

Layerscape LX2160A BSP v0.5



Contents

Chapter 1 About this document.....	11
Chapter 2 Acronyms and abbreviations.....	13
Chapter 3 Release notes.....	17
3.1 What's New.....	17
3.2 Components.....	17
3.3 Component location.....	20
3.4 Feature Support Matrix.....	21
3.5 Open, Fixed, and Closed Issues.....	24
Chapter 4 LX2160A BSP user guide.....	32
4.1 LSDK Quick Start.....	32
4.1.1 LSDK Quick Start Guide for LX2160ARDB.....	32
4.1.1.1 Introduction.....	32
4.1.1.2 Host system requirements.....	32
4.1.1.3 Download and assemble LSDK images.....	33
4.1.1.4 Deploy LSDK images on board.....	34
4.1.1.4.1 LX2160ARDB reference information.....	35
4.1.1.4.2 Option 1 - Deploy LSDK images using removable storage device.....	41
4.1.1.4.3 Option 2 - Deploy LSDK images directly to the storage device on a board.....	42
4.1.1.5 Bringing up DPPA2 network interfaces.....	45
4.1.1.5.1 Use Linux commands to list network interfaces.....	45
4.1.1.5.2 Use restool wrapper scripts to list DPAA2 objects.....	45
4.1.1.5.3 Add and destroy network interfaces.....	46
4.1.1.5.4 Save configuration to a custom DPL file (Optional).....	47
4.1.1.5.5 Assign IP addresses to network interfaces.....	47
4.2 How to build LSDK with Flexbuild.....	48
4.3 Procedure to Run Secure Boot.....	56
4.3.1 Prepare board for Secure Boot.....	56
4.3.2 Running secure boot on target platforms.....	57
4.3.3 Steps to run Chain of Trust with Confidentiality.....	58
4.4 LSDK Memory Layout.....	59
4.5 Build tools.....	61
Chapter 5 Bootloaders.....	63
5.1 General boot flow.....	63
5.2 Trusted Firmware – A (TF-A).....	64
5.2.1 Introduction.....	64
5.2.2 TF-A boot flow	64
5.2.3 Secure boot with TF-A boot.....	65
5.2.4 TF-A compilation and deployments steps.....	66
5.3 U-Boot.....	66
5.3.1 LSDK U-Boot uses distro boot feature.....	66
5.3.2 LSDK U-Boot flash image feature	69
5.4 UEFI.....	70

5.4.1 Introduction.....	72
5.4.2 UEFI overview.....	72
5.4.3 LSDK distro boot with UEFI.....	74
5.4.4 Product Execution.....	78
5.4.4.1 LX2160ARDB.....	79
5.4.4.2 LS1043ARDB.....	80
5.4.4.3 LS1046ARDB.....	81
5.4.4.4 LS2088ARDB.....	82
5.4.5 LSDK Distro Boot Logs.....	83
5.4.6 PXE Boot.....	86
5.4.6.1 Creating the PXE Boot Setup.....	86
5.4.6.2 Installing the Kernel.....	88
5.5 Migration guide for new boot flow (with TF-A).....	90
5.5.1 General boot flow.....	90
5.5.2 Traditional boot flow with PPA.....	90
5.5.3 New boot flow with TF-A.....	91
5.5.3.1 New flash images.....	92
5.5.3.1.1 bL2_<boot_mode>.pbl.....	93
5.5.3.1.2 FIP binary.....	93
5.5.3.2 Changes in flash layout.....	94
5.5.3.3 DDR FIP image (only required for LX2160A)	94
5.5.3.4 Changes in U-Boot.....	95
Chapter 6 Security.....	96
6.1 Secure boot.....	96
6.1.1 Hardware Pre-Boot Loader (PBL) based platforms.....	96
6.1.1.1 Introduction.....	96
6.1.1.2 Secure boot process.....	96
6.1.1.3 Pre-boot phase.....	97
6.1.1.4 ISBC phase.....	98
6.1.1.4.1 Flow in the ISBC code.....	98
6.1.1.4.2 Super Root Keys (SRKs) and signing keys.....	98
6.1.1.4.3 Key revocation.....	99
6.1.1.4.4 Alternate image support.....	99
6.1.1.4.5 ESBC with CSF header.....	100
6.1.1.5 ESBC phase.....	100
6.1.1.5.1 Boot script.....	101
6.1.1.6 Next executable (Linux phase).....	105
6.1.1.7 Product execution.....	106
6.1.1.7.1 Introduction.....	106
6.1.1.7.2 Chain of Trust with confidentiality.....	107
6.1.1.8 Troubleshooting.....	111
6.1.1.9 CSF Header Data Structure.....	111
6.1.1.10 ISBC Validation Error Codes.....	121
6.1.1.11 ESBC Validation Error Codes.....	125
6.1.1.12 Trust Architecture and SFP Information.....	126
6.1.2 Service Processor (SP) Based Platforms.....	127
6.1.2.1 Secure Boot Introduction.....	127
6.1.2.1.1 Secure Boot process.....	129
6.1.2.2 ISBC Phase.....	130
6.1.2.2.1 ISBC for PBI validation.....	130
6.1.2.2.2 ISBC for Boot1 (Boot Loader 1) validation.....	131
6.1.2.3 ESBC Phase.....	131
6.1.2.3.1 esbc_validate command.....	132

6.1.2.3.2 esbc_halt command.....	132
6.1.2.3.3 blob enc command.....	132
6.1.2.3.4 blob dec command.....	132
6.1.2.3.5 Boot Script.....	133
6.1.2.4 Next executable phase.....	135
6.1.2.5 Product Execution.....	135
6.1.2.5.1 Introduction.....	136
6.1.2.5.2 Chain of Trust with confidentiality.....	137
6.1.2.6 PBI structure.....	140
6.1.2.7 CSF header structure definition.....	141
6.1.2.8 CSF header structure definition.....	148
6.1.2.9 Secure boot specific RCW fields.....	154
6.1.2.10 ISBC error codes.....	155
6.1.2.11 ESBC error codes.....	162
6.1.2.12 Troubleshooting.....	164
6.1.3 Code Signing Tool.....	164
6.1.3.1 Key generation.....	165
6.1.3.1.1 gen_keys.....	165
6.1.3.1.2 gen_otpmk_drbg.....	167
6.1.3.1.3 gen_drv_drbg.....	168
6.1.3.2 Header creation.....	169
6.1.3.2.1 uni_pbi.....	169
6.1.3.2.2 uni_sign.....	172
6.1.3.3 Signature generation.....	176
6.1.3.3.1 gen_sign.....	178
6.1.3.3.2 sign_embed.....	179
6.2 IMA-EVM on LS Trust Architecture SoCs.....	180
6.2.1 Introduction.....	180
6.2.2 Secure key and its blob.....	181
6.2.3 EVM Key on user keyrings.....	181
6.2.4 Enabling secure key and encrypted key in kernel config.....	182
6.2.5 Enabling IMA EVM integrity options in kernel image.....	182
6.2.6 Modes of operation in IMA EVM.....	182
6.2.7 IMA EVM feature use cases.....	183
6.2.8 Appendix A: Steps to enable IMA EVM using flex builder.....	184
6.2.9 Appendix B: Standalone steps to enable IMA EVM.....	184
6.2.10 Appendix C: Steps to verify IMA EVM feature.....	186
6.2.11 Appendix - D: Points to remember.....	187
6.3 Trusted Execution (OP-TEE).....	187
6.3.1 Introduction.....	187
6.3.1.1 Support Platform.....	187
6.3.1.2 Test Sequence.....	187
6.4 Fuse Provisioning User Guide.....	188
6.4.1 Introduction.....	188
6.4.2 Fuse Programming Scenarios.....	189
6.4.2.1 Fuse Provisioning during OEM Manufacturing.....	189
6.4.3 Fuse Provisioning Utility.....	190
6.4.3.1 Fuse file structure.....	191
6.4.3.2 Sample input file for fuse provisioning tool.....	191
6.4.4 Steps to build fuse provisioning firmware image.....	193
6.4.5 Deploy and run fuse provisioning.....	193
6.4.5.1 Enable POVDD for SFP.....	193
6.4.5.2 Deploy firmware image on board.....	194
6.4.5.3 Run firmware image on board.....	194
6.4.6 Validation.....	194

6.4.7 Error Codes.....	194
6.5 PKCS#11 and Secure Object Library	196
6.5.1 Introduction.....	196
6.5.2 Supported APIs.....	198
6.5.2.1 PKCS#11 Library – libpkcs11.so.....	198
6.5.2.2 Secure Object Library – libsecure_obj.so.....	200
6.5.3 Integrating applications with Secure Object.....	203
6.5.3.1 Using PKCS#11 APIs.....	203
6.5.3.2 Using Secure Object APIs.....	203
6.5.3.3 Applications using OpenSSL APIs.....	203
6.5.3.3.1 Secure Object Library based OpenSSL Engine (libeng_secure_obj).....	204
6.5.3.3.2 PKCS#11 based OpenSSL Engine (Third party OpenSC/libp11).....	206
6.5.4 Board Bootup & Running applications.....	207
6.5.4.1 Board Bootup.....	207
6.5.4.2 Running applications.....	208
6.5.4.2.1 sobj_app.....	209
6.5.4.2.2 pkcs11_app.....	211
6.5.4.2.3 mp_app.....	218
6.5.4.2.4 mp_verify.....	218
6.5.4.2.5 sobj_eng_app.....	219
6.5.4.2.6 thread_test.....	220
6.5.5 Validation.....	220
6.5.6 Appendix.....	221
 Chapter 7 Linux kernel.....	 224
7.1 Configuring and building.....	226
7.2 Device Drivers.....	228
7.2.1 CAAM Direct Memory Access (DMA).....	229
7.2.2 Enhanced Secured Digital Host Controller (eSDHC).....	231
7.2.3 IEEE 1588.....	235
7.2.4 Integrated Flash Controller (IFC).....	238
7.2.4.1 Integrated Flash Controller NOR Flash User Manual.....	238
7.2.4.2 Integrated Flash Controller NAND Flash User Manual.....	243
7.2.5 PL011 Universal Asynchronous Receiver/Transmitter (UART).....	250
7.2.6 Flex Serial Peripheral Interface (FSPI).....	255
7.2.7 PCI Express Interface Controller	257
7.2.7.1 PCIe Linux Driver.....	257
7.2.7.2 PCIe Advanced Error Reporting User Manual.....	260
7.2.7.3 PCIe Remove and Rescan User Manual.....	262
7.2.8 Flex Serial Peripheral Interface (FSPI).....	263
7.2.9 Queue Direct Memory Access Controller (qDMA).....	265
7.2.10 Real Time Clock (RTC).....	268
7.2.11 Serial Advanced Technology Attachment (SATA).....	270
7.2.12 Security Engine (SEC).....	272
7.2.13 Universal Serial Bus Interfaces.....	285
7.2.13.1 USB 3.0 Controller (DesignWare USB3).....	286
7.2.13.2 USB 2.0 Controller	298
7.2.14 Watchdog.....	302
 Chapter 8 QorIQ networking technologies.....	 304
8.1 Summary of networking technologies.....	304
8.2 DPAA2-specific Software.....	304
8.2.1 DPAA2 Software Overview.....	304

8.2.1.1 Introduction.....	304
8.2.1.2 DPAA2 Hardware.....	305
8.2.1.2.1 Introduction.....	305
8.2.1.2.2 DPAA2 hardware.....	305
8.2.1.2.3 LS2088A block diagram.....	306
8.2.1.3 DPAA2 Linux Software.....	307
8.2.1.3.1 Introduction.....	308
8.2.1.3.2 Linux and DPAA2.....	308
8.2.1.3.3 DPAA2, Management Complex, and drivers.....	309
8.2.1.3.4 DPAA2 and plug-and-play.....	309
8.2.1.3.5 Datapath layout files and restool.....	310
8.2.1.4 DPAA2 Networking Subsystem Deeper Dive.....	310
8.2.1.4.1 DPAA2 hardware abstraction example.....	311
8.2.1.4.2 Management Complex: How DPAA2 objects are created and managed.....	318
8.2.1.4.3 Objects and topology.....	325
8.2.1.4.4 AIOP in DPAA2.....	327
8.2.2 DPAA2 Quick Start Guide	328
8.2.2.1 Data Path Resource Containers.....	328
8.2.2.1.1 Creating DPRCs.....	328
8.2.2.1.2 DPRCs and Hot Plug.....	328
8.2.2.2 Key Release Files: RCW, DPC and DPL.....	329
8.2.2.2.1 RCW.....	329
8.2.2.2.2 Data path configuration file (DPC).....	329
8.2.2.2.3 Data path layout file (DPL).....	330
8.2.2.3 Linux Ethernet.....	333
8.2.2.3.1 Features overview.....	333
8.2.2.3.2 Compiling and selecting Kconfig options.....	333
8.2.2.3.3 Creating a DPAA2 network interface.....	334
8.2.2.3.4 DPAA2 Ethernet features.....	339
8.2.2.4 Setting up Ethernet Switch Capability.....	352
8.2.2.4.1 Ethernet Switch overview.....	352
8.2.2.4.2 Switch object creation.....	352
8.2.2.4.3 Setting up the driver.....	355
8.2.2.4.4 Commands supported.....	356
8.2.2.5 Setting Up Edge Virtual Bridge Capability.....	358
8.2.2.5.1 EVB overview.....	358
8.2.2.5.2 EVB object creation.....	358
8.2.2.5.3 Setting up the EVB driver.....	361
8.2.2.5.4 EVB commands supported.....	362
8.2.2.5.5 Forwarding methods overview.....	363
8.2.2.6 Security Engine (SEC).....	366
8.2.2.7 Decompression Compression Engine (DCE).....	378
8.2.3 DPAA2 Standard Linux Documentation.....	378
8.2.3.1 Kernel Documentation Directory.....	379
8.2.3.2 DPAA2 Resource Management Tool (restool) User Manual	385
8.2.3.2.1 DPRC commands.....	385
8.2.3.2.2 DPNI Commands.....	394
8.2.3.2.3 DPIO Commands.....	398
8.2.3.2.4 DPSW Commands.....	401
8.2.3.2.5 DPBP Commands.....	404
8.2.3.2.6 DPCON Commands.....	406
8.2.3.2.7 DPCI Commands.....	408
8.2.3.2.8 DPSECI Commands.....	410
8.2.3.2.9 DPDMUX Commands.....	412
8.2.3.2.10 DPMCP Commands.....	415

8.2.3.2.11 DPMAC Commands.....	417
8.2.3.2.12 DPDCEI Commands.....	419
8.2.3.2.13 DPAIOP Commands.....	421
8.2.4 DPAA2 User Manual.....	423
8.2.5 DPAA2 API Reference Manual.....	423
8.2.6 Backplane Support on Layerscape.....	423
8.2.6.1 Overview.....	423
8.2.6.1.1 10GBase-KR Support on Layerscape Platforms.....	423
8.2.6.1.2 Physical Layer Signaling System.....	424
8.2.6.1.3 Auto-negotiation.....	424
8.2.6.1.4 Link Training.....	424
8.2.6.2 Enable Backplane Support on Layerscape.....	424
8.2.6.2.1 Setup.....	424
8.2.6.2.2 Enable Backplane Connection from MC.....	425
8.2.6.2.3 Enable Backplane Support in Linux Kernel.....	425
8.2.6.2.4 SerDes Setup.....	427
8.2.6.2.5 Board Configuration.....	427
8.2.6.3 Use Cases.....	428
8.2.7 Soft Parser Support.....	428
8.2.7.1 Soft Parser Configuration Tool.....	428
8.2.7.1.1 Introduction.....	428
8.2.7.1.2 Defining a custom protocol.....	428
8.2.7.1.3 Expressions.....	438
8.2.7.1.4 FAF – frame attribute flags.....	447
8.2.7.1.5 Subroutines support.....	451
8.2.7.1.6 SP Hardware configuration file.....	452
8.2.7.1.7 Tips and recommendations.....	455
8.2.7.1.8 Limitations.....	456
8.2.7.1.9 Running the Soft Parser tool.....	457
8.2.7.1.10 Output of the SPC tool.....	458
8.2.7.2 SPC on DPAA 2.x Based Platforms.....	458
8.2.7.2.1 Introduction.....	458
8.2.7.2.2 Applying Soft Parser Blob at U-Boot command line.....	460
8.2.7.2.3 Limitations.....	462

Chapter 9 Linux user space..... 463

9.1 Libraries.....	463
9.1.1 OpenSSL.....	463
9.1.1.1 Overview.....	463
9.1.1.2 Manual Build of OpenSSL with Cryptodev Engine Support.....	465
9.1.1.3 Hardware Offloading with OpenSSL.....	466
9.1.1.4 TLS Ciphersuites and TLS Protocol Versions.....	469
9.1.2 Runtime Assembler Library Reference.....	474
9.1.2.1 Runtime Assembler Library Reference.....	474
9.2 Data Plane Development Kit (DPDK).....	474
9.2.1 Introduction	475
9.2.1.1 Platform-specific Details.....	475
9.2.1.2 References.....	477
9.2.2 DPDK Overview.....	477
9.2.2.1 DPDK Platform Support.....	477
9.2.2.2 DPAA: Supported DPDK Features.....	479
9.2.2.3 DPAA2: Supported DPDK Features.....	480
9.2.2.4 PPFE supported DPDK features.....	480
9.2.3 Build DPDK.....	481

9.2.3.1 Build DPDK using Flexbuild.....	481
9.2.3.2 Standalone build of DPDK Libraries and Applications.....	483
9.2.3.3 DPDK based Packet Generator.....	487
9.2.3.4 Build OVS-DPDK using Flexbuild.....	488
9.2.3.5 Virtual machine (VM or guest) images.....	489
9.2.4 Executing DPDK Applications on Host.....	489
9.2.4.1 Prerequisites for running DPDK Applications.....	489
9.2.4.2 Booting up the Target board.....	490
9.2.4.3 Prerequisites for running DPDK Applications.....	492
9.2.4.3.1 Test Environment Setup.....	492
9.2.4.3.2 Generic Setup - DPAA.....	493
9.2.4.3.3 Generic Setup - DPAA2.....	495
9.2.4.3.4 Generic Setup - PPFE.....	496
9.2.4.3.5 DPAA2: Multiple parallel DPDK Applications.....	496
9.2.4.4 DPDK example applications.....	497
9.2.4.5 Command interface (CMDIF) demo application.....	505
9.2.5 OVS-DPDK and DPDK in VM with VIRTIO Interfaces.....	507
9.2.5.1 Generic steps.....	508
9.2.5.2 Configuring OVS	508
9.2.5.3 Launch Virtual Machine.....	511
9.2.5.4 Accessing virtual machine console.....	513
9.2.5.5 Launching two virtual machines.....	513
9.2.5.6 Running DPDK applications in VM.....	514
9.2.5.7 Multi Queue VIRTIO support.....	515
9.2.6 Enabling DPAA2 direct assignment for DPDK.....	517
9.2.6.1 Launch Virtual Machine.....	517
9.2.6.2 Accessing the virtual machine console.....	520
9.2.6.3 Running DPDK applications with direct device assignments.....	521
9.2.7 DPDK on Docker.....	521
9.2.7.1 Docker Overview.....	521
9.2.7.2 DPAA1-Platform.....	521
9.2.7.2.1 Running the Docker Container.....	522
9.2.7.2.2 Running the DPDK Application.....	522
9.2.7.3 DPAA2-Platform.....	523
9.2.7.3.1 Traffic Multiplexer/De-Multiplexer.....	523
9.2.7.3.2 Docker's Resource Setup.....	525
9.2.7.3.3 Running the Docker Container.....	527
9.2.7.3.4 Running the DPDK Application.....	528
9.2.7.3.5 Example Configuration: Using DPDMUX.....	528
9.2.7.3.6 Example Configuration: Using DPSW.....	530
9.2.8 Known Limitations and Future Work.....	532
9.2.9 Troubleshooting.....	533
9.2.10 DPDK Performance Reproducibility Guide.....	534

Chapter 10 Virtualization..... 542

10.1 KVM/QEMU User Guide and Reference.....	542
10.1.1 KVM/QEMU Overview.....	542
10.1.1.1 Using QEMU and KVM.....	543
10.1.1.1.1 Overview of Using QEMU.....	543
10.1.1.1.2 Virtual Machine Memory.....	545
10.1.1.1.3 Virtual network interfaces.....	546
10.1.1.1.4 Virtual block devices.....	546
10.1.1.1.5 Direct assigned devices	546
10.1.1.1.6 VMs and the Linux Scheduler.....	548

10.1.1.2 Virtual Machine Overview.....	549
10.1.1.3 Introduction to KVM and QEMU.....	550
10.1.1.4 Device Tree Overview.....	551
10.1.1.5 References.....	551
10.1.1.6 For More Information.....	552
10.1.1.7 Virtual machine reference.....	552
10.1.1.7.1 VM Overview.....	552
10.1.1.7.2 Memory Map of Virtual I/O Devices.....	552
10.1.1.7.3 Virtual machine state at initialization.....	552
10.1.1.7.4 Virtual CPUs.....	553
10.1.1.7.5 VGIC.....	553
10.1.2 Configuring and Building.....	554
10.1.2.1 Overview.....	554
10.1.2.2 Quick Start - Recommended Configuration Options.....	554
10.1.2.3 Host Kernel: Enabling KVM.....	555
10.1.2.4 Host Kernel: Enabling Virtual Networking.....	556
10.1.2.5 Host kernel: Enabling DPAA2 direct assignment	556
10.1.2.6 Host kernel: Enabling PCIE direct assignment	556
10.1.2.7 Guest kernel: Enabling console.....	557
10.1.2.8 Guest Kernel: Enabling Network and Block Virtual I/O.....	557
10.1.2.9 Building kernel with KVM support using flexbuild.....	557
10.1.2.10 Building QEMU.....	558
10.1.2.11 Creating a host Linux root filesystem.....	559
10.1.2.12 Creating a guest Linux root filesystem.....	559
10.1.3 KVM/QEMU How-to's.....	560
10.1.3.1 Quick-start Steps to Build and Deploy KVM.....	560
10.1.3.2 Quick-start Steps to Run KVM Using Hugetlbfs.....	560
10.1.3.3 How to Use Virtual Network Interfaces Using Virtio.....	562
10.1.3.4 How to use vhost-net with virtio.....	563
10.1.3.5 How to Use Virtual Disks Using Virtio.....	564
10.1.3.6 How to use virtual disks using virtio-blk-dataplane.....	566
10.1.3.7 How to use DPAA2 direct assignment without scripts	566
10.1.3.8 How to use DPAA2 direct assignment with scripts	568
10.1.3.9 How to use PCIE direct assignment.....	574
10.1.3.10 Passthrough of USB Devices	575
10.1.3.11 Debugging: How to Examine Initial Virtual Machine State with QEMU.....	575
10.1.3.12 Debugging: How to Profile Virtualization Overhead with KVM.....	577
10.1.3.13 Debugging virtual machines.....	578
10.1.3.13.1 QEMU Monitor.....	578
10.1.3.13.2 QEMU GDB Stub.....	579
10.2 Linux Containers User Guide.....	580
10.2.1 Introduction to Linux Containers.....	580
10.2.1.1 NXP LXC Release Notes.....	580
10.2.1.2 Overview.....	581
10.2.1.3 Comparing LXC and Libvirt.....	582
10.2.1.4 For Further Information.....	582
10.2.2 More Details.....	583
10.2.2.1 LXC: Command Reference.....	583
10.2.2.2 LXC: Configuration Files.....	584
10.2.2.3 LXC: Templates.....	585
10.2.2.4 Containers with Libvirt.....	585
10.2.2.5 Linux Control Groups (cgroups).....	587
10.2.2.6 Linux Namespaces.....	588
10.2.2.7 POSIX Capabilities.....	588
10.2.3 LXC How To's.....	589

10.2.3.1 LXC: Getting Started (with a Busybox System Container).....	589
10.2.3.2 LXC: How to configure non-virtualized networking (lxc-no-netns.conf).....	592
10.2.3.3 LXC: How to assign a physical network interface to a container (lxc-phys.conf).....	594
10.2.3.4 LXC: How to configure networking with virtual Ethernet pairs (lxc-veth.conf).....	595
10.2.3.5 LXC: How to configure networking with macvlan (lxc-macvlan.conf).....	596
10.2.3.6 LXC: How to configure networking using a VLAN (lxc-vlan.conf).....	598
10.2.3.7 LXC: How to monitor containers.....	600
10.2.3.8 LXC: How to modify the capabilities of a container to provide additional isolation.....	601
10.2.3.9 LXC: How to use cgroups to manage and control a containers resources.....	602
10.2.3.10 LXC: How to run an application in a container with lxc-execute.....	603
10.2.3.11 LXC: How to run an unprivileged container.....	604
10.2.3.12 LXC: How to run containers with Seccomp protection.....	606
10.2.4 Libvirt.....	608
10.3 Docker Containers.....	619
10.3.1 Introduction to Docker Containers.....	619
10.3.1.1 Overview.....	619
10.3.2 Docker How To's.....	619
10.3.2.1 Running a webserver container.....	619
Chapter 11 NFV OpenStack Compute Node.....	623
11.1 Network Function Virtualizatin OpenStack Compute Node.....	623
11.1.1 OpenStack Nova overview.....	623
11.1.2 Build OpenStack.....	623
11.2 Board bring up.....	623
11.2.1 Bringing up Openstack Controller (OS) Container.....	623
11.2.2 LX2160ARDB board bring up.....	624
11.2.3 Installing OpenStack Nova (Newton).....	625
Chapter 12 Power Management.....	626
12.1 Power Management User Manual.....	626
12.2 Power Management User Manual.....	628
12.3 CPU Frequency Switching User Manual.....	630
12.4 Thermal Management User Manual.....	632
12.5 System Monitor.....	635
12.5.1 Power Monitor User Manual.....	635
12.5.2 Thermal Monitor User Manual.....	638
Chapter 13 Benchmarking guidelines.....	640
13.1 Coremark.....	640
13.1.1 Test Environment.....	640
13.1.2 Test Procedure.....	641
13.2 Dhrystone.....	642
13.2.1 Test Environment.....	642
13.2.2 Test Procedure.....	643
13.3 EEMBC.....	645
13.3.1 Test Environment.....	645
13.3.2 Test Procedure.....	648
13.4 LMBench.....	656
13.4.1 Test Environment.....	656
13.4.2 Test Procedure.....	657

Chapter 1

About this document

About LX2160A BSP v0.5

NXP Layerscape LX2160A BSP v0.5 Software Release is a complete Linux kit for NXP QorIQ ARM-based LX2160A SoC and the reference and evaluation boards LX2160ARDB

It is a *hybrid form* of a Linux distribution because it combines the following major components to form a complete Linux system.

- NXP firmware components including:
 - Trusted Firmware-A (TF-A), a reference implementation of secure world software for Armv8-A.
 - NXP peripheral firmware components for DPAA2.
- NXP boot loaders. Two are offered:
 - U-Boot, based on denx.de plus patches.
 - UEFI, based on TianoCore.
- NXP Linux kernel, based on kernel.org upstream plus patches.
- NXP added user space components.
- Ubuntu standard user space file set (user land), including compilers and cross compiler.

The use of Ubuntu user land is what makes LX2160A BSP v0.5 a hybrid. It is not entirely an Ubuntu distribution because it uses an NXP kernel, but it still uses Ubuntu user space files. This hybrid is possible because NXP ARM SoC's are standards-based so programs like bash and thousands of others run without being recompiled.

The benefit of using Ubuntu user land is the easy availability of thousands of standard Linux user space packages. The experience of using the LX2160A BSP v0.5 is similar to using Ubuntu, but the kernel, firmware, and some special NXP packages are managed separately.

Accessing LX2160A BSP v0.5

LX2160A BSP v0.5 is distributed via FAEs

There are two ways to use the LX2160A BSP v0.5, as an integration and as a source of individual components.

LX2160A BSP v0.5 as an integration

With `flexbuild` component. You can clone it and run a script to create and install LX2160A BSP v0.5 onto a mass storage device as an integration, ready for use on an NXP reference or evaluation board. You can build NXP components from source using a script called `flex-builder` or install from binaries of NXP components using `flex-installer`. See [LX2160A BSP user guide](#) on page 32.

LX2160A BSP v0.5 as components

The component location chapter shows git repositories for individual components, for example the LX2160A BSP v0.5 Linux kernel. If you clone and examine this git, you will see a conventional kernel source tree. You can compile the kernel using `make` in the normal way, like a kernel.org kernel. However, notice the configuration fragment in `arch/arm64/configs`. See [Linux kernel](#) on page 224.

Having git access to components is helpful if you assemble your own Linux distribution or wish to form a hybrid with a user land other than Ubuntu's.

LX2160A BSP v0.5 git tags

LX2160A BSP v0.5 git repositories use git tags to indicate component revisions that have been release tested together. Use the `git tag` command to examine them and chose a tag to check out.

LX2160A BSP v0.5 Relies on Mass Storage Devices

Ubuntu user land is very convenient for evaluation because it is possible to use the command `apt-get install` on the standard Ubuntu components you need. It also provides native development tools.

But this richness means that the user space file is large, too large for RAM disks.

Therefore, LX2160A BSP v0.5 requires installation to and use of a mass storage device such as

- SD card
- USB flash drive
- USB hard drive
- SATA drive, spinning, or SSD (for boards with a SATA port)
- eMMC flash (when available on board)

LX2160A BSP v0.5 provides scripts that populate a mass storage device with the needed files. These scripts can run on a Linux PC. It is especially simple to use an SD card or USB flash drive because they are the easiest to move between a Linux PC and the NXP board.

Chapter 2

Acronyms and abbreviations

Term	Definition
AH	Authentication Header (RFC 4302) – a network protocol designed to provide authentication services in IPv4 and IPv6.
ACL	Access Control List
AMP	Asynchronous Multi-Processing, running multiple operating system images on different processors of the same machine without virtualization.
API	Application Programming Interface
ARP	Address Resolution Protocol
CAAM	Cryptographic Acceleration and Assurance Module
CCSR	Configuration and Control Status Register
CPU	Central Processing Unit, also known more generally as "Processor"
DCD	Device Configuration Data
DCE	Data Compression/Decompression Engine
DMA	Direct Memory Access
DSK	Device Secret Key
DTB	Device Tree Blob—the binary representation of device trees
DTS	Device Tree Syntax—the textual representation of device trees
DUT	Device Under Test
ESP	Encapsulating Security Payload (RFC 4303) – a network protocol designed to provide a mix of security services in IPv4 and IPv6.
EVB	Edge Virtual Bridge
FDB	Forwarding Data Base
FUID	Freescale Unique ID
GPP	General Purpose Processor
Guest/VM	The term 'Guest' is used for Linux running inside the virtual machine(s) that are in turn running over Host Linux operating system. VM and Guest have been used interchangeably in this guide.
Table continues on the next page...	

Table continued from the previous page...

Term	Definition
GUEST_CONSOLE_TELNET_PORT	Telnet port for accessing guest console of VM.
HAL	Hardware Abstraction Library
HIF	Host Interface
HSM	Hardware security modules
IBR	Internal Boot ROM
IP_ADDR_BRD	This term is used for LS1088ARDB and LS2088ARDB IP address.
IP_ADDR_IMAGE_SERVER	This term is used for IP address of the machine on which all the software images are kept.
IPC	Inter-Process Communication, can be interpreted as being communication between distinct application execution flows or between distinct hardware processing units.
inbound (traffic)	Encrypted traffic which is coming from an unprotected interface. This traffic will be terminated on the CPU.
IPFwd	IPv4 Forward
IPSec	IP Security – a communication standard defined and refined by several public RFCs (such as RFC-2401 and RFC-4301) where hosts exchange encrypted IP data packets.
IPSec Tunnel	A communication convention between two network gateways to IPSec process certain network traffic in a particular way. An IPSec tunnel has two endpoints (which are the gateways), a clearly delimited set of encryption and authentication methods, keys, encapsulation headers and security policies, which define the traffic that is sent through the tunnel.
ISBC	Internal Secure Boot Code
ISR	Interrupt Status Register
ITF	Intent to Fail
ITS	Intent to Secure
KASLR	Kernel Address Space Layout Randomization
KVM	Kernel-based Virtual Machine - A Linux kernel module that allows a user space program access to the hardware virtualization features of NXP processors.
LIODN	Logical I/O Device Number
MC	Management Complex
NAT	Network Address Translation

Table continues on the next page...

Table continued from the previous page...

Term	Definition
OEM	Original Equipment Manufacturer
OS	Operating System
OUID	OEM Unique ID
outbound (traffic)	Clear traffic which is coming from a software application which generates traffic that must be encrypted and forwarded via an unprotected interface.
PAMU	Peripheral Access Management Unit
PBL	Pre-Boot Loader
PCD	Parse, Classify, Distribute – a software architecture concept in NXP DPAA drivers which allows the user to configure the DPAA hardware (FMan) to do frame parsing, classification or distribution on a series of frame queues.
PDCP	Packet Data Convergence Protocol – It is one of the layers of the Radio Traffic Stack in UMTS/LTE and performs IP header compression and decompression, transfer of user data and maintenance of sequence numbers for Radio Bearers which are configured for lossless serving radio network subsystem (SRNS) relocation.
PME	Pattern Matcher Engine
PKCS	Public-Key Cryptography Standards
QEMU	Quick EMUlator - A hosted hypervisor that performs hardware virtualization.
RC	Route Cache
RCW	Reset Configuration Word
RFC	Request for Comments – a public document which describes a software standard.
RDB	Reference Design Board
SA	Security Association – a data record, defined by RFC 4301, which stores the information related to the IPSec processing needed for a specific network traffic type (such as encryption/decryption keys and algorithms, traffic endpoints description, authentication algorithms, and so on).
SAD	Security Association Database – the database holding all the valid SAs in a system.
SDK	Software Development Kit
SEC	Security Engine Coprocessor – a DPAA hardware block performing cryptographic acceleration and offloading hardware.
SFP	Secure Fuse Processor
SIP DIP	Source Internet Protocol and Destination Internal Protocol

Table continues on the next page...

Table continued from the previous page...

Term	Definition
SKMM	Secure Key Management Module
SMP	Symmetric Multi-Processing, running an operating system image on multiple CPUs simultaneously.
SNVS	Secure Non-Volatile Storage
SoC	System on a Chip, a chip integrating one or more processors and on-chip peripherals.
SP	Security Policy – a set of rules that network traffic must comply with in order to be eligible for IPSec processing.
SPD	Security Policy Database – the database storing all the SPs in a system.
SRE	Stateful Rule Engine
SRK	Super Root Key
SRKH	Super Root Key Hash
STP	Spanning Tree Protocol
SUI	String Under Inspection
TFTP_BASE_DIR	Base directory of TFTP server where all the images are kept.
TLB	Translation Lookaside Buffer
TTL	Time To Live
UDP	User Datagram Protocol
UID	Unique Device ID
UIO	User space I/O
VEB	Virtual Ethernet Bridge
VEPA	Virtual Ethernet Port Aggregator
VID	Voltage Identifier

Chapter 3

Release notes

3.1 What's New

What's New in LX2160A BSP v0.5

This release is based on LSDK18.12 release . Going forward LX2160A will be supported via LSDK releases as supported target. All LSDK feature will be regressed against LX2160ARDB.

This release adopts conventions of LSDK for easy migration. Some of changes that are coming in with this release are:

- U-boot v2018.09 version
- Linux 4.14.83 version
- TFA firmware (BSP0.5 onwards) replaces PPA firmware (BSP0.4) as trusted firmware in U-Boot boot flow.
- Bootflow is changed. Refer [Bootloaders](#) on page 63 for details.
- Flexbuilder replaces Yocto (BSP0.4) as Build System
- Ubuntu 18.04 replaces Yocto (BSP0.4) as userland filesystem.
 - BSP0.5 testing is done using Ubuntu userland. BSP0.5 also additionally provides a small RAM disk userland with the release.
 - Yocto based release will be available with next planned LSDK Yocto release in Q2-2019.
- MC upgrade to 10.14
- OVS-DPDK version upgrade to 2.10
- DPDK performance optimizations
- Refer [Open, Fixed, and Closed Issues](#) on page 24 for details about the issues.

3.2 Components

Processor Support

LX2160A Processor

Board Support

- LX2160ARDB board

Frequency Support

- Core: 2000MHz, DDR: 2900MT/s, Platform: 700MHz [DIMM , SoC part should support this]
- Core: 1900MHz, DDR: 2600MT/s, Platform: 600MHz [default]
- Core: 2000MHz, DDR: 2600MT/s, Platform: 800MHz [for 100G]
- Core: 2000MHz, DDR: 2400MT/s, Platform: 700MHz [for 25G]

Serdes Protocol Support

- Serdes1: 19, Serdes2: 5, Serdes3: 2 [default]
- Serdes1: 13, Serdes2: 3, Serdes3: 3 [100G feature on QDS]

- Serdes1: 7, Serdes2: 5, Serdes3: 2 [SGMII feature on QDS]
- Serdes1: 15, Serdes2: 3, Serdes3: 3 [50G feature on QDS]

Overall Component Summary

- Ubuntu userland
- Linux Kernel core, Virtualization and Linux Kernel Drivers
- Data Plane Development Kit (DPDK)
- Virtualization - OVS-DPDK
- Trust Firmware - A (TF-A)
- U-Boot Boot Loader
- Unified Extensible Firmware Interface (UEFI)
- Management Complex (MC) - DPAA2 Firmware

Ubuntu Userland

- Ubuntu host 18.04
- Toolchain: gcc: Ubuntu/Linaro 7.3.0-16ubuntu3~18.04, glibc-2.27, binutils-2.30-0, gdb-8.1
- Libvirt 4.0
- Linux Containers (LXC)
- OpenSSL 1.1
- QEMU 2.11

Linux Kernel Core and Virtualization

- LTS kernel 4.14.83
- ARM Cortex-A72 (AARCH64), Little Endian (default)
- Huge Pages (hugetlbfs)
- Timer
- SMP-boot
- Clock, UART, DDR4
- DSPI [QDS board only]
- eSDHC, eMMC, GIC, I2C, OCRAM
- PCIe-gen1(root-complex, endpoint), PCIe-gen3(root-complex), MSI
- USB, SATA, Flexspi access to NOR flash
- Networking interfaces: PCI-e1000, USXGMII, RGMII, XFI(10G), XLAUI(40G), 25G
- Networking interfaces: SGMII, 100G, 50G [QDS-board only]
- Management Complex Bus [DPAA2 processors]
- WRIOP, QBMAN
- MDIO
- PHY support: RGMII
- PHY support: SGMII [QDS-board only]

- SEC, QDMA
- System Memory Management Unit (SMMU)
- Virtualization: KVM, Guest Virtual Machines, Virtio-net, Docker Containers

Data Plane Development Kit (DPDK)

- Support of DPDK v17.11 as base
- Following DPDK Applications have been verified
 - l2fwd
 - l3fwd
 - l2fwd_crypto
 - ipsecgateway
- Fragmentation applications ip-reassembly and ip-fragmentation
- IPSEC protocol offload

Virtualization - OVS-DPDK

- OVS-DPDK 2.10
- OVS-DPDK working with vhost-virtio interfaces (Virtual Machines)

Trust Firmware - A (TF-A)

- Replaces PPA as trusted firmware and secure firmware

U-Boot Boot Loader

- U-Boot: 2018.09
- Unified memory map
- On ARM platforms, the U-Boot image includes the device tree
- Non-secure and Secure Boot (ESBC)
- Trusted Firmware-A (TF-A) integration.
- Boot from Flexspi NOR flash, SDCard
- Boot from NOR, NAND flash, QSPI, SDHC
- Clock, CPLD, UART, DDR4, DSPI, eSDHC, eMMC, GIC, I2C, OCRAM
- PCIe-gen1-rootcomplex, USB, SATA (one controller support)
- Flexspi access to NOR flash
- Management Complex Bus [DPAA2 processors]
- MDIO, PHY support
- Networking support using PCI-e1000, USXGMII, RGMII, XFI, XLAUI(40G), 25G
- Networking support using SGMII, 100G, 50G [QDS board only]
- Voltage ID (board specific)

UEFI bootloader [RDB board only]

- Non-secure
- Trusted Firmware-A : TFA

- Boot from Flexspi NOR flash
- A72 core, Timer, UART
- Clock, FPGA, DDR4
- SD, FAT32 filesystem
- I2C, RTC, SATA, USB
- DPAA2: XFI support
- SMP Linux boot via EFI_STUB on SD card
- MC High Mem Support

MC version 10.14.0

- Support for 25G, USXGMII, RGMII, XFI, XLAUI(40G), SGMII, 50G, 100G[one port]

Other Tools and Utilities

- Flexbuild
 - build tool
 - Support of iperf version 2.0.13a
 - Support of Soft Parser Configuration Tool
 - Support of TF-A
- RCW

3.3 Component location

The below table lists the component location.

Table 1. Component location

Component	CAF/github location	commit/tag
linux	https://source.codeaurora.org/external/qoriq/qoriq-components/linux/	lx2160a-early-access-bsp0.5
U-Boot	https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/	lx2160a-early-access-bsp0.5
cryptodev	https://source.codeaurora.org/external/qoriq/qoriq-components/cryptodev-linux/	LSDK-18.12
mc-utils	https://source.codeaurora.org/external/qoriq/qoriq-components/mc-utils/	LSDK-18.12
dpdk	https://source.codeaurora.org/external/qoriq/qoriq-components/dpdk/	lx2160a-early-access-bsp0.5
ovs-dpdk	https://source.codeaurora.org/external/qoriq/qoriq-components/ovs-dpdk/	lx2160a-early-access-bsp0.5
<i>Table continues on the next page...</i>		

Table 1. Component location (continued)

rcw	https://source.codeaurora.org/external/qorik/qorik-components/rcw/	LSDK-18.12
ddr-phy	https://github.com/NXP/ddr-phy-binary.git	LSDK-18.12
mc-binary	https://github.com/NXP/qorik-mc-binary	lx2160a-early-access-bsp0.5
openssl	https://source.codeaurora.org/external/qorik/qorik-components/openssl/	LSDK-18.12
cst	https://source.codeaurora.org/external/qorik/qorik-components/cst/	LSDK-18.12
atf	https://source.codeaurora.org/external/qorik/qorik-components/atf/	lx2160a-early-access-bsp0.5
restool	https://source.codeaurora.org/external/qorik/qorik-components/restool/	lx2160a-early-access-bsp0.5
dce	https://source.codeaurora.org/external/qorik/qorik-components/dce	LSDK-18.12
qbmman_userspace	https://source.codeaurora.org/external/qorik/qorik-components/qbmman_userspace/	LSDK-18.12
uefi	https://source.codeaurora.org/external/qorik/qorik-components/uefi/	LSDK-18.12
	https://source.codeaurora.org/external/qorik/qorik-components/edk2-platforms/	LSDK-18.12
	https://github.com/NXP/qorik-uefi-binary	LSDK-18.12
qorik-firmware-inphi	https://github.com/NXP/qorik-firmware-inphi	LSDK-18.12
flex-build	https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qorik-layerscape-arm-processors/qorik-lx2160a-reference-design-board:LX2160A-RDB?tab=Design_Tools_Tab	NA

3.4 Feature Support Matrix

The following tables show the features that are supported in this release.

Feature	Description
Flexspi	• Read/Write/Erase; Flexspi clock: 50MHz [Default] • Read in AHB mode with Octal command support • Write in IP mode • Erase size: 128KB (u-boot), 4KB (Linux) • Supported Flash device: MT35XU512G (LX2160ARDB) • Multiple CS support • 3-byte command address support added • Read/Write/Erase functionality verified on 100MHz FlexSPI clock
UART	• UART1, UART2 verified. • Default frequency: 115.2kbps
DDR	• Chip select interleaving, ECC support, SPD support • Supports UDIMM(18ASF1G72AZ-2G6B1) (LX2160ARDB) • Supports RDIMM • Default frequency: 2600 MT/s
PCI	• Ping/tftp on gen1 e1000 card in u-boot and Linux • Ping/scp on Gen3 X550 NIC under Linux • MSI-X on e1000 and X550 NIC • PCIe gen1 Endpoint support • Default RDB RCU supports • Note:>gen1 protocol requires RCW bits to be updated
SDHC	• Read/Write on class10 card (25MHz) and eMMC device
SATA	• One controller in u-boot • Four controllers in Linux
I2C	• RTC, EEPROM, Sensors • Default frequency: 100KHz
USB3.0	• USB1, USB2
DSPI [QDS]	• U-boot: Read/write on all 9 flashes-3 DSPI controller • Read/write on DSPI1, DSPI3 in Linux • Tested with: MT25QU128A, SST25WF, EN25S64
Virtualization	• KVM VMs, virtio-net, Docker Containers
Table continues on the next page...	

Table continued from the previous page...

RGMII (1G)	- Both ports enabled in u-boot, Linux - Support for TYPE_PHY Mode
XFI (10G)	Both ports enabled in u-boot, Linux NOTE XFI mode is now replaced with USXGMII mode on RDB board as default. Do ensure to upgrade to latest USXGMII phy firmware using steps mentioned in lx2-phy repo. See Component location on page 20 for lx2-phy repo location. XFI mode will only be limited support on QDS board
USXGMII	- U-boot / Linux: ping, iperf - Speed: 100M/1G/2.5G/10G - Default 10G mode on RDB - Modes: TYPE_FIXED, TYPE_PHY NOTE Old Boards were default shipped with Aquantia XFI phy firmware. Please ensure to upgrade to USXGMII phy firmware using steps mentioned in lx2-phy repo. See Component location on page 20 for lx2-phy repo location.
25G	- U-boot / Linux: ping, tftp, traffic with other link partners, back-to-back, loopback modules - Limitations: - RS-FEC is enabled by default on the 25G ports and cannot be disabled therefore RS-FEC must be enabled on the link partner for the 25G link to work - Only DPMAC.6 enabled in U-Boot 25G requires platform clock to be 700MHz

Table continues on the next page...

Table continued from the previous page...

XLAUI (40G)	U-boot / Linux: ping, tftp, netperf traffic validated on RDB board
50G	Two ports enabled in u-boot, Linux on QDS board [Fixed Link mode]
100G	One port enabled in u-boot, Linux on QDS board [Fixed Link mode]
DPAA2 networking (WRIOP, QBMAN)	- Restool configuration - Ping/ping flood/netperf, Change MAC, Large frames, S/G
SEC (DPAA2 interface & JobRing)	- Basic sanity of algorithms done using crypto self-test - DPSEC object creation with 16 queues - DPAA2 SEC CAAM JR, QI2 - selftest - Linux kernel IPsec - basic testing (ping between 2 back-to-back RDB boards, XFI ports) in tunnel and transport modes, both on Job Ring and DPSEC interfaces
QDMA	Basic sanity done in Linux
DPDK	- Basic applications like l2fwd, l3fwd, testpmd - L2fwd-crypto and ipsec-gw application - Fragmentation applications ip-reassembly and ip-fragmentation - Support for 16 cores and 10G/25G/40G/50G/100G interfaces - Support OVS-DPDK 2.10; vhost-virtio interfaces (Virtual Machines)

3.5 Open, Fixed, and Closed Issues

This section contains below tables:

- Table1: Open issues do not currently have a resolution. Workaround suggestions are provided where possible.
- Table2: Fixed/Closed Issues: Contains Issues which have a software fix that has been integrated into this Release or the issues are root cause and fix is outside the scope of this release.

Table 2. Open issues

ID	Description
QLINUX-10056	Some USB drive like Sony 16GB 3.0, Kingston 64GB 3.0, HP 64GB 3.0, Kingston 16GB 2.0 can't be found in Linux on LX2160ARDB.
<i>Table continues on the next page...</i>	

Table 2. Open issues (continued)

QLINUX-9809	While running PCIe stress test, sometimes AER correctable errors can be seen on console. As these errors are correctable, therefore it will not impact functionality.
QLINUX-9501	PCIe-gen3: Occasionally with large data transfers on long run, issues like rcu_preempt logs dump followed by Linux hang are observed on some cards. Debugging in-progress
DPDK-1518	There is performance drop when core0 is used for DPDK datapath. Workaround: Use other cores for DPDK datapath
QLINUX-10962	Linux ipfwd 2x25G performance has degraded as compared to bsp0.4. The issue will be fixed in next LSDK19.03 release

Table 3. Fixed and Closed issues

ID	Description	
DPDK-1355	There is initial packet drop on the 'first burst' of packets and after the initial packet drops there is no packet drop seen. Workaround to measure performance is to send few packets before starting measuring the performance. This can be seen as a 'system warming' burst." The issue is fixed in BSP0.5	Fixed
QUBOOT-4240	The issue is fixed in BSP0.2 Release CS4223 Phy Initialization from uboot solves the Link instabilities. In order to use the CS4223 Phy init from uboot (and not from EEPROM) - SW2[2] should be set to OFF.	Fixed
QUBOOT-4030	As per current u-boot code implementation, it expects unique phy address for each MAC. But if single phy is present on more than one MAC like IN112525phy, some commands like mdio will not display all MACs. This requires major framework change in u-boot, so no plan to fix.	It cannot be fixed
<i>Table continues on the next page...</i>		

Table 3. Fixed and Closed issues (continued)

QSDK-4973	Memtester failure is observed on some LX2 boards. The issue is root-caused to bad SoC early part. Workaround: Limiting voltage to 0.8V from 0.825 (default) to 0.8V resolves the issue on most of SoC parts. In BSP0.4, in u-boot RDB code, default env sets vid to 0.8V . For QDS boards: set switch SW9[1:3]='001' for 0.8V.	Fixed
QSDK-4709	LX2160AQDS: As per board design, flash connected at CS0 and CS1 can't be accessed simultaneously. So, only one Flexspi chip-select [CS0/CS1] is functional at a time in Linux/U-boot. QDS board hardware-issue: wont be fixed.	It cannot be fixed
<i>Table continues on the next page...</i>		

Table 3. Fixed and Closed issues (continued)

QSDK-5113	PCIe controller may not send the ACK to the received TLP in required time and on replay of the TLP it will send NAK For Workaround implementation, refer : U-Boot https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/commit/?h=lx2160a-early-access-bsp0.4&id=3ddd829eea3f76c4a7910319b6aca85ef0e4a769 https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/commit/?h=lx2160a-early-access-bsp0.4&id=dd43541d67eee1327551368e9e5ed6a1152c435a https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/commit/?h=lx2160a-early-access-bsp0.4&id=06765157f19ad2eeef57f4524369364f6b745c0f Linux: https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/commit/?h=lx2160a-early-access-bsp0.4&id=51a4b99d820af11d7763d9ffc4d7841a5f3f0b38 https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=082fc9ac50f038d5d3193d2a1be557502a7ec36d	Fixed
QSDK-5114	MSIx table entries of all the VFs are not accessible if the BAR1 size is less than the default value of 8MB For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/commit/?h=lx2160a-early-access-bsp0.4&id=57201cd58d86031b9ecc6d4a966184efc31c88e3	Fixed
Table continues on the next page...		

Table 3. Fixed and Closed issues (continued)

QSDK-5115	PCIe controller will report false Data Link Protocol error if it receives four or more back to back ACK. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=4548b3d5cf3a0d9d7177e44005bfb05f9530d2a https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=1c169051b009871730d0154969f5d99430355c71	Fixed
QSDK-5116	In PCIe-EP mode, hot-reset can generate false completer abort error on EP side. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=ca19abc0bc6a57ef052c20fccbe2dc974c6d0162	Fixed
QSDK-5117	False correctable replay timeout error reported by the PCIe controller. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=c0795d038e8a0de27a6937ae4aa889c4138c1409	Fixed
QSDK-5119	Hot reset sequence for RC is not upstreamable on Linux. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=aca6f1f46b609b3d3be24cb8009f438220d1669	Fixed

Table continues on the next page...

Table 3. Fixed and Closed issues (continued)

QSDK-5120	GPEX RC: AER interrupt handling is not PCIe specification compliant. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=06893cc7b971e8e564fa1fa47fcd16ad19a6f1ec https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=f9bc1914a086d3b6d7be95b97f89334a8b58da32 https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=5be0fc5c16f3588d138aee7165f54abdb2fba01d https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=017e6e9d7a3292070873dd88d5ee3c754e279119	Fixed
QSDK-5121	GPEX-EP: For Unsupported request error on inbound posted write transaction, PCIe controller reports advisory error instead of uncorrectable error message to RC. For Workaround implementation, refer: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=lx2160a-early-access-bsp0.4&id=3f08b342c13c2711a8bf97f06b9820d33b93970e	Fixed
QLINUX-9824	PCIe-gen3: Occasionally with large data transfers on long run, Malformed TLP PCIe error is observed. This leads to SError which might hangs the Linux. This has been root-caused to a hardware issue [TKT385917 contain the details].	It cannot be fixed

Table continues on the next page...

Table 3. Fixed and Closed issues (continued)

QLINUX-9823	PCIe controllers 1,2,4 and 6 reports false uncorrectable Data link protocol error. This is seen when EP accumulates ack packets 4 or more and send these ack packet in one go. This type of behavior is seen only with one typical EP device. The issue is root-cause as hardware issue. Workaround: If interrupt is received for DL protocol error, after sometime (1ms) check if REPLAY_NUM_ROLLOVER interrupt is set or not. If REPLAY_NUM_ROLLOVER is not set then ignore the DL protocol error (don't perform hot-reset) as it's due to false error. Otherwise perform the normal error recovery for real DL protocol error with hot-reset. patch workaround link: https://source.codeaurora.org/external/qoriq/qoriq-components/linux/commit/?h=linux-4.9-lx2160a-early-access&id=1c169051b009871730d0154969f5d99430355c71	It cannot be fixed
QLINUX-9803	On some LX2160ARDB boards after sometime, the RGMII Phy AR8035 may stop responding to MDIO commands. Traffic is not affected. Please refer to E-00005 from LX2160ARDB Errata documentation. Workaround is to use Fixed Link (DPC configuration). In RGMII Fixed link on low speed traffic there can be seen some frame loss on RGMII interfaces. Another proper fix is board rework as per board errata E-00005.	Fixed
QLINUX-9727	nfsboot failure observed with large file transfer: tcpdump analysis shows bad length packet; With tuning of nfsclient parameters by setting "rsize=1024,wsize,1024,timeo=600" in nfsroot of bootargs, 1.1 GB file transfer is verified. The issue is closed in BSP0.1-p2 release as it works fine with updated nfsclient settings.	Not an issue
Table continues on the next page...		

Table 3. Fixed and Closed issues (continued)

QLINUX-9637	Busy poll feature is not supported by the DPAA2 Ethernet driver. Starting with kernel 4.10, busy polling support is included in the network stack core, with no code needed on the driver side. For previous kernel versions, driver support is necessary, but the DPAA2 Ethernet driver doesn't include this support.	It cannot be fixed
QLINUX-9636	On LX2160A, Watchdog timer module is SBSA compliant. So, non-secure watchdog in Linux cannot reset the system.	It cannot be fixed
QLINUX-9562	LX2160ARDB - iperf performance using 40GE is low. Linux Kernel 4.9 has some performance issues related to SMMU. Workaround: is to use "iommu.passthrough=1" in bootargs. iperf3 application with '-P 16' does not create 16 different processes to be scheduled on different processor but rather it creates multiple flows of data. Switching back to iperf resolves the issue.	Not an issue
QLINUX-9416	On LX2160A RDB boards, 10G XFI copper ports (using AQR107 PHYs) with flow control enabled may freeze on the Rx side under certain conditions. More specifically, if high rate traffic is injected into a 10G port, it will attempt to transmit pause frames and after this no more ingress frames will be processed. Resolution is to upgrade Aquantia Phy firmware on boards. Meanwhile, as workaround, it is recommended to manually disable Tx pause frames for the 10G ports before using them in high-rate traffic scenarios: <code>ethtool -A <niX> tx off</code> .	Fixed

Chapter 4

LX2160A BSP user guide

4.1 LSDK Quick Start

4.1.1 LSDK Quick Start Guide for LX2160ARDB

4.1.1.1 Introduction

The following sections describe the procedure to download and assemble LSDK images and then to deploy these images on to LX2160ARDB. For more information on the different components of the board and on how to configure and boot the board, see [QorIQ LX2160A Reference Design Board Getting Started Guide](#).

4.1.1.2 Host system requirements

- Ubuntu 18.04 should be installed on the host machine.
- If this requirement is not fulfilled, see “Emulate Ubuntu 18.04 environment using Docker container” topic below.
- For root users, there is no limitation for the build. For non-root users, obtain sudo permission by running the command `sudoedit /etc/sudoers` and adding a line `<user-account-name> ALL=(ALL:ALL) NOPASSWD: ALL` in `/etc/sudoers`.
- To build the target Ubuntu userland for arm64/armhf arch, the user's network environment must have access to the remote Ubuntu official server.

Emulate Ubuntu 18.04 environment using Docker container (optional)

If a Linux distribution other than Ubuntu 18.04 is installed on the host machine, perform the following steps to create an Ubuntu 18.04 Docker container to emulate the environment.

- Install Docker on the host machine. See <https://docs.docker.com/engine/installation/> for information on how to install Docker on the host machine.
- To build the Ubuntu userland, the user's network environment must have *sudo* permission for Docker commands or the user must be added to a group called “docker” as specified below.

Change current group to “docker”

```
$ sudo newgrp - docker
```

NOTE

User can run the command `cut -d: -f1 /etc/group | sort` to check if *docker* group is included in the list of all the available groups.

Add your account to “docker” group

```
$ sudo usermod -aG docker <accountname>
$ sudo gpasswd -a <accountname> docker
```

Restart service

```
$ sudo service docker restart
```

- Logout from current terminal session, then login again to ensure user can run `docker ps -a`.

- The user's network environment must have access to the remote Ubuntu official server
- Run flex-builder command for docker as given in the next section.

4.1.1.3 Download and assemble LSDK images

In this LSDK Quick Start Guide, the build framework Flexbuild is used. Flexbuild contains three scripts: flex-builder, flex-installer, and flex-mkdistrorfs. In this section, "Download and assemble LSDK images" flex-builder is used to generate an Ubuntu userland.

In the next section, "Deploy LSDK images on board," the script flex-installer is used to deploy Ubuntu userland to a storage device (SD, USB, or SATA). In addition, flex-installer is used to deploy a boot partition and firmware images to the same storage device.

The boot partition contains a Linux kernel image and device tree blob. Firmware images include RCW (Reset Configuration Word), U-Boot and a FIT image. For a complete list of the boot partition components and the firmware components see [LSDK Memory Layout](#) on page 59.

See [Build tools](#) on page 61 for more information on flexbuild.

To download and assemble LSDK images, perform the steps below:

1. Download the flexbuild source tarball and set up flexbuild environment.
 - Go to "Component location" in "Release notes" and click the link to download flexbuild source tarball named "flexbuild_<version>.tgz"
 - Set up flexbuild environment.

```
$ tar xvzf flexbuild_<version>.tgz
$ cd flexbuild
$ source setup.env
```

Run the following two commands in addition to the above three commands only when building LSDK in Docker container.

```
$ flex-builder docker (This command emulates Ubuntu 18.04 environment on the host machine.)
$ source setup.env (Again, setup the flexbuild environment after entering Docker container.)
```

2. Download prebuilt images for boot partition and NXP-specific components tarball.

- Download prebuilt app components (e.g., dpdk, openssl).

```
wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/app_components_LS_arm64.tgz
```

- Download prebuilt images for boot partition and arm modules. There is a set of images for each of the Linux kernel versions supported by LSDK 1812.

```
wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/bootpartition_LS_arm64_lts_4.14.tgz
wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/lib_modules_LS_arm64_4.14.83.tgz
```

3. Generate LSDK Ubuntu userland, untar the prebuilt components tarball, and merge them into target userland.

- Generate Ubuntu arm64 userland.

```
$ flex-builder -i mkrfs -a arm64
```

- Extract app components.

```
$ tar xvzf app_components_LS_arm64.tgz -C build/apps
```

- Extract kernel modules (e.g., cryptodev).

```
$ sudo tar xvzf lib_modules_LS_arm64_<kernel_version>.tgz -C build/rfs/  
rootfs_ubuntu_bionic_LS_arm64/lib/modules
```

- Merge all components packages and kernel modules into target userland and compress ubuntu arm64 rootfs as .tgz tarball.

```
$ flex-builder -i merge-component -a arm64  
$ flex-builder -i compressrfs -a arm64
```

- Exit, if using Docker.

```
$ exit (optional, this command exits from docker when building in docker)
```

NOTE

- If the Linux host machine is in a subnet that needs HTTP proxy to access external Internet, set environment variable http_proxy and https_proxy as follows.

```
Add the following proxy settings in ~/.bashrc  
# No authentication:  
export http_proxy=http://<domain>:<port>  
export https_proxy=http://<domain>:<port>  
  
# With authentication:  
export http_proxy=http://<account>:<password>@<domain>:<port>  
export https_proxy=http://<account>:<password>@<domain>:<port>  
  
# Set no_proxy variable to bypass proxy for some local servers:  
export no_proxy="localhost,<local-server1>,<local-server2>"
```

4.1.1.4 Deploy LSDK images on board

This section assumes that the LSDK images have been assembled as given in “Download and assemble LSDK images”. There are two options for deploying LSDK images on the reference board:

1. Use a removable storage device such as an SD card, connect it to a local Linux host machine and deploy LSDK images on to the removable storage device. Then, connect this removable storage device to a reference board. This option is typically useful when the user has local access to a reference board and has a local Linux host machine.
2. If the user does not have access to a local host machine and/or a local reference board, LSDK images can be directly deployed to the storage device that is plugged into the board. This option is typically useful when a remote Linux host server is used, and/or the user is accessing the reference board remotely. For this option, a network connection to the reference board is required.

The deployment covers how to program LSDK composite firmware for "FlexSPI NOR flash boot" and "SD boot". It also covers how to deploy boot partition images and Ubuntu userland on different storage media (SD/USB/SATA).

NOTE

SD/USB/SATA capacity must be at least 8 GB.

Table 4. LSDK storage location

Boot source	Composite firmware	Kernel image and DTB	Ubuntu rootfs
FlexSPI NOR flash	FlexSPI NOR flash	SD/USB/SATA Partition 2	SD/USB/SATA Partition 3

Table continues on the next page...

Table 4. LSDK storage location (continued)

Boot source	Composite firmware	Kernel image and DTB	Ubuntu rootfs
SD	SD "raw partition"	"Boot Partition"	"Rootfs Partition"

4.1.1.4.1 LX2160ARDB reference information

This section provides general information about LX2160ARDB which may come in handy as a reference while completing steps for deploying LSDK that follow.

System memory map

Start address	End address	Size	Allocation	Comment
0x0000_0000_0000	0x0000_000F_FFFF	1 MB	CCSR - Boot ROM	64KB
0x0000_0010_0000	0x0000_00FF_FFFF	15 MB	Reserved	
0x0000_0100_0000	0x0000_0FFF_FFFF	240 MB	CCSR	
0x0000_1000_0000	0x0000_10FF_FFFF	16 MB	Reserved	SP alias this space to DCSR. Do not allocate.
0x0000_1100_0000	0x0000_11FF_FFFF	16 MB	Reserved	SP alias this space to DCSR. Do not allocate.
0x0000_1200_0000	0x0000_13FF_FFFF	32 MB	Reserved	SP alias this space to DCSR. Do not allocate.
0x0000_1400_0000	0x0000_17FF_FFFF	64 MB	Reserved	
0x0000_1800_0000	0x0000_181F_FFFF	2 MB	OCRAM	256 KB
0x0000_1820_0000	0x0000_18FF_FFFF	14 MB	Reserved	
0x0000_1900_0000	0x0000_19FF_FFFF	16MB	CoreSight STM	
0x0000_1A00_0000	0x0000_1BFF_FFFF	32MB	Reserved	
0x0000_1C00_0000	0x0000_1CFF_FFFF	16MB	Reserved	
0x0000_1D00_0000	0x0000_1FFF_FFFF	48MB	Reserved	
0x0000_2000_0000	0x0000_2FFF_FFFF	256MB	FlexSPI Region #1	
0x0000_3000_0000	0x0000_3FFF_FFFF	256 MB	Reserved	
0x0000_4000_0000	0x0000_5FFF_FFFF	512 MB	Reserved	
0x0000_6000_0000	0x0000_7FFF_FFFF	512 MB	Reserved	
0x0000_8000_0000	0x0000_9FFF_FFFF	512 MB	GPP DRAM Region #1 (0-2 GB)	
0x0000_A000_0000	0x0000_BFFF_FFFF	512 MB		
0x0000_C000_0000	0x0000_DFFF_FFFF	512 MB		
0x0000_E000_0000	0x0000_FFFF_FFFF	512 MB		
0x0001_0000_0000	0x0001_FFFF_FFFF	4 GB	Reserved	

Table continues on the next page...

Table continued from the previous page...

0x0002_0000_0000	0x0003_FFFF_FFFF	8 GB		
0x0004_0000_0000	0x0004_0FFF_FFFF	256 MB	FlexSPI Hole	Collapsed away by remapping logic to merge FlexSPI Region #1
0x0004_1000_0000	0x0004_FFFF_FFFF	3.75 GB	FlexSPI Region #2 (256 MB-4 GB)	3.75 GB
0x0005_0000_0000	0x0005_FFFF_FFFF	4 GB	Reserved	
0x0006_0000_0000	0x0006_FFFF_FFFF	4 GB	Reserved	
0x0007_0000_0000	0x0007_3FFF_FFFF	1 GB	DCSR	
0x0007_4000_0000	0x0007_FFFF_FFFF	3 GB	Reserved	
DPAA2 Portal Map				
0x0008_0000_0000	0x0008_03FF_FFFF	64 MB	Reserved	512 MB (0x0008_0000_0000-0x0008_1FFF_FFFF)
0x0008_0400_0000	0x0008_07FF_FFFF	64 MB	Reserved	
0x0008_0800_0000	0x0008_0BFF_FFFF	64 MB	Reserved	
0x0008_0C00_0000	0x0008_0FFF_FFFF	64 MB	MC - 1024 portals	
0x0008_1000_0000	0x0008_17FF_FFFF	128 MB	Reserved	
0x0008_1800_0000	0x0008_1FFF_FFFF	128 MB	QBMAN portals	
0x0008_2000_0000	0x000B_FFFF_FFFF	15.5 GB	Reserved	
0x000C_0000_0000	0x000F_FFFF_FFFF	16 GB	Reserved	
DPAA2 External address map				
0x0010_0000_0000	0x0010_FFFF_FFFF	4 GB	Reserved	(0x0010_0000_0000-0x001F_FFFF_FFFF)
0x0011_0000_0000	0x0011_FFFF_FFFF	4 GB	Reserved	
0x0012_0000_0000	0x0012_FFFF_FFFF	4 GB	Reserved	
0x0013_0000_0000	0x0013_FFFF_FFFF	4 GB	WRIOP access window	
0x0014_0000_0000	0x001B_FFFF_FFFF	32 GB	Reserved	
0x001C_0000_0000	0x001C_001F_FFFF	2 MB	Packet express buffer	
0x001C_4000_0000	0x001F_FFFF_FFFF	79 GB	Reserved	
0x0020_0000_0000	0x0020_7FFF_FFFF	2 GB	DRAM Hole	
0x0020_8000_0000	0x003F_FFFF_FFFF	126 GB	GPP DRAM Region #2	
0x0040_0000_0000	0x005F_FFFF_FFFF	128 GB	Reserved DRAM Hole Other "Normal" Memory	Collapsed by remap logic after MemNoC to merge DRAM Regions #1 and #2
0x0060_0000_0000	0x007F_FFFF_FFFF	128 GB	GPP DRAM Region #3	
High-speed I/O (PCI Express)				
Table continues on the next page...				

Table continued from the previous page...

0x0080_0000_0000	0x0087_FFFF_FFFF	32 GB	PCI Express 1	(0x0080_0000_0000-0x00FF_FF_FFFF)
0x0088_0000_0000	0x008F_FFFF_FFFF	32 GB	PCI Express 2	
0x0090_0000_0000	0x0097_FFFF_FFFF	32 GB	PCI Express 3	
0x0098_0000_0000	0x009F_FFFF_FFFF	32 GB	PCI Express 4	
0x00A0_0000_0000	0x00A7_FFFF_FFFF	32 GB	PCI Express 5	
0x00A8_0000_0000	0x00AF_FFFF_FFFF	32 GB	PCI Express 6	

Supported boot options

LX2160ARDB supports the following boot options:

- FlexSPI NOR flash (referred to as "FSPI" or "FSPI flash" in the following sections). CS refers to Chip Select.
- SD card (SDHC1)

On-board switch options

The RDBs have user selectable switches for evaluating different boot options for the LX2160A device as given in the table below ('0' is OFF, '1' is ON).

Boot source	SW1[1:8]	SW2[1:8]	SW3[1:8]	SW4[1:8]
FSPI NOR CS0 (default)	1111 1000	0000 0110	1111 1100	1011 1000
FSPI NOR CS1	1111 1001	0000 0110	1111 1100	1011 1000
SD Card (SDHC1)	1000 1000	0000 0110	1111 1100	1011 1000

Note: SW4[2] switch should be turned on [1], if user wants to power on the board as soon as power supply is turned on. This is useful in scenarios when the boards is to be used remotely.

Note that changing the boot device configuration from the default setting may require additional changes in the RCW or in other code images. For information on RCW naming convention for LX2160ARDB see <https://source.codeaurora.org/external/qorIQ/qorIQ-components/rcw/tree/lx2160ardb/README?h=github.com.qorIQ-os/integration>.

In addition to the above switch settings, make sure the following jumper settings are correct.

Table 5. LX2160ARDB jumpers

Jumper	Type	Name/function	Description
J6	1x2-pin connector	TA_BB_TMP_DETECT_B enable	Open: TA_BB_TMP_DETECT_B pin is grounded Shorted: TA_BB_TMP_DETECT_B pin is powered (default setting)
J7	1x2-pin connector	VBAT power for TA_BB_VDD enable	Not supported. Do not install J7. See <i>QorIQ LX2160A Reference Design Board Errata</i> for more details.

Table continues on the next page...

Table 5. LX2160ARDB jumpers (continued)

Jumper	Type	Name/function	Description
J8	1x2-pin connector	PROG_MTR voltage control (for NXP use only)	Open: PROG_MTR pin is powered off (default setting) Shorted: PROG_MTR pin is powered by OVDD (1.8 V)
J9	1x2-pin connector	TA_PROG_SFP voltage control (for NXP use only)	Open: TA_PROG_SFP pin is powered off (default setting) Shorted: TA_PROG_SFP pin is powered by OVDD (1.8 V)
J31	1x2-pin connector	USB1 mode setting	Open: USB1 works in Device mode Shorted: USB1 works in Host mode (default setting)
J33	1x2-pin connector	USB2 mode setting	Open: USB2 works in On-The-Go (OTG) mode (default setting) Shorted: USB2 works in Host mode
J56	2x3-pin connector	Inphi CS4223 GUI access	Normal: 1-2 short, 5-6 short (default setting) GUI mode: 1-2 open, 5-6 open
J57	1x2-pin connector	Inphi CS4223 GUI enable	Normal: Open (default setting) GUI mode: Short
J58	1x2-pin connector	Fan speed	Open: 100% speed Short: 50% speed (default setting)

FlexSPI NOR Flash Chip-select

FlexSPI NOR flash is a simple and convenient destination for deploying images so it is frequently used.

The benefit of this feature is that it allows more than one set of images to be independently deployed to the one NOR flash. This is very helpful during development because you can use the U-Boot image in one chip-select to program an image set into a different chip-select. If the new images are flawed, the old images are still functional to let you deploy corrected images.

The logic on the board usually allows the NOR flash to be accessed from different CS (chip select) option. Each CS is connected to dedicated NOR flash devices, these CS are called as DEV#0 and DEV#1. U-Boot prints which CS is loaded from. The output looks like following.

```
NOTICE: BL2: v1.5(release):lsdk18_12_ear1-6-g9277bc1c
NOTICE: BL2: Built : 21:09:32, Nov 27 2018
NOTICE: UDIMM 18ADF2G72AZ-3G2E1
NOTICE: DDR4 UDIMM with 2-rank 64-bit bus (x8)

NOTICE: 32 GB DDR4, 64-bit, CL=17, ECC on, 256B, CS0+CS1
NOTICE: BL2: Booting BL31
NOTICE: BL31: v1.5(release):lsdk18_12_ear1-6-g9277bc1c
NOTICE: BL31: Built : 21:09:28, Nov 27 2018
NOTICE: Welcome to LX2160 BL31 Phase
```

```

U-Boot 2018.09-09780-gd227337706 (Nov 20 2018 - 21:02:52 +0530)

SoC: LX2160ACE Rev1.0 (0x87360010)
Clock Configuration:
    CPU0(A72):2000 MHz  CPU1(A72):2000 MHz  CPU2(A72):2000 MHz
    CPU3(A72):2000 MHz  CPU4(A72):2000 MHz  CPU5(A72):2000 MHz
    CPU6(A72):2000 MHz  CPU7(A72):2000 MHz  CPU8(A72):2000 MHz
    CPU9(A72):2000 MHz  CPU10(A72):2000 MHz CPU11(A72):2000 MHz
    CPU12(A72):2000 MHz CPU13(A72):2000 MHz CPU14(A72):2000 MHz
    CPU15(A72):2000 MHz
    Bus:      700 MHz  DDR:      2900 MT/s
Reset Configuration Word (RCW):
    00000000: 50636338 24500050 00000000 00000000
    00000010: 00000000 0e010000 00000000 00000000
    00000020: 014001a0 00002580 00000000 00000096
    00000030: 00000000 00000000 00000000 00000000
    00000040: 00000000 00000000 00000000 00000000
    00000050: 00000000 00000000 00000000 00000000
    00000060: 00000000 00000000 00027000 00000000
    00000070: 08b30010 003f0020
Model: NXP Layerscape LX2160ARDB Board
Board: LX2160ACE Rev1.0-RDB, Board version: B, boot from FlexSPI DEV#0
FPGA: v5.0
SERDES1 Reference: Clock1 = 161.13MHz Clock2 = 161.13MHz
SERDES2 Reference: Clock1 = 100MHz Clock2 = 100MHz
SERDES3 Reference: Clock1 = 100MHz Clock2 = 100Hz
I2C:   ready
DRAM:  15.9 GiB
DDR     15.9 GiB (DDR4, 64-bit, CL=17, ECC on)
        DDR Controller Interleaving Mode: 256B
        DDR Chip-Select Interleaving Mode: CS0+CS1
Using SERDES1 Protocol: 19 (0x13)
Using SERDES2 Protocol: 5 (0x5)
Using SERDES3 Protocol: 2 (0x2)
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from SPI Flash... SF: Detected mt35xu512g with page size 256 Bytes, erase size
128 KiB, total 64 MiB
OK
EEPROM: NXID v1
In:     serial_pl01x
Out:    serial_pl01x
Err:    serial_pl01x
SCSI:
Net:    DPMAC3@xgmii: system interface USXGMII
DPMAC3@xgmii: Aquantia AQR107 Firmware Version 3.5.e
DPMAC4@xgmii: system interface USXGMII
DPMAC4@xgmii: Aquantia AQR107 Firmware Version 3.5.e
PCIe0: pcie@3400000 disabled
PCIe1: pcie@3500000 disabled
PCIe2: pcie@3600000 Root Complex: x1 gen1
PCIe3: pcie@3700000 disabled
PCIe4: pcie@3800000 Root Complex: no link
PCIe5: pcie@3900000 disabled
e1000: 68:05:ca:2c:db:73
        DPMAC3@xgmii, DPMAC4@xgmii, DPMAC17@rgmii-id, DPMAC18@rgmii-id, e1000#0
Warning: e1000#0 MAC addresses don't match:
Address in SROM is      68:05:ca:2c:db:73
Address in environment is 00:04:9f:05:a1:af

```

```
crc32+
fsl-mc: Booting Management Complex ... SUCCESS
fsl-mc: Management Complex booted (version: 10.11.2, boot status: 0x1)
Hit any key to stop autoboot: 0
=>
```

Boot option switching can be performed in U-Boot using the following statements.

- Switch to FlexSPI NOR flash 0 (default):

```
=>qixis_reset
```

- Switch to FlexSPI NOR flash 1:

```
=>qixis_reset altbank
```

- Switch to SD:

```
=>qixis_reset sd
```

U-Boot Environment Variables

- DPAA2-specific Environment Variables

- **mcboottimeout**: Defines Management Complex boot timeout in milliseconds. If this variable is not defined the compile-time value, `CONFIG_SYS_LS_MC_BOOT_TIMEOUT_MS` will be the default. Normally, users do not need to set this variable because the default is acceptable.
- **mcmemsize**: Defines amount of system DDR to be use by the Management Complex. If this variable is not defined, the compile-time value `CONFIG_SYS_LS_MC_DRAM_BLOCK_MIN_SIZE` will be the default. Normally, users do not need to set this variable because the default is acceptable.
- **mcinitcmd**: Contains commands to load and start the Management Complex automatically before the U-Boot count down to boot starts. If this variable is defined, its contents are run. The default value assumes that the Management Complex (MC) firmware and Data Path Control file are stored in FlexSPI flash/SD at fixed addresses. The default value for FlexSPI boot is `fsl_mc start mc 0x20a00000 0x20e00000`. The default value for SD boot is `mmc read 0x80000000 0x5000 0x800;mmc read 0x80100000 0x7000 0x800;fsl_mc start mc 0x80000000 0x80100000`. Users may change this variable as needed to load the MC files from sources other than FlexSPI into DDR and then start the MC using the `fsl_mc` command. For example, the files may be on a disk drive.

- Environment variables that are not specific to DPAA2

- **bootcmd**: Contains commands that are automatically executed when the U-Boot boot command is run. This happens automatically when the user does not interrupt U-Boot's initial count down. In normal usage, `bootcmd` should contain the command to apply the Management Complex Data Path Layout (DPL) file because this must be done before booting Linux. When booting from FlexSPI NOR, the default `bootcmd` is `bootcmd=env exists mcinitcmd && env exists secureboot && esbc_validate 0x20780000; env exists mcinitcmd && fsl_mc lazyapply dpl 0x20d00000;run distro_bootcmd;run xspl_bootcmd; env exists secureboot && esbc_halt;`. When booting from SD, the default `bootcmd` is `bootcmd=env exists mcinitcmd && mmcinfo; mmc read 0x80001000 0x6800 0x800; env exists mcinitcmd && env exists secureboot && mmc read 0x80780000 0x3C00 0x10 && esbc_validate 0x80780000;env exists mcinitcmd && fsl_mc lazyapply dpl 0x80001000;run distro_bootcmd;run sd_bootcmd;env exists secureboot && esbc_halt;`

For more information on U-Boot distro boot command, see “LSDK U-Boot uses distro boot feature” in LSDK documentation.

4.1.1.4.2 Option 1 - Deploy LSDK images using removable storage device

Given below are the steps to deploy LSDK images on to the reference board using a removable storage device (SD/USB/SATA). Option 1 can be used when the user has access to a local Linux host machine and to a local reference board. For this option, the storage device is plugged into the host machine and programmed. Once the programming is complete, the storage device is removed from the host machine and then inserted into the reference board.

1. Connect the removable storage device to the local Linux host machine.
2. The boot partition and rootfs images that were assembled during “Download and assemble LSDK images” will be available under “flexbuild_<version>” directory on the host machine. The images have following naming format:

- Boot partition:

- `bootpartition_LS_arm64_<version> or bootpartition_LS_arm64_<version>.tgz (64-bit)`

- rootfs:

- `rootfs_ubuntu_bionic_LS_arm64 or rootfs_ubuntu_bionic_LS_arm64.tgz (64-bit)`

3. Setup the environment for flex-installer to run:

```
$ cd flexbuild
$ source setup.env
```

4. Execute flex-installer with appropriate arguments to deploy LSDK images to the storage device.

For example:

```
$ flex-installer -b bootpartition_arm64_lts_<version>.tgz -r build/images/
rootfs_ubuntu_bionic_arm64_<timestamp>.tgz -d /dev/sdX
```

NOTE

The above command is valid assuming that U-boot is used as bootloader. If, UEFI is used as a bootloader instead of U-boot, option ‘-m <machine>’ must be specified.

For example:

```
$ flex-installer -b bootpartition_arm64_lts_<version>.tgz -r build/
images/rootfs_ubuntu_bionic_arm64_<timestamp>.tgz -d /dev/sdX -m
lx2160ardb
```

WARNING

- The SD/USB/SATA storage drive in the Linux PC is detected as /dev/sdX, where X is a letter such as a, b, c. Make sure to choose the correct device name, because data on this device will be replaced.
- Use the command `cat /proc/partitions` to see a list of devices and their sizes to make sure that the correct device names have been chosen.
- If the Linux host machine supports read/write SD card directly without an extra SD card reader device, the device name of SD card is typically `mmcblk0`.

5. After a successful installation, the message “installation finished successfully” appears. Execute the following command to unmount all mounted partitions of the target device.

Example:

```
$ sudo umount /run/media/sdX
```

- Unplug removable storage device from the Linux host and plugin into the reference board.
- Make sure the DIP switch settings on the board enable booting from FlexSPI NOR (or SD card). (Refer to “On-board switch options” in the preceding section for switch settings.)
- Power-on the board and stop autoboot to enter the U-Boot prompt.
- Program/update the composite firmware to FlexSPI NOR flash (or SD card) as described below.

Note : <filename> in below steps refers to `firmware_lx2160ardb_uboot_xspiboot.img` for FlexSPI NOR flash boot source and `firmware_lx2160ardb_uboot_sdboot.img` for SD card as boot source.

- Load firmware image from storage media.

Storage media	Command in U-Boot
USB	<pre>=> usb start => load usb 0:2 a0000000 <filename></pre>
SATA	<pre>=> load scsi 0:2 a0000000 <filename></pre>
SD	<pre>=> load mmc 0:2 a0000000 <filename></pre>

NOTE

If UEFI is used as a boot loader instead of U-boot, update the firmware name in the above command with the appropriate UEFI firmware.

- if FlexSPI NOR flash is to be used as boot source execute the commands below to program the CS1 flash.

```
=> i2c mw 66 50 20; sf probe 0:0; sf erase 0 +$filesize; sf write a0000000 0 $filesize
=> qixis_reset altbank
```

- if SD card is to be used as boot source execute the commands below to program the SD card.

```
=> mmc rescan; mmc dev 0; mmc write a0000000 0 0x25000
=> qixis_reset sd
```

- The system will automatically boot up LSDK Ubuntu distro available from the removable storage device.

4.1.1.4.3 Option 2 - Deploy LSDK images directly to the storage device on a board

If the user does not have a local Linux host machine that can connect to a removable media, and/or the user does not have local access to a reference board, the user can directly deploy LSDK images to the storage device on a reference board. The reference board may be accessed by the user remotely. For this option, a network connection to the reference board is required.

- Reboot the reference board from FlexSPI NOR CS0 (or SD card) and let it boot automatically.
 - If the DIP switch settings are configured to be default, the board will boot from FlexSPI NOR flash CS0 (or SD card) using the composite firmware which is present on the reference board by default.

2. After the reference board boots automatically, check whether the reference board boots TinyDistro or whether it boots Ubuntu distribution. TinyDistro is a non-customizable prebuilt ramdisk rootfs deployed in flash media on the reference board. This rootfs fits into the firmware image on flash and is therefore called tiny.

- If the reference board boots TinyDistro, proceed to step #4.
- If the reference board boots Ubuntu distribution, it means that an older Ubuntu distribution may already be present on the storage device that is plugged into the reference board. In this case, go to step #3 first, to force the board to boot TinyDistro.

3. Force the reference board to boot TinyDistro.

- Reboot the board and stop autoboot to enter U-Boot prompt in FlexSPI NOR CS0 flash (or SD card).
- if FlexSPI Nor is the boot source, run the following commands to force the reference board to boot TinyDistro present on FlexSPI NOR flash CS0.

```
$ fsl_mc apply dpl 0x20d00000
$ run xspi_bootcmd
```

- if SD card is the boot source, run the following commands to force the reference board to boot TinyDistro present on SD card.

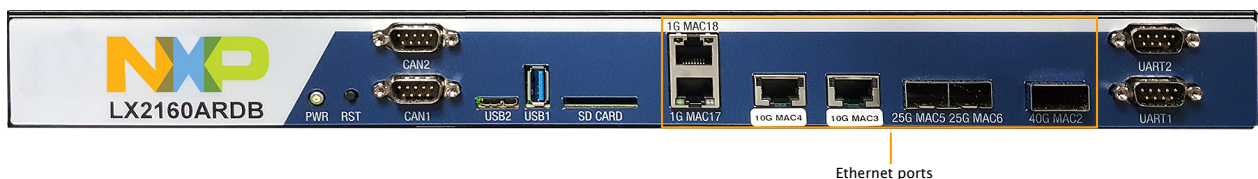
```
$ mmc rescan; mmc dev 0; mmc read 0x80001000 0x6800 0x800; fsl_mc apply dpl 0x80001000
$ run sd_bootcmd
```

4. Login to TinyDistro as “root” and bring up a network interface.

```
$ udhcpc -i <port name in TinyDistro>
```

For LX2160ARDB, in TinyDistro as well as in Ubuntu distribution, by default, only one MAC is enabled as a standard Kernel Ethernet Interface.

For LX2160ARDB, this interface is labeled 1G MAC17 on chassis front panel. MAC names corresponding to each of the seven ports on LX2160ARDB chassis are shown in diagram below. Note that ni0 corresponds to MAC17.



5. Use flex-installer to create and format the partitions for storage device (USB/SATA/SD).

Storage Device	Commands in Linux
USB	\$ flex-installer -i pf -d /dev/sdX
SATA	\$ flex-installer -i pf -d /dev/sdX
SD	\$ flex-installer -i pf -d /dev/mmcblk0

WARNING

- The USB/SATA storage drive in the Linux PC is detected as /dev/sdX, where X is a letter such as a, b, c. Make sure to choose the correct device name, because data on this device will be replaced.
- Use the command cat /proc/partitions to see a list of devices and their sizes to make sure that the correct device names have been chosen.

6. Download and deploy two tarballs (boot partition and Ubuntu userland) to USB/SATA/SD storage device. These tarballs were assembled during the “Download and Assemble LSDK Images” step.

Storage Device	Commands in Linux
USB	a. <code>\$ cd /run/media/sdX3</code> b. Download <code>bootpartition_<arch>_<version>.tgz</code> and <code>rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz</code> using the <code>wget</code> or <code>scp</code> command. c. <code>\$ flex-installer -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/sdX</code>
SATA	a. <code>\$ cd /run/media/sdX3</code> b. Download <code>bootpartition_<arch>_<version>.tgz</code> and <code>ubuntu_<codename>_<arch>_rootfs_<timestamp>.tgz</code> using the <code>wget</code> or <code>scp</code> command. c. <code>\$ flex-installer -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/sdX</code>
SD	a. <code>\$ cd /run/media/mmcblk0p3</code> b. Download <code>bootpartition_<arch>_<version>.tgz</code> and <code>ubuntu_<codename>_<arch>_rootfs_<timestamp>.tgz</code> using the <code>wget</code> or <code>scp</code> command. c. <code>\$ flex-installer -i install -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/mmcblk0</code>

7. Reboot the board with LSDK images. The system will automatically boot Ubuntu userland. Login using `root/root` or `user/user`.
8. After the above steps are completed, the reference board is ready to boot the latest Ubuntu userland. Optionally, to make sure that the reference board boots using the latest firmware, user may choose to update firmware using steps below:

- FlexSPI NOR firmware

- Reboot the board from FlexSPI NOR flash CS0 and stop autoboot to enter U-Boot prompt.
- Download the LSDK NOR composite firmware for LX2160ARDB using the command

```
$ wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/firmware_lx2160ardb_uboot_xspiboot.img
```


- c. Under U-Boot, download the firmware via TFTP to the reference board.

```
=> tftp a0000000 firmware_lx2160ardb_uboot_xspiboot.img
=> i2c mw 66 50 20; sf probe 0:0; sf erase 0 +$filesize; sf write a0000000 0 $filesize
=> qixis_reset altbank
```

In the steps above, the contents of the CS1 flash are overwritten with the new FlexSPI NOR firmware. The board is then rebooted using the CS1 flash.

- SD firmware

- a. Reboot the board from FlexSPI NOR flash CS0 (or SD card) and stop autoboot to enter U-Boot prompt.
- b. Download the LSDK SD composite firmware for LX2160ARDB using the command

```
$ wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/firmware_lx2160ardb_uboot_sdboot.img
```

- c. Under U-Boot, download the firmware via TFTP to the reference board.

```
=> tftp a0000000 firmware_lx2160ardb_uboot_sdboot.img
=> mmc rescan; mmc dev 0; mmc write a0000000 0 0x25000
=> qixis_reset sd
```

In the steps above, the contents of the SD card are overwritten with the new SD firmware. The board is then rebooted using the SD card.

4.1.1.5 Bringing up DPAA2 network interfaces

This section describes the procedure to bring up DPAA2 network interfaces.

4.1.1.5.1 Use Linux commands to list network interfaces

The Linux distribution boots with a default DPL file which enables only one network interface on DPAA2 by default as a standard kernel Ethernet interface. Run the following standard Linux command to get a list of enabled interfaces.

```
$ ip link show
```

The default interface is named eth0 (or eth1 if a PCI Express network interface card is discovered first).

4.1.1.5.2 Use restool wrapper scripts to list DPAA2 objects

User-friendly wrapper scripts are provided in the release rootfs to assist with dynamic creation of DPNI and associated dependencies. The wrapper scripts call restool commands.

Enter the following command for a list of the available wrapper scripts:

```
$ls-main
```

The Ethernet interfaces have corresponding DPAA2 objects associated with them. Run the following restool wrapper script to list the enabled data path network interface (DPNI) associated with ni0 (or ni1).

```
$ ls-listni
dprc.1/dpni.1 (interface: eth0, end point: dpmac.2)
dprc.1/dpni.0 (interface: eth1, end point: dpmac.17)
```

This indicates that the data path network interface named dpni.0 which belongs to the DPAA2 resource container dprc.1 is present. This DPNI object corresponds to the interface named ni0 which is connected to dpmac.17.

The following command can be used to list all DPMAC objects present in the system and what they are connected to (if anything).

```
$ ls-listmac
dprc.1/dpmac.18
dprc.1/dpmac.17 (end point: dpni.0)
dprc.1/dpmac.6
dprc.1/dpmac.5
dprc.1/dpmac.4
dprc.1/dpmac.3
dprc.1/dpmac.2 (end point: dpni.1)
```

For more information on DPAA2 objects and restool, see DPAA2-specific Software in LSDK documentation.

4.1.1.5.3 Add and destroy network interfaces

As mentioned in previous sections, interface ni0 corresponds to the data path network interface dpni.0 which is the only ones enabled by default DPL file. However, users may need more network interface enabled. Additional and fully featured DPNI objects can be created using restool. Once these objects are created, the configuration can be saved to a custom DPL file.

Running the command below is the simplest way of adding a DPNI object and connecting it to a DPMAC. In this example DPNI object is being connected to dpmac.4 using default options and arguments.

```
$ ls-addni dpmac.4
Created interface: ni2 (object:dpni.2, endpoint: dpmac.4)
```

Run the following command to display information about the newly created dpni.2 interface. The number of queues is shown to be 16, one queue per core for 16 cores which can receive traffic.

```
$ restool dpni info dpni.2
dpni version: 7.8
dpni id: 2
plugged state: plugged
endpoint state: 0
endpoint: dpmac.4, link is down
link status: 0 - down
mac address: ae:ff:05:f9:8e:02
dpni_attr.options value is: 0
num_queues: 16
num_rx_tcs: 1
num_tx_tcs: 1
mac_entries: 16
vlan_entries: 0
qos_entries: 0
fs_entries: 64
qos_key_size: 0
fs_key_size: 56
ingress_all_frames: 0
ingress_all_bytes: 0
ingress_multicast_frames: 0
ingress_multicast_bytes: 0
ingress_broadcast_frames: 0
ingress_broadcast_bytes: 0
egress_all_frames: 0
egress_all_bytes: 0
egress_multicast_frames: 0
egress_multicast_bytes: 0
egress_broadcast_frames: 0
egress_broadcast_bytes: 0
```

```
ingress_filtered_frames: 0
ingress_discarded_frames: 0
ingress_nobuffer_discards: 0
egress_discarded_frames: 0
egress_confirmed_frames: 0
```

If a user wants to connect DPMAC17 (which is connected to dpni.0 by default) to a fully featured data path network interface, the user will first need to unbind and destroy the existing interface by using the commands below.

Unbind dpni.0 from the driver

```
$ echo dpni.0 > /sys/bus/fsl-mc/drivers/fsl_dpaa2_eth/unbind
```

Destroy data path network interface dpni.0

```
$ restool dpni destroy dpni.0
dpni.0 is destroyed
```

Now add back dpmac.17 using the command below. Even though dpmac.17 is again connected to dpni.0, dpni.0 now uses 16 queues for traffic distribution.

```
$ ls-addni dpmac.17
Created interface: ni0 (object:dpni.0, endpoint: dpmac.17)
```

4.1.1.5.4 Save configuration to a custom DPL file (Optional)

Once the additional DPNI objects are created, a custom DPL file can be generated using the following command. This DPL file has a .dts format and is created on the reference board.

```
$ restool dprc generate-dpl dprc.1 > <file_name>.dts
```

The resulting .dts file must be compiled using the dtc tool to generate a .dtb file. Copy this file to a Linux host machine or server using SCP and run the following command to convert it to a .dtb file.

```
$ dtc -I dts -O dtb <file_name>.dts -o <file_name>.dtb
```

The newly created DPL file can be flashed onto the board and used to boot to Linux.

4.1.1.5.5 Assign IP addresses to network interfaces

Static IP addresses can be assigned to network interfaces using the standard ifconfig or ip commands.

```
ifconfig <interface_name_in_Linux> <ip_address>
OR
ip address add <ip_address> dev <interface_name_in_linux>
```

Alternatively, Static IP addresses can also be assigned using netplan. Create a file called “config.yaml” in /etc/netplan. Using a text editor, add the following lines to this config file and save it.

```
network:
  version: 2
  renderer: networkd
  ethernets:
    <interface_name_in_Linux>:
      addresses:
        - <ip_address>/24
```

After saving this file, run the following command to apply this netplan configuration and then reboot the board.

```
sudo netplan apply
```

Once the board reboots, bring up the desired interface by using “`ifconfig <interface_name_in_Linux> up`” or “`ip link set <interface_name_in_Linux> up`” command. The interface will be assigned the IP address that was entered in the “`config.yaml`” file.

Netplan can also be used for dynamic IP address assignment using DHCP. For dynamic IP assignment, replace the contents of the `config.yaml` file with the following.

```
network:
  version: 2
  renderer: networkd
  ethernets:
    <interface_name_in_Linux>:
      dhcp4: true
```

Follow the same procedure as for the static IP assignment using Netplan after saving the “`config.yaml`” file.

4.2 How to build LSDK with Flexbuild

Flexbuild provides cmdline for various build scenarios. The [LSDK Quick Start](#) on page 32 section introduces how to build the LSDK distro userland with prebuilt boot partition and component tarballs for quick deployment on the target board. This section introduces detailed steps to build LSDK with Flexbuild.

Go to **Tools and Software** tab at https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qoriq-layerscape-arm-processors/qoriq-lx2160a-reference-design-board:LX2160A-RDB?tab=Design_Tools_Tab. Download Layerscape LX2160A BSP v0.5 Software Release. Enter login details, accept the agreement to download the flexbuild source tarball in the name format `flexbuild_<version>.tgz`

```
$ tar xvfz flexbuild_<version>.tgz
$ cd flexbuild
$ source setup.env
$ flex-builder -h
```

Build TF-A with RCW and U-Boot/UEFI in Flexbuild

Layerscape platforms support TF-A (Trusted Firmware-A) which provides a reference implementation of secure world software for Armv7-A and Armv8-A.

`flex-builder` can automatically build the dependent RCW, U-Boot/UEFI, OPTEE and CST when building ATF to generate `bl2` and `fip` images for Layerscape platforms.

Use the commands below to build uboot-based or UEFI-based ATF image in Flexbuild.

```
Usage:
$ flex-builder -c atf -m <machine> -b <boottype> [-s]
Example:
$ flex-builder -c atf -m lx2160ardb -b sd # build uboot-based ATF image for SD boot on
lx2160ardb
$ flex-builder -c atf -m lx2160ardb -b xspi -s # build uboot-based ATF image for Flexspi-
NOR secure boot on lx2160ardb
$ flex-builder -c atf -m lx2160ardb -b xspi -B uefi # build uefi-based ATF image for Flexspi-NOR
boot on lx2160ardb
```

NOTE

In case users modify RCW source code (in packages/firmware/rcw) or specify different RCW binary rather than the default one, run 'flex-builder -c rcw' to regenerate RCW binary, then reconfigure rcw_<boottype> variable in configs/board/<machine>/manifest.

NOTE

If RCW or U-Boot source code is updated since last build, make sure clean the obsolete image by commands 'rm -rf build/firmware/u-boot/<machine>' and/or 'rm -rf build/firmware/rcw/<machine>', then re-build ATF by command 'flex-builder -c atf -m <machine> -b <boottype>'

NOTE

The '-s' option is used for secure boot, OPTEE and FUSE_PROVISIONING are not enabled by default, change CONFIG_BUILD_OPTEE=n to y and/or change CONFIG_FUSE_PROVISIONING=n to y in configs/build_lsdk.cfg to enable it if necessary.

Build Linux kernel with Flexbuild

Besides building LSDK kernel in stand-alone way (see [Configuring and building](#) on page 226), it is easy to automatically build LSDK kernel with flex-builder.

To build kernel using the default tree/branch/tag configurations specified in configs/build_lsdk.cfg:

```
$ flex-builder -c linux # for 64-bit mode of all armv8 Layerscape platforms
by default
$ flex-builder -c linux -a arm64 # for 64-bit mode of all armv8 Layerscape platforms
by default
```

To build kernel with specified tree/branch/tag and additional fragment config:

```
Usage:
$ flex-builder -c linux:<kernel-repo>:<branch|tag> -a arm64 -B fragment:<custom>.config
Example:
$ flex-builder -c linux:dash-lts:linux-4.14 -a arm64 -B fragment:lttng.config
$ flex-builder -c linux:linux:LSDK-18.12-V4.14 -a arm64
```

User can put a custom kernel fragment config file (given named test.config) in flexbuild/packages/linux/<kernel-repo>/arch/arm64/configs directory, then run the command below to compile kernel as per the default defconfig, lsdk.config and the additional test.config.

```
$ flex-builder -c linux -a arm64 -B fragment:test.config
```

Platform	Command for building Linux
LX2160AQDS 64bit LX2160ARDB 64bit	

Table continues on the next page...

Table continued from the previous page...

	<pre>\$ flex-builder -c linux:custom (optional, customize kernel config in interactive menu) or \$ flex-builder -c linux:<kernel-repo>:<branch/tag> -B fragment:<custom>.config (optional, add additional fragment config) \$ flex-builder -c linux (if -a <arch> is not specified, arm64 arch is used by default) \$ flex-builder -c linux:custom -a arm64 (optional, customize kernel config in interactive menu) \$ flex-builder -c linux -a arm64</pre>

Build LSDK composite firmware and boot partition

LSDK composite firmware consists of att bl2, att bl3 fip firmware, bootloader environment, secure headers, Ethernet MAC/PHY firmware, dtb, kernel and tiny ramdisk rfs, etc. This composite firmware can be programmed at offset 0x0 in flash device or at offset block# 8 in SD/eMMC card.

To generate LSDK composite firmware for Layerscape platform , directly run the following command:

```
Usage:
$ flex-builder -i mkfw -m <machine> -b <boottype> [-B <bootloader>] [-s]

Examples:
$ flex-builder -i mkfw -m lx2160ardb -b xspi
firmware_lx2160ardb_uboot_xspiboot.img will be generated.

$ flex-builder -i mkfw -m lx2160ardb -b xspi -s
firmware_lx2160ardb_uboot_xspiboot_secure.img will be generated.

$ flex-builder -i mkfw -m lx2160ardb -b xspi -B uefi
firmware_lx2160ardb_uefi_xspiboot.img will be generated.

$ flex-builder -i mkfw -m lx2160ardb -b sd
firmware_lx2160ardb_uboot_sdboot.img will be generated.

$ flex-builder -i mkfw -m lx2160ardb -b sd -s
firmware_lx2160ardb_uboot_sdboot_secure.img will be generated.
```

Similarly, the following composite firmware can be generated with the same usage of flex-builder command:

```
firmware_lx2160aqds_uboot_sdboot.img
firmware_lx2160aqds_uboot_sdboot_secure.img
firmware_lx2160aqds_uboot_xspiboot.img
firmware_lx2160aqds_uboot_xspiboot_secure.img
```

To generate LSDK boot partition tarball, run the command as below:

```
$ flex-builder -i mkbootpartition -a arm64
or
$ flex-builder -i mkbootpartition -a arm64 -s (for secure boot)
```

The command above will generate all needed images including kernel image, dtb, distro boot script, flex_linux_<arch>.itb, small ramdiskrfs, etc. Flex-builder automatically builds the dependent images if they are not present.

How to build application components in Flexbuild

The following commands are some examples of building application components.

```
Usage:
$ flex-builder -c <component> -a <arch> # build single application component for specified <arch>
Example:
$ flex-builder -c apps # build all apps components for arm64 arch
$ flex-builder -c dpdk # build DPDK component for SoCs integrated DPAA1 or DPAA2
$ flex-builder -c ovs-dpdk # build OVS-DPDK component
$ flex-builder -c restool # build RESTOOL component for arm64 arch,
$ flex-builder -c ptpd # build ptpd component for IEEE 1588 on arm64 platform
$ flex-builder -c cst # build cst component, needed for secure boot
(arm64 is the default arch if -a <arch> is not specified)
```

To add new application component in Flexbuild, follow the steps below:

1. Add new <component> to apps_repo_list and set CONFIG_BUILD_<component>=y in configs/build_xx.cfg.
2. Configure url/branch/tag/commit info for new <component_name> in configs/build_xx.cfg, default remote. Component git repository is specified by GIT_REPOSITORY_URL by default if <component>_url is not specified, user also can directly create the new component git repository in packages/apps directory.
3. Add build support of new component in packages/apps/Makefile..
4. Run flex-builder -c <component-name> -a <arch> to build the new component.
5. Run flex-builder -i merge-component -a <arch> to merge the new component package into target distro userland.

How to update existing Linux kernel with new custom kernel for Ubuntu on target board in case of non secure boot

User can quickly install custom kernel and lib modules after Ubuntu had been deployed in SD card on target board in case of non secure boot, follow the steps below.

```
After modify Linux kernel source code in $FBDIR/packages/linux/<kernel-repo> on demand, rebuild
kernel as below
$ flex-builder -c linux:custom (optional, to customize kernel config in interactive menu)
$ flex-builder -c linux
$ flex-builder -i mkbootpartition -a arm64
The new kernel image tarball $FBDIR/build/images/linux_4.14_LS_arm64_<timestamp>.tgz will be
generated.

Then login Ubuntu system on target board, update kernel image (take LX2160ARDB for example) as below:
root@localhost:/# dhclient -i eth2
root@localhost:~# cd /
root@localhost:/# wget <path>/linux_4.14_LS_arm64_<timestamp>.tgz (or by scp command)
root@localhost:/# tar xf linux_4.14_LS_arm64_<timestamp>.tgz
root@localhost:/# reboot
System will reboot and automatically boot to Ubuntu with new custom kernel.
```

Rebuild images after modifying the source code of NXP user space components locally

Flexbuild supports building specific components after the source code is changed. Flexbuild then deploys the changes to the target board.

1. Modify source code of user space components in the directory packages/apps/<apps-component> on demand.

2. Clean old build footprint under user space component.

```
$ make clean -C $FBDIR/packages/apps/<app-component>
$ rm -rf $FBDIR/build/apps/components_LS_arm64
```

3. Build the user space component and generate the compressed component tarball.

```
$ flex_builder -c <apps-component> -a arm64
$ flex-builder -i compressapps -a arm64
```

4. Upload and deploy the newly generated app_components_LS_arm64.tgz to target board on which Ubuntu distro had been installed in SD card.

```
=> run sd_bootcmd (or run xspi_bootcmd to enter Tiny distro Linux invironment)
Download app_components_LS_arm64.tgz via wget or scp command
root@TinyDistro:~# tar xf app_components_LS_arm64.tgz
root@TinyDistro:~# cp -a components_LS_arm64/* /run/media/mmcblk0p3
root@TinyDistro:~# reboot
```

How to generate a custom Ubuntu root filesystem with custom additional package list on x86 host machine

In Flexbulid, there are two default additional package lists for Ubuntu/Debian: additional_packages_list_moderate, and additional_packages_list_tiny.

```
$ flex-builder -i mkrfs -a arm64 (as per additional_packages_list_moderate with more packages for
Ubuntu rootfs by default)
$ flex-builder -i mkrfs -r ubuntu:tiny -a <arch> (as per additional_packages_list_tiny with less
packages for Ubuntu rootfs)
$ flex-builder -i mkrfs -r ubuntu -a <arch> -B <custom_packages_list>
```

Optionally, if you do not want to use default Ubuntu userland in certain use cases, you can generate Debian, CentOS or buildroot-based tiny userland by following instruction by Flexbuild, for examples:

```
$ flex-builder -i mkrfs -r debian:tiny:stretch -a arm64 (generate arm64 tiny Debian stretch rootfs
as per configs/ubuntu/additional_packages_list_tiny)
$ flex-builder -i mkrfs -r centos -a arm64 (generate arm64 LE CentOS rootfs as per
configs/centos/distro.cfg)
$ flex-builder -i mkrfs -r buildroot:tiny -a arm64 (generate arm64 LE buildroot userland as
per qoriq_arm64_tiny_defconfig)
$ flex-builder -i mkrfs -r buildroot:moderate -a arm64 (generate arm64 LE buildroot userland as
per qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:custom -a arm64 (generate arm64 LE buildroot userland as
per custom qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:custom -a arm64:be (generate arm64 big-endian buildroot
userland with custom qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:imaevm -a arm64 (generate arm64 LE initramfs userland with
qoriq_arm64_imaevm_defconfig used for secure boot with IMA EVM)
```

To install a new package to build/rfs/rootfs_ubuntu_bionic_LS_arm64 filesystem, run the following commands:

```
$ sudo chroot build/rfs/rootfs_ubunutu_bionic_LS_arm64
$ apt-get install <new_package_name>
# exit
```

How to add a new custom machine based on LSDK release in flexbuild

Assumption: Adding a new custom machine named LS1043AXX based on LS1043A SoC

1. Run flex-builder -i repo-fetch to fetch all git repositories of LSDK components for the first time
2. Add new machine support in U-Boot and ATF, example:

```
$ cd packages/firmware/u-boot
$ git checkout LSDK-18.12 -b LSDK-18.12-LS1043AXX
```

Add necessary U-Boot patches in U-boot tree and commit the patches in this branch for new machine. Add necessary ATF patches in ATF tree and commit the patches in this branch for new machine. Assume that you have added packages/firmware/u-boot/configs/ls1043axx_tfa_defconfig and added CONFIG_MACHINE_LS1043AXX=y in configs/build_lsdk.cfg, then run commands below to build TF-A image based on U-Boot for SD boot on LS1043AXX

```
$ cd packages/firmware/atf
$ git checkout LSDK-18.12 -b LSDK-18.12-LS1043AXX
$ flex-builder -c atf -m ls1043axx -b sd
```

3. Add new machine support in Linux kernel, take LSDK-18.12 for example:

```
$ cd packages/linux/linux
$ git checkout LSDK-18.12-V4.14 -b LSDK-18.12-V4.14-LS1043AXX
```

Now, you can add kernel patches and submit the patches in this branch for the new machine, then make a tag as below.

```
$ git tag LSDK-18.12-V4.14-LS1043AXX
```

Assume that you have added a new ls1043axx.dts in packages/linux/linux/arch/arm64/boot/dts directory, run as below to build kernel image for new LS1043AXX

```
$ flex-builder -c linux:linux:LSDK-18.12-V4.14-LS1043AXX
```

4. Add configs in flexbuild for new machine.
 - a. Add ls1043axx node in configs/linux/linux_arm64.its.
 - b. Add CONFIG_MACHINE_LS1043AXX=y in configs/build_lsdk.cfg.
 - c. Set linux_repo_tag to LSDK-18.12-V4.14-LS1043AXX and set u_boot_repo_tag to LSDK-18.12-LS1043AXX in configs/build_lsdk.cfg.
 - d. Set BUILD_DUAL_KERNEL to n in configs/build_lsdk.cfg if user doesn't need the second version of linux.
 - e. Optionally, user can use different memory layout from default settings by modifying them in configs/board/common/memorylayout.cfg.
 - f. Add manifest for new machine as below


```
$ mkdir configs/board/ls1043axx
$ cp configs/board/ls1043ardb/manifest configs/board/ls1043axx.
```
 - g. Update the settings in configs/board/ls1043axx/manifest on demand.
 - h. Generally, user can reuse the settings of rcw/fman/qe/eth-phy used for existing ls1043ardb if those components are same for new ls1043axx, otherwise user needs to add new support in packages/firmware/rcw.
 - i. Run flex-builder -i mklinux -a arm64 to generate lsdk_linux_arm64_tiny.itb image.
 - j. Run flex-builder -i mkfw -m ls1043ardb -b sd to generate the shared ppa/rcw/fman/qe/eth-phy images for new ls1043axx.
 - k. Run flex-builder -i mkfw -m ls1043axx -b sd to generate firmware_ls1043axx_uboot_sdboot.img.

- I. User can boot the new `lsdk_linux_arm64_tiny.itb` from U-Boot prompt during debugging stage on LS1043AXX machine.

```
=> tftp a0000000 lsd_k_linux_arm64_tiny.itb
=> bootm a0000000#ls1043axx
```

5. Build all other images for new custom machine with Ubuntu userland if required as below:

```
$ flex-builder -i mkrfs -a arm64
$ flex-builder -c apps -a arm64
$ flex-builder -i mkbootpartition -a arm64
$ flex-builder -i merge-component -a arm64
$ flex-builder -i compressrfs -a arm64
```

Finally, install `bootpartition_arm64_lts_<version>.tgz` and `rootfs_ubuntu_<codename>_<arch>.tgz` to SD/USB/SATA storage drive on LS1043AXX machine by flex-installer as below:

```
$ flex-installer -b bootpartition_LS_arm64_<version>.tgz -r build/rfs/
rootfs_ubuntu_bionic_LS_arm64 -d /dev/sdX
```

How to install distro to SD/USB/SATA storage drive

Use the LSDK flex-installer to install all the release binaries and distro userland onto a storage media (e.g. SD/eMMC card, USB/SATA disk) on the Linux host machine or on the target board.

Follow the instructions below:

Step 1: Plug SD/USB/SATA storage device to Linux Host machine or target board

Step 2: Install LSDK distro

- If the prebuilt distro tarball generated by Flexbuild is available on Linux host machine, run the following command:

```
$ flex-installer -b bootpartition_LS_arm64_lts_4.14.tgz -r rootfs_ubuntu_bionic_LS_arm64.tgz -m
lx2160ardb -d /dev/sdx
```

NOTE

sdx should be the actual device name on the host machine, for example: sdb, sdc, mmcblk0, etc.

- If the user wants to modify source code and build a custom LSDK distro with flexbuild, use the commands below on the Linux host machine:

```
$ flex-builder -c linux:custom -a <arch> # customize kernel config in interactive menu
$ flex-builder -c linux -a <arch>
$ flex-builder -i mkfw -m <machine> -b <boottype>
$ flex-builder -i mkrfs -a <arch>
$ flex-builder -c apps -a <arch>
$ flex-builder -i merge-component -a <arch>
$ flex-builder -i mkbootpartition -a <arch>
$ flex-installer -b build/images/bootpartition_LS_arm64_lts_4.14 -r build/rfs/
rootfs_ubuntu_bionic_LS_arm64 -d /dev/sdx
```

- If the user wants to install distro rootfs directly onto SD/USB/SATA disk on QorIQ board on which Linux is unavailable, download the prebuilt `flex_linux_<arch>.itb` image using the command:

```
$ wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/lsdk_linux_arm64_LS_tiny.itb
```

Optionally, locally build it using the command below:

```
$ flex-builder -i mklinux -a <arch> to generate lsd_k_linux_arm64_LS_tiny.itb
```

Put lsd_k_linux_arm64_LS_tiny.itb to a TFTP service directory, then download it to the target board under U-Boot prompt as below:

```
=> tftp a0000000 lsd_k_linux_arm64_LS_tiny.itb
=> bootm a0000000#<board-name>
```

The <board-name> can be: lx2160aqds, lx2160ardb.

- After booting and logging in to Linux on the target board, download the prebuilt distro tarballs generated by Flexbuild and install using the commands below:

```
$ flex-installer -i pf -d /dev/sdx
$ cd /run/media/{mmcblk0p3 or sdx3}, then download distro images to SD/USB/SATA storage disk via
wget or scp command
$ flex-installer -b bootpartition_LS_arm64_lts_<version>.tgz -r
rootfs_ubuntu_bionic_LS_arm64.tgz -d /dev/sdX
```

Step 3: Power on or reboot the target board after finishing the distro installation, the system will enter boot loader (U-Boot or UEFI) and automatically scan boot configuration script from the attached SD/USB/SATA disk and boot the target LSDK distro if found, otherwise it falls back to boot from Flexspi-NOR flash with tiny ramdisk distro.

How to program firmware to SD/Flexspi-NOR flash media

- **For SD/eMMC card (on all platforms):**

1. Download the prebuilt image (take LX2160ARDB for example):

- **Option 1:** Load the prebuilt image from SD card in U-Boot:

```
=> load mmc 0:2 a0000000 firmware_lx2160ardb_uboot_sdboot.img
```

- **Option 2:** Download the prebuilt image using the wget command:

```
$ wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/lsd_k_linux_arm64_LS_tiny.itb
```

- **Option 3:** To locally generate firmware_lx2160ardb_uboot_sdboot.img, run the command as below:

```
$ flex-builder -i mkfw -m lx2160ardb -b sd
```

2. Program firmware_<machine>_uboot_sdboot.img to SD card:

- **Under U-Boot:**

```
=> load mmc 0:2 a0000000 firmware_lx2160ardb_uboot_sdboot.img
=> mmc write a0000000 8 1fff8 (same on all platforms)
=> qixis_reset sd
```

- **Under Linux:**

```
$ flex-installer -f firmware_lx2160ardb_uboot_sdboot.img -d /dev/mmcblk0
```

- **For Flexspi-NOR flash:**

- On LX2160ARDB:

1. Download the image using the following options:

- **Option 1:** Load prebuilt image from SD card.

```
=> load mmc 0:2 a0000000 firmware_lx2160ardb_uboot_xspiboot.img
```

- **Option 2:** Download the prebuilt image using the wget command.

```
$ wget http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/firmware_lx2160ardb_uboot_xspiboot.img
```

- **Option 3:** To locally generate firmware_lx2160ardb_uboot_xspiboot.img, run the command as below:

```
$ flex-builder -i mkfw -m lx2160ardb -b xspi
```

2. Program firmware_lx2160ardb_uboot_xspiboot.img to Flexspi-NOR flash:

```
=> sf probe 0:1  
=> sf erase 0 +$filesize && sf write 0xa0000000 0 $filesize
```

- On LX2160AQDS:

1. Download the image using the following options:

- **Option 1:** Load prebuilt image from SD card.

```
=> load mmc 0:2 a0000000 firmware_lx2160aqds_uboot_xspiboot.img
```

- **Option 2:** Download the prebuilt image using the wget command.

```
http://www.nxp.com/lgfiles/sdk/lx2160a_bsp05/firmware_lx2160aqds_uboot_xspiboot.img
```

- **Option 3:** To generate firmware_lx2160aqds_uboot_xspiboot.img locally, run the command as below:

```
$ flex-builder -i mkfw -m lx2160aqds -b xspi
```

2. Program firmware_lx2160aqds_uboot_xspiboot.img to Flexspi-NOR flash:

```
=> sf probe 0:1  
=> sf erase 0 +$filesize && sf write 0xa0000000 0 $filesize
```

4.3 Procedure to Run Secure Boot

This section describes the steps to be followed to run secure boot on a platform, after building the images.

4.3.1 Prepare board for Secure Boot

Steps to blow fuses

1. Enable POVDD
 - a. LX2160A RDB platform
 - Put J9 to enable PROG_SFP
2. Write the required values to be fused in the corresponding SFP Registers. Check SFP Block Guide in the SoC RM for details..
3. Blow the fuses by writing PROGFB (0x2) in the INST field in INGR register of SFP.

Blowing One Time Programmable Master Key (OTPMK) fuse using the above procedure

- Check initial SNVS state

```
md 1e90014
88000900
```

The second nibble indicates that the OTPMK is not blown

- Enable POVDD
- Command to generate OTPMK

```
cd cst
./gen_otpmk_drbg 2
```

NOTE

For more information on gen_otpmk_drbg, refer Code Signing Tool section in Secure Boot User Guide

- Write OTPMK fuse values on shadow registers:

```
mw.l 1e80234 <OTPMK1>
mw.l 1e80238 <OTPMK2>
mw.l 1e8023c <OTPMK3>
mw.l 1e80240 <OTPMK4>
mw.l 1e80244 <OTPMK5>
mw.l 1e80248 <OTPMK6>
mw.l 1e8024c <OTPMK7>
mw.l 1e80250 <OTPMK8>
```

- Check SNVS state again. There should be no parity errors.

```
md 1e90014
80 000 900
```

Now you will see '0' in second nibble.

```
md 1e80024
00000000
```

No parity errors .

Use the below command write to INGR register :

For LX2160 use the below command:

```
mw 1e80020 0x2
```

- Reset and check that SNVS is in Check state

```
md 1e90014
80 000 900
```

4.3.2 Running secure boot on target platforms

1. Platform LX2160A

Below are the steps to put the LX2160 in RSP and write SRKH in SFP mirror registers:

```
ccs::config_chain {lx2160a dap}
jtag::lock
#To Read the Content of TPINSVSR SEL (TPINSVSR) register
jtag::scan_io 0 8 0x92
## Setting the override bit to 1 and the RSP enable bit to 0
jtag::scan_io 1 64 0x0 # this will give the content of the register as output
# To write to TAP Configuration Pin Override Control Register (TCPOVCR)
jtag::scan_io 0 8 0x93
## Setting the override bit to 1 and the RSP enable bit to 0
jtag::scan_io 1 64 0x00000103713F001F # in case of FlexSPI NOR for LX2160ARDB
jtag::scan_io 1 64 0x000001027100001F # in case of FlexSPI NOR for LX2160AQDS
jtag::unlock
After executing the above steps, do POR , and then run the following commands

ccs::config_chain {lx2160a sap2}
display ccs::get_config_chain
ccs::stop_core 1
ccs::write_mem 1 0x1E80254 4 0 <SRKH1>
ccs::write_mem 1 0x1E80258 4 0 <SRKH2>
ccs::write_mem 1 0x1E8025c 4 0 <SRKH3>
ccs::write_mem 1 0x1E80260 4 0 <SRKH4>
ccs::write_mem 1 0x1E80264 4 0 <SRKH5>
ccs::write_mem 1 0x1E80268 4 0 <SRKH6>
ccs::write_mem 1 0x1E8026c 4 0 <SRKH7>
ccs::write_mem 1 0x1E80270 4 0 <SRKH8>
# To get the board out of RSP
ccs::write_mem 1 0x101000D0 0x4 0x0 0x000c0000
ccs::run_core 1
```

After implementing all the steps, the board will boot and user will get the Linux prompt after **successful validation** of all the images.

NOTE

- To blow SRKH in production environment follow procedure similar to blowing OTPMK fuses. For more detail on production and development environment refer Flow A and Flow B under Product Execution section in Secure Boot User Guide

4.3.3 Steps to run Chain of Trust with Confidentiality

1. Generate all images

```
$ flex-builder -c firmware
$ flex-builder -c linux -a <arch>
```

2. Generate autoboot script with e flag

a. with encapsulation flag enabled

```
$ flex-builder -i mkdistroscr -e
```

(OR)

b. With encapsulation and key identifier (16 bytes)

```
$ flex-builder -i mkdistroscr -e -k <key_id>
Eg. Key_id = 0x20000000
```

NOTE

For more information on key identifier, refer Section 1.7.2 of Secure Boot Guide.

3. Signing all images

```
$ flex-builder -i signing -m <platform> -b <boottype> -s -e
```

4. Generating firmware image

```
$ flex-builder -i mkfw -m <platform> -b <boottype> -B uboot -s
```

5. Generating bootpartition

```
$ flex-builder -i mkbootpartition -a <arch> -s
```

6. Writing image to sd card

```
$ flex-installer -b build/images/bootpartition_arm64_lts_<version>.tgz -r build/rfs/  
rootfs_ubuntu_bionic_arm64 -m <platform> -d /dev/sdx
```

BOOT FLOW

First Boot: Encapsulaton Step (Shoudl happen in OEM's premises)

1. By default the enacap and decap bootscripts will be installed in the bootpartition.
2. When the board boots up for the first time after all images have been generated, Encap bootscript will execute. This bootscript:
 - a. Authenticates and encapsulates linux and dtb images and replaces the unencrypted linux and dtb images with newly encapsulated linux and dtb.
 - b. Replaces the encap bootscript and header with the decap bootscript and it's header, already present in the bootpartition.
 - c. Issues reset

Subsequent Boot

1. Uboot would execute script with decap commands
 - a. Un-blobify linux and dtb image in DDR
 - b. Pass control to these images

NOTE

Chain of trust with confidentiality is currently not supported for LS1012 in flexbuild

4.4 LSDK Memory Layout

Flash layout

The following table shows the memory layout of various firmware stored in NOR/NAND/QSPI flash device or SD card on all QorIQ Reference Design Boards.

Table 6. Unified 64MB memory layout of NOR/QSPI/NAND/SD media on all Layerscape platforms

Firmware Definition		MaxSize	Flash Offset	SD Start Block No.
RCW + PBI + BL2 (bl2.pbi)		1MB	0x00000000	0x00008
ATF FIP Image (fip.bin) BL31 + BL32 + BL33		4MB	0x00100000	0x00800
Boot firmware environment		1MB	0x00500000	0x02800
Secure boot headers		2MB	0x00600000	0x03000
Secure header or DDR PHY FW		512KB	0x00800000	0x04000
Fuse provisioning header		512KB	0x00880000	0x04400
DPAA1 FMAN ucode		256KB	0x00900000	0x04800
QE/uQE firmware		256KB	0x00940000	0x04A00
Ethernet PHY firmware		256KB	0x00980000	0x04C00
Script for flashing image		256KB	0x009C0000	0x04E00
DPAA2-MC or PFE firmware		3MB	0x00A00000	0x05000
DPAA2 DPL		1MB	0x00D00000	0x06800
DPAA2 DPC		1MB	0x00E00000	0x07000
Device tree(needed by uefi)		1MB	0x00F00000	0x07800
Kernel	lsdk_linux.itb	16MB	0x01000000	0x08000
Ramdisk rfs		32MB	0x02000000	0x10000

Table 7. Unified 2MB memory layout of QSPI/SD media on Layerscape platforms

Firmware definition	Max size	Location	SD Start Block No.
RCW+PBI+BL2 (bl2.pbi)	64KB	0x0000_0000 - 0x0000_FFFF	0x00008
Reserved	64KB	0x0001_0000 - 0x0001_FFFF	0x00080
PFE firmware	256KB	0x0002_0000 - 0x0005_FFFF	0x00100
FIP (BL31+BL32+BL33)	1MB	0x0006_0000 - 0x000D_FFFF	0x00300
Environment variables	64KB	0x001D_0000 - 0x001D_FFFF	0x00E80
Reserved	64KB	0x001E_0000 - 0x001E_FFFF	0x00F00
Secureboot headers	64KB	0x001F_0000 - 0x001F_FFFF	0x00F80

Storage layout on SD/USB/SATA for LSDK images deployment

With LSDK flex-installer, the LSDK distro can be installed into an SD/USB/SATA storage disk which should have at least 8GB of memory space.

Table 8. Storage Layout on SD/USB/SATA for LSDK Image Deployment

Region 1 (4KB)	Region 2 (RAW) 64MB Firmware	Region 3 (Partition-1 FAT32) 20MB EFI	Region 4 (Partition-2 EXT4) 1GB Boot partition	Region 5 (Partition-3 EXT4) Remaining space RootFS partition
MBR/GPT	RCW U-Boot or UEFI TF-A firmware Secure boot headers FMan firmware QE/uQE firmware Eth PHY firmware MC firmware DPC firmware DPL firmware DTB lsdk_linux_<arch>.itb	BOOTAA64.EFI grub.cfg	kernel image DTB lsdk_linux_<arch>.itb distro boot scripts secure headers composite firmware	Ubuntu or Android or CentOS or Debian or Tiny Buildroot RFS

4.5 Build tools

Flexbuild is a component-oriented build framework and integrated platform with capabilities of flexible, easy-to-use, scalable system build and distro installation. With flex-builder CLI tool, users can build various components (linux, u-boot, uefi, rcw, TF-A and miscellaneous custom userspace applications) and distro userland to generate composite firmware, hybrid rootfs with customizable userland. The following are Flexbuild's main features:

- Automatically build Linux, U-Boot, TF-A, RCW and miscellaneous user space applications.
- Generate machine-specific composite firmware for various boot types: SD/QSPI/NOR/NAND boot, secure boot, U-Boot, UEFI.
- Support integrated management with repo-fetch, repo-branch, repo-commit, repo-tag, repo-update for git repositories of all components.
- Support cross build on x86 Ubuntu 18.04 host machine for aarch64/armhf arch target.
- Support native build on aarch64/armhf machine for ARM arch target.
- Support creating an Ubuntu docker container and building LSDK inside it when the host machine is using CentOS, RHEL, Fedora, SUSE, Debian, non-18.04 Ubuntu, etc.
- Scalability of integrating various components of both system firmware and user space applications.

- Capability of generating custom aarch64/armhf Ubuntu userland integrated configurable packages and proprietary components.

Flexbuild can separately build each component or automatically build all components, it generates the boot firmware (RCW, U-Boot/UEFI, PHY firmware, kernel image, and ramdiskrfs), `lsdk_linux_<arch>_tiny.itb`, and the Ubuntu userland containing the specified packages and application components.

NOTE

Since LSDK-18.06, upgrading of toolchain is required for U-Boot v2018.03 or later, if your host machine is not a Ubuntu 18.04 system, there are two ways to use Ubuntu 18.04 toolchain as below:

- Run `sudo do-release-upgrade` command to upgrade existing Ubuntu 16.04 to Ubuntu 18.04
 - Run `flex-builder docker` command on the existing non Ubuntu 18.04 host to create a ubuntu 18.04 docker container in which GCC 7.3.0 is available, then build LSDK in docker.
-

Chapter 5

Bootloaders

5.1 General boot flow

NXP SoC Booting Principles

The high-level (some details omitted) boot flow of an ARMv8a SoC is:

1. SoC comes out of reset and reads and RCW/PBL image from a boot source such as NOR flash or an SD card or eMMC flash. The RCW/PBL image contains configuration bits that control things such as:
 - Pin muxing and the protocol selected for SerDes pins.
 - Clock parameters and PLL multipliers.
 - Device containing the first software (not in an internal SoC ROM) to run.
2. Code in an internal boot ROM starts running and configures low-level aspects of the SoC.
3. The boot ROM must then load the first external software to run from a boot device such as NOR flash or SD/eMMC. In today's LSDK, this first software is a boot loader, either U-Boot or UEFI.
4. The boot ROM transfers control to the boot loader.
5. The boot loader does additional system configuration and then loads and starts the PPA image from NOR flash or SD/eMMC.
6. The PPA is a special resident firmware that runs at the highest ARMv8A privilege level EL3. It provides services to both the boot loader and, later, the operating system. These services include controlling core power state and bringing additional cores out of reset.
7. Usually, the boot loader must also load peripheral firmware, firmware required to make peripherals such as Ethernet controllers work. The details of this differ from SoC to SoC.
8. When the boot loader finishes initialization, its job is to locate a Linux kernel image and a Linux device tree image. The device tree is a description of the board and SoC hardware that Linux uses, for example, to know which peripherals are available for use and to associate drivers with them. Often, boot loaders do some on-the-fly "fixups" to the device tree to pass information to Linux.
9. In summary, the boot loader read kernel and device tree images from memory or mass storage device. Because boot loaders have many drivers, there are many possible choices for the location of the images.
 - NOR flash (serial QSPI or parallel)
 - NAND
 - SD/eMMC
 - USB mass storage devices of all types.
 - SATA drives of all types.
 - Ethernet, normally via TFTP.
10. After the boot loader loads the kernel and device tree and does fixups, it puts kernel boot parameters and the device tree into DDR where the kernel can find them and passes control to the kernel. One of the key kernel parameters is "root=". It tells the Linux kernel what device contains the user space file set (user land). U-Boot stores kernel parameters in environment variable `bootargs`.
11. Because the Linux kernel supports even more device drivers than boot loaders support, the array of choices for the user land device is even larger.

- NOR flash (serial QSPI or parallel)
- NAND (using special file systems)
- SD/eMMC
- USB mass storage devices of all types.
- SATA drives of all types.
- Ethernet, normally via NFS.
- RAM disks (which the boot loader populates)
- Third party PCIe-based mass storage devices and controllers
 - a. SATA controllers
 - b. SAS controllers
 - c. Fibre Channel Host Bus Adaptors
 - d. NVMe cards
 - e. And more.

Once the kernel is up, it starts user land, starting with `systemd`. The startup process is part of the Ubuntu file set and conforms to normal Ubuntu procedures.

Notes on General Boot Principles

- In the future, the boot flow will be altered to have the boot ROM load the PPA before the boot loader. The PPA will then load and start the boot loader. This mechanism allows consolidation of more configuration into one place, the PPA. This configuration is then removed from the boot loaders.
- Secure boot does not change the overall sequence. The significant difference is that secure boot involves each component (starting with the boot ROM) validating the images it loads and starts. This sequence of image validations is called the “chain of trust”.

Linux often resets peripherals and reloads their firmware. This process is specific to SoC's.

5.2 Trusted Firmware – A (TF-A)

5.2.1 Introduction

TF-A Stands for Trusted Firmware – A (For ARM A-Series Core).

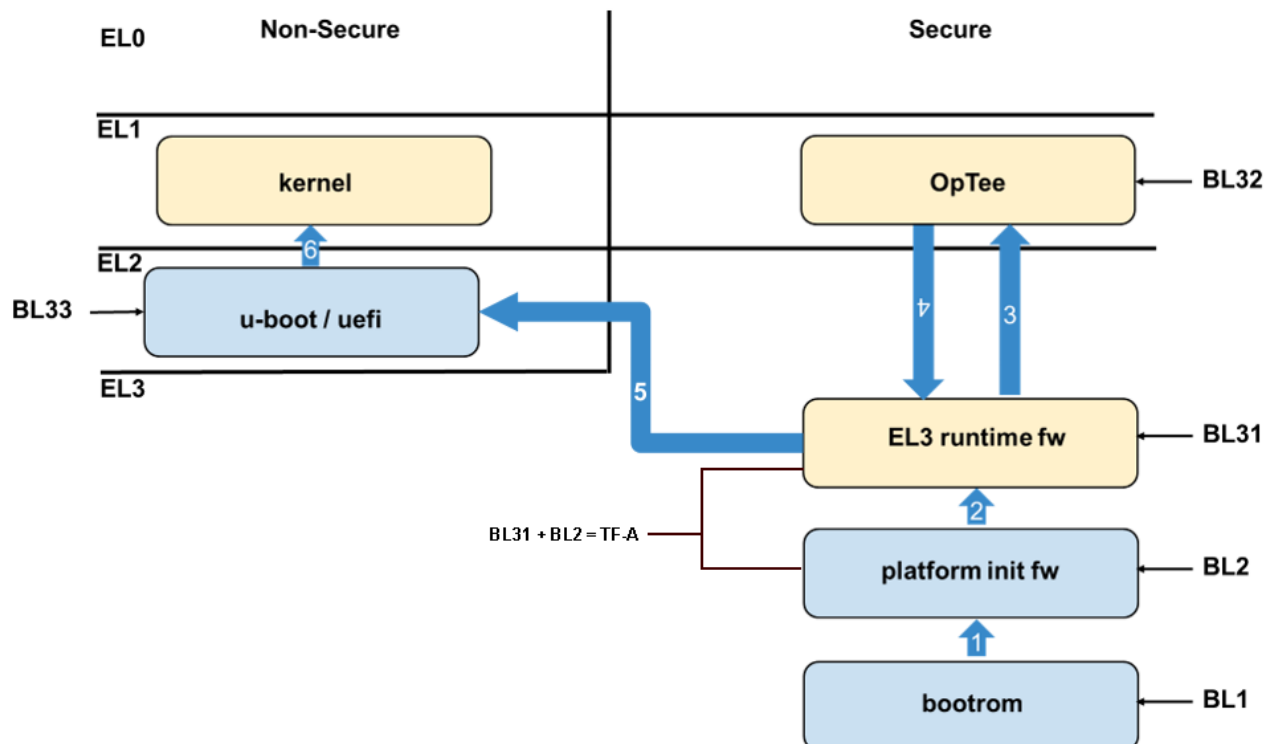
This section describes how to use the accompanying LSDK release on the QorIQ Layerscape platforms and how to boot LSDK distro with TF-A in non-secure as well as secure mode. The section also covers TF-A enablement on QorIQ Layerscape platforms.

5.2.2 TF-A boot flow

Below is the generic boot procedure for TF-A based platforms:

1. Out of reset, control transfers to HW PBL block (for chasis 2.0 based platforms) or to BootROM (for chasis 3.0 based platforms), which has the responsibility to execute the PBI commands. PBI commands are present to copy the BL2 binary for platform initialization to OCRAM. It also contains the PBI command to populate the BOOTLOC ptr to the location where bl2.bin is copied.
 - Upon successful execution of PBI commands control transfers to the BOOTLOC pointer i.e bl2.bin image.

2. BL2 firmware initializes the DRAM, configure TZASC and validate the FIP images before loading the FIP image (which constitutes of BL31, BL32, BL33 images) to DDR.
3. Various components of FIP image are categorised as :
 - a. BL31: Secure firmware
 - b. BL32: Trusted OS (e.g. OPTEE). This is optional to have in FIP image.
 - c. BL33: U-Boot/UEFI
4. Post validation of all the components of FIP image, BL2 passes execution control to the EL3 runtime firmware image named as “BL31”, one of the FIP images.
5. BL31 setup exception vector table at EL3 and passes execution control to:
 - [Optional] Trusted OS (OP-TEE in our case) image denoted as BL32 image, if BL32 image is present. (Execution flow 3 & 4 are optional)
6. [Optional] After execution of BL32 image, execution control comes back to BL31 image via SMC call.
7. BL31 passes execution control to BL33 image U-Boot/UEFI.
8. BL33 image is responsible to load and start the kernel and other firmware (if any) images.



5.2.3 Secure boot with TF-A boot

Secure boot image validation is done using CSF headers for the images, generated using Code signing tool. For details related to secure boot, refer to section [Secure boot](#) on page 96. Below section describes the TFA based secure boot flow.

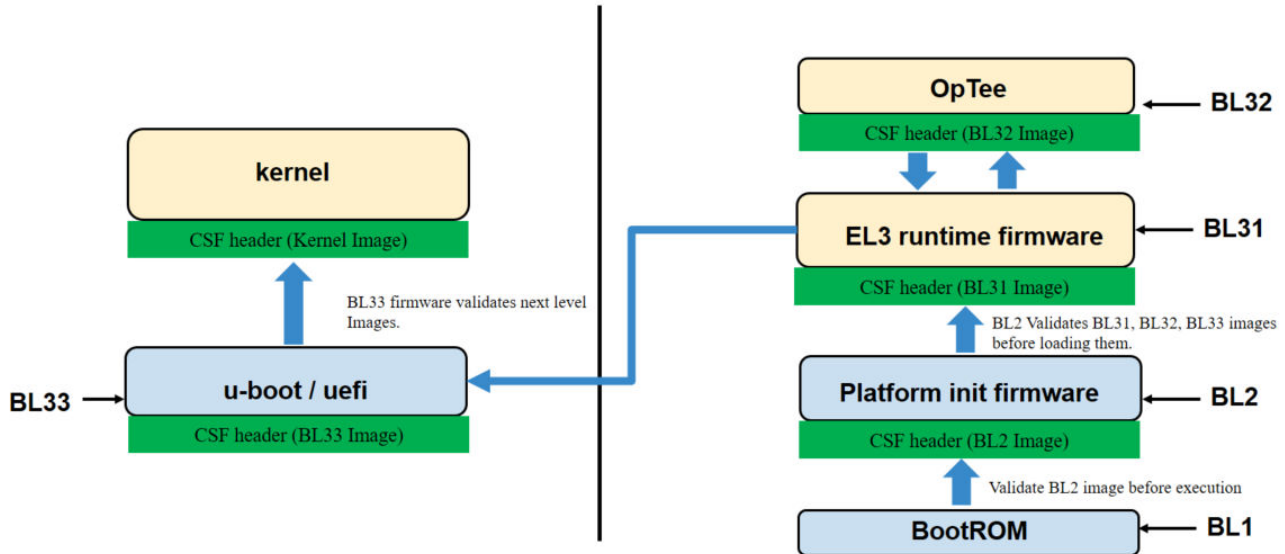
1. When SOC comes out of reset, control transfers to HW PBL block (for chasis 2.0 based platforms) or to SP BootROM (for chasis 3.0 based platforms), which is responsible for validation of BL2 image using its CSF header added with the BL2 image itself. Either SP BootROM (chasis 3.0) or the HW PBL block (chasis 2.0) reads the BOOTLOC pointer value to locate the BL2 image csf header and validates the image there after.

2. If the BL2 image is validated successfully control is passed for its execution. BL2 image further validates the components of FIP image using their respective CSF headers. FIP image constitutes of following images:

- CSF Header BL31 + BL31 image
- CSF Header BL32 + BL32 image
- CSF Header BL33 + BL33 image

3. BL33 (UBOOT) is responsible to perform the validation of the next level firmwares to establish the chain of trust.

Note: TF-A also support TBBR based secure boot validation. To enable this approach refer to readme in ATF repository. Readme is located at: `~/atf/plat/nxp/README.TRUSTED_BOOT`



5.2.4 TF-A compilation and deployments steps

1. Toolchain to compile TFA images can be obtained from below location:
 - a. [https://releases.linaro.org/components/toolchain/binaries/latest/aarch64-linux-gnu/ gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz](https://releases.linaro.org/components/toolchain/binaries/latest/aarch64-linux-gnu/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz)
2. For compilation & deployment steps refer the document file: `~/ATF/plat/nxp/README.TRUSTED_BOOT` in the TF-A source code.

References

- TF-A: <https://github.com/ARM-software/arm-trusted-firmware>
- OP-TEE OS: https://github.com/OP-TEE/optee_os

5.3 U-Boot

5.3.1 LSDK U-Boot uses distro boot feature

As in previous versions of the NXP SDK, the U-Boot variable `bootcmd` contains commands that represent the default boot process. LSDK is different in that it uses a standard U-Boot feature called `distro boot` to make automatic booting more flexible. In distro boot, `bootcmd` runs additional commands in the variable `distro_bootcmd`. These commands are the heart of the distro boot process.

Distro boot sequential examines partitions on mass storage devices looking for a script file. When U-Boot finds one, it loads and executes it to initiate the boot process.

The mass storage devices to be searched are defined in the U-Boot environment variable `boot_targets`. Set it to control which mass storage devices are searched and the order in which they are searched. For example,

```
=> printenv boot_targets
boot_targets=usb0 mmc0 scsi0 dhcp
```

The command above shows the search order USB device 0, MMC (or SD) device 0, SCSI (SATA) device 0, followed by DHCP.

The process of searching involves a number of U-Boot variables. It ends with the variables shown below in an example from an LS2088ARDB.

```
=> printenv scan_dev_for_scripts
scan_dev_for_scripts=for script in ${boot_scripts}; do if test -e ${devtype} ${devnum}:$
{distro_bootpart} ${prefix}${script}; then echo Found U-Boot script ${prefix}${script}; run
boot_a_script; echo SCRIPT FAILED: continuing...; fi; done => printenv boot_scripts
boot_scripts=ls2088ardb_boot.scr => printenv boot_a_script boot_a_script=load ${devtype} ${devnum}:
${distro_bootpart} ${scriptaddr} ${prefix}${script}; env exists secureboot && load ${devtype} $
{devnum}:${distro_bootpart} ${scripthdraddr} ${prefix}${boot_script_hdr} && esbc_validate $
{scripthdraddr};source ${scriptaddr}
```

The process searches for a script named by the variable `boot_scripts`. In this example, the search is for a script named `ls2088ardb_boot.scr`. When this script is located, it is loaded into RAM using the `load` command and run using the `source` command. This causes Linux to boot.

LSDK puts boot scripts into a file system on the second partition of a mass storage device. U-Boot can display files in a file system. Continuing the example, the following U-Boot commands list the files in the second partition of USB device 0 (do a `usb start` first):

```
=> ls usb 0:2
<DIR>      4096 .
<DIR>      4096 ..
<DIR>     16384 lost+found
33301280 firmware_ls1043ardb_uboot_norboot.img
33301280 firmware_ls1046ardb_uboot_qspiboot.img
33301280 firmware_ls1088ardb_uboot_qspiboot.img
33301280 firmware_ls2088ardb_uboot_norboot.img
16524064 flex_linux_arm64.itb
10005 fsl-ls1012a-frdm.dtb
10145 fsl-ls1012a-qds.dtb
8974 fsl-ls1012a-rdb.dtb
30832 fsl-ls1043a-qds.dtb
28619 fsl-ls1043a-rdb-sdk.dtb
29342 fsl-ls1043a-rdb-sdk.dtb
30134 fsl-ls1043a-rdb-usdpaa.dtb
30010 fsl-ls1046a-qds.dtb
27125 fsl-ls1046a-rdb-sdk.dtb
27909 fsl-ls1046a-rdb-sdk.dtb
28556 fsl-ls1046a-rdb-usdpaa.dtb
14692 fsl-ls1088a-qds.dtb
15451 fsl-ls1088a-rdb.dtb
19713 fsl-ls2080a-qds.dtb
19243 fsl-ls2080a-rdb.dtb
20651 fsl-ls2088a-qds.dtb
19545 fsl-ls2088a-rdb.dtb
<DIR>      4096 grub
1152 hdr_ls1043ardb_bs.out
1152 hdr_ls1046ardb_bs.out
```

Bootloaders

U-Boot

```
16654848 Image
7443102 Image.gz
703 ls1043ardb_boot.scr
703 ls1046ardb_boot.scr
862 ls1088ardb_boot.scr
853 ls2088ardb_boot.scr
9035568 perf
8948941 ramdisk_rootfs_arm64.ext4.gz
8949005 ramdisk_rootfs_arm64.ext4.gz.uboot
<DIR> 4096 secboot_hdrs
7443166 uImage-4.4.65-dirty
<SYM> 19 vmlinuz
0 c80546ca-02
```

It shows that this USB drive contains scripts (and necessary images) to boot any of the boards LS1043ARDB, LS1046ARDB, LS1088ARDB, and LS2088ARDB. For example, the LS2088ARDB boot script is `ls2088ardb_boot.scr`. The script files are binary. But one can boot Linux and look at them. LSDK mounts the boot partition containing the scripts at mount point `/boot`.

```
user@localhost:~$ ls /boot
c80546ca-02                fsl-ls2088a-qds.dtb
firmware_ls1043ardb_uboot_norboot.img  fsl-ls2088a-rdb.dtb
firmware_ls1046ardb_uboot_qspiboot.img  grub firmware_ls1088ardb_uboot_qspiboot.img
hdr_ls1043ardb_bs.out
firmware_ls2088ardb_uboot_norboot.img  hdr_ls1046ardb_bs.out
flex_linux_arm64.itb                Image
fsl-ls1012a-frdm.dtb                Image.gz
fsl-ls1012a-qds.dtb                lost+found
flash_images.scr                    ls1012afrwy_boot.scr
fsl-ls1012a-rdb.dtb                ls1043ardb_boot.scr
fsl-ls1043a-qds.dtb                ls1046ardb_boot.scr
fsl-ls1043a-rdb-sdk.dtb                ls1088ardb_boot.scr
fsl-ls1043a-rdb-sdk.dtb                ls1088ardb_boot.scr
fsl-ls1043a-rdb-usdpaa.dtb            ls2088ardb_boot.scr
fsl-ls1046a-qds.dtb                perf
fsl-ls1046a-rdb-sdk.dtb                ramdisk_rootfs_arm64.ext4.gz
fsl-ls1046a-rdb-usdpaa.dtb            ramdisk_rootfs_arm64.ext4.gz.uboot
fsl-ls1088a-qds.dtb                secboot_hdrs
fsl-ls1088a-rdb.dtb                uImage-4.4.65-dirty
fsl-ls2080a-qds.dtb                vmlinuz
fsl-ls2080a-rdb.dtb
```

The boot scripts are sophisticated due to secure boot. Ignore secure boot, and the key steps in a boot script are:

```
part uuid $devtype $devnum:3 partuuid3
setenv bootargs console=ttyS1,115200 earlycon=uart8250,mmio,0x21c0600 root=PARTUUID=$partuuid3 rw
rootwait $othbootargs default_hugepagesz=2m hugepagesz=2m hugepages=256 load $devtype $devnum:
2 $kernel_addr_r /vmlinuz; load $devtype $devnum:2 $fdt_addr_r /fsl-ls2088a-rdb.dtb; bootm
$kernel_addr_r - $fdt_addr_r
```

The distro boot search process sets the variables `devtype` and `devnum`. In this example, they would be `"usb"` and `"0"`.

The U-Boot `part` command sets variable `partuuid3` to the partition universal unique identifier of partition 3 of USB device 0. This value is used in `bootargs` to identify the root partition to the Linux kernel. This method is better than using a device name (like `/dev/sda3`) because it is not dependent on probe order.

The next steps are to load the kernel image (`vmlinuz`) and device tree (`fsl-ls2088a-rdb.dtb`) into RAM and then boot Linux using `bootm`.

In summary (and ignoring secure boot), the distro boot processes searches for a partition with a file system containing a boot script. It loads and runs the boot script. The boot script does the five steps above to boot Linux.

To boot your own kernel, replace the kernel and device tree images in `/boot` and reboot your system. But also install any needed kernel modules first.

There are two types of userland in LSDK: 1) Large standard distro (Ubuntu rootfs) deployed on external SD/USB/SATA media storage. 2) Prebuilt tiny ramdisk rootfs (currently non-customizable) deployed in flash media onboard for arm32/arm64 target.

If U-Boot is used as boot loader, after LSDK is installed by flex-installer and reboots the target board, U-Boot will first automatically search for boot script `<platform>_boot.scr` from SD/eMMC/USB/SATA storage media, if a valid `<platform>_boot.scr` is found, U-Boot will boot the external distro (Ubuntu as default) deployed on SD/USB/SATA media storage, otherwise U-Boot will fall back to boot the tiny distro deployed on flash media onboard.

In case of booting LSDK tiny rootfs from flash media:

The default U-Boot environment `bootargs` is used and user can directly modify `bootargs` for custom kernel on demand.

In case of booting LSDK distro from external SD/USB/SATA storage disk:

The default U-Boot environment 'bootargs' is NOT used by external distro, `bootargs` is preset in `<platform>_boot.scr`, users can indirectly modify `othbootargs` on demand, for example, `setenv othbootargs fsl_fm_max_frm=9600` in U-Boot prompt.

5.3.2 LSDK U-Boot flash image feature

In case user needs to flash different image (e.g. `bl2`, `atf bl3 fip`, `dtb`, `kernel`, etc) to current or other bank to evaluate certain feature on Layerscape board, for example, to evaluate TDM feature with the non-default `rcw_1600_qetdm.bin` on LS1043ARDB:

```
1. change default rcw_1600.bin to rcw_1600_qetdm.bin for rcw_nor variable in configs/board/ls1043ardb/manifest in Flexbuild.
2. clean the obsolete atf images
   $ rm -rf build/firmware/atf/ls1043ardb
3. re-generate ATF image with new RCW specified by step 1
   $ flex-builder -c atf -m ls1043ardb -b nor
4. copy the new BL2 image build/firmware/atf/ls1043ardb/bl2_nor.pbl to flash_images/ls1043ardb directory of bootpartition on SD card
5. run the following commands in uboot prompt on ls1043ardb
   => setenv board ls1043ardb
   => setenv bd_type mmc
   => setenv bd_part 0:2
   => setenv bank other
   => ls $bd_type $bd_part flash_images/ls1043ardb
   # to update RCW in BL2
   => setenv img bl2
   => setenv bl2_img flash_images/ls1043ardb/bl2_nor.pbl
   => load $bd_type $bd_part $load_addr flash_images.scr
   => source $load_addr

   # similarly, to update dtb
   => setenv img dtb
   => setenv dtb_img fsl-ls1043a-rdb-usdpaa.dtb
   => source $load_addr
```

- To flash all images to current or other bank, set environment variable `img` to all by executing commands 'setenv img all' and 'source \$load_addr'.
- To flash single image, set environment variable `img` to one of following: `bl2`, `fip`, `mcfw`, `mcdpc`, `mcdpl`, `fman`, `qe`, `pfe`, `phy`, `dtb` or `kernel`
- If needed, you can override the default setting of variable `bd_part`, `flash_type`, `bl2_img`, `fip_img`, `dtb_img`, `kernel_img`, `qe_img`, `fman_img`, `phy_img`, `mcfw_img`, `mcdpl_img`, `mcdpc_img` before running 'source \$load_addr'.

5.4 UEFI

Release Description

This section provides information about the LSDK UEFI release on QorIQ LS boards. The following features are supported in this release:

- DUART
- I2C
- DSPI
- SATA
- SD, FAT32 filesystem
- GPIO Support
- RTC, Watchdog
- TF-A Integration
- PCIe – e1000 NIC card support
- DPAA 1.x support – XFI, RGMII, and SGMI
- DPAA2.x support - XFI
- SMP Linux boot via EFI_STUB on SD card
- PXE boot via PCIe and DPAA interfaces
- Ubuntu Distro boot support
- KASLR Support in UEFI
- QSPI boot
- USB 3.0
- Prefetch configuration support
- MC High Mem Support
- RTC Support for LS1043A x2 Board

Feature Summary

Features\Board	LS1043ARDB	LS2088ARDB	LS1046ARDB	LX2160ARDB
UART	YES	YES	YES	YES
I2C	Yes	YES	YES	YES
DSPI	Yes	Yes	No	NO
SATA	N/A	YES	YES	YES
SD,FAT32 filesystem	YES	YES	YES	YES
GPIO	YES	NO	YES	NO
Table continues on the next page...				

Table continued from the previous page...

IFC-NAND	YES	YES	NO	N/A
IFC-NOR	YES	YES	N/A	N/A
RTC	YES	YES	YES	YES
Watchdog	YES	YES	YES	YES
TF-A Integration	YES	YES	YES	YES
PCIe – e1000 NIC	YES	YES	YES (PCIe Using legacy interrupt)	YES
DPAA 1.x	YES	N/A	YES	N/A
DPAA 2.x	N/A	YES	N/A	YES
SMP Linux boot via EFI_STUB on SD card	YES	YES	YES	YES
PXE boot via PCIe and DPAA interfaces	YES	YES	YES	YES
Ubuntu Distro boot support	YES	YES	YES	YES
KASLR Support in UEFI	YES	YES	YES	YES
QSPI boot	No	NO	YES	YES (FSPI)
USB 3.0	No	YES	NO	YES
Prefetch Config support	No	YES	NO	NO
MC High Mem Support	N/A	YES	N/A	YES
Silicon Rev	1.0/1.1	1.0	1.0	1.0

Tool Chain

- gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu- Used to compile UEFI firmware (can be downloaded from https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/aarch64-linux-gnu/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu.tar.xz)

Known issues

QUEFI-780: ls1043ardb_uefi reconnect hang

Limitation

On LS1046ARDB, QSPI flash is disabled during device tree fix-up and Linux will not be able to use QSPI flash as UEFI run time services will be using it for variables storage.

5.4.1 Introduction

Purpose

This section describes how to use the accompanying LSDK release on the QorIQ Layerscape platforms and how to boot LSDK distro with UEFI. The section covers UEFI enablement on QorIQ Layerscape platforms and does not describe UEFI specifications in detail.

References

- [Unified Extensible Firmware Interface Specification](#)
- [QorIQ LS1043A Reference Manual](#)
- [QorIQ LS1046A Reference Manual](#)
- [QorIQ LS2088A Reference Manual](#)
- [QorIQ LX2160A Reference Manual](#)

5.4.2 UEFI overview

UEFI (Unified Extensible Firmware Interface) describes an interface between the operating system (OS) and the platform firmware. The interface consists of data tables that contain platform-related information, plus boot and runtime service calls that are available to the operating system and its loader. Together, these provide a standard, modern environment for booting an operating system and running pre-boot applications.

UEFI implementations are governed by the UEFI specifications, which are designed as a pure interface specification. As such, the specification defines the set of interfaces and structures that platform firmware must implement.

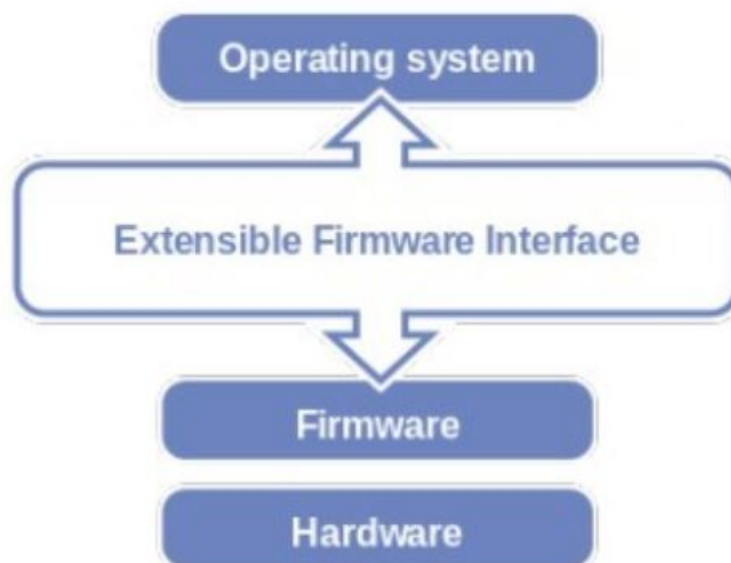


Figure 1. UEFI

For the latest version of UEFI, refer bellow references:

References

- [Unified Extensible Firmware Interface Specification](#)
- [QorIQ LS1043A Reference Manual](#)

- [QorIQ LS1046A Reference Manual](#)
- [LX2160ARM RevD_draft_review.PDF](#)
- [LS2088A Reference Manual](#)

UEFI Bootflow

The following is the Boot Execution Order on QorIQ LS boards:

- Execution begins in the PBI State Machine when the SoC comes out of reset
- After PBI, execution starts with SP boot core which gives control to GPP Boot core
- GPP boot core gives control to TFA.
- TF-A starts with BL2 image and load BL33 (UEFI image)
- TF-A jumps control to UEFI in EL2 mode.
- UEFI starts with DXE phase followed by BDS phase
- In BDS phase, UEFI loads OS (Linux) or OS loader (such as Grub)
- Linux starts in EL1 mode
- Linux Kernel invokes PSCI (cpu_on) to release secondary core.

Here, the TF-A firmware is the platform and runtime security firmware which allows configuring and enforcing platform specific security policies.

Environment requirements

Hardware requirements

- Host PC: Ubuntu (64-bit variant with at least 2 GB RAM) host is preferred to compile/build the UEFI firmware.
- Board: QorIQ Layerscape, with UART cable.
- SD Card: Preferably from well-known vendors like SanDisk.

Software requirements

- To build the UEFI firmware on the Ubuntu host, install uuid-dev.

```
$ sudo apt-get install uuid-dev
```

- To build the UEFI firmware on the ubuntu Host, Install Linaro GCC-4.9 toolchain on your Host machine using the following commands.

```
$ wget https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/aarch64-linux-gnu/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu.tar.xz
$ tar -xvf gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu.tar.xz
```

Add path of these toolchains to the PATH environment variable:

```
$ export PATH=/home/user/linaro_4.9_2016/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu/bin:
$PATH
```

- Ensure that Python (2.7 or higher) is installed on the build machine, for successful compilation.

```
$ python -version
Python 2.7.12
```

- Ensure DTC compiler version is 1.4.3 or later. Below are the steps to Check and Update DTC tool version

```
$ dtc --version
Version: DTC 1.4.0
$ mkdir ~/dtc
$ cd ~/dtc
$ git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
$ cd dtc/
$ git checkout -b v1.4.7 tags/v1.4.7
$ cd ../
$ make -C dtc DESTDIR=$(pwd) install
$ export PATH = $(pwd)/$HOME/bin:$PATH
$ dtc --version
Version: DTC 1.4.7
```

5.4.3 LSDK distro boot with UEFI

Boot up Image Requirements

The following images are required for UEFI boot.

- BL2 and BL33 (UEFI Image)
- FMAN (Frame Manager) Micro Code - Required for DPAA1 (Data Path Acceleration Architecture) interfaces, applicable for LS1043ARDB board.
- Mc Firmware, DPL and DPC binaries for DPAA2 Interfaces, applicable for LS2088ARDB/LX2160ARDB board.
- Phy Firmware (for Cortina Phy), applicable for LS2088ARDB board.

UEFI Boot Order

The UEFI boot manager will try to boot from all entries as they appear in the UEFI boot menu. Boot entries can be divided into the following three categories.

- Boot entries for Block devices
- Boot entries for Network Boot (PXE boot)
- Boot entry for UEFI Shell

For block devices, the UEFI boot manager will look for an EFI application with a predefined name (`BOOT{machine type short name}.EFI`) in `/EFI/BOOT`. If found, the boot manager will automatically run the EFI application. In our case (AArch64 ARM platforms), the application should be named as `BOOTAA64.EFI`.

Linux Boot Storage Media Layout

Table 9. Media Layout

Region 1 (4 KB)	Region 2 (64 MB)	Region 3 (20 MB)	Region 4 (1GB)	Region 5 (left space of disk)
MBR/GPT	Loaders & FW UEFI TF-A firmware FMan or MC firmware lsdk_linux_arm64_tiny .itb	Partition 1 (FAT 16/32) LABEL: EFI BOOTAA64.EFI grub.cfg	Partition 2 (EXT4) LABEL: boot Kernel DTB lsdk_linux_arm64_tiny .itb distro boot scripts other	Partition 3 (EXT4) LABEL: rootfs Ubuntu or Ubuntu-Core or CenOS or Debian

In the Linux environment, fdisk utility can be used to partition and format the target as per the table above and then copy the required images (Kernel image, Rootfs) to the target.

For LSDK, flex-builder & flex-installer utility can be used to partition and install required images to target as per above layout. For more information on how to build and install LSDK using flex-builder and flex-installer, refer to [LX2160A BSP user guide](#) on page 32.

Image name	Partition
BOOTAA64.EFI, grub.cfg	EFI partition
Kernel image	boot partition
Root file system	rootfs partition

NOTE

- The Kernel Image should be the standard (uncompresses) kernel images build as arch/arm64/boot/Image (for arm64).
- The device tree has to be stored in flash at a fixed offset (board specific) as per the LSDK flash layout. For e.g. for LS1043ARDB NOR flash, default value is (0x60F00000 to 0x60FFFFFF). Update 'PcdFdtAddress' PCD in platform description file (.dsc) for a different flash layout.

Sample files

• BOOTAA64.EFI

Represents grub boot loader. It will load the grub.cfg kept in the same directory and provides grub menu to the user. The user can select the required menu entry. Follow 'Generate BOOTAA64.EFI' for compilation steps.

• grub.cfg

grub.cfg provides boot options to user. For example, the grub.cfg can be used for booting the LSDK distro.

```
set default="1"
set timeout=10

menuentry 'LSDK on QorIQ ARM64 ls1043ardb' {
    search --no-floppy --file /42013eab-02 --set root
    linux /Image console=ttyS0,115200 root=PARTUUID=42013eab-03 rootwait rw
    earlycon=uart8250,mmio,0x21c0600
}
```

NOTE

The example above uses a LSDK distribution. Specific kernel boot arguments could vary per distribution.

The table below represents the configurable parameters in grub.cfg.

Option	Explanation	Comments
<i>search --no-floppy --file /<FileName> --set root</i>	Search all partitions for the given file name. First Partition containing the specified file is set at root so that Grub will look for required image (kernel image) in this partition.	For example, for LSDK, flex-installer will update the FileName as PARTUUID of the partition containing Kernel image. This removes the ambiguity of finding a specific partition based on UUID/LABEL when multiple devices are connected.
<i>root=PARTUUID=XXXXXXXX-YY</i>	It represents the PARTUUID of the partition containing rootfs. This is passed as boot argument to kernel.	For example, for LSDK, flex-installer will update it with PARTUUID of the partition containing rootfs. This makes sure that correct rootfs is passed to kernel and removes the ambiguity of finding the correct rootfs based on UUID/LABEL when multiple devices are connected.
<i>set timeout=N</i>	If defined, GRUB will wait 'N' Seconds, before booting the default menu entry. If not defined, user always has to select the required menu entry.	Adjust the timeout value as per requirement.

LSDK Distro Bootflow with UEFI

- All required files (BL2, BL33 (fip) fman and UEFI images) are stored in NOR flash.
- UEFI Boot starts from DDR
- When prompted, Press ESCAPE to select a Boot Option (UEFI SHELL/PXE Boot) OR else UEFI Boot Manager tries to boot from all boot entries starting from 'Removable Media' followed by 'Network Boot' and 'UEFI SHELL'
- If a Removable Media (e.g. SD card) has a FAT formatted partition with /EFI/BOOT/BOOT{machine type short name}.EFI (Ex: BOOTAA64.EFI for arm64), it will be executed by UEFI Boot Manager.
- BOOTAA64.EFI will load 'grub.cfg'.
- grub.cfg contains menu entry for distro along with required parameter to identify kernel image and pass 'rootfs' path to kernel as boot argument.

Flow diagram

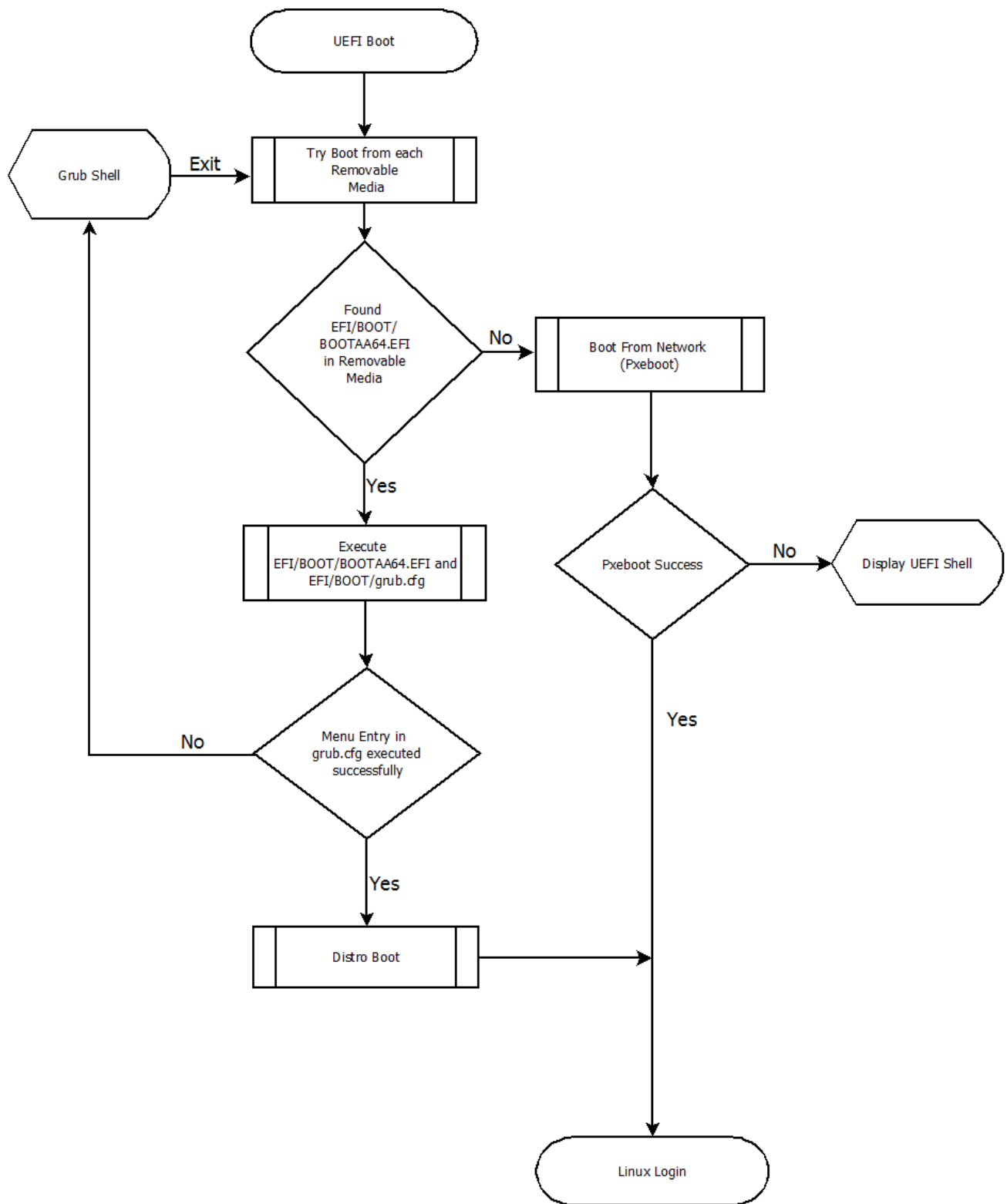


Figure 2. UEFI Bootflow Diagram

Generate BOOTAA64.EFI

Follow the steps below to compile the grub source code to generate `BOOTAA64.EFI`. In general, `BOOTAA64.EFI` is provided by the distribution like LSDK provides prebuild `BOOTAA64.EFI`.

- Get the grub source code and install the prerequisites as mentioned in `INSTALL` file.

```
$ git clone git://git.savannah.gnu.org/grub.git;  
$ cd grub
```

- Use grub-2.02 tag.

```
$ git checkout tags/grub-2.02
```

- Set the toolchain path and set `CROSS_COMPILE` environment variable. See the Software Requirements section to download and install the toolchain.

```
$ export PATH=/home/user/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu/bin:$PATH  
$ export CROSS_COMPILE=aarch64-linux-gnu-
```

- Configure and compile source code for target (arm64)

```
$ ./autogen.sh  
$ ./configure --target=aarch64-linux-gnu  
$ make
```

- Create standalone GRUB image.

```
$ echo 'configfile ${cmdpath}/grub.cfg' > grub.cfg  
$ grub-mkstandalone --directory=./grub-core -O arm64-efi -o BOOTAA64.EFI --modules "part_gpt  
part_msdos" /boot/grub/grub.cfg=./grub.cfg
```

NOTE

- GRUB standalone application has all the modules embedded in application itself and capability to recognize different file system (ext2, ext4, and so on), thus removing the need for having a separate directory populated with all of the GRUB UEFI modules and other related files.
- 'configfile \${cmdpath}/grub.cfg' instruct GRUB EFI (`BOOTAA64.EFI`) to use `grub.cfg` placed in same directory. Thus, making them portable.
- Option `--modules="part_gpt part_msdos"` (with the quotes) is necessary for `${cmdpath}` feature to work properly and to recognize MBR and GPT partitioning.

Conventions for UEFI and U-Boot compatibility

UEFI needs the following firmware:

- Firmware (RCW, TF-A, Fman/MC ucode, boot loader, and so on) is in NOR flash
- Kernel image is on `/boot` on a mass storage device
- Root file system is in a mass storage device (/)

U-Boot must boot Linux with the dtb in NOR (not in a kernel itb) but the kernel image (not itb form) is stored in `/boot`. This means that U-Boot `boot.scr` must extload a kernel image but not the dtb. It is consistent with what the LSDK specification says about using `booti`.

5.4.4 Product Execution

5.4.4.1 LX2160ARDB

By default (per board switch settings), the boot loader (U-Boot) image located in FlexSPI NOR flash DEV#0 runs on power on. Press any key while U-Boot is counting down to stop U-Boot from automatically running the "bootcmd" variable and booting Linux.

As the U-Boot boots to its prompt, users can use the commands listed below to deploy new images onto the RDB.

```
$i2c mw 66 50 20; sf probe 0:0;

$tftp a0000000 <path>/fip_ddr_all.bin; sf erase 0x800000 +40000; sf write 0xa0000000 0x800000 0x40000;

$tftp a0000000 <path>/bl2_flexspi_nor.pbl; sf erase 0 +0x40000; sf write 0xa0000000 0x0 0x40000;

$tftp a0000000 <path>/fip.bin; sf erase 0x100000 +40000; sf write 0xa0000000 0x100000 0x40000;

$tftp c0000000 <path>/mc.itb; sf erase a00000 +200000;sf write c0000000 a00000 200000;

$tftp d0000000 <path>/dpl-eth.19.dtb; sf erase d00000 +c0000;sf write d0000000 d00000 c0000;

$tftp e0000000 <path>/dpc-warn.dtb; sf erase e00000 +c0000;sf write e0000000 e00000 c0000;

$tftp a0000000 <path>/fsl-lx2160a-rdb.dtb; sf erase f00000 +c0000;sf write a0000000 f00000 c0000;

Once images are flashed, reset the board to boot from device#1

$qixis_reset altbank
```

This will start booting the UEFI, UEFI boot manager can be preempted by pressing 'esc' key. Or UEFI will start booting as defined in boot-manager, it will start booting from PXE interfaces first.

Booting Linux

Please follow below steps to boot Linux using EFI_STUB:

1. Copy kernel (Image), rootfs (fsl-image-core-lx2160ardb.ext2.gz and boot.nsh to FAT32 formatted SD card and insert it into your LX2160 RDB board. You can also tftp these images on SD card from UEFI shell using either XFI/RGMII interface of DPAA or PCIe.
2. Start UEFI and goto UEFI Shell (follow above mentioned steps).
3. Go to the SD card filesystem on Shell (e.g. FS0 in attached snapshot), by inputting FS0.
4. Run `boot.nsh`

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s):HD0b:;BLK1:
      VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBBCF)/HD(1,GPT,728F8E16-5C99-784
D-B263-7E1778D03A20,0x800,0x1CE97DF)
BLK0: Alias(s):
      VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBBCF)
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> ls
Directory of: FS0:\
08/31/2018  20:50          18,616,832  Image
08/31/2018  18:25              262  boot.nsh
09/05/2018  10:15        45,134,721  fsl-image-core-lx2160ardb.ext2.gz
          3 File(s)  63,751,815 bytes
          0 Dir(s)
FS0:\> █
```

5.4.4.2 LS1043ARDB

By default (per board switch settings), the boot loader (U-Boot) image located in IFC NOR flash vbank 0 runs on power on. Press any key while U-Boot is counting down to stop U-Boot from automatically running the "bootcmd" variable and booting Linux.

As the U-Boot boots to its prompt, users can use the commands listed below to deploy new images onto the RDB.

```
=> tftp 0x80000000 <path>/bl2_nor.pbl; erase 0x64000000 +$filesize; cp.b 0x80000000
0x64000000 $filesize

=> tftp 0x80000000 <path>/fip.bin; erase 0x64100000 +$filesize; cp.b 0x80000000
0x64100000 $filesize

=> tftp 0x80000000 <path>/fsl_fman_ucode_ls1043_r1.1_108_4_9.bin; protect off 0x64900000 +
$filesize; erase 0x64900000 +$filesize; cp.b 0x80000000 0x64900000 $filesize; protect on
0x64900000 +$filesize

=> tftp 0x80000000 <path>/fsl-ls1043a-rdb.dtb; protect off 0x64f00000 +$filesize; erase
0x64f00000 +$filesize; cp.b 0x80000000 0x64f00000 $filesize
```

Once images are flashed, reset the board to boot from alternate bank i.e. vbank4.

```
$cp1d reset altbank
```

This will start booting the UEFI, UEFI boot manager can be preempted by pressing 'esc' key.

Or UEFI will start booting as defined in boot-manager, it will start booting from PXE interfaces first.

Booting Linux

Please follow below steps to boot Linux using EFI_STUB:

1. Copy kernel (Image), rootfs (ramdisk_rootfs_arm64.ext4.gz) and boot.nsh to FAT32 formatted SD card and insert it into your LS1043 RDB board. You can also tftp these images on SD card from UEFI shell using either XFI/RGMII/SGMII interface of DPAA or PCIe.

2. Start UEFI and goto UEFI Shell (follow above mentioned steps).
3. Go to the SD card filesystem on Shell (e.g. FS0 in attached snapshot), by inputting FS0.
4. Run `boot.nsh`

```

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
  FS0: Alias(s):HD0b:;BLK1:
        VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBBCF)/HD(1,GPT,728F8E16-5C99-784
        D-B263-7E1778D03A20,0x800,0x1CE97DF)
  BLK0: Alias(s):
        VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBBCF)
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> ls
Directory of: FS0:\
08/31/2018  20:50          18,616,832  Image
08/31/2018  18:25              262  boot.nsh
09/05/2018  10:15        45,134,721  fsl-image-core-lx2160ardb.ext2.gz
          3 File(s)  63,751,815 bytes
          0 Dir(s)
FS0:\>

```

5.4.4.3 LS1046ARDB

Booting LS1046A-RDB to UEFI prompt (via QSPI BOOT)

LS1046A QSPI flash map

Base address for Primary Bank (VBank0 (Bank 0) 64MB) is 0x40000000 and Base address for Secondary Bank (VBank4 (Bank 4) 64MB) is 0x44000000.

Flashing UEFI images on QSPI flash bank 4 (alternate QSPI flash bank)

Boot to u-boot prompt from QSPI flash primary bank (Bank 0).

- Setup serial port connection on host machine to capture logs from the target LS1046A RDB board.
- Reset the board to boot u-boot on bank 0, make sure that there is a valid u-boot image flashed there

Copy Images to QSPI flash alternate bank using following commands:

```

=> sf probe 0:1
=> setenv bl2 'tftpboot 0x82000000 bl2_qspi.pbl; sf erase 0 +$filesize && sf write 0x82000000
0 $filesize'
=> run bl2
=> setenv fip 'tftpboot 0x82000000 fip.bin; sf erase 100000 +$filesize && sf write 0x82000000
100000 $filesize'
=> run fip
=>setenv fman 'tftpboot 0x82000000 fsl_fman_ucode_ls1046_r1.0_108_4_9.bin; sf erase 900000 +
$filesize && sf write 0x82000000 900000 $filesize'
=> run fman
=>setenv dtb 'tftpboot 0x82000000 fsl-ls1046a-rdb.dtb; sf erase f00000 +$filesize && sf write
0x82000000 f00000 $filesize'
=> run dtb

=>setenv uefi_nv 'tftpboot 0x82000000 LS1046ARDBNV_EFI.fd; sf erase 500000 +$filesize && sf write

```

```
0x82000000 500000 $filesize'
=> run uefi_nv (This is optional, needed if first fast boot needed)
```

Once images are flashed, reset the board to boot from alternate bank.

```
$cp1d reset altbank
```

This will start booting the UEFI, UEFI boot manager can be preempted by pressing 'esc' key.

Or UEFI will start booting as defined in boot-manager, it will start booting from PXE interfaces first.

5.4.4.4 LS2088ARDB

By default (per board switch settings), the boot loader (U-Boot) image located in IFC NOR flash BANK#0 runs on power on. Press any key while U-Boot is counting down to stop U-Boot from automatically running the "bootcmd" variable and booting Linux.

As the U-Boot boots to its prompt, users can use the commands listed below to deploy new images onto the RDB.

```
$tftp a0000000 <path>/ bl2_nor.pbl; erase 0x584000000 +$filesize; cp.b 0xa0000000
0x584000000 $filesize';
$tftp a0000000 <path>/ fip.bin; erase 0x584100000 +$filesize; cp.b 0xa0000000 0x584100000 $filesize'
$tftp a0000000 <path>/ LS2088ARDBNV_EFI.fd; erase 0x584500000 +$filesize; cp.b 0xa0000000
0x585100000 $filesize' <If fast first boot is needed>
$tftp 0xa0000000 <path>/mc.itb ; erase 0x584A00000 0x584FFFFFF; cp.b 0xa0000000
0x584A00000 $filesize ;
$ tftp 0xa0000000 <path>/dpc.0x2A_0x41.dtb ; cp.b 0xa0000000 0x584E00000 $filesize ;
$ tftp 0xa0000000 <path>/dpl-eth.0x2A_0x41.dtb ; cp.b 0xa0000000 0x584D00000 $filesize
$ 'tftp 0x80000000 cs4315-cs4340-PHY-ucode.txt; erase 0x584980000 0x5849BFFFF ; cp.b 0x80000000
0x584980000 $filesize' <If needed flash cortina phy firmware>
$tftp 0xa0000000 <path>/fsl-ls2088a-rdb.dtb ; erase 0x584F00000 0x584FFFFFF; cp.b 0xa0000000
0x584F00000 $filesize ;
```

Once images are flashed, reset the board to boot from bank#1

```
$qixis_reset altbank
```

This will start booting the UEFI, UEFI boot manager can be preempted by pressing 'esc' key.

Or UEFI will start booting as defined in boot-manager, it will start booting from PXE interfaces first.

Booting Linux

Please follow below steps to boot Linux using EFI_STUB:

1. Copy kernel (Image), rootfs (fsl-image-core-ls2088ardb.ext.gz and boot.nsh to FAT32 formatted SD card and insert it into your LS2088 RDB board. You can also tftp these images on SD card from UEFI shell using either DPAA interface or PCIe.
2. Start UEFI and goto UEFI Shell (follow above mentioned steps).
3. Go to the SD card filesystem on Shell (e.g. FS0 or FS1 in attached snapshot), by inputting FS0.
4. Run `boot.nsh`

```

UEFI v2.70 (EDK II, 0x00010000)
Mapping table
  FS0: Alias(s):HD0b0b;;BLK1:
        VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,000010030000000000)/USB(0x1
0x0)/HD(1,MBR,0xC094289C,0x20000,0xA001)
  FS1: Alias(s):HD3b;;BLK7:
        VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBB CF)/HD(1,MBR,0xD55D926A,0x800,
0x1DAC400)
  BLK5: Alias(s):
        VenHw(616FE8D8-F4AA-42E0-A393-B332BDB2D3C1)
  BLK8: Alias(s):
        VenHw(DA2406B3-FEEB-455B-AF1F-C45E34888524)
  BLK6: Alias(s):
        VenHw(7948A4CA-2F2E-41CA-90A2-D4420CECBB CF)
  BLK0: Alias(s):
        VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,000010030000000000)/USB(0x1
0x0)
  BLK2: Alias(s):
        VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,000010030000000000)/USB(0x1
0x0)/HD(2,MBR,0xC094289C,0x2A001,0x200001)
  BLK3: Alias(s):
        VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,000010030000000000)/USB(0x1
0x0)/HD(3,MBR,0xC094289C,0x22A002,0x71B5FFE)
  BLK4: Alias(s):
        VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,000011030000000000)/USB(0x1
0x0)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
shell> fs)l
'fs)l' is not recognized as an internal or external command, operable program, o
r script file.
shell> fs0:
FS0:\> ls
ls: File Not Found - 'FS0:\'
FS0:\> fs1:
FS1:\> ls
Directory of: FS1:\
0/24/2018  12:34          21,318,144  vmlinuz-4.14.67
0/24/2018  12:34          20,298,240  vmlinuz-4.9.124
0/24/2018  12:35          11,079,672  ramdisk_rootfs_arm64.ext4.gz
0/24/2018  12:36              208      startup.nsh
0/24/2018  12:37        37,035,440  firmware_ls2088ardb_uboot_norboot.img
0/26/2018  05:11          26,710      fsl-ls2088a-rdb-uboot.dts
0/26/2018  05:11          20,809      fsl-ls2088a-rdb-uboot.dtb
0/26/2018  05:24           196      boot.nsh
            8 File(s)  89,779,419 bytes
            0 Dir(s)
FS1:\> cat boot.nsh
vmlinuz-4.14.67 initrd=ramdisk_rootfs_arm64.ext4.gz console=ttyS1,115200 root=/d
ev/ram0 earlycon=uart8250,mmio,0x21c0600 ramdisk_size=0x2000000 default_hugepage
sz=2m hu[gepagesz]=2m hugepages=256
FS1:\> █

```

5.4.5 LSDK Distro Boot Logs

Below steps are given to build source code, this is not necessary to build all source code. Needed source code can be build using below steps UEFI build is supported with GCC49 and GCC54, download toolchain from linaro web <https://releases.linaro.org/components/toolchain/binaries/4.9-2017.01/>.

1. RCW:

git clone the rcw repository.

Fetch repository for RCW as:

```
$git clone ssh://git@bitbucket.sw.nxp.com/dash/dash-rcw.git
$cd dash-rcw
$git checkout -b devel remotes/origin/devel
```

Compile RCW as:

```
$make
```

2. Linux:

git clone the linux repository and prepare the cross-compile environment to build linux (install toolchain provided with Yocto).

Fetch repository for RCW as:

```
$git clone ssh://git@bitbucket.sw.nxp.com/dash/dash-lts.git
$cd dash-lts
$git checkout -b linux-4.14 remotes/origin/linux-4.14
```

Compile Linux as:

```
$source toolchain_path/environment-setup-aarch64-fsl-linux
$export LDFLAGS="--hash-style=gnu"
$make distclean
$ ./scripts/kconfig/merge_config.sh arch/arm64/configs/defconfig arch/arm64/configs/lsdk.config
$make -j4 all
```

3. UEFI:

git clone the uefi repository and prepare the cross-compile environment to build uefi (install toolchain gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu).

Fetch repository for RCW as:

a. Get the edk2

```
$git clone ssh://git@bitbucket.sw.nxp.com/dnnpi/edk2.git
$git checkout -b master remotes/origin/master

$cd edk2
TAG: LSDK-18.12
```

b. Get the edk2-platforms

```
$git clone ssh://git@bitbucket.sw.nxp.com/dnnpi/edk2-platforms.git
$git checkout -b edk2_upstream_re_arch remotes/origin/ edk2_upstream_re_arch
TAG: LSDK-18.12
```

Compile UEFI as:

```
$ cd edk2
$ source edksetup.sh
$ cd edk2-platforms/Platform/NXP
$ source Env.cshrc

$ Build base tools (This is one-time activity, Italic Bold text below)
// remove following files if exists
$ rm -rf (Base_dir)/edk2/Conf/build_rule.txt
$ rm -rf (Base_dir)/edk2/Conf/tools_def.txt
```



```
$ rm -rf (Base_dir)/edk2/Conf/target.txt
$ make -C (Base_dir)/edk2/BaseTools/Source/C

$ ./build.sh <SOC> <BOARD> RELEASE clean
$ ./build.sh <SOC> <BOARD> RELEASE
```

Where SOC could be LX2160, LS2088, LS1043 or LS1046

BOARD value will be RDB e.g for LS1046 build (./build.sh LS1046 RDB RELEASE)

Compilation will generate two files <SOC>ARDB_EFI.fd and <SOC>ARDBNV_EFI.fd in
edk2/Build/<SOC>aRdbPkg/RELEASE_GCC49/FV directory.

(e.g LS1046ARDB_EFI.fd and LS1046ARDBNV_EFI.fd in edk2/Build/LS1046aRdbPkg/RELEASE_GCC49/FV directory)

If fast boot is needed, then <SOC>ARDBNV_EFI.fd should be flashed at 5MB offset in flash. If not flashed, UEFI will create this file on first boot, subsequent boot will be fast except first.

4. ATF:

This section is added for completeness, please refer ATF documentation and release to be in sync with latest development.

git clone the atf repository from [ATF](#) and checkout branch dev03, prepare the cross compile environment to build atf (install toolchain gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu).

```
$cd atf
```

Copy UEFI <SOC>ARDB_EFI.fd image and RCW into ATF folder.

NOTE

UEFI and RCW images can be taken from pre-build images.

```
$ export CROSS_COMPILE= aarch64-linux-gnu-
```

LX2160 build

```
$ make PLAT=lx2160ardb bl2 pbl BOOT_MODE=flexspi_nor RCW=rcw_2000_700_2400_19_5_2.bin
$make PLAT=lx2160ardb bl31
```

Above two steps will create bl2_flexspi_nor.pbl in atf/build/lx2160ardb/release/ directory

```
$make PLAT=lx2160ardb fip BL33=LX2160ARDB_EFI.fd
```

It will create fip.bin in atf/build/lx2160ardb/release/ directory

```
$ tools/fiptool/fiptool create --ddr-immem-udimm-1d ddr4_pmu_train_imem.bin --ddr-immem-udimm-2d
ddr4_2d_pmu_train_imem.bin --ddr-dmmem-udimm-1d ddr4_pmu_train_dmem.bin --ddr-dmmem-udimm-2d
ddr4_2d_pmu_train_dmem.bin --ddr-immem-rdimm-1d ddr4_rdimm_pmu_train_imem.bin --ddr-immem-
rdimm-2d ddr4_rdimm2d_pmu_train_imem.bin --ddr-dmmem-rdimm-1d ddr4_rdimm_pmu_train_dmem.bin --
ddr-dmmem-rdimm-2d ddr4_rdimm2d_pmu_train_dmem.bin fip_ddr_all.bin
```

It will create fip_ddr_all.bin in atf directory

LS1043 build

```
make PLAT=ls1043ardb pbl bl2 BOOT_MODE=nor RCW=rcw_uefi_1500.bin
make PLAT=ls1043ardb fip BL33=LS1043ARDB_EFI.fd
This will create fip.bin and bl2_nor.pbl in /build/ ls1043ardb/release/ folder
```

LS1046 build

```
make PLAT=ls1046ardb bl2 pbl BOOT_MODE=qspi RCW=rcw_1600_qspiboot.bin  
make PLAT=ls1046ardb fip BL33=LS1046ARDB_EFI.fd
```

This will create fip.bin and bl2_qspi.pbl in /build/ls1046ardb/release/ folder

LS2088 build

```
make PLAT=ls2088ardb pbl bl2 BOOT_MODE=nor RCW=rcw_2100.bin  
make PLAT=ls2088ardb fip BL33=LS2088ARDB_EFI.fd
```

This will create fip.bin and bl2_qspi.pbl in build/ls2088ardb/release/ folder

5.4.6 PXE Boot

This section describes the steps required to boot the Linux kernel using PXE boot. UEFI is the primary bootloader. It loads the GRUB2 bootloader image. PXE boot is used to load the kernel and root file system images .

Hardware Requirements

- **Host PC:** Ubuntu (64-bit variant with at least 2GB RAM) host is preferred to compile/build the UEFI firmware.
- **Board:** QorIQ Layerscape reference development board (RDB) with a UART cable.

Software Requirements

- To build the UEFI firmware on the Ubuntu Host, install `uuid-dev`. Run the following command:

```
$ sudo apt-get install uuid-dev
```

- Ensure that Python (2.7 or higher) is installed on the build machine.
- DHCP server: `isc-dhcp-server` Version 4.2.4 or higher.
- Tftp Server: Any of below tftp server should be installed on host machine.
 - `tftpd-hpa`
 - `atftpd`
 - `dnsmasq`
- To build the grub bootloader on the ubuntu Host, Install Linaro GCC-4.9 toolchain on your Host machine using the following commands:

```
$ wget https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/aarch64-linux-gnu/  
gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu.tar.xz  
$ tar -xvf gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu.tar.xz  
$ export PATH=/home/user/linaro_4.9_2016/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu/bin:  
$PATH
```

5.4.6.1 Creating the PXE Boot Setup

Setting up DHCP server for PXE boot

1. Open `/etc/dhcp/dhcpd.conf` with write permissions on the server machine.

2. Add a configuration block for PXE boot to the file.

```
host ls1043rdbboard15 {
    hardware ethernet 26:5E:3D:21:00:02;
    fixed-address 192.168.3.41;
    next-server 192.168.3.161;
    filename "grub.efi"
}
```

NOTE

Use the MAC address for which PXE boot entry is created.

3. Restart the dhcp server (The command below is for Ubuntu. It may change for a different Host)

```
sudo service isc-dhcp-server restart
```

4. Place the following files in the tftp server root directory with execute permission.

- ramdisk_rootfs_arm64.ext4.gz (Root file system) : It can be fetched using flex-builder.

Run `flex-builder -i repo-fetch` to download the Root File System.

Path: `packages/installer/ramdiskrfs/ramdisk_rootfs_arm64.ext4.gz`.

Refer to [#unique_57](#) for more info flex-builder usage.

- Image (Kernel Image) : It can be generated using flex-builder.

Run `flex-builder -c linux -a arm64` to generate kernel image (Image).

Refer to [#unique_57](#) for more info flex-builder usage.

NOTE

Image is the standard kernel image generated at `arch/arm64/boot/Image`.

- grub.cfg : Below is the sample grub.cfg for LS1043ARDB.

```
set default="0"
function load_video {
    if [ x$feature_all_video_module = xy ]; then
        insmod all_video
    else
        insmod efi_gop
        insmod efi_uga
        insmod ieee1275_fb
        insmod vbe
        insmod vga
        insmod video_bochs
        insmod video_cirrus
    fi
}
load_video
set gfxpayload=keep
set timeout=10
menuentry 'LSDK 1706 on QorIQ ARM64' --class red --class gnu-linux --class gnu --class os {
    linux /Image console=ttyS0,115200 root=/dev/ram0 rw earlycon=uart8250,mmio,0x21c0500
    ramdisk_size=500000 default_hugepagesz=2m hugepagesz=2m hugepages=256
    initrd /ramdisk_rootfs_arm64.ext4.gz
}
```

NOTE

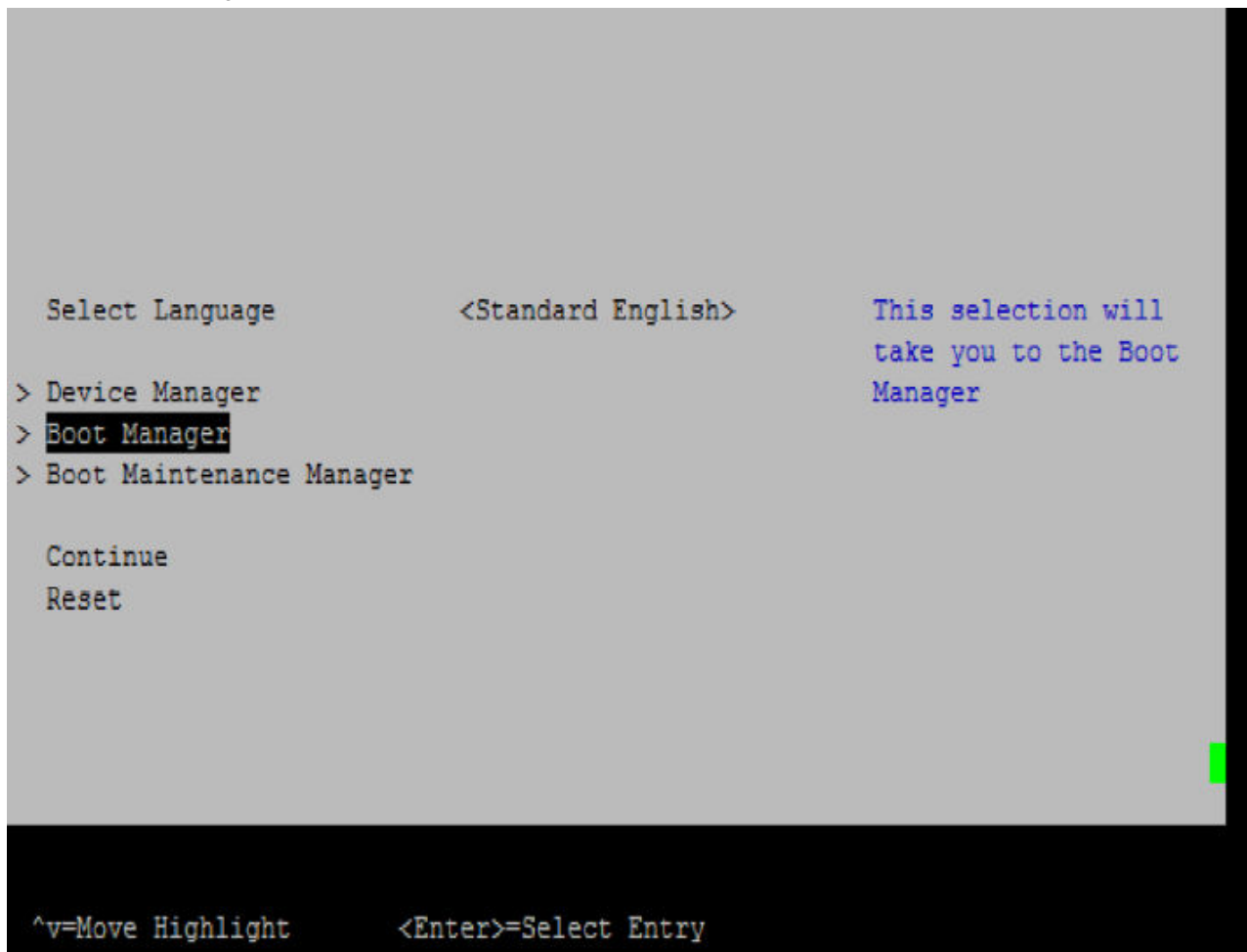
This is a sample grub configuration file for LS1043ARDB. Kernel boot arguments may change for different QorIQ LS board.

5. grub.efi: Grub bootloader image. The steps below can be followed to generate grub.efi:

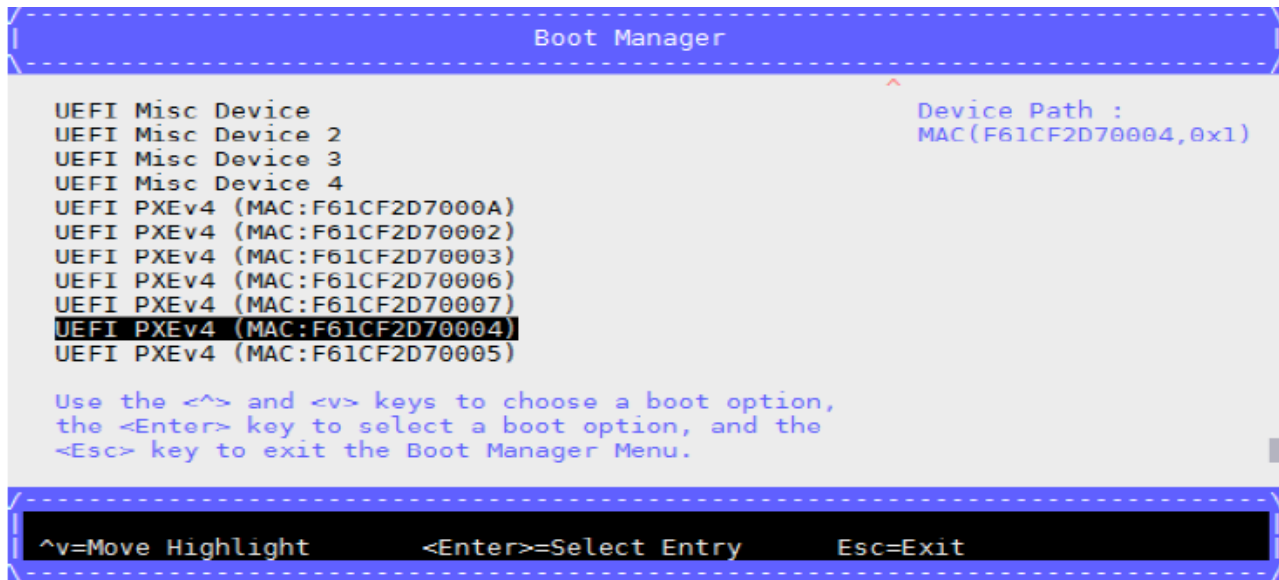
```
git clone git://git.savannah.gnu.org/grub.git;cd grub
git checkout tags/grub-2.02
export PATH=/home/user/gcc-linaro-4.9-2016.02-x86_64_aarch64-linux-gnu/bin:$PATH
export CROSS_COMPILE=aarch64-linux-gnu-
./autogen.sh
./configure --target=aarch64-linux-gnu
Make
echo ' set root=(tftp)' > grub.cfg
echo 'configfile /grub.cfg' >> grub.cfg
grub-mkstandalone --directory=./grub-core -O arm64-efi -o grub.efi --modules "tftp net efinet
gzio linux efifwsetup part_gpt part_msdos font gfxterm all_video" /boot/grub/grub.cfg=./grub.cfg
```

5.4.6.2 Installing the Kernel

- Boot UEFI to prompt using NOR Flash.
- Press Esc when prompted to enter the Boot Menu.
- Enter the Boot Manager



- Choose PXE boot option. Make sure that connectivity to dhcp server is fine and dhcp server is set up for PXE boot.



- From grub menu select the option to install linux kernel.

```

DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC1 ...
EFI stub: Booting Linux Kernel...
EFI stub: Using DTB from configuration table
EFI stub: Exiting boot services and installing virtual address map...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC9 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC1 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC2 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC5 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC6 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC3 ...
DPAA1: Disabling DPAA1 Ethernet physical device for MEMAC4 ...
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC9 ...
DPAA1 ERROR: Failed to uninstall UEFI driver model protocols for DPAA1 device 0xFE839018 (error 15)
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC2 ...
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC5 ...
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC6 ...
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC3 ...
DPAA1: Stopping DPAA1 Ethernet physical device for MEMAC4 ...
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Initializing cgroup subsys cpuacct
[ 0.000000] Linux version 4.4.65 (b48164@uefi-workstation) (gcc version 5.4.0 20160609 (Ubuntu/Linaro 5.4.0-6ubuntu1~16.04.4) ) #1 SMP PREEMPT Mon May 29 16:38:31 IST 2017
[ 0.000000] Boot CPU: AArch64 Processor [410fd034]
[ 0.000000] earlycon: Early serial console at MMIO 0x21c0500 (options '')
[ 0.000000] bootconsole [uart0] enabled
[ 0.000000] efi: Getting EFI parameters from FDT:
[ 0.000000] EFI v2.60 by EDK II
[ 0.000000] efi:
[ 0.000000] Reserved memory: initialized node qman-fqd, compatible id fsl,qman-fqd
[ 0.000000] Reserved memory: initialized node qman-pfdr, compatible id fsl,qman-pfdr
[ 0.000000] Reserved memory: initialized node bman-fbpr, compatible id fsl,bman-fbpr
[ 0.000000] cma: Reserved 16 MiB at 0x00000000fb000000
[ 0.000000] psci: probing for conduit method from DT.
[ 0.000000] psci: PSCIv0.2 detected in firmware.
[ 0.000000] psci: Using standard PSCI v0.2 function IDs
[ 0.000000] psci: Trusted OS migration not required
[ 0.000000] PERCPU: Embedded 21 pages/cpu @ffff80007fb6a000 s45336 r8192 d32488 u86016
[ 0.000000] Detected VIPT I-cache on CPU0

```

- Login after successful installation.

```

INIT: version 2.88 booting
Starting udev
[ 5.565793] udevd[2224]: starting version 182
[ 5.875032] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[ 5.895077] EXT4-fs (ram0): re-mounted. Opts: (null)
[ 5.923480] random: dd: uninitialized urandom read (512 bytes read, 9 bits of entropy available)
Populating dev cache
tar: dev/disk/by-partlabel/Basicx20datax20partition: No such file or directory
tar: error exit delayed from previous errors
udev-cache: update failed!
Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...
INIT: Entering runlevel: 5un-postinsts exists du
Configuring network interfaces... done.
Starting syslogd/klogd: done

QorIQ LSDK (FSL Reference Distro) 2.0 flex-installer /dev/ttyS0

flex-installer login:
QorIQ LSDK (FSL Reference Distro) 2.0 flex-installer /dev/ttyS0

flex-installer login: root
root@flex-installer:~# uname -a
Linux flex-installer 4.4.65 #1 SMP PREEMPT Mon May 29 16:38:31 IST 2017 aarch64 GNU/Linux

```

5.5 Migration guide for new boot flow (with TF-A)

The document describes the changes in the boot flow with introduction of opensource secure firmware, Trusted firmware from ARM (TF-A). TF-A replaces the proprietary secure EL3 firmware of NXP referred to as PPA in the boot flow.

For details about TF-A, refer [Trusted Firmware – A \(TF-A\)](#) on page 64

5.5.1 General boot flow

Few steps are common in both the traditional boot flow (with PPA) and the new boot flow (with TF-A). These steps are as follows:

1. SoC comes out of reset and reads RCW/PBL image from a boot source, such as NOR flash, SD card, or eMMC flash. The RCW/PBL image contains configuration bits that control:
 - Pin muxing and the protocol selected for SerDes pins.
 - Clock parameters and PLL multipliers.
 - Device containing the first software (not in an internal SoC ROM) to run.
2. Code in an internal BootROM starts running and configures low-level aspects of the SoC.
3. The BootROM then loads the first external software to run from a boot device, such as NOR flash or SD/eMMC. The external software which is loaded by the BootROM code varies with the boot flow.

5.5.2 Traditional boot flow with PPA

The steps followed in the traditional boot flow with PPA once the BootROM passes control to the first external software are as follows:

1. In traditional boot flow, the first external software is either U-Boot or UEFI. For XIP (Execute in Place) boot devices, such as NOR flash, the bootloader runs directly from the XIP memory. However, for SD/eMMC, the bootloader is first loaded to OCRAM via ROM/HW.
2. The BootROM transfers control to the bootloader.
3. The bootloader performs additional system configuration including:

- DDR initialization
 - Platform interconnect settings
 - Any other configuration required in EL3/secure world
4. It then loads and starts the PPA image from NOR flash or SD/ eMMC.
 5. PPA is a special resident firmware that runs at the highest ARMv8A privilege level EL3. It provides services to both bootloader and operating system. These services include controlling core power state and bringing additional cores out of reset.
 6. PPA further loads and executes OPTEE (Trusted OS) if present and passes back control to bootloader which now runs in non-secure world.
 7. U-Boot/UEFI is responsible to load and pass control to the kernel.

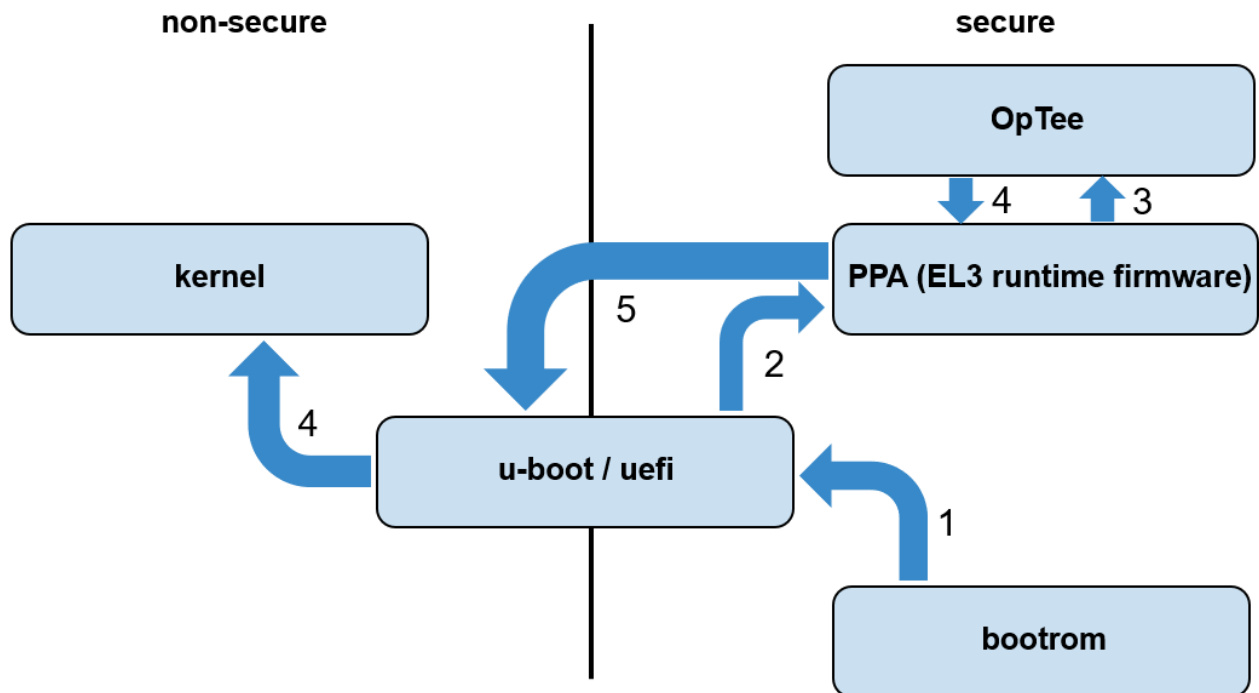


Figure 3. Traditional boot flow with PPA

5.5.3 New boot flow with TF-A

The steps followed in the new boot flow with TF-A once the BootROM passes the control to the first external software are as follows:

1. In the new boot flow, the first external software is a binary called BL2 in ARM terminology. BL2 is always loaded and executed from OCRAM by the ROM/PBI commands irrespective of the boot source.
2. BL2 is a platform initialization software which runs at the EL3 execution level. It does the following:
 - Initializes DDR to allow loading of other bootloader images
 - Configures TZASC
 - Platform Interconnect settings
 - Loads the next set of images -> BL31 (EL3 run time firmware), BL32 (OPTEE if present), and BL33 (Bootloader – U-Boot/UEFI)
3. BL2 transfers control to BL31 which is the EL3 run time firmware. It is responsible for doing the following:

- Any security related settings (TZPC)
 - Providing services to both boot loader and operating system. These services include controlling core power state and bringing additional cores out of reset
4. BL31 is responsible for passing control to OPTEE if present. OPTEE after initialization returns control back to BL31.
 5. BL31 then transfers control to the bootloader (U-Boot/UEFI) in EL2 mode.
 6. U-Boot/UEFI is responsible to load and pass control to kernel.

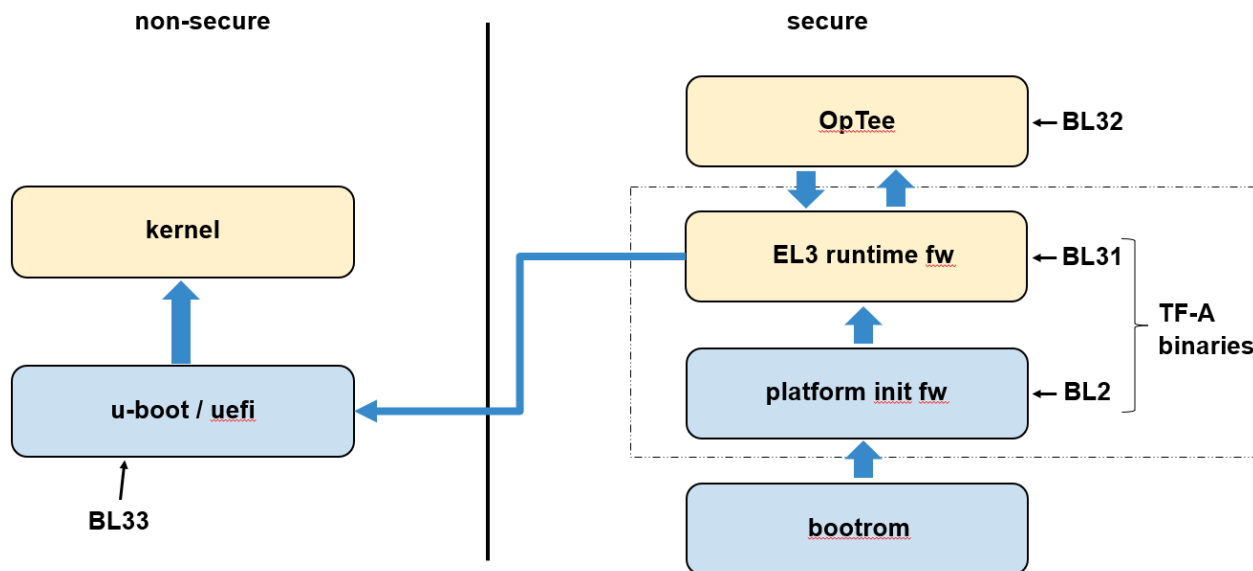


Figure 4. New boot flow with TF-A

Following will not change with new boot flow:

- Kernel will not know or care – no changes required
 - Interface between the non-secure and secure worlds is a defined interface
 - PPA and TF-A implement exactly the same std APIs
 - PPA supports PSCI v0.2 while TF-A supports PSCI v1.1. This is automatically taken care by the PSCI version call already implemented in kernel. No additional changes are required.

Following will change with new boot flow:

- Flash images
- DDR FIP image needs to be used instead of DDR FIT image for LX2.
- Flash layout
- U-Boot

5.5.3.1 New flash images

The following table lists the new flash images in the boot flow with TF-A.

S.No	Binary Name	Components
1.	bl2_<boot_mode>.pbl where boot_mode = flexspi_nor, sd for LX2	RCW + PBL commands + bl2.bin
2.	fip.bin	bl31.bin bl32.bin (Trusted OS image), bl33 (U-Boot image)

NOTE

For steps for compiling and deploying these images, refer the document file: `~/<tf-a repo>/plat/nxp/README.TRUSTED_BOOT` in the TF-A source code.

5.5.3.1.1 bL2_<boot_mode>.pbl

The components of `bL2_<boot_mode>.pbl` are:

- BL2 binary: Platform Initialization binary
- RCW binary of the required boot source

BL2 binary is loaded on OCRAM by the ROM code. Tools in TF-A generate a combined binary containing RCW, BL2 binary, and PBI commands to load the BL2 binary. RCW is provided externally to the TF-A repository.

Steps for generating `bl2_<boot_mode>.pbl`:

1. Compile TF-A to generate `bl2.bin`.
2. Generate `bl2_<boot_mode>.pbl` by executing the tools.
 - a tool in TF-A adds the `bl2` binary to the RCW file as the PBI commands.

The `bl2` binary is generated automatically if you use target option `pbl` and provide `RCW` binary file as input to the TF-A `make` command.

```
make PLAT=<platform> pbl RCW=<rcw bin> BOOT_MODE=<boot_mode>
```

```
where PLAT= ls1012aqds|ls1012ardb|ls1043aqds|ls1043ardb|ls1046aqds|ls1046ardb|ls1088aqds|ls1088ardb|ls2088aqds|ls2088ardb|lx2160ardb
```

```
BOOT_MODE= nor, nand, sd, emmc, qspi, flexspi_nor
```

Files generated are available at:

- `<tf-a repo>/build/<debug/release>/bl2.bin`
- `<tf-a repo>/build/<debug/release>/bl2_<boot_mode>.pbl`

The `*.pbl` needs to be loaded at the location of RCW. For any change in `bl2` source code or RCW, this file needs to be regenerated.

5.5.3.1.2 FIP binary

The components of FIP image are:

- BL31: Secure runtime firmware
BL31 binary is generated by compiling TF-A source code.
- BL32: Trusted OS (for example, OPTEE). This is optional to have in FIP image.

BL32 binary needs to be compiled from OPTEE source repository and its path needs to be provided when compiling TF-A to generate the FIP image.

- BL33: U-Boot/UEFI

BL33 binary is the U-Boot binary and its path needs to be provided when compiling TF-A to generate the FIP image.

Any change in BL31/BL32 or BL33 image requires recompilation of the FIP binary.

Generate bl31.bin

```
make PLAT=<platform> bl31
```

platform - ls1043ardb, ls1046ardb, lx2160ardb

Generate fip.bin

TF-A has an inbuilt tool to generate the FIP binary.

```
make PLAT=<platform> fip BL33=<U-Boot/UEFI bin> SPD=opteed BL32=<tee.bin>
```

File is generated at: <tf-a_repo>/build/<debug/release>/fip.bin

For any change in any of the components of the FIP image, the FIP file needs to be regenerated.

5.5.3.2 Changes in flash layout

The following figure shows the flash layout with PPA.

Definition	Max Size	NOR/QSPI/NAND	SD Card
RCW+PBI	1MB	0x00000000	0x00008
Boot firmware (U-Boot or UEFI)	2MB	0x00100000	0x00800
Boot firmware Environment	1MB	0x00300000	0x01800
PPA firmware	2MB	0x00400000	0x02000
Secure boot headers	3MB	0x00600000	0x03000

Figure 5. Flash layout with traditional boot flow with PPA

The following figure shows the flash layout with TF-A.

Changed Definition	Max Size	NOR/QSPI/NAND	SD Card
RCW+PBI+BL2 - bl2.pbi	1 MB	0x00000000	0x00008
FIP Image (BI31 + BL32(optee.bin) + BI33(uboot/uefi) + headers for secure boot)	4 MB	0x00100000	0x00800
Boot firmware environment	1MB	0x00500000	0x2800
Secure boot headers	3 MB	0x00600000	0x03000

Figure 6. Changed flash layout with TF-A

5.5.3.3 DDR FIP image (only required for LX2160A)

A pre-built FIP image is already provided in the release. To regenerate the fip image, perform the following steps:

1. cd <tfa_repo>/tools/fiptool
2. Make
3. fiptool create
 - --ddr-immem-udimm-1d <ddr4_pmu_train_imem.bin>
 - --ddr-immem-udimm-2d <ddr4_2d_pmu_train_imem.bin>

- **--ddr-dmmem-udimm-1d** <ddr4_pmu_train_dmem.bin>
 - **--ddr-dmmem-udimm-2d** <ddr4_2d_pmu_train_dmem.bin>
 - **--ddr-immem-rdimm-1d** <ddr4_rdimmem_pmu_train_imem.bin>
 - **--ddr-immem-rdimm-2d** <ddr4_rdimmem2d_pmu_train_imem.bin>
 - **--ddr-dmmem-rdimm-1d** <ddr4_rdimmem_pmu_train_dmem.bin>
 - **--ddr-dmmem-rdimm-2d** <ddr4_rdimmem2d_pmu_train_dmem.bin>
- <dd_fip_file_name>

5.5.3.4 Changes in U-Boot

- In the changed boot flow, no DDR initialization is required in U-Boot. DDR initialization is a part of TF-A.

In TF-A, DDR init code can be added to <atf_dir>/plat/nxp/soc-<soc-name>/<soc-name>ardb/ddr_init.c

For example, for LX2160A, the DDR init code should be added to <atf_dir>/plat/nxp/soc-lx2160/lx2160ardb/ddr_init.c

The DDR drivers for various controllers can be found at plat/nxp/drivers/ddr

- Any changes in Interconnect initialization can be added to soc.c file at <tf-a_repo>/plat/nxp/soc-<soc-name>/
- A single defconfig needs to be created that is used for all the boot sources. For example, FlexSPI boot, SD boot.
- The following points need to be taken care of while creating the defconfig:
 - PPA is not required, its support needs to be disabled.
 - Environment support needs to be enabled for all the boot sources, for example FlexSPI, SD boot.
- Other changes:
 - Boot command changes are required to support Flexbuild Linux autoboot. This is similar to changes required for Flexbuild support. Following variables need to be defined:
 - FSPI_NOR_BOOTCOMMAND
 - SD_BOOTCOMMAND
 - MC init command changes are required to provide the MC init command as per boot source:
 - XSPI_MC_INIT_CMD
 - SD_MC_INIT_CMD

Chapter 6

Security

6.1 Secure boot

6.1.1 Hardware Pre-Boot Loader (PBL) based platforms

6.1.1.1 Introduction

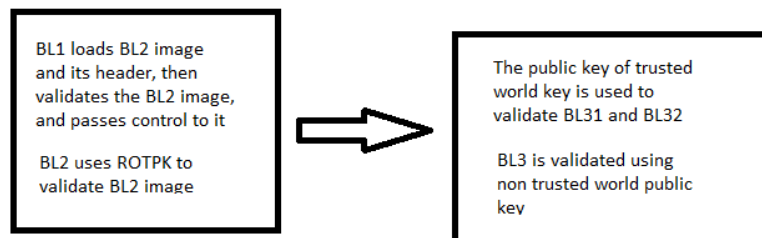
This section is intended for end-users to demonstrate the image validation process. The image validation can be split into stages, where each stage performs a specific function and validates the subsequent stage before passing control to that stage.

Chain Of Trust:

CoT starts from a set of implicitly trusted components.

The following images are included in the CoT:

- BL1
- BL2
- BL31
- BL32
- BL33
- Linux



For more details on the CoT refer `trusted-board-boot.rst` in the TF-A repository

6.1.1.2 Secure boot process

Secure boot process uses a digital signature validation routine already present in Internal BOOT ROM. This routine performs validation using HW bound RSA public key to decrypt the signed hash and compare it to a freshly calculated hash over the same system image. If the comparison passes, the image can be considered as authentic.

The complete process can be broken down into following phases:

- Pre-Boot Phase
 1. PBL
 2. SFP
- ISBC

- ESBC

The complete secure boot process is shown in the figure below.

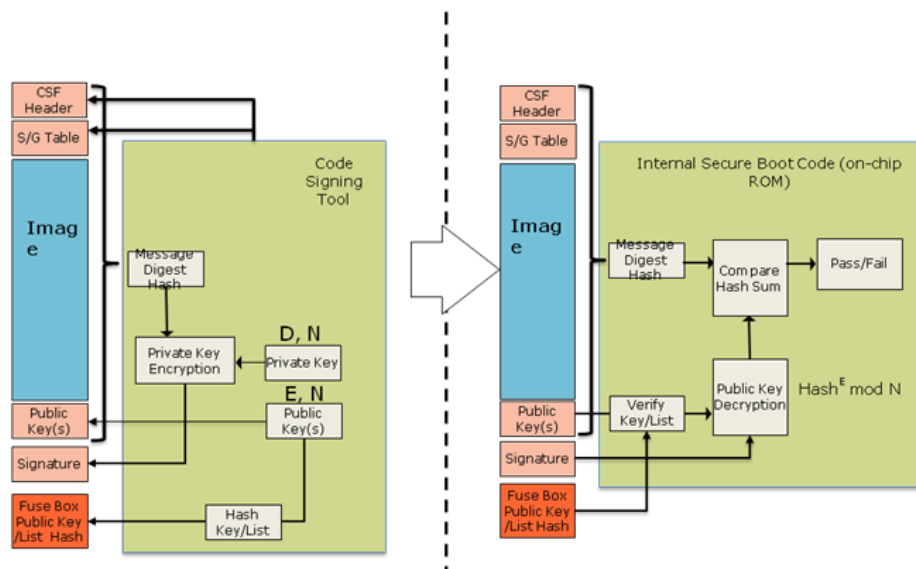


Figure 7. Secure boot process

6.1.1.3 Pre-boot phase

When the processor is powered on, reset control logic blocks all device activities (including scan and debug activity) until fuse values can be accurately sensed. The most important fuse value at this stage of operation is the 'Intent to Secure' (ITS) bit. When an OEM sets ITS, they intend for the system to operate in a secure and trusted manner.

The two main components involved during this process are:

The Security Fuse Processor (SFP) has two roles. The first is to physically burn fuses during device provisioning. The second is to use these provisioned values to enforce security policy in the pre-boot phase, and to securely pass provisioned keys and other secret values to other hardware blocks when the system is in a trusted/secure state.

Pre-Boot Loader (PBL) is the micro-sequencer that can simplify system boot by configuring the DDR memory controllers to more optimal settings and copying code and data from low speed memory into DDR. This allows subsequent phases of boot to operate at higher speed. The setting of ITS determines where the PBL is allowed to read and write. The use of the PBL is mandatory when performing secure boot. At a minimum, the PBL must read a command file from a location determined by the Reset Configuration Word (RCW) and perform a store of a value to the ESBC Pointer Register within the SoC. If the PBL does not perform this operation (or sets the ESBC pointer to the wrong value), the ISBC will fail to validate the ESBC. Once the PBL has completed any operations defined by its command file, the PBL is disabled until the next Power on Reset and the Boot Phase begins.

The ISBC is capable of reading from NOR flash connected to the local bus, on-chip memory configured as SRAM, or main memory. Unless the ESBC is stored in NOR flash, the developer is required to create a PBL Image that copies the image to be validated from NVRAM to main memory or internal SRAM prior to writing the SCRATCHRW1 Register and executing the ISBC code.

To assist with the creation of PBL Images (for both normal and Trust systems), NXP offers a PBL Image Tool.

Note that it is possible for an attacker to modify the board to direct the PBL to the wrong non-volatile memory interface, or change the PBL Image and CSF Header pointer, however this will result in a secure boot failure and the system remaining in an idle loop indefinitely.

6.1.1.4 ISBC phase

6.1.1.4.1 Flow in the ISBC code

With the PBL disabled and all external masters blocked by the PAMUs, CPU 0 is released from boot hold-off and begins executing instructions from a hardwired location within the Internal BOOT ROM. The instructions inside the Internal BOOT ROM are NXP developed code known as the Internal Secure Boot Code (ISBC). The ISBC leads CPU 0 to perform the following actions:

1. **Who am I check?** - CPU 0 reads its Processor ID Register, and if it finds any value besides physical CPU 0, the CPU enters a loop. This insures that only CPU 0 executes the ISBC.
2. **Sec_Mon check** - CPU 0 confirms that the Sec_Mon is in the Check state. If not, it writes a 'fail' bit in a Sec_Mon control register, leading to a state transition.
3. **ESBC pointer read** - CPU 0 reads the ESBC (External Secure Boot Code) Pointer Register, and then reads the word at the indicated address, which is the first word of the Command Sequence File Header which precedes the ESBC itself. If the contents of the word do not match a hard coded preamble value, the ISBC takes this to mean it has not found a valid CSF and cannot proceed. This leads to a fail, as described in #2 above.
4. **CSF parsing and public key check** - If CPU 0 finds a valid CSF header, it parses the CSF header to locate the public key to be used to validate the code. There can be a single public key or a table of 4 public keys present in the header. The Secure Fuse Processor does not actually store a public key, it stores a SHA-256 hash of the public key/table of 4 keys. This is done to allow support for up to 4096b keys without an excessively large fuse block. If the hash of the public key fails to match the stored hash, secure boot fails.
5. **Signature validation** - With the validated public key, CPU 0 decrypts the digital signature stored with the CSF header. The ISBC then uses the ESBC lengths and pointer fields in the CSF header to calculate a hash over the code. The ISBC checks that the CSF header is included in the address range to be hashed. Option flags in the CSF header tell the ISBC whether the NXP Unique ID and the OEM Unique ID (in the Secure Fuse Processor) are included in the hash calculation. Including these IDs allows the image to be bound to a single platform. If the decrypted hash and generated hash do not match, secure boot fails.
6. **ESBC First Instruction Pointer check** - One final check is performed by the ISBC. This check confirms that the First Instruction Pointer in the CSF header falls within the range of the addresses included in the previous hash. If the pointer is valid, the ISBC writes a 'PASS' bit in a Sec_Mon command register, the state machine transitions to 'Trusted', and the OTPMK is made available to the SEC.
7. In case of failure, for Trust v2.0 devices, secondary flag is checked in the CSF header. If set, ISBC reads the CSF header pointer from SCRATCHRW3 location and repeats from step 4.

There are many reasons the ISBC could fail to validate the ESBC. Technicians with debug access can check the SCRATCHRW2 Register to obtain an error code. For a list of error codes, refer ISBC Validation Error Codes.

6.1.1.4.2 Super Root Keys (SRKs) and signing keys

These are RSA public and private key pairs. Private keys are used to sign the images and public keys are used to validate the image during ISBC and ESBC phase.

Public keys are embedded in the header and the hash of SRK table is fused in SRKH register of SFP.

These are Hardware Bound Keys, once the hash is fused the public private key pair cannot be modified.

Keys of sizes 1k, 2k, and 4k are supported in FSL Secure Boot Process.

It is the OEM's responsibility to tightly control access to the RSA private signature key. If this key is ever exposed, attackers will be able to generate alternate images that will pass secure boot.

If this key is ever lost, the OEM will be unable to update the image.

6.1.1.4.3 Key revocation

Trust Architecture 2.x introduces support for revoking the RSA public keys used by the ISBC to verify the ESBC. The RSA public keys used for this purpose are called Super Root Keys (SRK's).

OEM can use either a single key or a list of upto 4 SRK's in the Trust Arch v2.x devices.

In the NXP Code Signing Tool (CST), the OEM defines whether the device uses a single SRK, or offers a list of SRK's. If using a single SRK, a new flag bit in the CSF header will indicate "Key", otherwise the flag will indicate "Key List". Assuming key list, the OEM can populate a list of up to 4 SRK's for trust arch v2.x onwards platforms and can calculate a SHA-256 hash over the list. This hash is written to the SRKH registers in the SFP.

As part of code signing, the OEM defines which key in the key list is to be used for validating the image. This key number is included as a new field in the CSF header.

During secure boot, the ISBC determines whether a key list is in use. If the key list is valid, the ISBC checks the key number indicated in the CSF header against the revocation fuses in the SFP's OEM Security Policy Register (SFP_OSPPR). If the key is revoked, the image validation fails.

NOTE

In order to prevent unauthorized revocation of keys, SFP provides a bit (Write Disable). If the bit is set, the Key revocation bits cannot be written to.

In regular operation, the ESBC (early Trusted S/W) needs to set the SFP Write Disable bit. When circumstances call for revoking a key, the OEM will use an ESBC image with "Write Disable" bit not set. So, the SFP will be in a state in which key revocation fuses can be set.

Logically after revoking the required key(s), the OEM would then load a new signed ESBC image with code to set the "Write Disable" bit, with new CSF header indicating which of the remaining non-revoked key to use.

So, only the possessor of a legitimate RSA private key can enable key revocation.

One possible motivation for an OEM to revoke an SRK is the loss of the associated RSA private key to an attacker. If the attacker has gained access to a legitimate RSA private key, and the attacker can turn on power to the fuse programming circuitry, then the attacker could maliciously revoke keys. To prevent this from being used to permanently disable the system, one SRK does not have an associated revocation fuse.

6.1.1.4.4 Alternate image support

Trust 2.0 onwards will support a primary and alternate image, where failure to find a valid image at the primary location will cause the ISBC to check a configured alternate location.

To execute, the alternate image must be validated using a non-revoked public key as defined by its CSF Header. A valid alternate image has same rights and privileges as a valid primary image.

This feature helps to reduce risk of corrupting single valid image during firmware update or as a result of flash block wear-out.

To enable this feature, create PBI with pointers for both primary and alternate images (HW PBL uses SCRATCHRW1 & SCRATCHRW3).

6.1.1.4.5 ESBC with CSF header

ESBC is the generic name for the code that the ISBC validates. A few ESBC scenarios are described in later sections.

The figure below provides an example of an ESBC with CSF (Command Sequence File) header. The CSF header includes lengths and offset which allow the ISBC to locate the operands used in ESBC image validation, as well as describe the size and location of the ESBC image itself.

Note: CSF header and ESBC header may be used synonymously in this and other NXP Trust Architecture documentation.

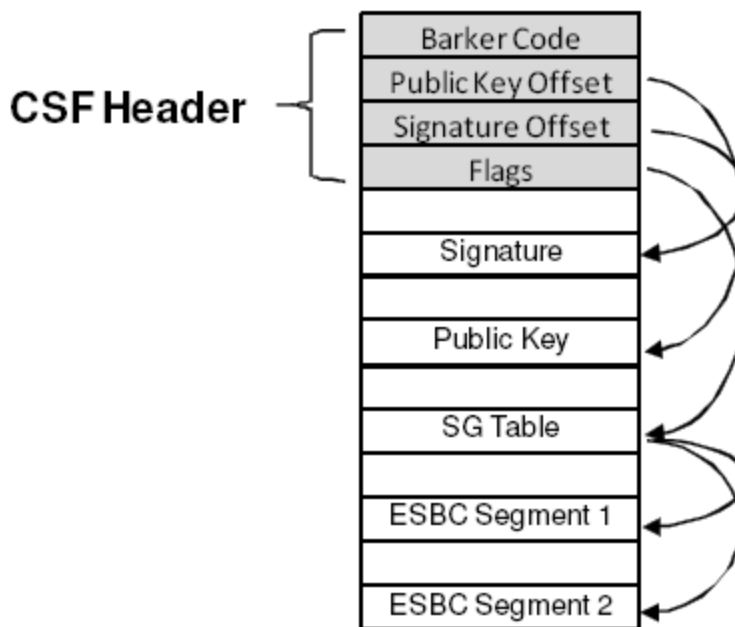


Figure 8. ESBC with CSF header

6.1.1.5 ESBC phase

Unlike ISBC, which is an internal ROM and unchangeable, ESBC is NXP – supplied reference code, and can be changed by OEMs. ESBC is the BL2 image, which is signed using private key. That image then loads a FIP image that includes, BL31 (EL3 runtime software), BL32(optional image for platform storage) and BL33 (Uboot) to DDR and their headers to DDR, then validates these images.

BL33 (U-Boot) which has been signed using a private key. U-Boot reserves a small space for storing environment variables. This space is typically one sector above or below the U-Boot and is stored on persistent storage devices like NOR flash if macro CONFIG_ENV_IS_IN_FLASH is used. In case of secure boot, macro CONFIG_ENV_IS_NOWHERE is used and so, environment is compiled in BL33 (U-Boot) image and is called default environment. This default environment cannot be stored on flash devices. User won't be able to edit this environment also as he cannot reach to U-Boot prompt in case of secure boot. There is default boot command for secure boot in this default environment which executes on autoboot.

ESBC validates a file called boot script and on successful validation, execute the commands in the boot script.

There are many reasons ESBC could fail to validate Client images or boot script. The error status message along with the code is printed on the U-Boot console. For a list of error codes, refer ESBC Validation Error Codes.

Users are free to use NXP ESBC as it is provided or to use it as reference to modify their own secure boot system.

NOTE

On Soc's with ARMv8 core (For example, LS1043, LS1046, and LS1012), during ISBC phase in internal BOOT ROM, SMMU (which by default is in by-pass mode) is configured to allow only secure transactions from CAAM.

The security policy with respect to the SMMU in ESBC phase must be decided by the user/customer. So, currently in ESBC (U-Boot), SMMU is configured back to by-pass mode allowing all transactions (secure as well as non-secure).

6.1.1.5.1 Boot script

Boot script is a U-Boot script image which contains U-Boot commands. ESBC would validate this boot script before executing commands in it.

NOTE

1. Boot script can have any commands which U-Boot supports. No checking on the allowed commands in boot script. Since it is validated image, assumption is that commands in boot script would be correct.
2. If some basic scripting error done in boot script (like unknown command, missing arguments), the required usage of that command and core is put in infinite loop.
3. After execution of commands in boot script, if control reaches back in U-Boot, error message would be printed on U-Boot console and core would be put in spin loop by command `esbc_halt`.
4. Scatter gather images are not supported with validate command.
5. If ITS fuse is blown, any error in verification of the image would result in system reset. The error would be printed on console before system goes for a reset.

6.1.1.5.1.1 Where to place the boot script?

NXP's ESBC U-Boot expects the boot script to be loaded in flash as specified in address map. ESBC U-Boot code assumes that the public/private key pair used to sign the boot script is same as that was used while signing the U-Boot image. If user used different key pair to sign the image, hash of the N and E component of the key pair should be defined in macro:

CONFIG_BOOTSCRIPT_KEY_HASH.

Note: The hash defined should be hex value, 256 bits long.

Both the above macros can be defined or changed in the configuration file `secure_boot.h` at the following location in U-Boot code:

```
u-boot/arch/arm/include/asm/fsl_secure_boot.h
```

Two new commands called `esbc_validate` and `esbc_halt` have been added in NXP ESBC U-Boot.

Two more commands are present, 'blob enc' and 'blob dec' for running Chain of Trust with confidentiality.

6.1.1.5.1.2 Chain of Trust

Boot script contains information about the next level of images, For example, Linux, HV, and so on. ESBC validates these images as per their public keys and then executes `bootm` command to pass-on the control to next image.

Users are free to use NXP ESBC as it is provided or to use it as reference to modify their own secure boot system.

The following figure shows the Chain of Trust established for validation with this ESBC U-Boot.

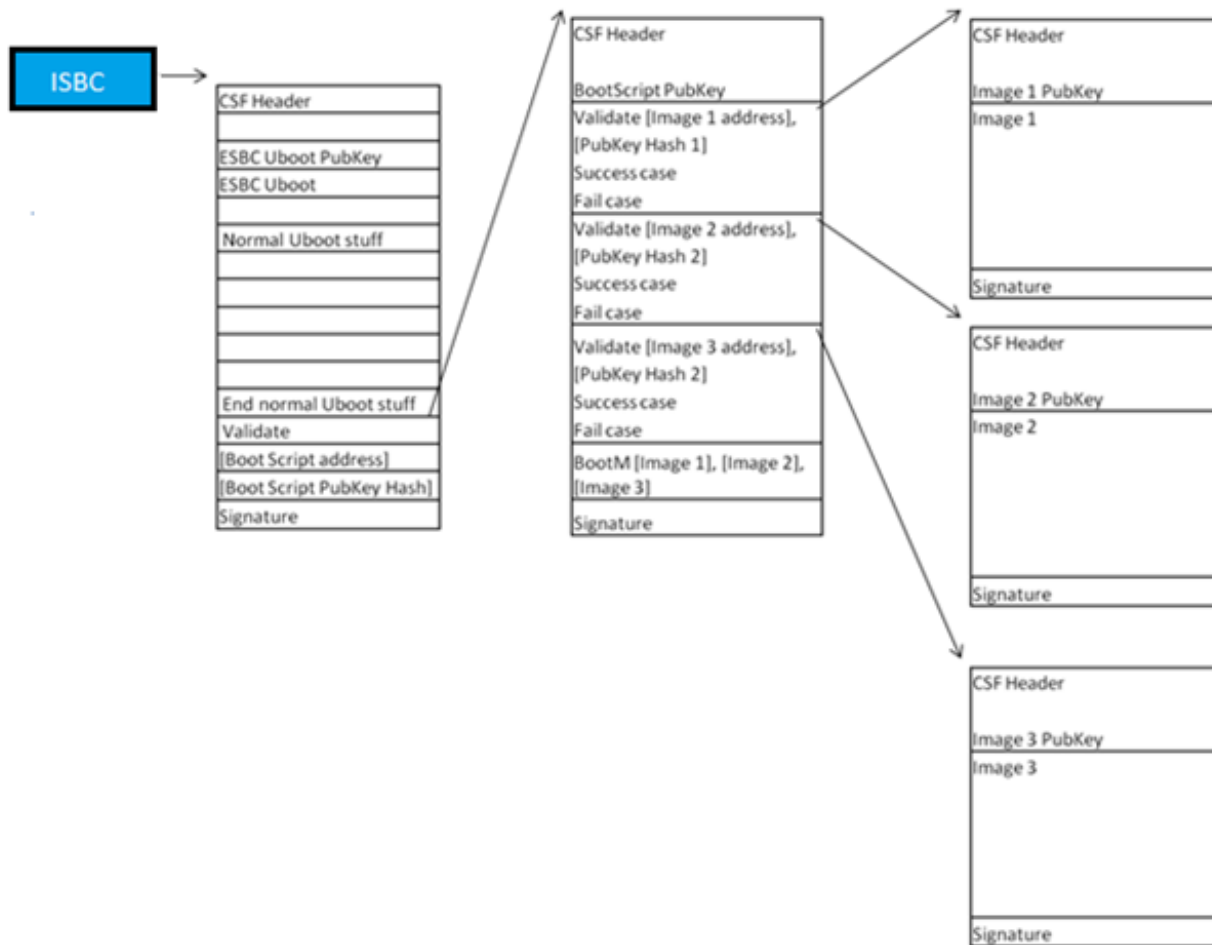


Figure 9. Secure boot flow (Chain of Trust)

6.1.1.5.1.2.1 Sample boot script

A sample boot script would look like:

```

...
esbc_validate <Img1 header addr> <pub_key hash>
esbc_validate <Img2 header addr> <pub_key hash>
esbc_validate <Img3 header addr> <pub_key hash>
...
bootm <img1 addr> <img2 addr> <img3 addr>

```

6.1.1.5.1.2.1.1 esbc_validate command

esbc_validate img_hdr [pub_key_hash]

Input arguments:

img_hdr - Location of CSF header of the image to be validated

pub_key_hash - hash of the public key used to verify the image. This is an optional parameter. If not provided, code makes the assumption that the key pair used to sign the image is same as that used with ISBC. So the hash of the key in the header is checked against the hash available in SRK fuse for verification.

Description:

The command would do the following:

- Perform CSF header validation on the address passed in the image header. During parsing of the header, image address is stored in an environment variable which is later used in source command in default secure boot command.
- Signature checks on the image

6.1.1.5.1.2.1.2 esbc_halt command

esbc_halt (no arguments)

Description:

The command would do the following:

This command puts core in spin loop.

After successful validation of images, bootm command in bootscript should execute and control should never reach back to U-Boot. If somehow, control reaches back to U-Boot (for example, bootm not present in bootscript), core should just spin.

6.1.1.5.1.3 Chain of Trust with confidentiality

To establish Chain of Trust with confidentiality, cryptographic blob mechanism can be used. In this Chain of Trust, validated image is allowed to use the One Time Programmable Master Key to decrypt system secrets.

Two bootscripts are to be used. First encap bootscript is used which creates a blob of the Linux images and saves them. After that, the system is booted after replacing the encap bootscript with decap bootscript which decapsulates the blobs and boot the Linux with the images.

The following figures show the Chain of Trust with confidentiality (Encapsulation and Decapsulation).

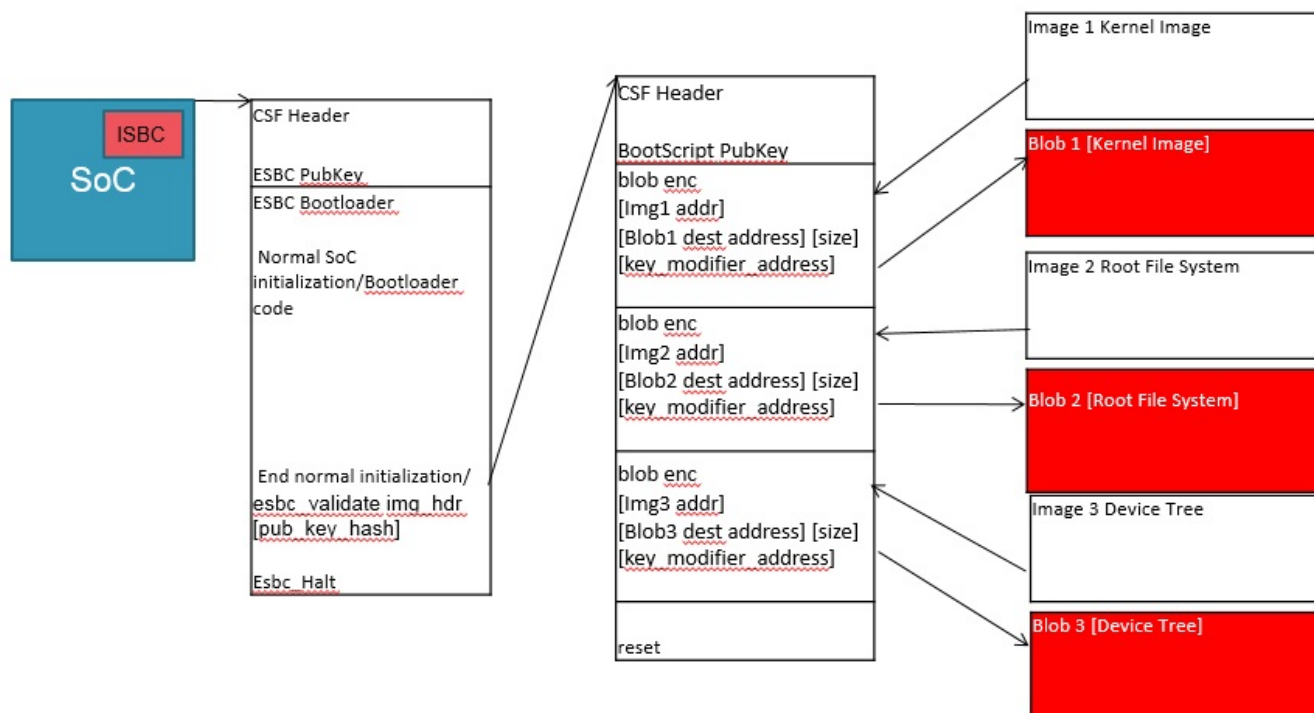


Figure 10. Chain of Trust with confidentiality (Encapsulation)

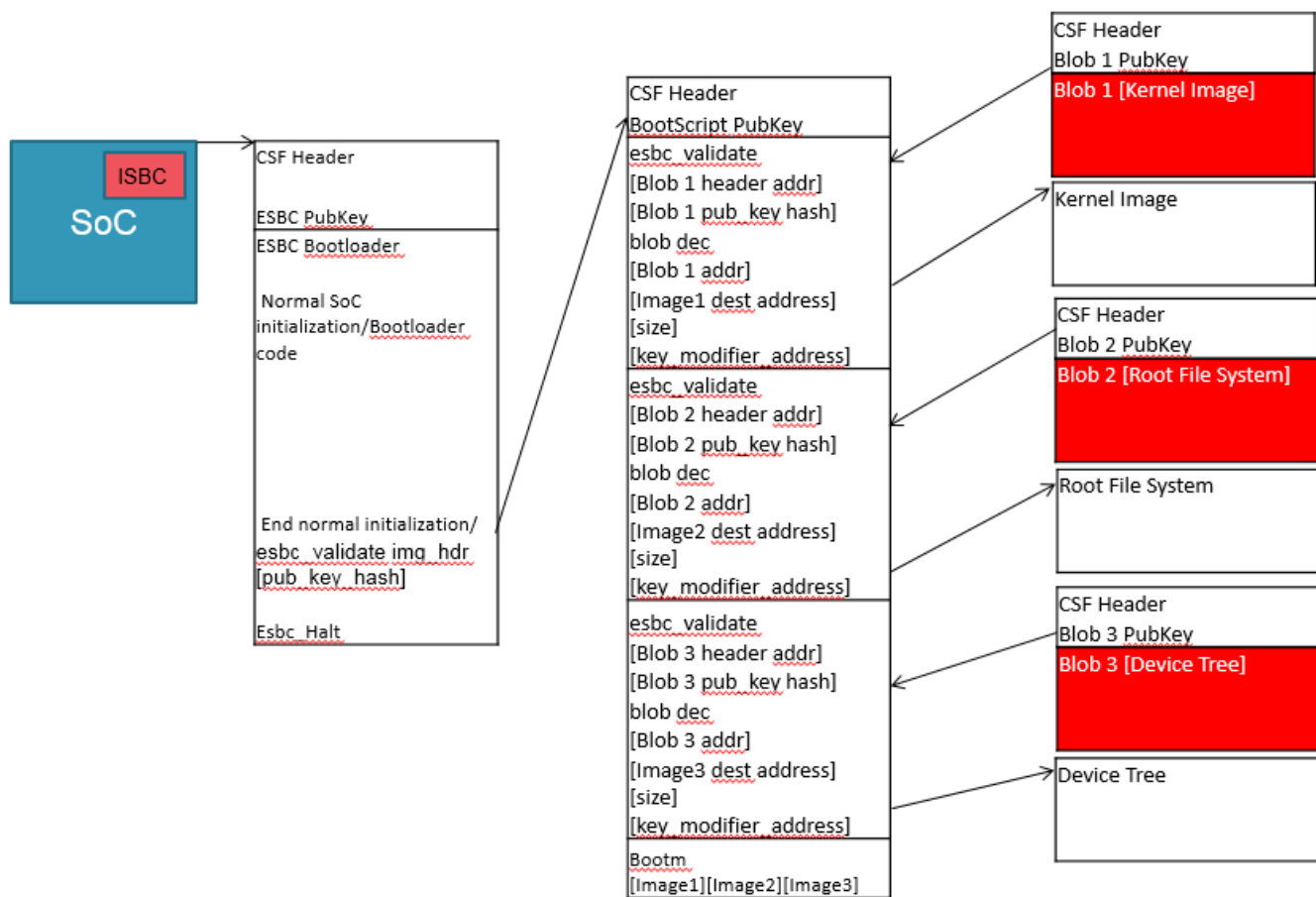


Figure 11. Chain of Trust with Confidentiality (Decapsulation)

6.1.15.1.3.1 blob enc command

blob enc <src location> <dst location> <length> <key_modifier address>

Input arguments:

src location - Address of the image to be encapsulated

dst location - Address where the blob will be created

length - Size of the image to be encapsulated

key_modifier address - Address where a random number 16 bytes long(key modifier) is placed

Description:

The command would do the following:

- Create a cryptographic blob of the image placed at src location and place the blob at dst location.

6.1.15.1.3.1.1 Sample encap boot script

A sample encap boot script would look like:

```
...
blob enc <Img1 addr> <Img1 dest addr> <Img1 size> <key_modifier address>
erase <encap Img1 addr> +<encap Img1 size>
cp.b <Img1 dest addr> <encap Img1 addr> <encap Img1 size>
```

```

blob enc <Img2 addr> <Img2 dest addr> <Img2 size> <key_modifier address>
erase <encap Img2 addr> +<encap Imag2 size>
cp.b <Img2 dest addr> <encap Img2 addr> <encap Imag2 size>

blob enc <Img3 addr> <Img3 dest addr> <Img3 size> <key_modifier address>
erase <encap Img3 addr> +<encap Imag3 size>
cp.b <Img3 dest addr> <encap Img3 addr> <encap Imag3 size>

...

```

6.1.15.13.2 blob dec command

blob dec <src location> <dst location> <length> <key_modifier address>

Input arguments:

src location - Address of the image blob to be decapsulated

dst location - Address where the decapsulated image will be placed

length - Expected Size of the image after decapsulation.

key_modifier address - Address where key modifier (Same as that used for Encapsulation) is placed

Description:

The command would do the following:

- Decapsulate the blob placed at src location and place the decapsulated data of expected size at dst location.

6.1.15.13.2.1 Sample Decap Boot Script

A sample decap boot script would look like:

```

...
blob dec <Img1 blob addr> <Img1 dest addr> <expected Img1 size> <key_modifier address>
blob dec <Img2 blob addr> <Img2 dest addr> <expected Img2 size> <key_modifier address>
blob dec <Img3 blob addr> <Img3 dest addr> <expected Img3 size> <key_modifier address>
...
bootm <Img1 dest addr> <Img2 dest addr> <Img3 dest addr>

```

6.1.16 Next executable (Linux phase)

The bootloader finishes the platform initialization and passes control to the Linux image. The boot-chain can be further extended to be able to sign application which would be running on Linux prompt. Further, integrate RTIC to verify memory regions using Security Engine (SEC) during run time.

Chain of Trust

To execute Chain of Trust, follow the steps below:

Instruction on demo:

1. Make sure that the ISBC code validate the BL2 code.
2. BL2 loads FIP (BL31 (Secure Firmware) + BL32 (Optional) + BL33 (Uboot)) and validate them.
3. On successful validation, BL31 and BL32 passes for necessary configurations.
4. After configuration, U-Boot code runs and validates the boot script.
5. On successful validation of boot script, U-Boot code executes the commands.
6. The Boot script also contains commands to validate next level images, such as rootfs, Linux ulmage, and device tree.
7. After boot script validating all the images, U-Boot executes the `bootm` command to pass control to Linux.

*Rest of the content in the topic remains the same.

Chain Of trust with confidentiality

Step 1: Creating blobs

1. Make sure that the ISBC code validate the BL2 code.
2. BL2 loads FIP (BL31 (Secure Firmware) + BL32 (Optional) + BL33 (Uboot)) and validate them.
3. On successful validation, BL31 and BL32 passes for necessary configurations.
4. After configuration, U-Boot code runs and validates the boot script.
5. On successful validation of boot script, U-Boot code executes the commands.
6. The boot script contains commands that encapsulates next level images, such as linux ulmage and device tree.

Step 2: Decrypting blob and booting

1. Make sure that the ISBC code validate the BL2 code.
2. BL2 loads FIP (BL31 (Secure Firmware) + BL32 (Optional) + BL33 (Uboot)) and validate them.
3. On successful validation, BL31 and BL32 passes for necessary configurations.
4. After configuration, U-Boot code runs and validates the boot script.
5. On successful validation of boot script, U-Boot code executes the commands.
6. The boot script contains commands that decapsulate or decrypt next level images, such as rootfs, linux ulmage and device tree.
7. After decryption, U-Boot code executes the `bootm` command in boot script to pass control to Linux.

*Rest of the content in the topic remains the same.

Running Secure Boot (Chain Of Trust)

Change wherever ESBC Uboot is used

6.1.1.7 Product execution

This section presents the steps need to be followed in order to properly run the software product according to its intended use and functionalities.

6.1.1.7.1 Introduction

Chain of Trust

This section presents the steps need to be followed in order to execute Chain of Trust.

Steps in the demo would be:

1. ISBC code would validate the BL2 image code.
2. On successful validation, BL2 code would run, which would then validate the BL31, BL32, BL33 images.
3. On successful validation of boot script by BL33 image, commands in boot script would be executed.
4. Boot script contains commands to validate next level images, that is, rootfs, Linux ulmage, and device tree.
5. Once all the images are validated, bootm command in boot script would be executed which would pass control to Linux.

Running Secure boot (Chain of Trust)

1. Setup the board for secure boot flow. You can choose any if the flows mentioned below.

a. Flow A

Program the ITS fuse. Use RCW with SB_EN=0

Or

b. Flow B

For prototyping phase, don't blow the ITS fuse, but use rcw with SB_EN = 1.

Blow other required fuses on the board. (OTPMK and SRK hash^[1]) For more details regarding fuse blowing, CCS and Boot Hold Off, refer to Platform reference manual and Trust Architecture User Guide.

NOTE

SRK hash in the fuse should be same as the hash of the key pair being used to sign the ESBC u-boot.

For testing purpose, the SRK Hash can be written in the mirror registers.

gen_otpmk_drbg utility in cst can be used to generate otpmk key.

2. Flash all the generated images at locations as described in the address map.
 - a. **Flow A** - All the images would have to be flashed at the current bank addresses. Once ITS fuse is blown, the control would automatically shift to ISBC on power on.
 - b. If you are using **Flow B**, you can use alternate bank for demo purpose. This would mean flashing the images on alternate bank addresses from Bank0 and then switching to Bank4.
3. Give a power on cycle to the board.
 - a. For **Flow A** and **Flow B** (*Secure boot Images flashed on default Bank*)
 - On power on, ISBC code would get control, validate the ESBC image.
 - ESBC image would further validate the signed linux, rootfs and dtb images
 - Linux would come up
 - b. **Flow B** (*Secure boot Images flashed on alternate Bank*)
 - On power on cycle, u-boot prompt on bank 0 would come up.
 - On switching to alternate bank, the secure boot flow as mentioned above would execute.

Two additional features are provided in secure boot:

1. Chain of Trust with confidentiality
2. ISBC Key Extension

6.1.1.7.2 Chain of Trust with confidentiality

This section presents the steps need to be followed to execute Chain of Trust with confidentiality.

The demo is divided into two parts:

1. Creating or encrypting images in form of blobs.
2. Decrypting images, and booting from decrypted images.

Steps in the demo are:

Step 1: Creating blobs

1. ISBC code would validate the ESBC code.
2. On successful validation, ESBC code would run, which would then validate the boot script.

[1] Blowing of **OTPMK** is essential to run secure boot for both Production (Flow A) and Prototyping/Development (Flow B).

For **SRK Hash**, in Development Mode (Flow B), there is a workaround to avoid blowing fuses. For this use RCW with BOOT_HO = 1. This will put the core in Boot Hold off stage. Then a CCS can be connected via JTAG.

Write the SRK Hash value in SFP mirror registers and then release the core out of Boot Hold off by writing to Core Release Register in DCFG.

3. On successful validation of boot script, commands in boot script would be executed.
4. The boot script contains commands to encapsulate next level images, that is rootfs, linux ulmage and device tree.

blob encapsulation command::

blob enc src dst len km - Encapsulate and create blob of data

\$len - Number of bytes to be encapsulated.

\$src - The address where image to be encapsulated is present.

\$dst - The address where encapsulated image is stored.

\$km - The address where the key modifier is stored. The modifier is required and used as key for cryptographic operation. Key modifier should be 16 bytes long.

Step 2: Decrypting blob and booting

1. ISBC code would validate the ESBC code.
2. On successful validation, ESBC code would run, which would then validate the boot script.
3. On successful validation of boot script, commands in boot script would be executed.
4. The boot script contains commands to decapsulate or decrypt next level images, that is rootfs, linux ulmage, and device tree.
5. After decryption, bootm command would be executed in boot script to pass control to Linux.

blob decapsulation command::

blob dec src dst len km - Decapsulate the image and recover the data

\$len - Number of bytes to be decapsulated.

\$src - The address where encapsulated image is present.

\$dst - The address where decapsulated image will be stored.

\$km - The address where the key modifier is stored. The modifier is required and used as key for cryptographic operation. Key modifier should be 16 bytes long. It should be same as passed while encapsulating the image.

6.1.1.7.2.1 Other images required for the demo

Apart from SDK images described above, the following images are also required:

1. Encap boot script

Sample Boot script

```
load \${devtype} \${devnum:2} \${kernelheader_addr_r} /secboot_hdrs/ls1046aradb/hdr_linux.out;
esbc_validate \${kernelheader_addr_r};
load \${devtype} \${devnum:2} \${fdtheader_addr_r} /secboot_hdrs/ls1046aradb/hdr_dtb.out; esbc_validate
\${fdtheader_addr_r};
size \${devtype} \${devnum:2} /vmlinuz; echo Encapsulating linux image;setenv key_addr 0x87000000;
mw \${key_addr} \${key_id_1};
setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_2};setexpr \${key_addr} \${key_addr} + 0x4;
mw \${key_addr} \${key_id_3};setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_4};
blob enc \${kernelheader_addr_r} \${load_addr} \${filesize} \${key_addr}; setexpr blobsize \${filesize}
+ 0x30;echo Saving encrypted linux ;save \${devtype} \${devnum:2} \${load_addr} /vmlinuz \
\${blobsize};size \${devtype} \${devnum:2} /fsl-ls1046a-rdb-sdk.dtb;
echo Encapsulating dtb image; blob enc \${fdtheader_addr_r} \${load_addr} \${filesize} \${key_addr}; setexpr
blobsize \${filesize} + 0x30;echo Saving encrypted dtb; save \${devtype} \${devnum:2} \${load_addr} /fsl-
ls1046a-rdb-sdk.dtb \${blobsize}; size \${devtype} \${devnum:2} /ls1046aradb_dec_boot.scr;
load \${devtype} \${devnum:2} \${load_addr} /ls1046aradb_dec_boot.scr;
echo replacing Bootscript; save \${devtype} \${devnum:2} \${load_addr} /ls1046aradb_boot.scr \
\${filesize};size \${devtype} \${devnum:2} /secboot_hdrs/ls1046aradb/hdr_ls1046aradb_bs_dec.out;
load \${devtype} \${devnum:2} \${load_addr} /secboot_hdrs/ls1046aradb/hdr_ls1046aradb_bs_dec.out ;echo
```



```
Replacing bootscript header; save \${devtype} \${devnum}:2 \${load_addr} /hdr_ls1046ardb_bs.out \
${filesize};reset;'
```

2. Decap boot script

```
size \${devtype} \${devnum}:2 /vmlinuz;setexpr imgsize \${filesize} - 0x30 ;
echo Decapsulating linux image; setenv key_addr 0x87000000; mw \${key_addr} \${key_id_1};setexpr \
\${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_2};setexpr \${key_addr} \${key_addr} + 0x4; mw \
\${key_addr} \${key_id_3};setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_4};
blob dec \${kernel_addr_r} \${load_addr} \${imgsize} \${key_addr}; cp.b \${load_addr} \${kernel_addr_r} \
\${filesize} ;size \${devtype} \${devnum}:2 /fsl-ls1046a-rdb-sdk.dtb;setexpr imgsize \${filesize} - 0x30 ;
echo Decapsulating dtb image; blob dec \${fdt_addr_r} \${load_addr} \${imgsize} \${key_addr}; cp.b \
\${load_addr} \${fdt_addr_r} \${filesize} ;
```

6.1.1.7.2.2 Running secure boot (Chain of Trust with confidentiality)

1. Setup the board for secure boot flow. You can choose any of the flows mentioned below.

a. Flow A

Program the ITS fuse. Use RCW with SB_EN=0

Or

b. Flow B

For prototyping phase, do not blow the ITS fuse, instead use rcw with SB_EN = 1.

2. Blow other required fuses on the board. (OTPMK and SRK hash)^[2] For more details regarding fuse blowing, CCS and Boot Hold Off, refer to Platform Reference Manual and Trust Architecture User Guide.

NOTE

SRK hash in the fuse should be same as the hash of the key pair being used to sign the ESBC U-Boot.

For testing purpose, the SRK hash can be written in the mirror registers.

gen_otpmk_drbg utility in cst can be used to generate otpmk key.

3. Flash all the generated images at locations as described in the address map.

- a. **Flow A** - All the images would have to be flashed at the current bank addresses. Once ITS fuse is blown, the control would automatically shift to ISBC on power on.
- b. **Flow B** - You can use alternate bank for demo purpose. This would mean flashing the images on alternate bank addresses from Bank0 and then switching to Bank4.

4. Give a power on cycle to the board.

- a. For **Flow A** and **Flow B** (*Secure boot images flashed on default bank*)

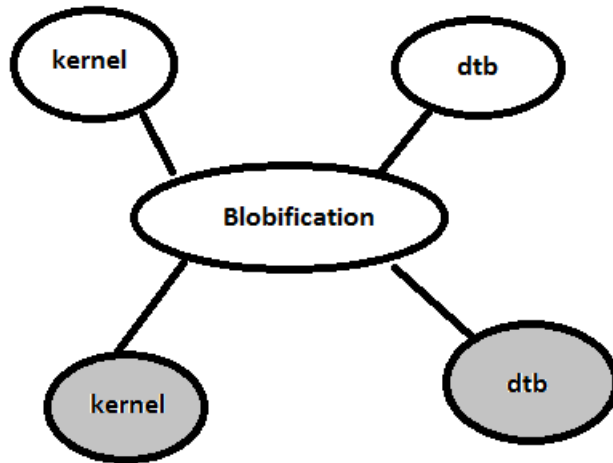
- On power on, ISBC code would get control, validate the ESBC image.
- **First Boot: Encapsulaton Step (Should happen in OEM's premises)**
 - i. By default the enacap and decap bootscripts will be installed in the bootpartition.

[2] Blowing of **OTPMK** is essential to run secure boot for both Production (Flow A) and Prototyping/Development (Flow B).

For **SRK Hash**, in Development Mode (Flow B), there is a workaround to avoid blowing fuses. For this use RCW with BOOT_HO = 1. This will put the core in Boot Hold off stage. Then a CCS can be connected via JTAG.

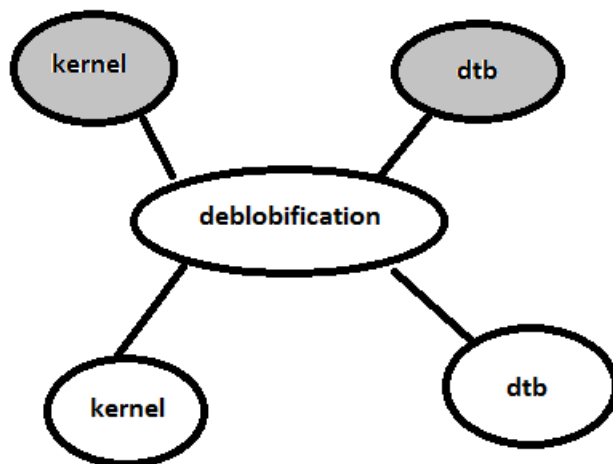
Write the SRK Hash value in SFP mirror registers and then release the core out of Boot Hold off by writing to Core Release Register in DCFG.

- ii. When the board boots up for the first time after all images have been generated, Encap bootscript will execute. This bootscript:
 - i. Authenticates and encapsulates linux and dtb images and replaces the unencrypted linux and dtb images with newly encapsulated linux and dtb.
 - ii. Replaces the encap bootscript and header with the decap bootscript and it's header, already present in the bootpartition.
 - iii. Issues reset



• **Subsequent Boot .**

- i. Uboot would execute script with decap commands
 - i. Un-blobify linux and dtb image in DDR
 - ii. Pass control to these images



b. **Flow B** (Secure boot images flashed on alternate bank)

- On power on cycle, U-Boot prompt on bank0 would come up.
- On switching to alternate bank, the secure boot flow as mentioned above would execute.

6.1.1.8 Troubleshooting

Table 10. Troubleshooting

	Symptoms	Reasons and/or Recommended actions
1.	No print on UART console.	• Check the status register of sec mon block (location 0xfe314014). Refer to the details of the register from the Reference Manual. Bits OTPMK_ZERO, OTPMK_SYNDROME and PE should be 0 otherwise there is some error in the OTPMK fuse blown by you. • If OTPMK fuse is correct (see Step 1), check the SCRATCHRW2 register for errors. Refer to Section for error codes. • If Error code = 0 then check the Security Monitor state in HPSR register of Sec Mon. Sec Mon in Check State (0x9) If ITS fuse = 1, then it means ISBC code has reset the board. This may be due to the following reasons: Hash of the public key used to sign the ESBC U-BOOT does not match with the value in SRK hash fuse Or Signature verification of the image failed Sec Mon in Trusted State (0xd) or Non-Secure State (0xb) Check the entry point field in the ESBC header. It should be 0xcffffffc for the demo described in Section 4. If entry point is correct, ensure that U-BOOT image has been compiled with the required secure boot configuration.
2.	Instead of linux prompt, you get a U-BOOT command prompt instead of linux prompt.	You have not booted in secure boot mode. You never get a U-BOOT prompt in secure boot flow. You would reach this stage if ITS = 0 and you are using rcw where sben0 is present in its name.
3	U-BOOT hangs or board resets	Some validation failure occurred in ESBC U-BOOT. Error code and description would be printed on U-BOOT console.

6.1.1.9 CSF Header Data Structure

The CSF Header provides the ISBC with most of the information needed to validate the image.

LS1 Platform

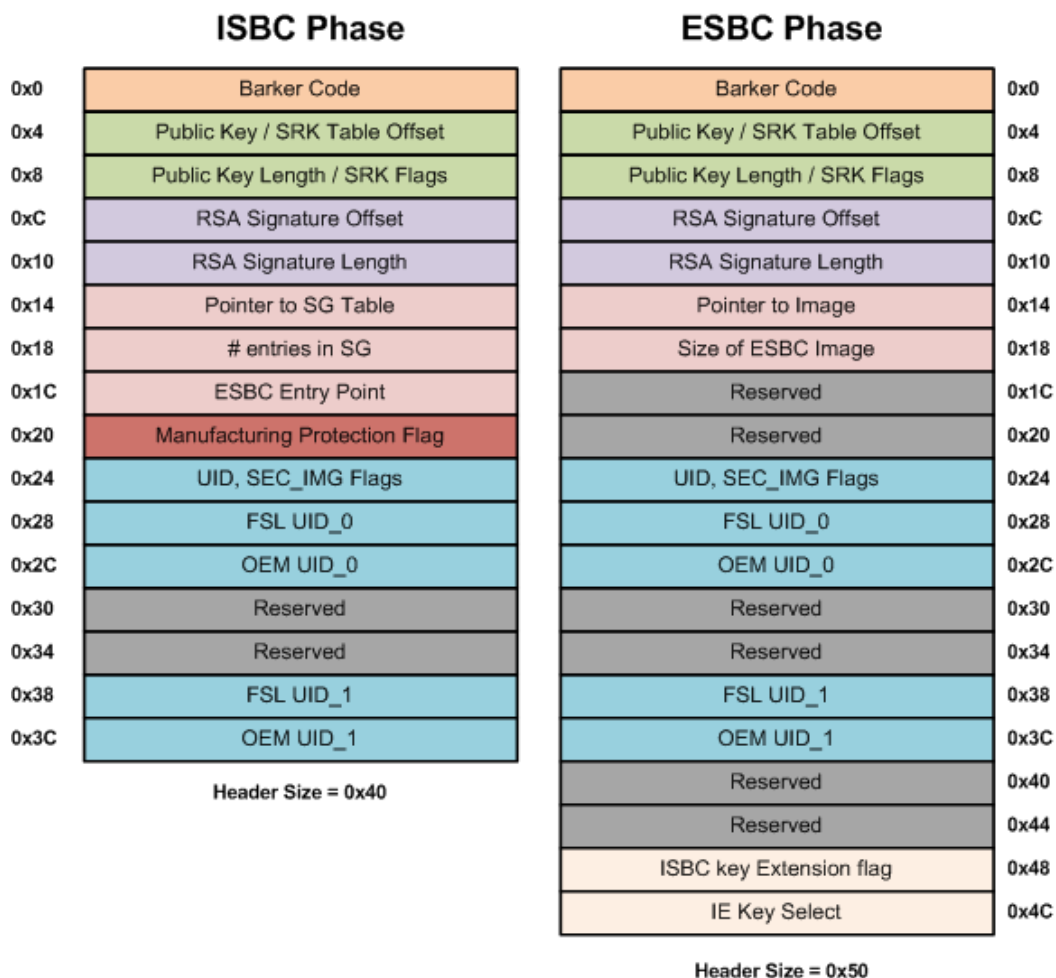


Figure 12. CSF Header for LS1 (ISBC and ESBC Phase)

Table 11. CSF Header Format (LS1 Platform)

Offset	Data Bits [0:31]
0x00-0x03	Barker code. This location should contain the value: 0x68392781. The ISBC code searches for this Barker code. If the value in this location does not match the Barker code, the ISBC stops execution and reports error.
0x07-0x04	If the srk_table_flag is not set : • Public key offset: This location contains an address which is the offset of the public key from the start of CSF header. Using this offset and the public key length, the public key is read. If srk_table_flag is set: • Srk table offset: This location contains an address which is the offset of the srk table from the start of CSF header. Using this offset and the number of entries is SRK Table, the SRK table is read.

Table continues on the next page...

Table 11. CSF Header Format (LS1 Platform) (continued)

Offset	Data Bits [0:31]
0x08	Srk table flag. This flag indicates whether hash burnt in srk fuse is of a single key or of srk table.
0x0b-0x09	If the srk_table_flag is not set : • 0x0b-0x09 -- Public key length: This location contains the length of the public key in bytes. If srk_table_flag is set: • 0x09 – Key Number from srk table which is to be used for verification. • 0x0b-0x0a – Number of entries in srk table. Minimum number of entries in table = 1, Maximum = 4.
0x0f-0x0c	RSA Signature offset. This location contains an offset(in bytes) of the RSA signature from the start of CSF header. Using this offset and the Signature length, the RSA signature is read. The RSA signature is calculated over CSF Header, Scatter Gather table and ESBC images.
0x13-0x10	RSA Signature length in bytes.
0x17-0x14	For ISBC Phase: SG Table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG Table, the SG table is read. For ESBC Phase: Address of the image to be validated.
0x1b-0x18	For ISBC Phase: Number of entries in SG Table (Earlier ,Based on the Scatter gather flag in CSF Header, this location can either be treated as number of entries in SG table or ESBC image size in bytes.). For ESBC Phase Size of image to be validated
0x1f-0x1c	For ISBC Phase: ESBC entry point. ISBC transfers control to this location upon successful validation of ESBC image(s). For ESBC Phase: Reserved
0x21-0x20	Manufacturing Protection Flag Indicates if manufacturing protection has to be enabled or not in ISBC.
0x23-0x22	Reserved .(Earlier this field was SG Flag. SG flag is always assumed to be 1 in unified implementation.)
Table continues on the next page...	

Table 11. CSF Header Format (LS1 Platform) (continued)

Offset	Data Bits [0:31]
0x24	For ISBC Phase: Reserved For ESBC Phase: Reserved
0x25	For ISBC Phase Secondary Image flag Indicates if user has a secondary image available in case of failures in validating primary image. Valid in case of primary Images's Header. For ESBC Phase: Reserved
0x27-0x26	Unique ID Usage This location contains a flag which specifies one of these possibilities • 0x00 - No UID's present • 0x01 - FSL UID and OEM UID are present • 0x02 - Only FSL UID is present • 0x04 - Only OEM UID is present
0x2b-0x28	NXP unique ID 0 Upper 32 bits of a unique 64 bit value, which is specific to NXP. This value is compared with the FSL ID 1 in Secure Fuse Processor 's FSL-ID registers
0x2f-0x2c	OEM unique ID 0 Upper 32 bits of a unique 64 bit value, which is specific to OEM. This value is compared with the OEM ID 0 in Secure Fuse Processor 's OEM-ID registers
0x37-0x30	Reserved
0x3b-0x38	NXP unique ID 1 Lower 32 bits of a unique 64 bit value, which is specific to NXP. This value is compared with the FSL ID 1 in Secure Fuse Processor 's FSL-ID registers
0x3f-0x3c	OEM unique ID 1 Lower 32 bits of a unique 32 bit value, which is specific to OEM. This value is compared with the OEM ID 1 in Secure Fuse Processor 's OEM-ID registers
0x40-0x47	For ISBC Phase: Not Applicable For ESBC Phase: Reserved
0x48-0x4b	For ISBC Phase: Not Applicable For ESBC Phase: ISBC key Extension flag If this flag is set, key to be used for validation needs to be picked up from IE Key table.
Table continues on the next page...	

Table 11. CSF Header Format (LS1 Platform) (continued)

Offset	Data Bits [0:31]
0x4c-0x4f	For ISBC Phase: Not Applicable For ESBC Phase: IE Key Select Key Number to be used from the IE Key Table if IE flag is set.

Table 12. Scatter Gather Table Format (LS1 Platform)

Offset	Data Bits [0:31]
0x00-0x03	Length. This location specifies the length in bytes of the ESBC image 1.
0x04-0x07	Target where the ESBC Image 1 can be found. This field is ignored in case of PBL based SOC's.
0x08-0x0b	Source Address of ESBC Image 1
0x0c-0x0f	Destination Address of ESBC Image 1 If the target address is 0xffffffff, the image is not copied to the target. This field is ignored in case of PBL based SOC's.
0x10-0x13	Length. This location specifies the length in bytes of the ESBC image 2.
0x14-0x17	Target where the ESBC Image 2 can be found. This field is ignored in case of PBL based SOC's.
0x18-0x1b	Source Address of ESBC Image 2
0x1c-0x1f	Destination Address of ESBC Image 2 If the target address is 0xffffffff, the image is not copied to the target. This field is ignored in case of PBL based SOC's.

Table 13. Signature (LS1 Platform)

Offset	Data Bits [0:31]
0x00-size	The RSA signature calculated over CSF Header, Scatter Gather table and ESBC image(s).

Table 14. Public key (LS1 Platform)

Offset	Data Bits [0:31]
0x00-size	Public Key Value. The hash of this public key is compared with the hash stored in Secure Fuse Processor SRKH registers.

Table 15. SRK Table (LS1 Platform)

Offset	Data Bits [0:31]
0x00-0x03	Key 1 length
0x04-0x403	Key 1 value. (Remaining bytes will be padded with zero)
0x404-0x407	Key 2 length
0x408-0x807	Key 2 value. (Remaining bytes will be padded with zero)
0x808-0x80b	Key 3 length
0x80c-0xb0b	Key 3 value. (Remaining bytes will be padded with zero)
0xb0c-0xb0f	Key 4 length
0xb10-0xe10	Key 4 value. (Remaining bytes will be padded with zero)

LS1043/LS1046/LS1012 Platforms

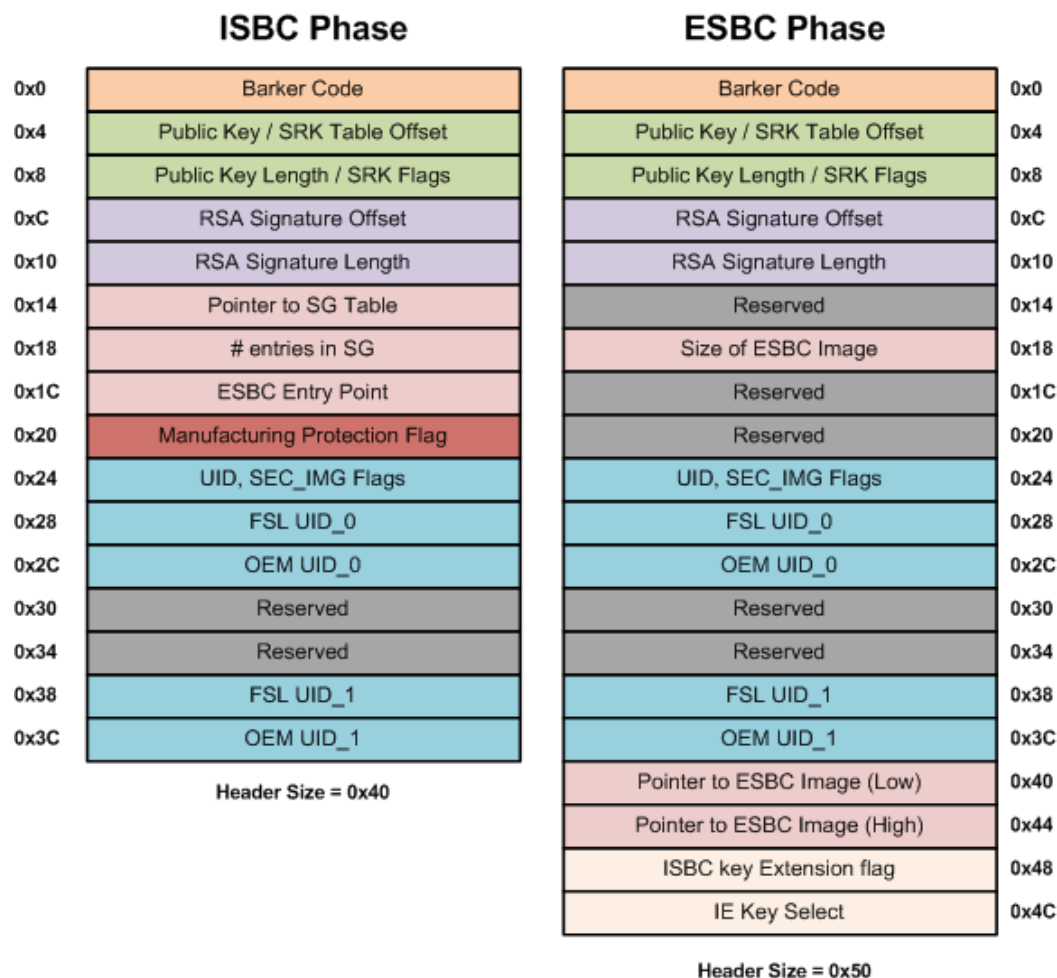


Figure 13. CSF Header for LS1043/LS1046/LS1012 (ISBC and ESBC Phase)

Table 16. CSF Header Format (LS1043/LS1046/LS1012 Platforms)

Offset	Data Bits [0:31]
0x00-0x03	Barker code. This location should contain the value: 0x68392781. The ISBC code searches for this Barker code. If the value in this location does not match the Barker code, the ISBC stops execution and reports error.
0x07-0x04	If the srk_table_flag is not set : • Public key offset: This location contains an address which is the offset of the public key from the start of CSF header. Using this offset and the public key length, the public key is read. If srk_table_flag is set: • Srk table offset: This location contains an address which is the offset of the srk table from the start of CSF header. Using this offset and the number of entries is SRK Table, the SRK table is read.

Table continues on the next page...

Table 16. CSF Header Format (LS1043/LS1046/LS1012 Platforms) (continued)

Offset	Data Bits [0:31]
0x08	Srk table flag. This flag indicates whether hash burnt in srk fuse is of a single key or of srk table.
0x0b-0x09	If the srk_table_flag is not set : • 0x0b-0x9 -- Public key length: This location contains the length of the public key in bytes. If srk_table_flag is set: • 0x09 – Key Number from srk table which is to be used for verification. • 0x0b-0x0a – Number of entries in srk table. Minimum number of entries in table = 1, Maximum = 4.
0x0f-0x0c	RSA Signature offset. This location contains an offset(in bytes) of the RSA signature from the start of CSF header. Using this offset and the Signature length, the RSA signature is read. The RSA signature is calculated over CSF Header, Scatter Gather table and ESBC images.
0x13-0x10	RSA Signature length in bytes.
0x17-0x14	For ISBC Phase: SG Table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG Table, the SG table is read. For ESBC Phase: Reserved
0x1b-0x18	For ISBC Phase: Number of entries in SG Table (Earlier ,Based on the Scatter gather flag in CSF Header, this location can either be treated as number of entries in SG table or ESBC image size in bytes.). For ESBC Phase Size of image to be validated
0x1f-0x1c	For ISBC Phase: ESBC entry point. ISBC transfers control to this location upon successful validation of ESBC image(s). For ESBC Phase: Reserved
0x21-0x20	Manufacturing Protection Flag Indicates if manufacturing protection has to be enabled or not in ISBC.
0x23-0x22	Reserved .(Earlier this field was SG Flag. SG flag is always assumed to be 1 in unified implementation.)
Table continues on the next page...	

Table 16. CSF Header Format (LS1043/LS1046/LS1012 Platforms) (continued)

Offset	Data Bits [0:31]
0x24	For ISBC Phase: Reserved For ESBC Phase: Reserved
0x25	For ISBC Phase Secondary Image flag Indicates if user has a secondary image available in case of failures in validating primary image. Valid in case of primary Images's Header. For ESBC Phase: Reserved
0x27-0x26	Unique ID Usage This location contains a flag which specifies one of these possibilities • 0x00 - No UID's present • 0x01 - FSL UID and OEM UID are present • 0x02 - Only FSL UID is present • 0x04 - Only OEM UID is present
0x2b-0x28	NXP unique ID 0 Upper 32 bits of a unique 64 bit value, which is specific to NXP. This value is compared with the FSL ID 1 in Secure Fuse Processor 's FSL-ID registers
0x2f-0x2c	OEM unique ID 0 Upper 32 bits of a unique 64 bit value, which is specific to OEM. This value is compared with the OEM ID 0 in Secure Fuse Processor 's OEM-ID registers
0x37-0x30	Reserved
0x3b-0x38	NXP unique ID 1 Lower 32 bits of a unique 64 bit value, which is specific to NXP. This value is compared with the FSL ID 1 in Secure Fuse Processor 's FSL-ID registers
0x3f-0x3c	OEM unique ID 1 Lower 32 bits of a unique 32 bit value, which is specific to OEM. This value is compared with the OEM ID 1 in Secure Fuse Processor 's OEM-ID registers
0x40-0x47	For ISBC Phase: Not Applicable For ESBC Phase: 64 bit pointer to ESBC image
0x48-0x4b	For ISBC Phase: Not Applicable For ESBC Phase: ISBC key Extension flag If this flag is set, key to be used for validation needs to be picked up from IE Key table.
Table continues on the next page...	

Table 16. CSF Header Format (LS1043/LS1046/LS1012 Platforms) (continued)

Offset	Data Bits [0:31]
0x4c-0x4f	For ISBC Phase: Not Applicable For ESBC Phase: IE Key Select Key Number to be used from the IE Key Table if IE flag is set.

Table 17. Scatter Gather Table Format (LS1043/LS1046/LS1012 Platforms)

Offset	Data Bits [0:31]
0x00-0x03	Length. This location specifies the length in bytes of the ESBC image 1.
0x04-0x07	Target where the ESBC Image 1 can be found. This field is ignored in case of PBL based SOC's.
0x08-0x0b	Source Address of ESBC Image 1
0x0c-0x0f	Destination Address of ESBC Image 1 If the target address is 0xffffffff, the image is not copied to the target. This field is ignored in case of PBL based SOC's.
0x10-0x13	Length. This location specifies the length in bytes of the ESBC image 2.
0x14-0x17	Target where the ESBC Image 2 can be found. This field is ignored in case of PBL based SOC's.
0x18-0x1b	Source Address of ESBC Image 2
0x1c-0x1f	Destination Address of ESBC Image 2 If the target address is 0xffffffff, the image is not copied to the target. This field is ignored in case of PBL based SOC's.

Table 18. Signature (LS1043/LS1046/LS1012 Platforms)

Offset	Data Bits [0:31]
0x00-size	The RSA signature calculated over CSF Header, Scatter Gather table and ESBC image(s).

Table 19. Public key (LS1043/LS1046/LS1012 Platforms)

Offset	Data Bits [0:31]
0x00-size	Public Key Value. The hash of this public key is compared with the hash stored in Secure Fuse Processor SRKH registers.

Table 20. SRK Table (LS1043/LS1046/LS1012 Platforms)

Offset	Data Bits [0:31]
0x00-0x03	Key 1 length
0x04-0x403	Key 1 value. (Remaining bytes will be padded with zero)
0x404-0x407	Key 2 length
0x408-0x807	Key 2 value. (Remaining bytes will be padded with zero)
0x808-0x80b	Key 3 length
0x80c-0xb0b	Key 3 value. (Remaining bytes will be padded with zero)
0xb0c-0xb0f	Key 4 length
0xb10-0xe10	Key 4 value. (Remaining bytes will be padded with zero)

6.1.1.10 ISBC Validation Error Codes

LS1/LS1043/LS1046/LS1012 platforms

Errors in the system can be of following types:

1. Core Exceptions
2. System State Failures
3. Header Checking Failures
 - a. General Failures
 - b. Key/Signature/UID related errors
4. Verification Failures
5. SEC/PAMU errors

Table 21. Core Exceptions (LS1 platform)

Value	Code	Definition
0x1	ERROR_UNDEFINED_INSTRUCTION	Occurs if neither the processor nor any attached co-processor recognizes the currently executing instruction.
0x2	ERROR_SWI	Software Interrupt is a user-defined interrupt instruction. It allows a program running in User mode, for example, to request privileged operations that run in Supervisor mode.
0x3	ERROR_PREFETCH_ABORT	Occurs when the processor attempts to execute an instruction that has been prefetched from an illegal address.
0x4	ERROR_DATA_ABORT	Occurs when a data transfer instruction attempts to load or store data at an illegal address.

Table continues on the next page...

Table 21. Core Exceptions (LS1 platform) (continued)

Value	Code	Definition
0x5	ERROR_IRQ	Occurs when the processor external interrupt request pin is asserted (LOW) and IRQ interrupts are enabled.
0x6	ERROR_FIQ	Occurs when the processor external fast interrupt request pin is asserted (LOW) and FIQ interrupts are enabled.

Table 22. Core Exceptions (LS1043/LS1046/LS1012 platforms)

Error Code	Value
Current EL with SP0	
ERROR_EXCEPTION_SYNC_SP0	0x01
ERROR_EXCEPTION_IRQ_SP0	0x02
ERROR_EXCEPTION_FIQ_SP0	0x03
ERROR_EXCEPTION_ERROR_SP0	0x04
Current EL with SPx	
ERROR_EXCEPTION_SYNC_SPX	0x05
ERROR_EXCEPTION_IRQ_SPX	0x06
ERROR_EXCEPTION_FIQ_SPX	0x07
ERROR_EXCEPTION_ERROR_SPX	0x08
Lower EL using AArch64	
ERROR_EXCEPTION_SYNC_L64	0x11
ERROR_EXCEPTION_IRQ_L64	0x12
ERROR_EXCEPTION_FIQ_L64	0x13
ERROR_EXCEPTION_ERROR_L64	0x14
Lower EL using AArch32	
ERROR_EXCEPTION_SYNC_L32	0x15
ERROR_EXCEPTION_IRQ_L32	0x16
ERROR_EXCEPTION_FIQ_L32	0x17
ERROR_EXCEPTION_ERROR_L32	0x18

Table 23. System State Failures (LS1/LS1043/LS1046/LS1012 platforms)

Value	Code	Definition
0x100	ERROR_CORE_NON_ZERO	ISBC is not running on CPU0
<i>Table continues on the next page...</i>		

Table 23. System State Failures (LS1/LS1043/LS1046/LS1012 platforms) (continued)

Value	Code	Definition
0x101	ERROR_STATE_NOT_CHECK	SEC_MON State Machine not in CHECK state at start of ISBC. Some Security violation could have occurred.
0x102	ERROR2_STATE_NOT_CHECK	SEC_MON State Machine not in CHECK state, when trying to transition it to Trusted/Non Secure/Soft Fail state
0x103	ERROR_SSM_TRUSTSTS	SEC_MON State Machine not in TRUSTED state at end of ISBC.

Table 24. General Header Checking Failures (LS1/LS1043/LS1046/LS1012 platforms)

Value	Code	Definition
0x301	ERROR_ESBC_HDR_LOC	ESBC header location is not in 3.5G space
0x302	ERROR_ESBC_HEADER_BARKER	Barker code in the header is incorrect.
0x303	ERROR_ESBC_HEADER_SG_ENTRIES_NOT_IN_3_5G	SG table/ESBC image address (header address + image offset in sg table) is beyond 3.5G
0x303	ERROR_ESBC_HEADER_SG_ENTRIES_ON_OCRAM	One Entry in the SG table is on OCRAM
0x304	ERROR_ESBC_HEADER_SG_ESBC_EP	ESBC entry point in header not within ESBC address range
0x305	ERROR_SGL_ENTIRES_NOT_SUPPORTED	Number of entries in SG table exceeds maximum limit i.e 8
0x306	ERROR_ESBC_HEADER_HKAREA_LENGTH_ZERO	Houskeeping area not provided in header
0x307	ERROR_ESBC_HEADER_HKAREA_NOT_IN_3_5G	House keeping area not in 3.5G boundary
0x308	ERROR_ESBC_HEADER_HKAREA_LENGTH_INSUFFICIENT	Housekeeping area length provided is not sufficient.
0x309	ERROR_SG_TABLE_NOT_IN_3_5	SG Table is not in 3.5G boundary
0x309	ERROR_SG_TABLE_ON_OCRAM	SG table is on OCRAM
0x310	ERROR_ESBC_HEADER_HKAREA_NOT_4K_ALIGNED	House keeping area is not aligned to 4K boundary
0x311	ERROR_SGL_ENTRIES_SIZE_ZERO	SG table has entry with size zero.

Table 25. Key/Signature/UID related errors (LS1/LS1043/LS1046/LS1012 platforms)

Value	Code	Definition
0x320	ERROR_ESBC_HEADER_KEY_LEN	Length of public key in header is not one of the supported values.
Table continues on the next page...		

Table 25. Key/Signature/UID related errors (LS1/LS1043/LS1046/LS1012 platforms) (continued)

Value	Code	Definition
0x321	ERROR_ESBC_HEADER_KEY_LEN_NOT_TWICE_SIG_LEN	Public key is not twice the length of the RSA signature
0x322	ERROR_ESBC_HEADER_KEY_MOD_1	Most significant bit of modulus in header is zero.
0x323	ERROR_ESBC_HEADER_KEY_MOD_2	Modulus in header is even number
0x324	ERROR_ESBC_HEADER_SIG_KEY_MOD	Signature value is greater than modulus in header
0x325	ERROR_FSL_UID	FSL_UID in ESBC Header did not match the FSL_UID in SFP if fsl uid flag is 1
0x326	ERROR_OEM_UID	OEM_UID in ESBC Header did not match the OEM_UID in SFP if oem uid flag is 1.
0x327	ERROR_INVALID_SRK_NUM_ENTRY	Number of entries field in CSF Header is > 4(This is when srk_flag in header is 1)
0x328	ERROR_INVALID_KEY_NUM	Key number to be used from srk table is not present in table. (This is when srk_flag in header is 1)
0x329	ERROR_KEY_REVOKED	Key selected from srk table has been revoked(This is when srk_flag in header is 1)
0x32a	ERROR_INVALID_SRK_ENTRY_KEY_LENGTH	Key length specified in one of the entries in srk table is not one of the supported values (This is when srk_flag in header is 1)
0x32b	ERROR_SRK_TBL_NOT_IN_3_5	SRK Table is not in 3.5G boundary (This is when srk_flag in header is 1)
0x32b	ERROR_SRK_TBL_ON_OCRAM	SRK Table is on OCRAM
0x32c	ERROR_KEY_NOT_IN_3_5G	Key is not in 3.5G boundary
0x32c	ERROR_KEY_ON_OCRAM	Key on OCRAM

Table 26. Verification Failures (LS1/LS1043/LS1046/LS1012 platforms)

Value	Code	Definition
0x340	ERROR_HASH_COMPARE_KEY	Super Root Key Hash Comparison failure. Mismatch in the hash of the public key/srk table as present in the header with the value in the SRK HASH fuse.
0x341	ERROR_HASH_COMPARE_EM	RSA signature check failure. Signature provided by you in the header doesn't match with the signature of the ESBC image generated by ISBC. The ESBC image loaded by you may be different than the image used while generating the signature(using CST)

Table 27. SEC/PAMU Failures (LS1/LS1043/LS1046/LS1012 platforms)

Value	Code	Definition
0x700	ERROR_SEC_ENQ	Error when enqueueing to SEC
0x701	ERROR_SEC_DEQ	Sec Block returned some error when dequeuing from it.
0x702	ERROR_SEC_DEQ_TO	Timeout when trying to deq from SEC
0x800	ERROR_PAMU	Error while programming PAACT/SPAACT tables in PAMU (For PowerPC platforms only)

6.1.1.11 ESBC Validation Error Codes

For trust arch version 1.x and 2.x.

Table 28. ESBC Validation Failures

Value	Code	Definition
0x0	ERROR_ESBC_CLIENT_MAX	NULL
0x4	ERROR_ESBC_CLIENT_HEADER_BARKER	Wrong barker code in header
0x8	ERROR_ESBC_CLIENT_HEADER_KEY_LEN	Wrong public key length in header
0x10	ERROR_ESBC_CLIENT_HEADER_SIG_LEN	Wrong signature length in header
0x11	ERROR_ESBC_CLIENT_HEADER_KEY_REVOKED	Key used to sign the image revoked
0x12	ERROR_ESBC_CLIENT_HEADER_INVALID_SRK_NUM_ENTRY	Wrong key entry
0x13	ERROR_ESBC_CLIENT_HEADER_INVALID_KEY_NUM	Selected key no. not in SRK table
0x14	ERROR_ESBC_CLIENT_HEADER_INVALID_SRK_ENTRY_KEYLEN	Unsupported key length of key in SRK table
0x15	ERROR_ESBC_CLIENT_HEADER_IE_KEY_REVOKED	Selected key in IE key table revoked
0x16	ERROR_ESBC_CLIENT_HEADER_INVALID_IE_NUM_ENTRY	Wrong IE Key entry
0x17	ERROR_ESBC_CLIENT_HEADER_INVALID_IE_KEY_NUM	Selected key no. not in IE Key table
0x18	ERROR_ESBC_CLIENT_HEADER_INVALID_IE_ENTRY_KEYLEN	Unsupported key length of key in IE Key table
0x19	ERROR_IE_TABLE_NOT_FOUND	information about IE table missing

Table continues on the next page...

Table 28. ESBC Validation Failures (continued)

Value	Code	Definition
0x20	ERROR_ESBC_CLIENT_HEADER_KEY_LEN_NOT_TWICE_SIG_LEN	Public key length not twice of signature length
0x21	ERROR_KEY_TABLE_NOT_FOUND	SRK Key/key table not found
0x40	ERROR_ESBC_CLIENT_HEADER_KEY_MOD_1	Public key Modulus most significant bit not set
0x80	ERROR_ESBC_CLIENT_HEADER_KEY_MOD_2	Public key Modulus in header not odd
0x100	ERROR_ESBC_CLIENT_HEADER_SIG_KEY_MOD	Signature not less than modulus
0x200	ERROR_ESBC_CLIENT_HEADER_SG_ENTRY_ERROR	Entry Point error
0x400	ERROR_ESBC_CLIENT_HASH_COMPARISON_FAILED	Public key hash comparison failed
0x800	ERROR_ESBC_CLIENT_HASH_VERIFICATION_FAILED	RSA verification failed
0x1000	ERROR_ESBC_CLIENT_SSM_TRUSTSTATE	SNVS not in TRUSTED state
0x2000	ERROR_ESBC_CLIENT_BAD_ADDRESS	Bad address error
0x4000	ERROR_ESBC_CLIENT_MISC	Miscellaneous error
0x8000	ERROR_ESBC_CLIENT_HEADER_SG_ENTRIES_BAD	Incorrect entries in SG table
0x10000	ERROR_ESBC_CLIENT_HEADER_NO_SG_SUPPORT	No SG support
0x20000	ERROR_ESBC_CLIENT_HEADER_INVALID_IMAGE_SIZE	Invalid Image size
0x40000	ERROR_ESBC_WRONG_CMD	Failure in command/Unknown command/Wrong arguments of boot script.
0x80000	ERROR_ESBC_MISSING_BOOTM	Bootm command missing from boot script.

6.1.1.12 Trust Architecture and SFP Information

SoC	Trust Arch. Version	SFP Version	POVDD	DRVR	OTPMK	SNVS/SFP Register to check Hamming Error

Table continues on the next page...

Table continued from the previous page...

				Algo (CST)	Register to check Hamming Error	Algo (CST)	Register to check Hamming Error	
LS1020A	2.1	3.3	1.89 V	A	SFP	2	SFP	
LS1043A	2.1	3.3	1.89 V	A	SFP	2	SFP	
LS1046A	2.1	3.3	1.89 V	A	SFP	2	SFP	
LS1012	2.1	3.3	1.89 V	A	SFP	2	SFP	

6.1.2 Service Processor (SP) Based Platforms

6.1.2.1 Secure Boot Introduction

There are three steps in the boot flow:

1. Service Processor Boot ROM

SP provides pre-boot initialization and secure boot capabilities. The on-chip SP Boot ROM offers read-only, non-volatile storage for the Boot ROM code, including the internal secure boot code (ISBC) sub-routine for image validation. This Boot ROM is an integral part in the booting of the SOC in non-secure and secure boot modes.

2. GPP Boot ROM

The on-chip GPP Boot ROM executes when the GPP cores are released from reset and is responsible for passing control to next step in the boot flow i.e. the boot-loader validated by the SP.

3. Boot Loader

The Boot Loader might further contain the External Secure Boot Code (ESBC) sub-routine for validation of next level images.

This document is intended for end-users to demonstrate the image validation process which happens in ISBC and ESBC.

The image validation can be split into stages, where each stage performs a specific function and validates the subsequent stage before passing control to that stage.

The Root of Trust is already established in the ISBC code residing in the Boot ROM which validates the Boot Loader 1.

Boot Loader 1 performs minimal SoC configuration before validating the Next Executable(s) which is/are known as ESBC image(s).

The flow includes validation of all ESBC images by a previously validated image before its execution to form a **Chain of TRUST**. The reference ESBC code also contains the functionality to form a **Chain of TRUST with confidentiality** where the next level images are kept on flash after encryption.

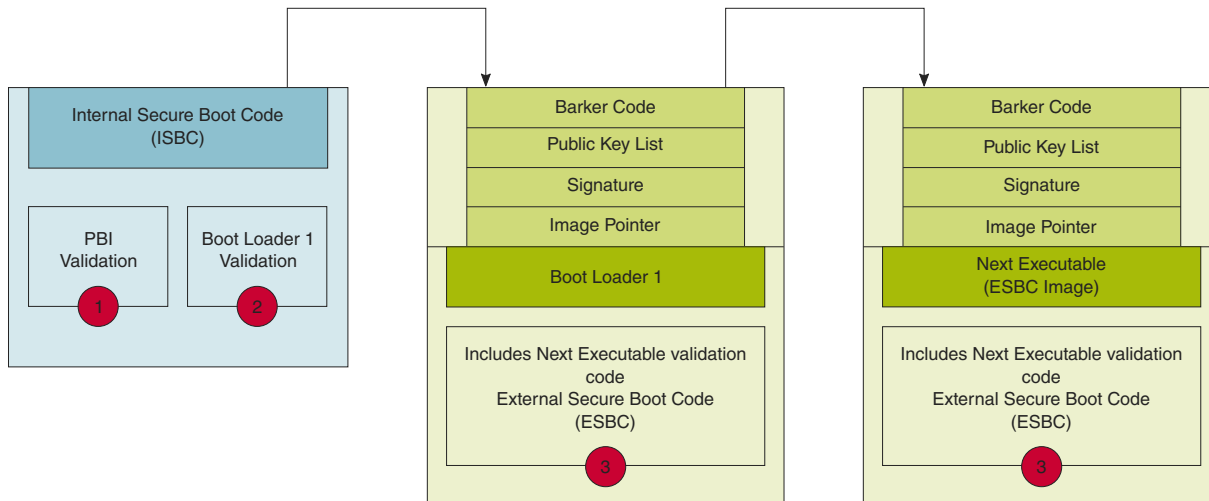


Figure 14. Chain of Trust

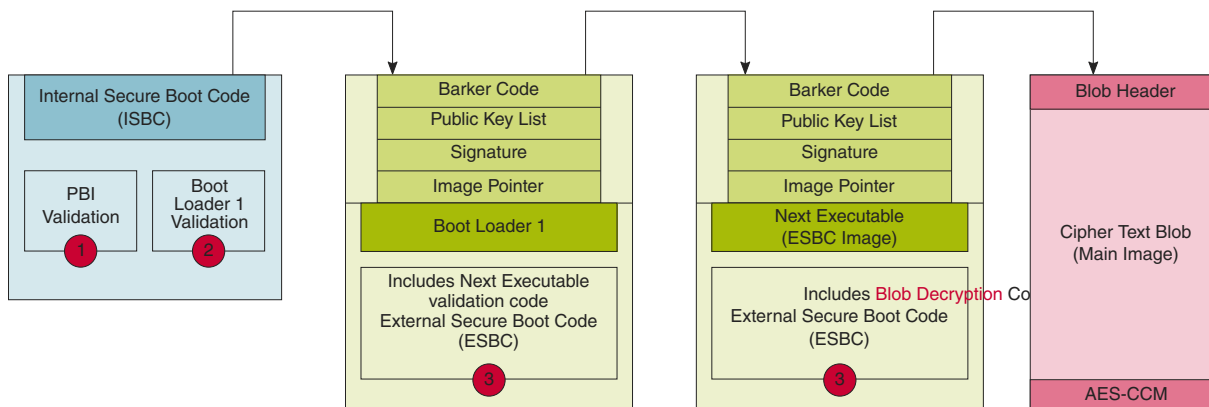


Figure 15. Chain of Trust with confidentiality

The validated ESBC image is allowed to use the One Time Programmable Master Key to decrypt system secrets. Cryptographic blob mechanism is used to establish Chain of trust with confidentiality.

The above is explained in detail in coming sections.

As depicted in figure(s) above, there are three types of images which need to be validated as part of Secure Boot.

1. PBI image by ISBC
2. Boot Loader 1 image by ISBC
3. Next level image(s) by ESBC

Typically ESBC images would include:

Boot Loader 2 - n In case Boot Loader is split in to multiple stages (Typical example is in case of NAND, SD Boot in which there is a mini-boot loader loaded on OCRAM which is Boot Loader stage 1 verified by ISBC. Boot Loader 2 is loaded on DDR, which must be validated by the Boot Loader 1.

MC/AIOP images Management Complex images

LINUX The operating system to be executed on the SoC.

6.1.2.1.1 Secure Boot process

Secure boot process uses a digital signature validation routine already present in ISBC residing in SP Boot ROM. This routine performs validation using HW bound RSA public key to decrypt the signed hash and compare it to a freshly calculated hash over the same system image. If the comparison passes, the image can be considered as authentic.

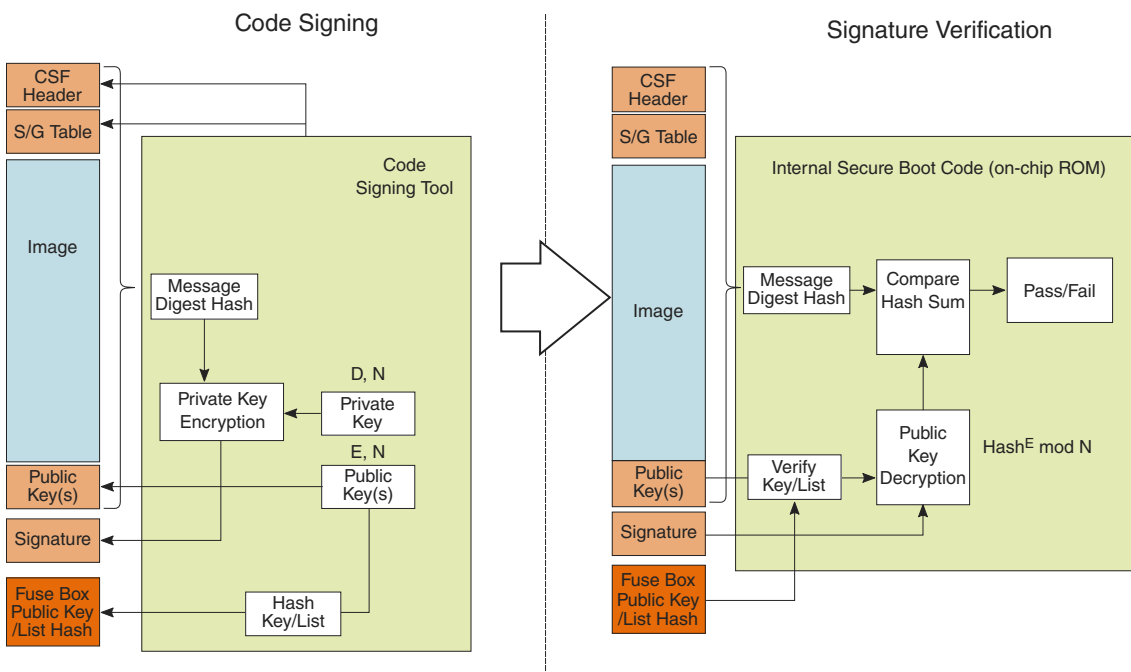


Figure 16. Secure Boot Process

As shown in the left side of the figure, the Code Signing Tool adds the following:

- CSF Header** Command Sequence File Header
This header provides the ISBC with flags, pointers, offsets, and lengths necessary to perform image validation.
- S/G Table** Scatter Gather Table
Optional (N/A for some stages which support only single image)
Allows support for multiple non-contiguous images.
- Public Key list** SRK (Super Root Key) Table
One or more public keys is appended to the image. The CSF header indicates which of the keys is to be used in signature validation.
- Signature** The SHA-256 hash of the CSF header + S/G table + Image + Public Key(s), encrypted with a RSA private key corresponding to one of the public keys in the key list.

As part of the code signing process, the CST also supports:

- Generating RSA public and private key pairs** The RSA private key is exported for the OEM to store securely
The CST also supports using public & private keys input by the OEM
- Hashing the public key or public key list** This hash becomes the Super Root Key Hash which is stored in the SFP

At a high level, the secure boot process runs code signing in reverse.

1. The ISBC locates the CSF header and S/G table to further locate the image, public key list, and signature

2. The public key (list) is hashed and compared to the SRKH
3. If the public key is good, it is used to decrypt the signature (recover the hash)
4. The CSF header + S/G table + Image + Public key list are hashed, with the result compared to above. If the two hashes match, image is considered to be authentic.

6.1.2.1.1 Super Root Key (SRK)

These are RSA public and private key pairs. Private keys are used to sign the images and public keys are used to validate the image during ISBC and ESBC phase.

Public keys are embedded in the header and the hash of SRK table is fused in SRKH register of SFP.

These are Hardware Bound Keys, once the hash is fused the public private key pair can not be modified.

Keys of sizes 1k, 2k and 4k are supported in NXP Secure Boot Process.

It is the OEM's responsibility to tightly control access to the RSA private signature key. If this key is ever exposed, attackers will be able to generate alternate images that will pass secure boot.

If this key is ever lost, the OEM will be unable to update the image.

Key	Trust Architecture provides support for revoking the RSA public keys used by the ISBC to verify the ESBC.
Revocation	The RSA public keys used for this purpose are called super root keys. The SRK table supports maximum of 8 public keys and user has the option to revoke up to 7 keys. During secure boot, the ISBC checks the key number indicated in the CSF header against the revocation fuses in the SFP's OEM Security Policy Register (SFP_OSPR). If the key is revoked, the image validation fails.

6.1.2.2 ISBC Phase

At reset, Service Processor core is released and begins executing instructions from reset vector address 0x0 which is mapped to Internal Boot ROM. The Internal Boot ROM contains the code known as Internal Secure Boot Code (ISBC). The main steps in ISBC flow are defined below.

6.1.2.2.1 ISBC for PBI validation

1. **Sec_Mon check:** Confirms that the Sec_Mon is in the Check state. If not, it writes a 'fail' bit in a Sec_Mon control register, leading to a state transition.
2. **PBI command check:** Verify that the first PBI command is 'LOAD SEC HDR'. If not found, an error is raised.
3. **Valid header check:** Check for a valid preamble and correct B0/1 flag set as 0 in the header. If not, an error is raised.
4. **CSF parsing and public key check:** If ISBC finds a valid CSF header, it parses the CSF header to locate the public key from SRK (Super Root Key) table to be used to validate the code. There can be a table of maximum 8 public keys present in the header. The Secure Fuse Processor doesn't actually store a public key, it stores a SHA-256 hash of the table. If the hash of the SRK table fails to match the stored hash, secure boot fails.
5. **Signature validation:** With the validated public key, ISBC decrypts the digital signature stored with the CSF header. The ISBC then uses the PBI length field in the RCW to calculate a hash over all PBI commands (CSF header is also a part of PBI commands) along with the SRK table. Optional flags in the CSF header tell the ISBC whether the FSL Unique ID and the OEM Unique ID (in the Secure Fuse Processor) are to be checked or not. Including these IDs allows the image to be bound to a single platform. If the decrypted hash and generated hash don't match, secure boot fails.
6. **Sec_Mon Transition:** If ISS (Increment Security State) flag is set in the header, transition the SNVS state from Check to Trusted.

NOTE

1. PBI commands in Secure Boot must have a command '**Load Boot 1 CSF Header Ptr**' to inform the ISBC about location of CSF Header for BOOT1 image (ESBC).
2. Boot1 image and header must be placed on an XIP memory before execution of next phase (ISBC validation of Boot1/ESBC). If these images are placed on memories like NAND/SD/eMMC, then they must be copied to an XIP memory like OCRAM, DDR via PBI commands.

6.1.2.2.2 ISBC for Boot1 (Boot Loader 1) validation

1. **Valid header check:** Check for a valid preamble and correct B0/1 flag set as 1 in the header. If not, an error is raised.
2. **CSF parsing and public key check:** If ISBC finds a valid CSF header, it parses the CSF header to locate the public key from SRK (Super Root Key) table to be used to validate the code. There can be a table of maximum 8 public keys present in the header. The Secure Fuse Processor doesn't actually store a public key, it stores a SHA-256 hash of the table. If the hash of the SRK table fails to match the stored hash, secure boot fails.
3. **Signature validation:** With the validated public key, ISBC decrypts the digital signature stored with the CSF header. The image information is stored in a SG (scatter gather) table with support of up to 8 discrete images. The ISBC calculates a hash over the CSF header, SRK table, SG table and all entries in SG table (i.e. images). Optional flags in the CSF header tell the ISBC whether the FSL Unique ID and the OEM Unique ID (in the Secure Fuse Processor) are to be checked or not. Including these IDs allows the image to be bound to a single platform. If the decrypted hash and generated hash do not match, secure boot fails.
4. **Entry Point check:** One final check is performed by the ISBC. This check confirms that the Entry Point to be updated in Boot Location Pointer falls within one of the SG entries which have been validated by the ISBC.
5. **Sec_Mon Transition:** If ISS (Increment Security State) flag is set in the header, transition the SNVS state from Check to Trusted or Trusted (if transitioned in PB phase) to Secure.

NOTE

1. After End of ISBC, Entry Point parsed from header is written to Boot LOC PTR register.
2. GPP is waken up.
3. SP goes to sleep.

There are many reasons the ISBC could fail to validate the PBI or Boot1. Technicians with debug access can check the DCFG SCRATCHRW3 register to obtain an error code. For a list of error codes refer ISBC Validation Error Codes.

6.1.2.3 ESBC Phase

Unlike the ISBC, which is in an internal ROM and therefore unchangeable, the ESBC is reference code, and can be changed by OEMs. The remainder of this section is the description of a reasonable secure boot chain of trust based on NXP's reference software for secure boot. The reference ESBC code is also part of the Boot 1 image validated by ISBC and would be used to validate further ESBC images such as U-Boot. U-Boot further has reference ESBC code to validate further images such as MC, AIOP, and LINUX and so on.

NXP provided ESBC consists of secure firmware (BL2, BL31) and standard u-boot which has been signed using a private key. If the Boot Mode is secure, user can't reach to uboot prompt as the environment variable **bootdelay** is defined to 0.

There is default boot command for secure boot in the environment which executes on auto boot. This default **bootcmd** validates a file called boot script and on successful validation execute the commands in the boot script.

There are many reasons ESBC could fail to validate Client images or boot script. The error status message along with the code is printed on the u-boot console. For a list of error codes refer ESBC Validation Error Codes.

Users are free to use NXP ESBC as it is provided or to use it as reference to modify their own secure boot system.

To establish the Secure Boot Chain of Trust, some U-Boot Commands have been added in the ESBC Code which will be discussed in detail in coming sections.

6.1.2.3.1 esbc_validate command

```
esbc_validate img_hdr [pub_key_hash]
```

Input arguments:

img_hdr – Location of CSF header of the image to be validated

pub_key_hash – hash of the public key used to verify the image. This is optional parameter. If not provided, code makes the assumption that the key pair used to sign the image is same as that used with ISBC. So the hash of the key in the header is checked against the hash available in SRK fuse for verification.

Description:

The command would do the following:

Perform CSF header validation on the address passed in the image header. During parsing of the header, image address is stored in an environment variable which is later used in source command in default secure boot command.

Signature checks on the image.

6.1.2.3.2 esbc_halt command

```
esbc_halt (no arguments)
```

Description:

This command puts core in spin loop.

6.1.2.3.3 blob_enc command

```
blob_enc <src location> <dst location> <length> <key_modifier address>
```

Input Arguments:

src location	Address of the image to be encapsulated
dst location	Address where the blob will be created
length	Size of the image to be encapsulated
key_modifier address	Address where a random number 16 bytes long (key modifier) is placed

Description:

This command would create a cryptographic blob of the image placed at src location and place the blob at dst location.

6.1.2.3.4 blob_dec command

```
blob_dec <src location> <dst location> <length> <key_modifier address>
```

Input Arguments:

src location	Address of the image blob to be decapsulated
dst location	Address where the decapsulated image will be placed
length	Expected Size of the image after decapsulation
key_modifier address	Address where a random number 16 bytes long(key modifier) is placed

Description:

This command would decapsulate the blob placed at src location and place the decapsulated data of expected size at dst location.

6.1.2.3.5 Boot Script

Boot script is a U-Boot script image which contains u-boot commands. ESBC would validate this boot script before executing commands in it.

1. Boot script can have any commands which u-boot supports. No checking is done on the allowed commands in boot script. Since it is validated image, assumption is that commands in boot script would be correct.
2. If some basic scripting error is done in boot script like unknown command, missing arguments, the required usage of that command and core is put in infinite loop.
3. After execution of commands in boot script, if control reaches back in u-boot, error message would be printed on u-boot console and core would be put in spin loop by command esbc_halt.
4. Scatter gather images are not supported with validate command.
5. If ITS fuse is blown, any error in verification of the image would result in system reset. The error would be printed on console before system goes for a reset.

Where to place the boot script?

ESBC u-boot expects the boot script to be loaded in flash. ESBC u-boot code assumes that the public/private key pair used to sign the boot script is same as that was used while signing the u-boot image. If user used different key pair to sign the image, hash of the N and E component of the key pair should be defined in macro:

```
CONFIG_BOOTSCRIPT_KEY_HASH
```

6.1.2.3.5.1 Chain of Trust

Boot script contains information about the next level images, e.g. MC, LINUX etc. ESBC validates these images as per their public keys. MC is started with validated MC images if required and finally bootm command is executed to pass control to LINUX image.

Users are free to use NXP ESBC as it is provided or to use it as reference to modify their own secure boot system. Figure below shows the Chain of Trust established for validation with this ESBC.

Sample Boot Script

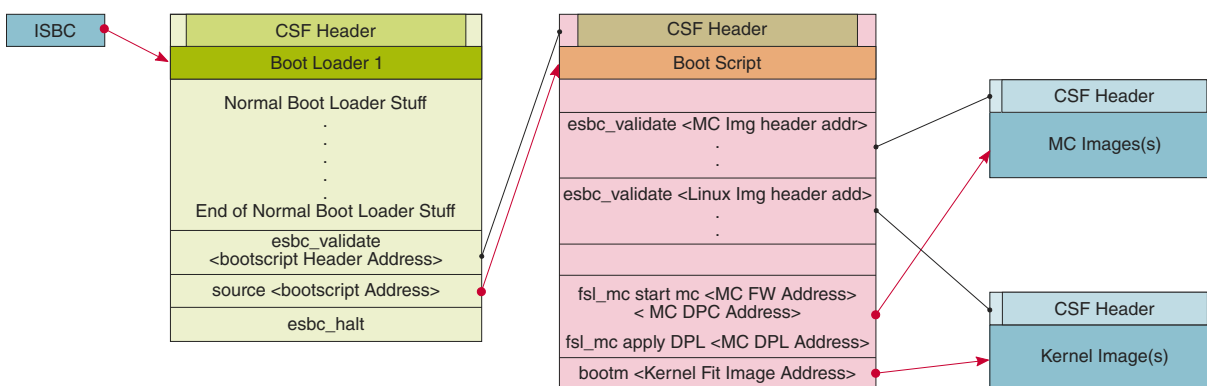


Figure 17. Secure Boot Flow (Chain of Trust)

```
# Get Images and Headers on DDR
.
```

```
.
# Validate the Images. (<pub_key_hash> is optional)
esbc_validate <Image1 Header Address> <pub_key_hash>
esbc_validate <Image2 Header Address> <pub_key_hash>
.
.
.

# Start MC with validated images
fsl_mc start mc <MC FW Address> < MC DPC Address>
fsl_mc apply DPL <MC DPL Address>

# Boot the Linux
bootm <Kernel Fit Image Address>
```

6.1.2.3.5.2 Chain of Trust with confidentiality

To establish chain of trust with confidentiality, cryptographic blob mechanism can be used. In this chain of trust, validated image is allowed to use the One Time Programmable Master Key to decrypt system secrets. Two bootscripts are to be used. First encap bootscripts is used which creates a blob of the next level images(e.g. MC, LINUX etc.) and saves them on flash. After this the system is booted after replacing the encap bootscript with decap bootscript which decapsulates the blobs and start MC and LINUX.

Sample Encap Boot Script

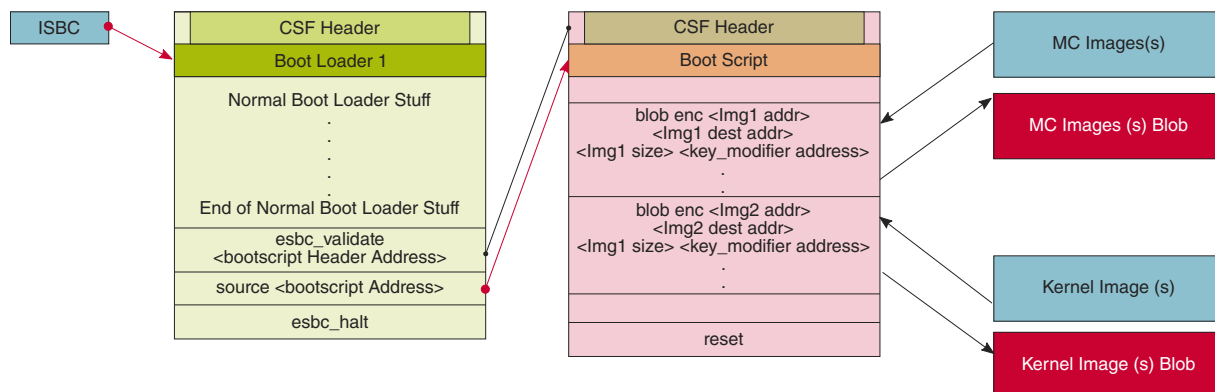


Figure 18. Chain of Trust with Confidentiality (Encapsulation)

```
# Get Images on DDR
.
.
.

# Create the Blobs
blob enc <Img1 addr> <Img1 dest addr> <Img1 size> <key_modifier address>
blob enc <Img2 addr> <Img2 dest addr> <Img2 size> <key_modifier address>
blob enc <Img3 addr> <Img3 dest addr> <Img3 size> <key_modifier address>
.
.
.

Save The Blobs created on Flash
.
.
.
```

```
# End of Encap Boot Script (This is one time only and must be replaced with decap Boot Script)
```

Sample Decap Boot Script

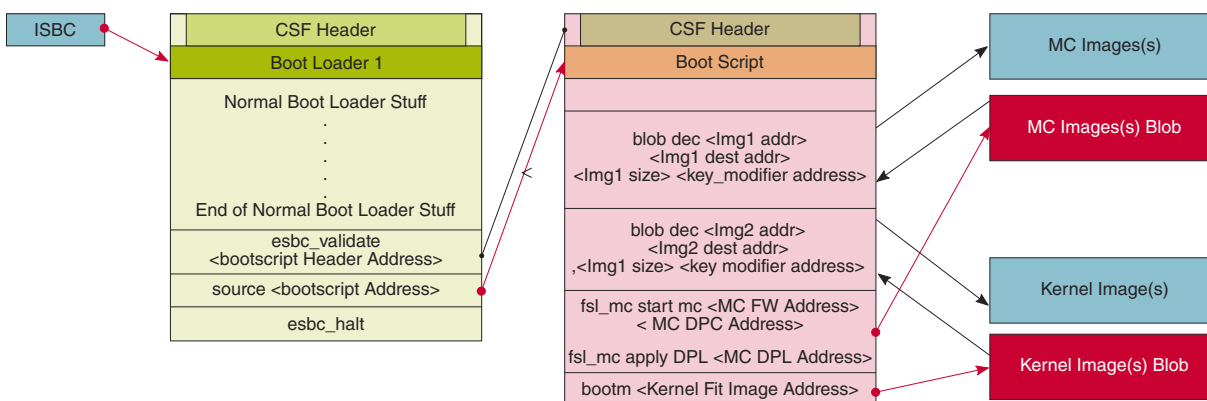


Figure 19. Chain of Trust with Confidentiality (Decapsulation)

```
# Get Images Blobs on DDR
.
.
.

# Decap the Blobs to get the actual images
blob dec <Img1 blob addr> <Img1 dest addr> <expected Img1 size> <key_modifier address>
blob dec <Img2 blob addr> <Img2 dest addr> <expected Img2 size> <key_modifier address>
blob dec <Img3 blob addr> <Img3 dest addr> <expected Img3 size> <key_modifier address>
.
.
.

# Start MC with validated images
fsl_mc start mc <MC FW Address> < MC DPC Address>
fsl_mc apply DPL <MC DPL Address>

# Boot the Linux
bootm <Kernel Fit Image Address>
```

6.1.2.4 Next executable phase

The boot loader (ESBC) finishes the platform initialization and passed control to the Linux image. The boot chain can be further extended to be able to sign application which would be running on Linux prompt. Further RTIC can be integrated to verify memory regions using Security Engine (SEC) during run time.

6.1.2.5 Product Execution

This section presents the steps to be followed in order to properly run the software product according to its intended use and functionalities.

Steps in the demo would be:

1. ISBC code would validate PBI and Boot Loader 1.
2. On Successful validation, PBI commands would be executed by SP BootROM.
3. Boot Loader 1 (BL2) execution will begin on GPP.

4. The ESBC code in BL2 will authenticate BL31, BL32, BL33 (U-Boot/UEFI). After successful validation BL2 passes control to BL31 which is the resident secure firmware (running in EL3 mode). If BL32 is present, BL31 would load and execute BL32 before passing execution control to BL33.
5. The ESBC code in BL33 would validate and execute and bootscript.^[3]
6. The boot script would contain the commands to validate and execute next level images as described in [Boot Script](#) on page 133.

6.1.2.5.1 Introduction

Running secure boot (Chain of Trust)

1. Setup the board for secure boot flow. You can choose any of the flows mentioned below.
 - a. **Flow A**
Program the ITS fuse.
 - b. **Flow B**
For prototyping phase, do not blow the ITS fuse, secure boot can be enabled by RCW with SB_EN = 1.
2. Blow other required fuses (TPMK and SRK Hash^[4]) in the SFP in silicon. For more details regarding fuse blowing, CCS and Reset Pause, refer to *Platform Reference Manual* and *Trust Architecture User Guide*.

NOTE

SRK hash in the fuse should be same as the hash of the key pair being used to sign the PBI and U-Boot image.

For testing purpose, the SRK hash can be written in the mirror registers.

gen_otpmk_drbg utility in CST can be used to generate otpmk key.

3. Flash all the generated images at locations as described in the address map
 - a. **Flow A** - All the images would have to be flashed at the current bank addresses.
 - b. If you are using **Flow B**, you can use alternate bank for demo purpose. This would mean flashing the images on alternate bank addresses from Bank0 and then switching to Bank1.
4. Give a power on cycle to the board.
 - a. For Flow A and Flow B (If secure boot images flashed on default bank)
 - On power on, ESBC code in SP boot ROM would validate the PBI image followed by validation of BL2 .
 - BL2 would validate BL31, BL32, BL33 image.
 - ESBC code in BL33 image would further validate the ESBC images (Boot Script, LINUX, MC, and so on)
 - MC and LINUX would be started.
 - b. For Flow B (If secure boot images flashed on alternate bank), the user must first do the switch settings^[5] for booting from alternate bank and also to enable reset pause.

[3] In case the boot loader is split into two parts, the validation and execution of boot script would happen in the final boot loader i.e Boot Loader2. Boot Loader 1 will validate and transfer control to Boot Loader 2.

[4] Blowing of OTPMK is essential to run secure boot for both Production (Flow A) and Prototyping/ Development (Flow B).

For SRK Hash, in Development Mode (Flow B), there is a workaround to avoid blowing fuses. The SoC can be put in a **Reset Pause** state. This will pause the Reset State Machinery after RCW Loading. Then CCS can be connected via JTAG.

Write the SRK Hash value in SFP mirror registers and then get the system out of Reset Pause State.

[5] This may also be done via writing to FPGA registers from the U-Boot Prompt of U-Boot running in Non-Secure Mode on Bank0. Refer the Platform FPGA guide for the same.

- On power on after the correct switch setting, Reset State Machinery will be paused after RCW loading.
- Write the SRKH to SFP mirror registers and get the system out of Reset Pause via CCS.
- Secure boot flow as mentioned above would execute.

Two additional features are provided in secure boot:

1. Chain of Trust with confidentiality
2. ISBC Key Extension

6.1.2.5.2 Chain of Trust with confidentiality

This section presents the steps to be followed in order to execute chain of trust with confidentiality.

The demo would be divided into two parts:

1. Creating /encrypting images in form of blobs.
2. Decrypting the images, and booting from decrypted images.

The execution steps remain same as specified above in [Product Execution](#) on page 135. In first phase the Boot Script would contain the commands to encrypt and create blobs of the images. After that the Boot Script is replaced and in second phase the Boot Script would contain commands to decrypt the blobs to get back the images and boot LINUX, AIOP using these images.

6.1.2.5.2.1 Other images required for demo

Apart from SDK images described above, the following images are also required:

1. Encap boot script

Sample Encap boot script

```
load \${devtype} \${devnum:2} \${kernelheader_addr_r} /secboot_hdrs/ls2088ardb/hdr_linux.out;
esbc_validate \${kernelheader_addr_r};
load \${devtype} \${devnum:2} \${fdtheader_addr_r} /secboot_hdrs/ls2088ardb/hdr_dtb.out; esbc_validate
\${fdtheader_addr_r};
size \${devtype} \${devnum:2} /vmlinuz; echo Encapsulating linux image;setenv key_addr 0x87000000;
mw \${key_addr} \${key_id_1};
setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_2};setexpr \${key_addr} \${key_addr} + 0x4;
mw \${key_addr} \${key_id_3};setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_4};
blob enc \${kernel_addr_r} \${load_addr} \${filesize} \${key_addr}; setexpr blobsize \${filesize}
+ 0x30;echo Saving encrypted linux ;save \${devtype} \${devnum:2} \${load_addr} /vmlinuz \
\${blobsize};size \${devtype} \${devnum:2} /fsl-ls1046a-rdb-sdk.dtb;
echo Encapsulating dtb image; blob enc \${fdt_addr_r} \${load_addr} \${filesize} \${key_addr}; setexpr
blobsize \${filesize} + 0x30;echo Saving encrypted dtb; save \${devtype} \${devnum:2} \${load_addr} /fsl-
ls1046a-rdb-sdk.dtb \${blobsize}; size \${devtype} \${devnum:2} /ls1046ardb_dec_boot.scr;
load \${devtype} \${devnum:2} \${load_addr} /ls2088ardb_dec_boot.scr;
echo replacing Bootscript; save \${devtype} \${devnum:2} \${load_addr} /ls2088ardb_boot.scr \
\${filesize};size \${devtype} \${devnum:2} /secboot_hdrs/ls2088ardb/hdr_ls2088ardb_bs_dec.out;
load \${devtype} \${devnum:2} \${load_addr} /secboot_hdrs/ls2088ardb/hdr_ls2088ardb_bs_dec.out ;echo
Replacing bootscript header; save \${devtype} \${devnum:2} \${load_addr} /hdr_ls2088ardb_bs.out \
\${filesize};reset;'
```

2. Decap boot script

```
size \${devtype} \${devnum:2} /vmlinuz;setexpr imgsize \${filesize} - 0x30 ;
echo Decapsulating linux image; setenv key_addr 0x87000000; mw \${key_addr} \${key_id_1};setexpr \
\${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_2};setexpr \${key_addr} \${key_addr} + 0x4; mw \
\${key_addr} \${key_id_3};setexpr \${key_addr} \${key_addr} + 0x4; mw \${key_addr} \${key_id_4};
blob dec \${kernel_addr_r} \${load_addr} \${imgsize} \${key_addr}; cp.b \${load_addr} \${kernel_addr_r} \
\${filesize} ;size \${devtype} \${devnum:2} /fsl-ls2088a-rdb.dtb;setexpr imgsize \${filesize} - 0x30 ;
```

```
echo Decapsulating dtb image; blob dec \${fdt_addr_r} \${load_addr} \${imgsize} \${key_addr}; cp.b \
${load_addr} \${fdt_addr_r} \${filesize} ;
```

6.1.2.5.2.2 Running secure boot (Chain of Trust with confidentiality)

1. Setup the board for secure boot flow. You can choose any of the flows mentioned below.
 - a. **Flow A**
Program the ITS fuse.
 - b. **Flow B**
For prototyping phase, do not blow the ITS fuse, secure boot can be enabled by RCW with SB_EN = 1.
2. Blow other required fuses(OTPMK and SRK hash^[6]) in the SFP in silicon. For more details regarding fuse blowing, CCS and Reset Pause, refer to Platform Reference Manual and Trust Architecture User Guide.

NOTE

*SRK hash in the fuse should be same as the hash of the key pair being used to sign the PBI and U-Boot image.

*For testing purpose, the SRK hash can be written in the mirror registers.

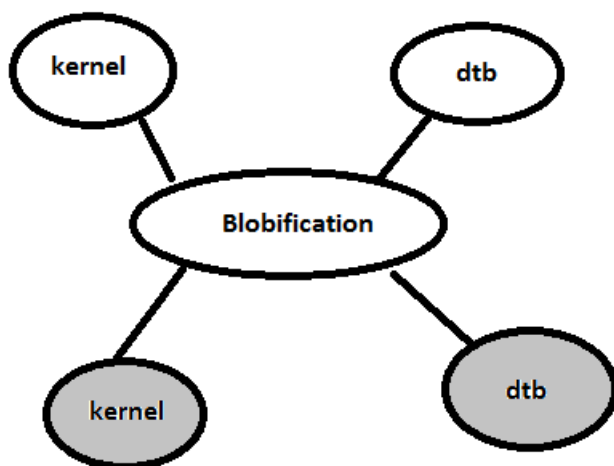
*gen_otpmk_drbg utility in CST can be used to generate otpmk key.

3. Flash all the generated images at locations as described in the address map.
 - a. **Flow A** - All the images would have to be flashed at the current bank addresses.
 - b. If you are using **Flow B**, you can use alternate bank for demo purpose. This would mean flashing the images on alternate bank addresses from Bank0 and then switching to Bank4.
4. Give a power on cycle to the board.
 - a. For Flow A and Flow B (If Secure Boot images flashed on default bank)
 - On power on, ISBC code in SP Boot ROM would validate the PBI image followed by validation of Boot Loader1 (U-Boot)
 - First Boot: Encapsulaton Step (Should happen in OEM's premises)
 - i. By default the enacap and decap bootscripts will be installed in the bootpartition.
 - ii. When the board boots up for the first time after all images have been generated, Encap bootscript will execute. This bootscript:
 - i. Authenticates and encapsulates linux and dtb images and replaces the unencrypted linux and dtb images with newly encapsulated linux and dtb.
 - ii. Replaces the encap bootscript and header with the decap bootscript and it's header, already present in the bootpartition.
 - iii. Issues reset

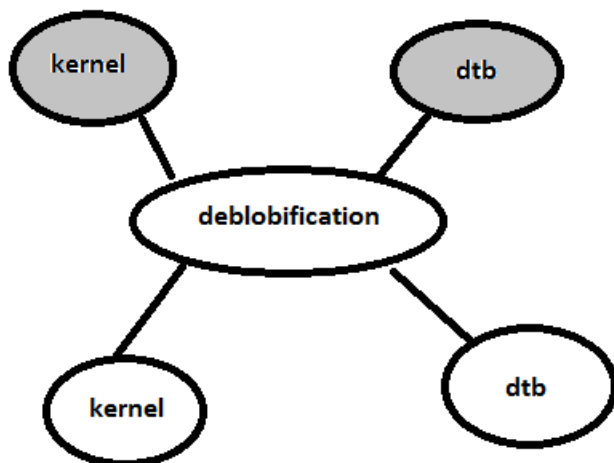
[6] Blowing of OTPMK is essential to run secure boot for both Production (Flow A) and Prototyping/ Development (Flow B).

For SRK Hash, in Development Mode (Flow B), there is a workaround to avoid blowing fuses. The SoC can be put in a **Reset Pause** state. This will pause the Reset State Machinery after RCW Loading. Then CCS can be connected via JTAG.

Write the SRK Hash value in SFP mirror registers and then get the system out of Reset Pause State.



- Subsequent Boot .
 - i. Uboot would execute script with decap commands
 - i. Un-blobify linux and dtb image in DDR
 - ii. Pass control to these images



- b. For Flow B (If Secure Boot images flashed on alternate bank), the user must first do the switch settings^[7] for booting from alternate bank and also to enable Reset Pause.
 - On power on after the correct switch setting, Reset State Machinery will be paused after RCW loading.
 - Write the SRKH to SFP mirror registers and get the system out of reset pause via CCS.
 - Secure Boot flow as mentioned above would execute.

[7] This may also be done via writing to FPGA registers from the U-Boot Prompt of U-Boot running in Non-Secure Mode on Bank0. Please refer the Platform FPGA guide for the same.

6.1.2.6 PBI structure

	Fields	Offset	Size (In 32-bit word)
RCW	Preamble (RCW)	0x00	1
	Load RCW command	0x04	1
	RCW words	0x08 – 0x87	32
	RCW checksum	0x88	1
PBI commands	Load security header	0x8c	1
	CSF header	0x90 – 0xdf	20
	Load boot 1CSF header	0xe0	1
	Boot 1 pointer	0xe4	1
	Other PBI commands	0xe8	N
	STOP command (With/ Without CRC)	0xe8 + (4*N)	2
SRK table	SRK table	0x90 + SRK table offset in CSF header	(No. of keys * Key length)
RSA signature	Signature	0x90 + Sign offset in CSF header	Sign length

RCW

- Preamble** The preamble is always the first element in a PBI image. It contains a standard pattern that identifies the memory location as the beginning of a valid PBI image. The preamble is a 4-byte pattern defined as “0xaa55aa55”.
- Load RCW command** The next word is load RCW command. This command loads the 1024-bit Reset Configuration Word from the interface specified by Power-on-Reset (POR) configuration strapping pins. It has the following two formats.
1. Load RCW with Checksum (0x10): Read Reset Configuration Word performs simple 32-bit checksum, and update RCW registers.
 2. Load RCW without Checksum (0x11): Read Reset Configuration Word and update RCW registers without performing checksum. The version without the checksum includes padding with zeroes in the place of the checksum value.
- RCW words** 1024 RCW bits that is 32 words of 32 bits.
- RCW checksum** It is calculated as a 32-bit unsigned integer summation of the RCW Preamble, the Load RCW with checksum command, and each of the 32 words (32-bit) of the RCW. A simple 32-bit checksum is used for the validation of the command.

```
checksum(RCW_WORD[]) {
    unsigned_32 sum = 0xAA55AA55 + 0x80100000 + Load RCW Command;
    for(i=0; i<32; i++)
        sum+=RCW_WORD[i];
}
```



```
return (sum);  
}
```

NOTE

Checksum will have to be updated by CST tool as the fields like SB_EN, PBI_LEN in the RCW words are changed.

PBI commands	Load security header	This command loads information required for authentication of the PBI image. The security header includes pointers to an SRK key table and RSA signatures as well as other flags and IDs. The CSF Header is part of the command. Please refer the CSF header structure in .
	Load boot 1 CSF header	This command loads a pointer to a CSF Header used for authentication of the Boot 1 Secondary Program Loader. This 32-bit value used by the Boot 0 ISBC and is required for secure boot.
	Other PBI commands	Other PBI commands input by user.
	STOP command	This command ends the PBI sequence and has two variants (with and without CRC). The CRC check value covers all commands from the first command after the RCW up to and including this CRC and Stop command, regardless of whether any are skipped by Jump commands during execution. In Stop command without CRC, it ends the PBI sequence immediately. It does not include a CRC value, but it instead has a 32-bit padding with zeroes so that it is the same size as the Stop with CRC command.

NOTE

CST tool updates the PBI commands by adding Load Security Header command and Load Boot 1 Security Header command. So, CRC must also be updated.

SRK table	Table of public keys is used in secure boot validation. It is kept at an offset from the CSF header. The offset is specified in the CSF header.
RSA signature	RSA signature is calculated over all PBI commands and SRK table. It is kept at an offset from the CSF header. The offset is specified in the CSF header.

6.1.2.7 CSF header structure definition

Table 29. Trust architecture and SFP information

SoC	Trust Arch. version	SFP version	POVDD	DRVR		OTPMK	
				Algo (CST)	Register to check Hamming Error	Algo (CST)	Register to check Hamming Error
LS1088A	3.1	3.5	1.89 V	A2	SFP Secret Value Hamming	2	SFP Secret Value Hamming

Table continues on the next page...

Table 29. Trust architecture and SFP information (continued)

LS2088A (LS2 Rev2)	3.1	3.5	1.89 V	A2	Error Status Register (SFP_SVHE SR)	2	Error Status Register (SFP_SVHE SR)

	PBI & ISBC Phase (Trust 3.0)	PBI & ISBC Phase (Trust 3.1)	ESBC Phase (Trust 3.0 & 3.1)
0x0	Barker Code	Barker Code	Barker Code
0x4	SRK Table Offset	SRK Table Offset	SRK Table Offset
0x8	Flags + Key Info	Flags + Key Info	Flags + Key Info
0xC	UID Flags	UID Flags	UID Flags
0x10	RSA Signature Offset	RSA Signature Offset	RSA Signature Offset
0x14	RSA Signature Length	RSA Signature Length	RSA Signature Length
0x18	Pointer to SG Table	Pointer to SG Table	64 bit Image Address Low
0x1C	# entries in SG	# entries in SG	64 bit Image Address High
0x20	32 bit Entry Point	64 bit Entry Point Low	Image Size
0x24	FSL UID_0	64 bit Entry Point High	Reserved
0x28	FSL UID_1	FSL UID_0	FSL UID_0
0x2C	OEM UID_0	FSL UID_1	FSL UID_1
0x30	OEM UID_1	OEM UID_0	OEM UID_0
0x34	OEM UID_2	OEM UID_1	OEM UID_1
0x38	OEM UID_3	OEM UID_2	OEM UID_2
0x3C	OEM UID_4	OEM UID_3	OEM UID_3
0x40	Reserved	OEM UID_4	OEM UID_4
0x44	Reserved	Reserved	Reserved
0x48	Reserved	Reserved	Reserved
0x4C	Reserved	Reserved	Reserved

Header Size = 0x50 Header Size = 0x50 Header Size = 0x50

Figure 20. CSF header structure

Table 30. CSF header structure (ISBC trust 3.0)

Offset	Description
0x00	Barker Code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)

Table continues on the next page...

Table 30. CSF header structure (ISBC trust 3.0) (continued)

Offset		Description
0x04		SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK table, the SRK table is read. * Description of fields in SRK table is mentioned below.
0x08	0x08	No. of keys This field specifies the no. of keys in the SRK table
	0x09	Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a	Field Reserved
	0x0b	IE : Reserved MP : Execute Manufacturing Protection Routine ISS : Increment Security State; indicates whether the ISBC should increment the SNVS SSM upon successful verification B01 : Identifies whether this is the CSF header of a boot 0 image (PBI) or a boot 1 image (SPL) LW : Leave Writeable; when set; ISBC does not set the SFP Write Disable
0x0C	0x0C	Reserved
	0x0D	Reserved
	0x0E	Reserved
	0x0F	OIDx : when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
0x10		RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF header, Scatter Gather table, and ESBC images.
0x14		RSA signature length This location contains the length of the RSA signature in bytes.
0x18		SG table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG Table, the SG table is read. * Description of fields in SG table is mentioned below.
Table continues on the next page...		

Table 30. CSF header structure (ISBC trust 3.0) (continued)

Offset	Description
0x1C	No. of entries This field specifies the number of entries present in SG table.
0x20	Entry point (32 bit) ISBC transfers control to this location upon successful validation of ESBC image(s).
0x24	FSL UID 0
0x28	FSL UID 1
0x2c	OEM UID 0
0x30	OEM UID 1
0x34	OEM UID 2
0x38	OEM UID 3
0x3c	OEM UID 4
0x40	Reserved
0x44	Reserved
0x48	Reserved
0x4C	Reserved

Table 31. CSF header structure (ISBC trust 3.1)

Offset	Description
0x00	Barker Code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)
0x04	SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK table, the SRK table is read.
0x08	0x08 No. of keys This field specifies the no. of keys in the SRK table

Table continues on the next page...

Table 31. CSF header structure (ISBC trust 3.1) (continued)

	0x09	Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a	Field Reserved
	0x0b	IE : ISBC Extension (Reserved) MP : Execute Manufacturing Protection Routine ISS : Increment Security State; indicates whether the ISBC should increment the SNVS SSM upon successful verification B01 : identifies whether this is the CSF header of a boot 0 image (PBI) or a boot 1 image (SPL) LW : Leave Writeable; when set; ISBC does not set the SFP Write Disable
0x0C	0x0C	Reserved
	0x0D	Reserved
	0x0E	Reserved
	0x0F	OIDx : when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
0x10		RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF header, Scatter Gather table, and ESBC images.
0x14		RSA signature length This location contains the length of the RSA signature in bytes.
0x18		SG table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG table, the SG table is read.
0x1C		No. of entries This field specifies the number of entries present in SG table.
0x20		Entry point (64 bit) ISBC transfers control to this location upon successful validation of ESBC image(s).
0x28		FSL UID 0
0x2c		FSL UID 1
0x30		OEM UID 0
Table continues on the next page...		

Table 31. CSF header structure (ISBC trust 3.1) (continued)

0x34	OEM UID 1
0x38	OEM UID 2
0x3c	OEM UID 3
0x40	OEM UID 4
0x44	Reserved
0x48	Reserved
0x4C	Reserved

Table 32. CSF header structure (ESBC trust 3.0 and trust 3.1)

Offset	Description
0x00	Barker Code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)
0x04	SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK Table, the SRK table is read.
0x08	0x08 No. of keys This field specifies the no. of keys in the SRK table
	0x09 Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a Field Reserved
	0x0b IE : ISBC Extension Flag
0x0C	0x0C Reserved
	0x0D Reserved
	0x0E Reserved

Table continues on the next page...

Table 32. CSF header structure (ESBC trust 3.0 and trust 3.1) (continued)

0x0F	OIDx: when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
0x10	RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF header, Scatter Gather table and ESBC images.
0x14	RSA signature length This location contains the length of the RSA signature in bytes.
0x18	Image address (64 bit)
0x20	Image size
0x24	IE Key Select
0x28	FSL UID 0
0x2c	FSL UID 1
0x30	OEM UID 0
0x34	OEM UID 1
0x38	OEM UID 2
0x3c	OEM UID 3
0x40	OEM UID 4
0x44	Reserved
0x48	Reserved
0x4c	Reserved

Table 33. SRK table structure

Offset	Description
0x00	SRK 0 Length (Length of Modulus or Exponent; Modulus length always equals Exponent length)
0x04	SRK 0 Value (Modulus)
0x04 + K	SRK 0 Value (Exponent)
0x04 + 2K	SRK 0 (Padding; 8Kb - (Exponent+Modulus))
0x04 * 1 + (10 * 1)K	SRK 1 Length (Length of Modulus or Exponent; Modulus length always equals Exponent length)
Table continues on the next page...	

Table 33. SRK table structure (continued)

Offset	Description
$0x04 * 2 + (10 * 1) K$	SRK 1 Value (Modulus)
$0x04 * 2 + (10 * 1) + 1k$	SRK 1 Value (Exponent)
$0x04 * 2 + (10 * 1) + 2K$	SRK 1 (Padding; 8Kb - (Exponent+Modulus))
$0x04 * (N-1) + (10 * (N-1))K$	SRK N Length (Length of Modulus or Exponent; Modulus length always equals Exponent length)
$0x04 * N + (10 * (N-1))K$	SRK N Value (Modulus)
$0x04 * N + (10 * (N-1)) + 1K$	SRK N Value (Exponent)
$0x04 * N + (10 * (N-1)) + 2K$	SRK N (Padding; 8Kb - (Exponent+Modulus))

Table 34. SG Table Structure

Offset	Description
0x00	Length
0x04	Reserved
0x08	SRC Address Low
0x0C	SRC Address High

6.1.2.8 CSF header structure definition

Table 35. Trust architecture and SFP information

SoC	Trust Arch. version	SFP version	POVDD	DRVR		OTPMK	
				Algo (CST)	Register to check Hamming Error	Algo (CST)	Register to check Hamming Error
LS2080A (LS2 Rev1)	3.0	3.4	1.89 V	A2	SFP Secret Value	2	SFP Secret Value
LS1088A	3.1	3.5	1.89 V	A2	Hamming Error Status	2	Hamming Error Status
LS2088A (LS2 Rev2)	3.1	3.5	1.89 V	A2	Register (SFP_SVHE SR)	2	Register (SFP_SVHE SR)

	PBI & ISBC Phase (Trust 3.0)	PBI & ISBC Phase (Trust 3.1)	ESBC Phase (Trust 3.0 & 3.1)
0x0	Barker Code	Barker Code	Barker Code
0x4	SRK Table Offset	SRK Table Offset	SRK Table Offset
0x8	Flags + Key Info	Flags + Key Info	Flags + Key Info
0xC	UID Flags	UID Flags	UID Flags
0x10	RSA Signature Offset	RSA Signature Offset	RSA Signature Offset
0x14	RSA Signature Length	RSA Signature Length	RSA Signature Length
0x18	Pointer to SG Table	Pointer to SG Table	64 bit Image Address Low
0x1C	# entries in SG	# entries in SG	64 bit Image Address High
0x20	32 bit Entry Point	64 bit Entry Point Low	Image Size
0x24	FSL UID_0	64 bit Entry Point High	Reserved
0x28	FSL UID_1	FSL UID_0	FSL UID_0
0x2C	OEM UID_0	FSL UID_1	FSL UID_1
0x30	OEM UID_1	OEM UID_0	OEM UID_0
0x34	OEM UID_2	OEM UID_1	OEM UID_1
0x38	OEM UID_3	OEM UID_2	OEM UID_2
0x3C	OEM UID_4	OEM UID_3	OEM UID_3
0x40	Reserved	OEM UID_4	OEM UID_4
0x44	Reserved	Reserved	Reserved
0x48	Reserved	Reserved	Reserved
0x4C	Reserved	Reserved	Reserved

Header Size = 0x50 Header Size = 0x50 Header Size = 0x50

Figure 21. CSF header structure

Table 36. CSF header structure (ISBC trust 3.0)

Offset	Description
0x00	Barker code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)
0x04	SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK table, the SRK table is read.
0x08	0x08 No. of keys This field specifies the no. of keys in the SRK table

Table continues on the next page...

Table 36. CSF header structure (ISBC trust 3.0) (continued)

Offset		Description
	0x09	Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a	Field Reserved
	0x0b	IE : Reserved MP : Execute Manufacturing Protection Routine ISS : Increment Security State; indicates whether the ISBC should increment the SNVS SSM upon successful verification B01 : Identifies whether this is the CSF header of a boot 0 image (PBI) or a boot 1 image (SPL) LW : Leave Writeable; when set, ISBC does not set the SFP Write Disable
0x0C	0x0C	Reserved
	0x0D	Reserved
	0x0E	Reserved
	0x0F	OIDx : when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
0x10		RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF Header, Scatter Gather table and ESBC images.
0x14		RSA signature length This location contains the length of the RSA signature in bytes.
0x18		SG table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG Table, the SG table is read.
0x1C		No. of entries This field specifies the number of entries present in SG table.
0x20		Entry point (32 bit) ISBC transfers control to this location upon successful validation of ESBC image(s).
0x24		FSL UID 0
0x28		FSL UID 1
Table continues on the next page...		

Table 36. CSF header structure (ISBC trust 3.0) (continued)

Offset	Description
0x2c	OEM UID 0
0x30	OEM UID 1
0x34	OEM UID 2
0x38	OEM UID 3
0x3c	OEM UID 4
0x40	Reserved
0x44	Reserved
0x48	Reserved
0x4C	Reserved

Table 37. CSF header structure (ISBC trust 3.1)

Offset	Description
0x00	Barker Code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)
0x04	SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK Table, the SRK table is read.
0x08	0x08 No. of keys This field specifies the no. of keys in the SRK Table
	0x09 Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a Field Reserved

Table continues on the next page...

Table 37. CSF header structure (ISBC trust 3.1) (continued)

	0x0b	IE : ISBC Extension (Reserved) MP : Execute Manufacturing Protection Routine ISS : Increment Security State; indicates whether the ISBC should increment the SNVS SSM upon successful verification B01 : identifies whether this is the CSF header of a boot 0 image (PBI) or a boot 1 image (SPL) LW : Leave Writeable; when set, ISBC does not set the SFP Write Disable
0x0C	0x0C	Reserved
	0x0D	Reserved
	0x0E	Reserved
	0x0F	OIDx: when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
0x10		RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF header, Scatter Gather table and ESBC images.
0x14		RSA signature length This location contains the length of the RSA signature in bytes.
0x18		SG table offset This location contains an address which is the offset of the SG table from the start of CSF header. Using this offset and the number of entries in SG Table, the SG table is read.
0x1C		No. of entries This field specifies the number of entries present in SG table.
0x20		Entry point (64 bit) ISBC transfers control to this location upon successful validation of ESBC image(s).
0x28		FSL UID 0
0x2c		FSL UID 1
0x30		OEM UID 0
0x34		OEM UID 1
0x38		OEM UID 2
0x3c		OEM UID 3
0x40		OEM UID 4
Table continues on the next page...		

Table 37. CSF header structure (ISBC trust 3.1) (continued)

0x44	Reserved
0x48	Reserved
0x4C	Reserved

Table 38. CSF header structure (ESBC trust 3.0 and trust 3.1)

Offset		Description
0x00		Barker code Fixed value which the ISBC uses to confirm it has located the start of a CSF header. (12_19_20_01) 0x00 – 0x12 0x01 – 0x19 0x02 – 0x20 0x03 – 0x01 It is numeric encoding of LSTA (LS Series Trust Architecture)
0x04		SRK table offset This location contains an address which is the offset of the SRK table from the start of CSF header. Using this offset and the number of entries in SRK table, the SRK table is read.
0x08	0x08	No. of keys This field specifies the no. of keys in the SRK table
	0x09	Key No. for verification Key # to use for verification; the key in the table which the ISBC uses to attempt signature verification
	0x0a	Field Reserved
	0x0b	IE : ISBC Extension Flag
0x0C	0x0C	Reserved
	0x0D	Reserved
	0x0E	Reserved
	0x0F	OIDx : when set, the corresponding OEM UID field in the SFP is included in the digital signature verification. For each bit set, the corresponding OUID field is included in the CSF header. FUID : when set, the 64b FUID is included in the digital signature verification and the FUID is included in the CSF header Other bits are reserved.
Table continues on the next page...		

Table 38. CSF header structure (ESBC trust 3.0 and trust 3.1) (continued)

0x10	RSA signature offset This location contains an address which is the offset of the RSA signature from the start of CSF header. Using this offset and the signature length, the RSA signature is read. The RSA signature is calculated over CSF header, Scatter Gather table and ESBC images.
0x14	RSA signature length This location contains the length of the RSA signature in bytes.
0x18	Image address (64 bit)
0x20	Image size
0x24	IE Key Select
0x28	FSL UID 0
0x2c	FSL UID 1
0x30	OEM UID 0
0x34	OEM UID 1
0x38	OEM UID 2
0x3c	OEM UID 3
0x40	OEM UID 4
0x44	Reserved
0x48	Reserved
0x4c	Reserved

6.1.2.9 Secure boot specific RCW fields

This section describes the various fields in RCW which are relevant to the ISBC code executed in the Service Processor Boot ROM.

SB_EN	Secure Boot Enable Bit(s): 266 0 Secure Boot is not enabled^[8] 1 Secure Boot is enabled
PBI_LENGTH	Pre-Boot Initialization Length Bit(s): 287-276 Size in words of the PBI commands.
SDBGEN	Secure Debug Enable Bit(s): 288 Secure Debug (CoreSight SPIDEN) is enabled after RCW is loaded if this RCW bit is set and the 'Intent to Secure' fuse value is cleared.

[8] Secure Boot is enabled if either this RCW bit is set or the Intent to Secure fuse value is set.

- 0 Secure debug is not enabled
- 1 Secure debug is enabled if the ITS fuse is not burned to asserted

GPIO_LED_EN GPIO LED Enable

Bit(s): 311

If the OEM chooses to implement a LED to indicate secure boot failure, the LED will be connected to a GPIO. The SP Boot ROM code sequence turns on the LED (if RCW[GPIO_LED_EN] = 1) by configuring one GPIO direction (GPDIR) register bit as an output and writing the corresponding output in a GPIO block data (GPDAT) register.

GPIO_LED_NUM GPIO Number for LED

Bit(s): 310-304

If GPIO_LED_EN is set, these bits specify the GPIO number to which LED is connected.

- 0x1f - 0x00 : GPIO_1
- 0x3f - 0x20 : GPIO_2
- 0x5f - 0x40 : GPIO_3
- 0x7f - 0x60 : GPIO_4

NOTE

The GPIO output assigned to the LED is driven high to whatever VDD voltage is supplied by the integrated device for the chosen GPIO output. Since GPIO pins at the time of SoC reset are initially configured as inputs, and since there will be some indeterminate period of time from the assertion of SoC reset to when the GPIO pin is configured by SP Boot ROM as an output, the GPIO pin chosen must be terminated with a weak pulldown to ground.

6.1.2.10 ISBC error codes

Error handling in production environment (ITS = 1)

- Error code would be logged in DCFG SCARTCH register.
- SNVS would be transitioned to **soft fail** state.
- Activate the LED. If the OEM chooses to implement a LED to indicate secure boot failure, the LED will be connected to a GPIO. The information of GPIO is specified via bits in RCW.

GPIO_LED_EN Bit(s): 311

The SP Boot ROM code sequence turns on the LED (if RCW[GPIO_LED_EN] = 1) by configuring one GPIO direction (GPDIR) register bit as an output and writing the corresponding output in a GPIO block data (GPDAT) register.

GPIO_LED_NUM Bit(s): 310-304

If GPIO_LED_EN is set, these bits specify the GPIO number to which LED is connected.

- 0x1f - 0x00 : GPIO_1
- 0x3f - 0x20 : GPIO_2
- 0x5f - 0x40 : GPIO_3
- 0x7f - 0x60 : GPIO_4

- Soft reset would be issued

- Cores would then enter infinite loop (If Reset is disabled)^[9]

Error handling in development environment (ITS = 0, SB_EN = 1)

- Error code would be logged in DCFG SCARTCH register.
- SNVS would be transitioned to **non-secure** state.
- Further actions depends on the type of failure

Fatal Errors Core is put in infinite Loop

Non-Fatal Error Application software is allowed to execute

Error codes

The Error codes reported by SP Boot ROM can be categorized in following sections.

1. Core exceptions
2. Device errors
3. RCW/PBI errors
4. Validation errors

Table 39. ISBC error codes

When error generated	Error code	Value	Description
Core exceptions			
Random	ERROR_UNDEFINED_INSTRUCTION	0x1	Occurs if neither the processor nor any attached co-processor recognizes the currently executing instruction.
Random	ERROR_SWI	0x2	Software Interrupt is a user-defined interrupt instruction. It allows a program running in User mode, for example, to request privileged operations that run in Supervisor mode.
Random	ERROR_PREFETCH_ABORT	0x3	Occurs when the processor attempts to execute an instruction that has been prefetched from an illegal address.
Random	ERROR_DATA_ABORT	0x4	Occurs when a data transfer instruction attempts to load or store data at an illegal address.
Random	ERROR_IRQ	0x5	Occurs when the processor external interrupt request pin is asserted (LOW) and IRQ interrupts are enabled.
Random	ERROR_FIQ	0x6	Occurs when the processor external fast interrupt request pin is asserted (LOW) and FIQ interrupts are enabled.
Device Errors – I2C			
<i>Table continues on the next page...</i>			

[9] To debug the root cause of failure and view the error code, Reset has to be disabled on the SoC.

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
Random	ERROR_I2C_TIMEOUT	0x11	
Random	ERROR_I2C_RESTART	0x12	
Random	ERROR_I2C_NODEV	0x13	
Random	ERROR_I2C_NOT_IDLE	0x14	
Random	ERROR_I2C_NOT_BUSY	0x15	
Random	ERROR_I2C_INVALID_OFFSET	0x16	
Random	ERROR_I2C_NO_WAKEUP_INIT	0x17	
Random	ERROR_I2C_NO_WAKEUP_READ	0x18	
Random	ERROR_I2C_NOACK	0x19	
Random	ERROR_READ_TIMEOUT	0x1a	
Random	ERROR_SLAVE_ADDR_TIMEOUT	0x1b	
Random	ERROR_MEM_ADDR_TIMEOUT	0x1c	
Device Errors – ESDHC			
Random	ERROR_ESDHC_CARD_DETECT_FAIL	0x31	
Random	ERROR_ESDHC_UNUSABLE_CARD	0x32	
Random	ERROR_ESDHC_COMMUNICATION_ERROR	0x33	
Random	ERROR_ESDHC_BLOCK_LENGTH	0x34	
Device Errors – QSPI			
Random	ERROR_QSPI_INVALID_OFFSET	0x41	
Phase – “RCW”			
RCW Phase	ERROR_PREAMBLE	0x50	Preamble not found.
RCW Phase	ERROR_RCW_CMD_NOT_FOUND	0x51	RCW command not found
RCW Phase	ERROR_RCW_CHECKSUM_MISMATCH	0x52	Checksum mismatch in RCW
RCW Phase	ERROR_RCW_SRC_INVALID	0x58	RCW_SRC is not a valid source
RCW Phase	ERROR_RCW_REQ_NOT_SET	0x59	RCW_REQ bit never set by Reset state machine (RSM)
RCW Phase	ERROR_PBI_REQ_NOT_SET	0x60	PBI_REQ bit never set (by RSM)
Phase = PBI			
PBI Phase	ERROR_SEC_CAAM_INIT	0x61	CAAM init failed (Would rarely occur)
<i>Table continues on the next page...</i>			

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
PBI Phase	ERROR_SEC_CAAM_NOT_FOUND	0x62	CAAM block not found in case of secure boot
PBI Phase	ERROR_PBI_SRC_NOT_SAME_AS_RCW_SRC	0x64	Mismatch between RCW_SRC and PBI_SRC fields
PBI Phase	ERROR_PBI_LENGTH	0x65	PBI length defined in RCW[PBI_LEN] field is invalid
PBI Phase	ERROR_PBI_LAST_CMD_NOT_STOP	0x66	STOP or CRC&STOP not found at the end of the specified PBI Length.
PBI Phase	ERROR_PBI_COMMAND_UNKNOWN	0x67	An invalid command parsed by PBI Parser
PBI Phase	ERROR_CAAM_SELF_TEST	0x6a	CAAM self-test failed
PBI Phase	ERROR_PBI_COPY_INVALID_SRC_TYPE	0x70	Copy command, src field does not match the RCW_SRC field
PBI Phase	ERROR_PBI_COPY_INVALID_DST_ADDR	0x71	Copy command, dest field is not 0x00
PBI Phase	ERROR_PBI_COPY_INVALID_SRC_ADDR_SRC_ADDR	0x72	SRC address is invalid (ROM/ OCRM reserved for SP)
PBI Phase	ERROR_PBI_CCSR_BYTE_COUNT	0x74	Byte count in CCSR Write not valid
PBI Phase	ERROR_PBI_CCSR_4_BYTE_ALIGNED	0x75	Offset is not 4 byte aligned
PBI Phase	ERROR_PBI_CCSR_OFFSET_INVALID	0x76	Offset is invalid that is less than allowed CCSR Base 0x0100_0000
PBI Phase	ERROR_PBI_ACSR_INVALID_ADDRESS	0x78	Source address in ACSR invalid (invalid addresses - OCRM or ROM address)
PBI Phase	ERROR_PBI_ACSR_BYTE_COUNT	0x79	Byte count in ACSR write command not valid
PBI Phase	ERROR_PBI_ACSR_WINDOW_NOT_SET	0x7a	ATU Window is not configured
PBI Phase	ERROR_PBI_ACSR_OFFSET_ALIGNED	0x7b	ACSR offset is invalid and trying to write to Reserved space on OCRM.
PBI Phase	ERROR_PBI_ALTCFG_WNDW_INVALID	0x7c	ATU Window is invalid
PBI Phase	ERROR_PBI_JUMP_OUT_LENGTH	0x80	Offset specified in JUMP command does not lie in PBI length range
PBI Phase	ERROR_PBI_JUMP_4_BYTE_ALIGNED	0x81	Offset specified in JUMP command is not 4 byte aligned
PBI Phase	ERROR_PBI_JUMP_OFFSET_0	0x82	Offset specified in JUMP command is 0

Table continues on the next page...

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
PBI Phase	ERROR_PBI_LOADC_4_BYTE_ALIGN ED	0x84	Address specified in LOAD condition command is not 4 byte aligned
PBI Phase	ERROR_PBI_JUMPC_OUT_LENGTH	0x88	Offset specified in JUMP command does not lie in PBI length range
PBI Phase	ERROR_PBI_JUMPC_4_BYTE_ALIGN ED	0x89	Offset specified in JUMP conditional command is not 4 byte aligned
PBI Phase	ERROR_PBI_JUMPC_CONDITION_NO T_SET	0x8a	Jump conditional command encountered before condition is set using Load Condition
PBI Phase	ERROR_PBI_CRC_MISMATCH	0x90	CRC mismatch
PBI Phase	ERROR_PBI_POLL	0x91	Poll timeout
PBI Phase	ERROR_PBI_POLL_4_BYTE_ALIGN ED	0x92	Address being polled is not 4 byte aligned
PBI Phase	ERROR_PBI_BOOT1_CSF_INVALID_A DDR	0x94	Address of CSF header is not valid
PBI Phase	ERROR_PBI_BOOT1_CSF_ALLIGNED	0x95	Address of CSF header is not 4 byte aligned
Phase = Verify (System State Errors (Secure boot))			
Before PBI verification	ERROR_STATE_NOT_CHECK	0xf0	SEC_MON State Machine not in CHECK state at start of ISBC in primary flow. Some Security violation could have occurred.
Before PBI verification	ERROR_STATE_NOT_CHECK_TRUST ED	0xf1	SEC_MON State Machine not in CHECK/ Trusted state at start of ISBC in secondary flow.
Phase = Verify (Secure Boot Fatal errors)			
Verify PBI	ERROR_PBI_COMMANDS_NOT_FOU ND	0xf4	Not having PBI commands in RCW is error scenario for secure boot
Verify PBI	ERROR_SEC_HDR_NOT_FOUND	0xf5	Error if security header command not found in RCW. Expected location of Security Header command • After Preamble for hard coded RCW • After preamble and rcw for other RCW sources
Phase = Verify (Secure Boot Fatal (Header parsing errors))			
Verify PBI	ERROR_HEADER_LOC	0xf8	Header location is invalid
Verify PBI	ERROR_HEADER_BARKER	0xf9	Barker code in the header is incorrect
Verify PBI	ERROR_HEADER_INVALID	0xfa	Flag B01 in the header identifies this as SPL header
Phase = Verify (Secure Boot Non Fatal (Key/UID related errors))			
Table continues on the next page...			

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
Verify PBI	ERROR_INVALID_SRK_ENTRY_KEYLEN	0x210	Length of public key specified in one of the entries in srk table is not one of the supported values. (1k, 2k, or 4k)
Verify PBI	ERROR_KEY_LEN_NOT_TWICE_SIG_LEN	0x211	Public key is not twice the length of the RSA signature
Verify PBI	ERROR_KEY_MOD_1	0x212	Most significant bit of modulus in header is zero.
Verify PBI	ERROR_KEY_MOD_2	0x213	Modulus in header is even number
Verify PBI	ERROR_SIG_KEY_MOD	0x214	Signature value is greater than modulus in header
Verify PBI	ERROR_INVALID_SRK_NUM_ENTRY	0x215	Number of entries field in CSF Header is > 8 (This is when srk_flag in header is 1)
Verify PBI	ERROR_INVALID_KEY_NUM	0x216	Key number to be used from srk table is not present in table. (This is when srk_flag in header is 1)
Verify PBI	ERROR_KEY_REVOKED	0x217	Key selected from srk table has been revoked (This is when srk_flag in header is 1)
Verify PBI	ERROR_FSL_UID	0x220	FSL_UID in ESBC header did not match the FSL_UID in SFP if fsl uid flag is 1
Verify PBI	ERROR_OEM_UID0	0x221	OEM_UID0 in ESBC header did not match the OEM_UID0 in SFP if oem uid0 flag is 1.
Verify PBI	ERROR_OEM_UID1	0x222	OEM_UID1 in ESBC header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify PBI	ERROR_OEM_UID2	0x223	OEM_UID1 in ESBC header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify PBI	ERROR_OEM_UID3	0x224	OEM_UID1 in ESBC header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify PBI	ERROR_OEM_UID4	0x225	OEM_UID1 in ESBC header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Phase = Verify (Header Verification Failure) Secure Boot Non Fatal			
Verify PBI	ERROR_HASH_COMPARE_KEY	0x240	Super Root Key Hash Comparison failure. Mismatch in the hash of the public key/srk table as present in the header with the value in the SRK HASH fuse.
Table continues on the next page...			

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
Verify PBI	ERROR_HASH_COMPARE_EM	0x241	RSA signature check failure. Signature provided by you in the header does not match with the signature of the ESBC image generated by ISBC. The ESBC image loaded by you may be different than the image used while generating the signature (using CST)
Phase = Verify (Secure Boot Fatal (Header parsing errors))			
Verify Boot1	ERROR_HEADER_LOC	0x100f8	Header location is invalid
Verify Boot1	ERROR_HEADER_BARKER	0x100f9	Barker code in the header is incorrect.
Verify Boot1	ERROR_HEADER_INVALID	0x100fa	Flag B01 in the header identifies this as SPL header.
Phase = Verify (Secure Boot Fatal (SG Table related errors))			
Verify Boot1	ERROR_SG_ENTRY_POINT	0x10200	Entry point is not within any of SG entries
Verify Boot1	ERROR_SG_NUM_ENTRY	0x10201	No. of entries in SG table is 0 or >8
Verify Boot1	ERROR_SG_SIZE_ZERO	0x10202	A SG entry has size 0
Phase = Verify (Secure Boot Non-Fatal (Key/UID related errors))			
Verify Boot1	ERROR_INVALID_SRK_ENTRY_KEYLEN	0x10210	Length of public key specified in one of the entries in srk table is not one of the supported values. (1k, 2k, or 4k)
Verify Boot1	ERROR_KEY_LEN_NOT_TWICE_SIG_LEN	0x10211	Public key is not twice the length of the RSA signature
Verify Boot1	ERROR_KEY_MOD_1	0x10212	Most significant bit of modulus in header is zero
Verify Boot1	ERROR_KEY_MOD_2	0x10213	Modulus in header is even number
Verify Boot1	ERROR_SIG_KEY_MOD	0x10214	Signature value is greater than modulus in header
Verify Boot1	ERROR_INVALID_SRK_NUM_ENTRY	0x10215	Number of entries field in CSF Header is > 8 (This is when srk_flag in header is 1)
Verify Boot1	ERROR_INVALID_KEY_NUM	0x10216	Key number to be used from srk table is not present in table. (This is when srk_flag in header is 1)
Verify Boot1	ERROR_KEY_REVOKED	0x10217	Key selected from srk table has been revoked (This is when srk_flag in header is 1)
Verify Boot1	ERROR_FSL_UID	0x10220	FSL_UID in ESBC Header did not match the FSL_UID in SFP if fsl uid flag is 1
Table continues on the next page...			

Table 39. ISBC error codes (continued)

When error generated	Error code	Value	Description
Verify Boot1	ERROR_OEM_UID0	0x10221	OEM_UID0 in ESBC Header did not match the OEM_UID0 in SFP if oem uid0 flag is 1.
Verify Boot1	ERROR_OEM_UID1	0x10222	OEM_UID1 in ESBC Header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify Boot1	ERROR_OEM_UID2	0x10223	OEM_UID1 in ESBC Header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify Boot1	ERROR_OEM_UID3	0x10224	OEM_UID1 in ESBC Header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Verify Boot1	ERROR_OEM_UID4	0x10225	OEM_UID1 in ESBC Header did not match the OEM_UID1 in SFP if oem uid1 flag is 1.
Phase = Verify (Header Verification Failure) Secure Boot Non-Fatal			
Verify Boot1	ERROR_HASH_COMPARE_KEY	0x10240	Super Root Key Hash Comparison failure. Mismatch in the hash of the public key/srk table as present in the header with the value in the SRK HASH fuse.
Verify Boot1	ERROR_HASH_COMPARE_EM	0x10241	RSA signature check failure. Signature provided by you in the header does not match with the signature of the ESBC image generated by ISBC. The ESBC image loaded by you may be different than the image used while generating the signature(using CST)
Verify Boot1	ERROR_PRIVATE_KEY_DERIVATION	0x10250	Error in Private key derivation when enabling Manufacturing Protection.

6.1.2.11 ESBC error codes

Table 40. ESBC validation failures

Value	Code	Definition
0x4	ERROR_ESBC_CLIENT_HEADER_BARKER	Wrong barker code in header
0x8	ERROR_ESBC_CLIENT_HEADER_KEY_LEN	Wrong public key length in header
0x10	ERROR_ESBC_CLIENT_HEADER_SIG_LEN	Wrong signature length in header
0x20	ERROR_ESBC_CLIENT_HEADER_KEY_LEN_NOT_TWICE_SIG_LEN	Public key length not twice of signature length
Table continues on the next page...		

Table 40. ESBC validation failures (continued)

Value	Code	Definition
0x40	ERROR_ESBC_CLIENT_HEADER_KEY_MOD_1	Public key Modulus most significant bit not set
0x80	ERROR_ESBC_CLIENT_HEADER_KEY_MOD_2	Public key Modulus in header not odd
0x100	ERROR_ESBC_CLIENT_HEADER_SIG_KEY_MOD	Signature not less than modulus
0x400	ERROR_ESBC_CLIENT_HASH_COMPAR_E_KEY	Public key hash comparison failed
0x800	ERROR_ESBC_CLIENT_HASH_COMPAR_E_EM	RSA verification failed
0x10000	ERROR_ESBC_CLIENT_HEADER_SG	No SG support
0x20000	ERROR_ESBC_WRONG_CMD	Failure in command/Unknown command/Wrong arguments of boot script.
0x40000	ERROR_ESBC_MISSING_BOOTM	Bootm command missing from boot script.

6.1.2.12 Troubleshooting

	Symptoms	Reasons and/or Recommended actions
1.	No print on UART console.	Check the status register of sec mon block. Refer to the details of the register from the Reference Manual. Bits OTPMK_ZERO, OTMPK_SYNDROME and PE should be 0 otherwise there is some error in the OTPMK fuse blown by you. If OTMPK fuse is correct (see Step 1), check the DCFG SCRATCHRW3 register for error code. For a list of error codes, see ISBC error codes on page 155 If Error code = 0 then check the Security Monitor state in HPSR register of Sec Mon. Sec Mon in Check State (0x9) If ITS fuse = 1, then it means ISBC code has reset the board. This may be due to the following reasons “ Hash of the public key used to sign the ESBC U-Boot does not match with the value in SRK Hash Fuse Or Signature verification of the image failed Sec Mon in Trusted State (0xd) or Non-Secure State (0xb) Check the entry point field in the CSF header. If entry point is correct, ensure that U-Boot image has been signed with the correct input file.
2.	Instead of Linux prompt, you get a U-Boot command prompt.	You have not booted in secure boot mode. You never get a U-Boot prompt in secure boot flow. You would reach this stage if ITS = 0 and you are running normal U-Boot.
3	U-Boot hangs or board resets	Some validation failure occurred in U-Boot. Error code and description would be printed on U-Boot console. See ESBC error codes on page 162 for more details on errors.

6.1.3 Code Signing Tool

To assist with signing of various images and creation of CSF header, NXP offers a Code Signing Tool (CST). It is generally expected that the CST signs images in an offline process

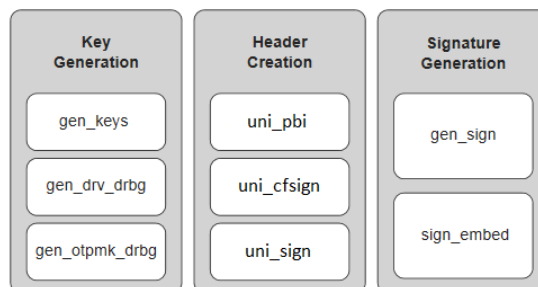


Figure 22. Tool in CST Package

6.1.3.1 Key generation

The CST begins by generating a RSA public and private key pair using OPENSSL APIs. The key pair consists of 3 parts; N, E, and D.

N - Modulus

E - Encryption exponent

D - Decryption exponent

Public Key - It is a combination of E and N components.

Private Key - It is a combination of D and N components.

The application allows the user to feed 3 key sizes for generating keys. The key sizes are 1024 bits, 2048 bits, and 4096 bits.

It is the OEM's responsibility to tightly control access to the RSA private signature key. If this key is ever exposed, attackers will be able to generate alternate images that will pass secure boot. If this key is ever lost, the OEM will be unable to update the image.

6.1.3.1.1 gen_keys

This utility generates a RSA public and private key pair using OPENSSL APIs. The key pair consists of 3 parts; N, E, and D.

N – Modulus

E – Encryption exponent

D – Decryption exponent

Public Key - It is a combination of E and N components.

Private Key - It is a combination of D and N components.

It is the OEM's responsibility to tightly control access to the RSA private signature key. If this key is ever exposed, attackers will be able to generate alternate images that will pass secure boot. If this key is ever lost, the OEM will be unable to update the image.

Features

- The application allows the user to generate 3 sizes keys. The key sizes are 1024 bits, 2048 bits, and 4096 bits.
- It generates RSA key pairs in PEM format.
- Keys are generated and stored in the files. User can provide file names through command line option.

Usage

`./gen_keys [OPTION] SIZE`

SIZE refers to size of public key in bits. (Modulus size).

Size supported -- 1024, 2048, 4096. The generated keys would be in PEM format.

Options:

`-h,--help` Usage of the command

`-k,--pubkey` File where Public key would be stored in PEM format (default = `srk.pub`)

`-p,--privkey` File where Private key would be stored in PEM format (default = `srk.priv`)

Usage Example

```
$ ./gen_keys 1024

#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#

=====
This product includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit (http://www.openssl.org/)
This product includes cryptographic software written by
Eric Young (eay@cryptsoft.com)
=====

Generated SRK pair stored in :
    PUBLIC KEY srk.pub
    PRIVATE KEY srk.priv
```

```
$ ./gen_keys 4096 -k my.pub -p my.pri

#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#

=====
This product includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit (http://www.openssl.org/)
This product includes cryptographic software written by
Eric Young (eay@cryptsoft.com)
=====
```

```
Generated SRK pair stored in :
PUBLIC KEY my.pub
PRIVATE KEY my.pri
```

6.1.3.1.2 gen_otpmk_drbg

This utility in the Code Signing Tool inserts hamming code in a user defined 256b hexadecimal string, or generate a 256b hexadecimal random number and inserts the hamming code in it which can be used as OTPMK value.

NOTE

For random number generation, Hash_DRBG library is used. The Hash_DRBG is an implementation of the NIST approved DRBG (Deterministic Random Bit Generator), specified in SP800-90A. The entropy source is the Linux /dev/random.

Features:

- Generates random numbers, which can be used if user defined string is not provided, to generate OTPMK value.
- Calculates and embeds the hamming code in the hexadecimal string.

Usage:

```
./gen_otpmk_drbg -b <bit_order> --s [string]
```

<bit_order> : (1 or 2) OTPMK Bit Ordering Scheme in SFP

1 : TA1.x

2 : TA2.x, TA3.x

<string> : 32 byte string

In case string is not specified, the utility generates a 32 bytes random number and embeds hamming code in it.

Usage Example:

```
$ gen_otpmk_drbg -b 1
```

```
#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

```
Input string not provided
Generating a random string
```

```
* Hash_DRBG library invoked
* Seed being taken from /dev/urandom
```

```
OTPMK[255:0] is:
d2f63a662f69a1faa4c2406f83eedde7647fbd3c62ac442c67fad2d4cda8b3a0
```

NAME	BITS	VALUE
OTPMKR 0	31- 0	cda8b3a0
OTPMKR 1	63- 32	67fad2d4
OTPMKR 2	95- 64	62ac442c
OTPMKR 3	127- 96	647fbd3c
OTPMKR 4	159-128	83eedde7
OTPMKR 5	191-160	a4c2406f

```
OTPMKR 6 | 223-192 | 2f69a1fa
OTPMKR 7 | 255-224 | d2f63a66
```

```
$ ./gen_otpmk_drbg -b 2 --s 111111112222222333333334444444455555556666666777777788888888
```

```
#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

```
OTPMK[255:0] is:
1111111122222223333333444444455555556666666777777788888888
```

NAME	BITS	VALUE
OTPMKR 0	255-224	11111111
OTPMKR 1	223-192	22222222
OTPMKR 2	191-160	33333333
OTPMKR 3	159-128	44444444
OTPMKR 4	127- 96	55555555
OTPMKR 5	95- 64	66666666
OTPMKR 6	63- 32	77777777
OTPMKR 7	31- 0	88888888

6.1.3.1.3 gen_drv_drbg

This utility in the Code Signing Tool inserts hamming code in a user defined 64b hexadecimal string, or generate a 64b hexadecimal random number and inserts the hamming code in it which can be used as Debug Response Value.

NOTE

For random number generation, Hash_DRBG library is used. The Hash_DRBG is an implementation of the NIST approved DRBG (Deterministic Random Bit Generator), specified in SP800-90A. The entropy source is the Linux /dev/random.

Features:

- Generates random numbers, which can be used if user defined string is not provided, to generate Debug Response value.
- Calculates and embeds the hamming code in the hexadecimal string.

Usage:

```
./gen_drv_drbg <Hamming_algo> [string]
```

Hamming_algo : Platforms

A1 : T10xx, T20xx, T4xxx, P4080rev1, B4xxx

A2 : LSx

B : P10xx, P20xx, P30xx, P4080rev2, P4080rev3, P50xx, BSC913x, C29x

string : 8 byte string

In case string is not specified, the utility generates an 8 byte random number and embeds hamming code in it.

Usage Example:

```
$ ./gen_drv_drbg A2

#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

Input string not provided
Generating a random string

```
-----
* Hash_DRBG library invoked
* Seed being taken from /dev/random
-----
```

Random Key Generated is:

f4bfc65e16284dbb

DRV[63:0] after Hamming Code is:

f4bfc65f16294daf

NAME	BITS	VALUE
DRV 0	63 - 32	f4bfc65f
DRV 1	31 - 0	16294daf

```
$ ./gen_drv_drbg A2 1652afe595631dec
```

```
#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

DRV[63:0] after Hamming Code is:

1652afe495631cea

NAME	BITS	VALUE
DRV 0	63 - 32	1652afe4
DRV 1	31 - 0	95631cea

6.1.3.2 Header creation

6.1.3.2.1 uni_pbi

Following options are available with the uni_pbi command.

```
$ ./uni_pbi
--verbose    Display header Info after Creation. This option is invalid for TA2 platform
--hash       Print the SRK(Public key) hash. This option is invalid for TA2 platform
--img_hash   Header is generated without Signature.
              Image Hash is stored in a separate file. This option is invalid for TA2
platform
--help       Show the Help for Tool Usage.
```

The input to this tool will be an input file specifying the platform. Based on that, there are two separate behaviour of the tool.

uni_pbi for TA2.x platforms is used for the following:

- To add boot location pointer and set SB_EN and BOOT_HO value for secure boot
- (optional) To add PBI commands (ACS write commands to add U-Boot spl and its header to OCRAM from Non-XIP memory).
- (optional) To append images (U-Boot, Boot script, and their headers) to RCW file.

Refer [Hardware Pre-Boot Loader \(PBL\) based platforms](#) on page 96 for TA2.x based platforms

uni_pbi for Service processor based platforms

- uni_pbi tool is used for creating signature and header over PBI commands.

Table 41. Description of fields in input files of both type of platforms (TA2.x and TA3.x)

Field name	Description	Platform supported
PLATFORM	The platform for which tool is being used	TA 2.x and TA 3.x
RCW_PBI_FILENAME	Input image file name. The rcw file which has to be modified.	TA 2.x and TA 3.x
BOOT1_PTR	Address of ISBC (Boot1) CSF Header	TA 2.x and TA 3.x
OUTPUT_RCW_PBI_FILENAME	To identify the platform for which the tool is being used. This field is optional. If not specified, it will take default name.	TA 2.x
BOOT_SRC	Only to be specified in case of SD boot	TA 2.x
SB_EN	Field to enable or disable secure boot, by setting SB_EN bit in rcw file to 1	TA 2.x
BOOT_HO	To put core in hold-off state to fuse key hash in case of secure boot, by setting BOOT_HO bit in rcw file to 1	TA 2.x
COPY_CMD	To add ACS write commands to write U-Boot spl and is header to OCRAM. This is an optional field. If not mentioned, won't add the command.	TA 2.x
APPEND_IMAGES	To append U-Boot, Boot script, and their headers to the new rcw generated. It is an optional field. This is an optional field, if not specified, no images will be appended.	TA 2.x
KEY_SELECT	Specify the key to be used in signature generation from the SRK table	TA 3.x
PRI_KEY	Private key file name in PEM format. The maximum keys supported are 8.	TA 3.x
FSL_UID_x	FSL UID(s) to be populated in the header	TA 3.x
OEM_UID_x	OEM UID(s) to be populated in the header	TA 3.x
OUTPUT_HDR_FILENAME	Output file name of the header. An output file name is generated with rcw commands appended with signed PBI commands.	TA 3.x
IMAGE_HASH_FILENAME	used with '--img_hash' option (Name of file in which Image Hash is stored)	TA 3.x
MP_FLAG	Manufacturing Protection Flag	TA 3.x
ISS_FLAG	Increment Security State Flag	TA 3.x
Table continues on the next page...		

Table 41. Description of fields in input files of both type of platforms (TA2.x and TA3.x) (continued)

Field name	Description	Platform supported
LW_FLAG	Leave Writeable Flag	TA 3.x
VERBOSE	Specify VERBOSE as 1, if you want to display header information. This can also be done with '--verbose' option	TA 3.x
IE_TABLE_ADDR	64-bit address of IE table(to be used in case of IE key extension feature usage)	TA 3.x

Sample input files are present in the CST tool at location: input_files/uni_pbi/<platform>/

For example, input_files/uni_pbi/ls1/input_pbi_sd_secure

NOTE

In TA 3.x, SB_EN and BOOT_HO fields are by default set to 1 to enable secure boot.

NOTE

TA 3.x : LS1088, LS2088. To know platforms under TA 2.x, refer [Trust Architecture and SFP Information](#) on page 126

6.1.3.2.1.1 Sample Input File

Sample input file for TA2 based platforms

```

/*
 * Copyright 2016 NXP
 */
-----
# For PBI Creation
# Name of RCW + PBI file [Mandatory]
RCW_PBI_FILENAME= u-boot-with-spl-pbl.bin
-----
# Specify the output file name [Optional].
# Default Values chosen in Tool
OUTPUT_RCW_PBI_FILENAME=u-boot-with-spl-pbl-sec.bin
-----
#specify the boot src
BOOT_SRC=SD_BOOT
# Specify the platform
PLATFORM=LS1020
# Specify the RCW Fields. (0 or 1) - [Optional]
SB_EN=1
BOOT_HO=1
BOOT1_PTR=10016000
-----
# Specify the PBI commands - [Optional]
# Argument: COPY_CMD = (src_offset, dest_offset, Image name)
# Split hdr_uboot_spl.out in PBI commads
COPY_CMD={ffffffff,10016000,hdr_uboot_spl.out;}
-----
# Specify the Images to be appended
# Arguments: APPEND_IMAGES=(Image name, Offset from start)
APPEND_IMAGES={u-boot-dtb.bin,00022000;}
APPEND_IMAGES={hdr_uboot.out,00122000;}
APPEND_IMAGES={hdr_bs.out, 00124000;}

```

```
APPEND_IMAGES={bootscript,00128000;}
```

Sample input file for SP based platforms

```
-----  
# Specify the platform. [Mandatory]  
# Choose Platform -  
# TRUST 3.1: LS2088, LS1088  
PLATFORM=LS1088  
-----  
# Specify the Key Information.  
# PUB_KEY [Mandatory] Comma Separated List  
# Usage: <srk1.pub>, <srk2.pub> .....  
PUB_KEY=srk.pub  
# KEY_SELECT [Mandatory]  
# USAGE (for TRUST 3.1): (between 1 to 8)  
KEY_SELECT=1  
# PRI_KEY [Mandatory] Comma Separated List for Signing  
# USAGE: <srk.pri>, <srk2.pri>  
PRI_KEY=srk.pri  
-----  
# For PBI Signing  
# Name of RCW + PBI file [Mandatory]  
RCW_PBI_FILENAME=rcw.bin  
# Address of ISBC (Boot1) CSF Header [Mandatory]  
BOOT1_PTR=20c00000  
-----  
# Specify OEM AND FSL ID to be populated in header. [Optional]  
# e.g FSL_UID_0=11111111  
FSL_UID_0=  
FSL_UID_1=  
OEM_UID_0=  
OEM_UID_1=  
OEM_UID_2=  
OEM_UID_3=  
OEM_UID_4=  
-----  
# Specify the output file names [Optional].  
# Default Values chosen in Tool  
OUTPUT_HDR_FILENAME=rcw_sec.bin  
IMAGE_HASH_FILENAME=  
-----  
# Specify The Flags. (0 or 1) - [Optional]  
MP_FLAG=0  
ISS_FLAG=1  
LW_FLAG=0  
-----  
# Specify VERBOSE as 1, if you want to Display Header Information [Optional]  
VERBOSE=0
```

6.1.3.2.2 uni_sign

uni_sign tool can be used for the following functions.

- CSF header generation along with signature for both ISBC and ESBC phase
- CSF header generation without signature if private key is not provided

uni_sign tool (with ESBC = 0 in input file) is used for creating signature and header over Boot1 image to be verified by ISBC

uni_sign tool (with ESBC = 1 in input file) is used for creating signature and header over images to be verified by ESBC

Following options are available with the uni_sign command.

Usage:

To view usage of tool:

```
./uni_sign
--verbose      Display header Info after Creation
--hash         Print the SRK(Public key) hash
--img_hash     Header is generated without Signature. Image Hash is stored in a separate file
--out <file>   Header file name
--in <file>    Input file for signature calculation. This option would override the filename in
IMAGE_1 in input_file if present
--app <file>   File to be appended to the header
--app_off <offset> Offset at which file will be appended to the header
--help        Show the Help for Tool Usage
```

For example:

```
./uni_sign --in <inp_file> --out <op file> --app_off <offset> --app <file> <input_file>
```

NOTE

There are scenarios when a build script using the tool needs to modify the input file name or the output header file name. These command line options provide a way to override the values as specified in the input file.

Table 42. Description of fields

Field	Field description	Platform supported
PLATFORM	To identify the platform/SoC for which CF header needs to be created.	All
ESBC	Do not set this flag when code signing is being performed on the image directly verified by the ISBC. For later images in the chain of trust, set this flag.	TA3.x
ENTRY_POINT	Entry point address or Image start address field in the header.	All
PRI_KEY	Private key file name to be used for signing the image. (File has to be in PEM format) (default = srk.pri generated by gen_keys command) FILE1 [,FILE2, FILE3, FILE4]. Multiple key support for Trust Arch v2.x devices only.	All
PUB_KEY	Public key file name in PEM format. (default = srk.pub generated by gen_keys) FILE1 [,FILE2, FILE3, FILE4]. Multiple key support for Trust Arch v2.x devices only.	All
KEY_SELECT	Specify the key to be used in signature generation when more than one key has been given as input. (Default=1, first key will be selected)	All
IMAGE_1 - IMAGE_8	Create Entries for SG table in the format { IMAGE_NAME, SRC_ADDR, DST_ADDR }	All
Table continues on the next page...		

Table 42. Description of fields (continued)

Field	Field description	Platform supported
OEM_UID_x	OEM UID to be populated in the header.	All
FSL_UID_x	FSL UID to be populated in the header.	All
HK_AREA_POINTER	House Keeping Area Starting Pointer required by Sec (Required for Trust Arch v2.x devices only when esbc option is not provided)	TA2.x
HKAREA_SIZE	House Keeping Area Size (Required for Trust Arch v2.x devices only when esbc option is not provided)	TA2.x
OUTPUT_HDR_FILENAME	Name of the combined header binary to be created by tool	All
SG_TABLE_ADDR	Specify SG_TABLE Address where Scatter Gather table is present for 2041/3041/4080/5020/5040 when ESBC=0.	TA1.x
OUTPUT_SG_BIN	Specify the output file name of sg table.	TA1.x
IMAGE_TARGET	Specify the target where image will be loaded. For example,NOR_8B/NOR_16B/NAND_8B_512/NAND_8B_2K/NAND_8B_4K/ NAND_16B_512/ NAND_16B_2K/NAND_16B_4K/SD/MMC/SPI	All
SEC_IMG	Flag for Secondary Image. Required for Trust Arch v2.x devices only	TA2.x
MP_FLAG	Specify Manufacturing Protection Flag. Available for LS1 only.	All, only needed in ISBC phase
VERBOSE	Specify Verbose option. Contents of header generated will be printed.	All
IMAGE_HASH_FILENAME	used with '--img_hash' option (Name of file in which Image Hash is stored)	TA3.x
ISS_FLAG	Increment Security State Flag	TA3.x, only needed in ISBC phase
LW_FLAG	Leave Writeable Flag	TA3.x, only needed in ISBC phase
ESBC_HDRADDR	32-bit address where header generated will be placed. Used to calculate IE key table address	TA3.x, only to be used in case of IE key extension feature usage
IE_KEY	Comma separated list of files containing public keys(IE Keys)	TA3.x, only to be used in case of IE key extension feature usage
Table continues on the next page...		

Table 42. Description of fields (continued)

Field	Field description	Platform supported
IE_REVOC	Comma separated list of numbers that are to be revoked from IE table	TA3.x, only to be used in case of IE key extension feature usage
IE_KEY_SEL	No. of keys in IE table that is to be used to validate image	TA3.x, only to be used in case of IE key extension feature usage

Sample input files can be referred to, from input_files/uni_sign/l<platform>

For IE keys, you can refer to input_files/uni_sign/l<platform>/ie_ke

TA3.x: LS2088 and LS1088. To know platforms under TA1.x and TA 2.x, refer [Trust Architecture and SFP Information](#) on page 126

6.1.3.2.2.1 Sample Input File

The input files will not have ESBC field (ESBC=0).

```

-----
# Specify the platform. [Mandatory]
# Choose Platform -
# TRUST 3.1: LS2088, LS1088
# TRUST 1.0, 1.1, 2.0, 2.1: 1010/1040/2041/3041/4080/5020/5040/9131/9132/9164/4240/C290/LS1
PLATFORM=LS2088
-----
# Entry Point/Image start address field in the header. [Mandatory]
# (default=ADDRESS of first file specified in images)
# Address can be 64 bit
ENTRY_POINT=30008000
-----
# Specify the Key Information.
# PUB_KEY [Mandatory] Comma Separated List
# Usage: <srk1.pub> <srk2.pub> .....
PUB_KEY=srk.pub
# KEY_SELECT [Mandatory]
# USAGE (for TRUST 3.1): (between 1 to 8)
KEY_SELECT=1
# PRI_KEY [Mandatory] Comma Separated List for Signing
# USAGE: <srk.pri>, <srk2.pri>
PRI_KEY=srk.pri
-----
# Specify IMAGE, Max 8 images are possible.
# DST_ADDR is required only for Non-PBL Platform. [Mandatory]
# USAGE : IMAGE_NO = {IMAGE_NAME, SRC_ADDR, DST_ADDR}
# Address can be 64 bit
IMAGE_1={u-boot.bin,30008000,ffffffff}
IMAGE_2={,,}
IMAGE_3={,,}
IMAGE_4={,,}
IMAGE_5={,,}
IMAGE_6={,,}

```

```
IMAGE_7={,,}  
IMAGE_8={,,}  
-----  
# Specify OEM AND FSL ID to be populated in header. [Optional]  
# e.g FSL_UID_0=11111111  
FSL_UID_0=  
FSL_UID_1=  
OEM_UID_0=  
OEM_UID_1=  
OEM_UID_2=  
OEM_UID_3=  
OEM_UID_4=  
-----  
# Specify the output file names [Optional].  
# Default Values chosen in Tool  
OUTPUT_HDR_FILENAME=hdr_uboot.out  
IMAGE_HASH_FILENAME=  
RSA_SIGN_FILENAME=  
-----  
# Specify The Flags. (0 or 1) - [Optional]  
MP_FLAG=0  
ISS_FLAG=1  
LW_FLAG=0  
-----  
# Specify VERBOSE as 1, if you want to Display Header Information [Optional]  
VERBOSE=0  
-----  
# Following fields are Required for 4240/9164/1040/C290 only  
  
# Specify House keeping Area  
# Required for 4240/9164/1040/C290 only when ESBC flag is not set. [Mandatory]  
HK_AREA_POINTER=  
HK_AREA_SIZE=  
-----  
# Following field Required for 4240/9164/1040/C290 only  
# Specify Secondary Image Flag. (0 or 1) - [Optional]  
# (Default is 0)  
SEC_IMAGE=  
-----  
# Specify SG table address, only for (2041/3041/4080/5020/5040) with ESBC=0 - [Optional]  
SG_TABLE_ADDR=
```

6.1.3.3 Signature generation

The tools in this category are provided in case the user does not want to share the Private Key with the CST tool. The **--img_hash** option in [Header creation](#) on page 169 tools provides OEMs with the ability to perform code signing in a secure environment which does not run the NXP Code Signing Tool.

--img_hash option

- Generates hash file in binary format which contains SHA256 hash of the components required for signature.
- Generates output header binary file based on the fields specified in input file.
- Output header binary file does not contain signature.

- Provides flexibility to manually append signature at the end of output header file. Users can use their own custom tool to generate the signature. The signature offset chosen in the header is such that the signature can be appended at the end of the header file.
- This option does not require private key to be provided. But the corresponding public key from the public/ private key pair must be provided to calculate correct SHA256 hash.
- The SHA256 hash generated over CF header (in case of TA1.x platforms)) is then signed using RSA algorithm (OPENSSL APIs) with the private key. This encrypted hash is known as digital signature. This signature is placed at an offset from the CF header, which is later read by IBR.
- The SHA256 hash generated over CSF header, the public Key, the S/G table and the ESBC is also signed using RSA algorithm with the same private key. The signature generated is placed at an offset from the CSF header, which is again later read by IBR.

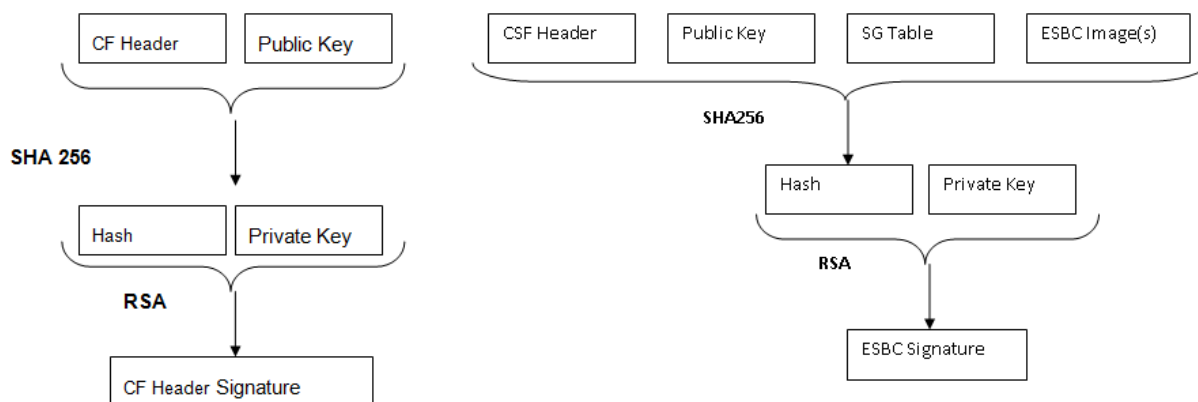


Figure 23. Dual signature generation

Usage example

```
$ ./uni_sign --img_hash --verbose input_files/uni_sign/<platform>/input_uboot_nor_secure
```

```
#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

```
=====
This tool includes software developed by OpenSSL Project
for use in the OpenSSL Toolkit (http://www.openssl.org/)
This product includes cryptographic software written by
Eric Young (eay@cryptsoft.com)
=====
```

```
Input File is input_files/uni_sign/<platform>/input_uboot_nor_secure
```

```
-----
- Dumping the Header Fields
-----
```

```
- SRK Information
- SRK Offset : 200
- Number of Keys : 1
- Key Select : 1
```

```
- Key List :
- Key1 srk.pub(100)
- UID Information
- UID Flags = 00
- FSL UID = 00000000_00000000
- OEM UID0 = 00000000
- OEM UID1 = 00000000
- OEM UID2 = 00000000
- OEM UID3 = 00000000
- OEM UID4 = 00000000
- FLAGS Information
- MISC Flags = 60
- ISS = 1
- MP = 0
- LW = 0
- B01 = 1
- Image Information
- SG Table Offset : 800
- Number of entries : 1
- Entry Point : 30008000
- Entry 1 : u-boot.bin (Size = 000c0000 SRC = 30008000 DST = ffffffff)
- RSA Signature Information
- RSA Offset : a00
- RSA Size : 80
```

Image Hash:

8588c174dd92f4a1b114b9029fc647e18cac4aaa46f03a6538ef20531e796e8f

* Image Hash Stored in File: hash.out

* Header File is w/o Signature appended

Header File Created: hdr_uboot.out

SRK (Public Key) Hash:

7df50d4256c4cbde4ef4ae9931042b1e44ff13aeb5107a7e0e9ee07e0fbfc236

SFP SRKHR0 = 7df50d42

SFP SRKHR1 = 56c4cbde

SFP SRKHR2 = 4ef4ae99

SFP SRKHR3 = 31042b1e

SFP SRKHR4 = 44ff13ae

SFP SRKHR5 = b5107a7e

SFP SRKHR6 = 0e9ee07e

SFP SRKHR7 = 0fbfc236

The tools are provided to create the signature file and embed the signature at the end of header file.

6.1.3.3.1 gen_sign

This tool is provided for the user to calculate signature for a given hash using CST tool. The tool requires only the hash file and private key file from the user as input. It would generate signature file as output.

It uses RSA_sign API of openssl to calculate signature over hash provided.

Usage

./gen_sign [option] <HASH_FILE> <PRIV_KEY_FILE>

--sign_file SIGN_FILE Provides file name for signature to be generated as operand. SIGN_FILE is generated containing signature calculated over hash provided through HASH_FILE using private key provided through PRIV_KEY_FILE. With this option, HASH_FILE and PRIV_KEY_FILE are compulsory while SIGN_FILE is optional. The default value of SIGN_FILE is signout.

HASH_FILE Name of hash file containing hash over signature needs to be calculated.

PRIV_KEY_FILE Name of key file containing private key.

Usage example

After the hash file has been created as described in [Signature generation](#) on page 176, the tool can be used as described below.

```
$ ./uni_sign --img_hash input_files/uni_sign/<platform>/input_uboot_nor_secure
.
.
.

*****
* Image Hash Stored in File: hash.out
* Header File is w/o Signature appended
*****

Header File Created: hdr_uboot.out

$ ./gen_sign hash.out srk.pri

#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#

Signature Length = 80
Hash in hash.out is signed with srk.pri
Signature is stored in file : sign.out
```

6.1.3.3.2 sign_embed

This tool embeds signature in the header file generated using img_hash option which generates header but does not embed signature in the header. This option opens header file and copies signature at the end of the file.

The header file generated with 'img_hash' option has padding added till signature offset, so that signature can be directly embedded to the end of the file.

Usage

`./sign_embed <hdr_file> <sign_file>`

hdr_file Name of header file in which signature needs to be embedded

sign_file Name of sign file containing signature which needs to be embedded

Usage example

```
$ ./sign_embed hdr_uboot.out sign.out

#-----#
```

```
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

```
hdr_uboot.out is appended with file sign.out (0x80)
```

NOTE

User can generate the complete header along with signature in a single step using uni_sign/uni_pbi tool without any option.

```
./uni_sign <input_file>
```

Or

User may wish to do it in three separate steps:

1. `./uni_sign --img_hash <input_file>` (Create header file without signature and store the hash in a separate file)
2. `./gen_sign[10] [option] <HASH_FILE> <PRIV_KEY_FILE>` (Sign the image hash using private key)
3. `./sign_embed <hdr_file> <sign_file>` (Embed the signature at the end of header file)

6.2 IMA-EVM on LS Trust Architecture SoCs

6.2.1 Introduction

Layerscape trust architecture supports image validation using secure boot feature. Secure boot performs image validation in various stages and kernel image is the last one to get verified in chain of trust. Linux-based IMA-EVM feature helps in extending the chain of trust (COT) until rootfs installed over persistent storage devices.

Integrity Measurement Architecture (IMA): is the linux integrity subsystem used to detect if files have been accidentally or maliciously altered. It appraises a file's measurement against a "good" value stored as an extended attribute (security.ima) and enforces local file integrity checks. The extended attribute (security.ima) of a file is the hash value (SHA-1, SHA-256 or SHA-512) of its content. IMA maintains a list of hash values over all executables and other sensitive system files loaded at runtime into the system.

Extended Verification Module (EVM): protects a file's extended attributes against integrity attacks. The extended security attribute (security.evm) stores the HMAC value over other extended attributes associated with the file such as security.selinux, security.SMACK64, security.ima.

EVM depends on the kernel key retention system and requires an encrypted key named evm-key for the HMAC operation. The key is loaded onto the root user keyring using keyctl utility. EVM is enabled by setting an enable flag in securityfs/evm file.

[10] This may be done by user's own tool in case he does not want to share the private key with the CST tool.

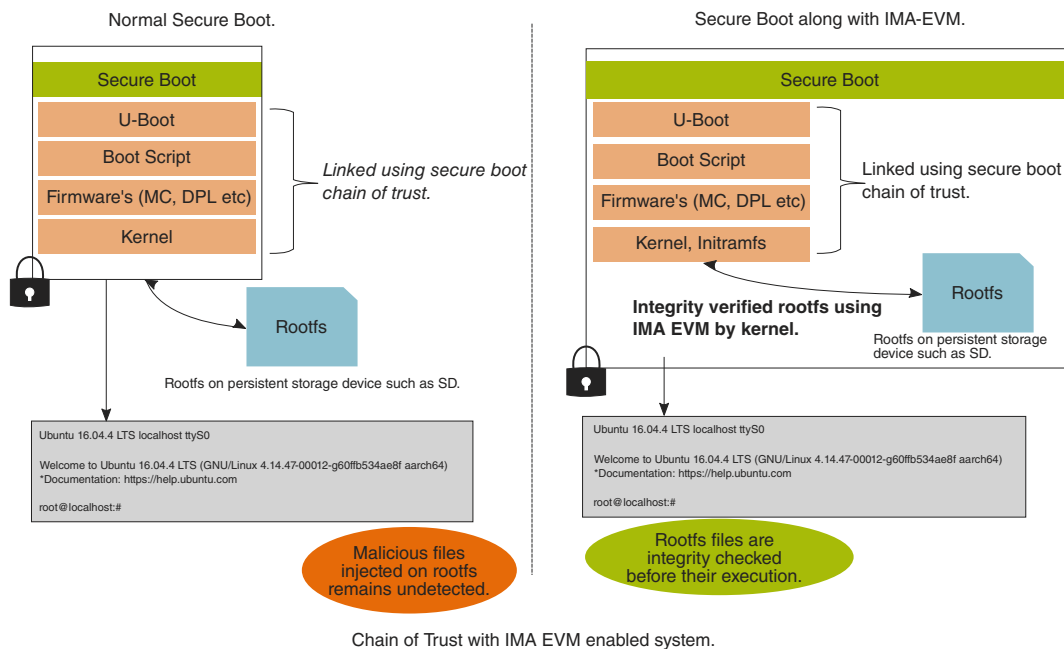


Figure 24. Chain of Trust with IMA EVM enabled system

In normal secure boot process, contents of root file system mounted over persistent storage device are not validated by any mechanism and hence cannot be trusted. Any malicious changes in non-trusted rootfs contents are undetected. IMA EVM is the linux standard mechanism to verify the integrity of the rootfs. Integrity checks over file attributes and its contents are performed by linux IMA EVM module before its execution. IMA EVM depends on encrypted key loaded on user's keyring. Loading keys to root user keyring and enabling EVM is typically done using initramfs image. The initramfs image is validated using secure boot process and becomes the part of chain of trust. Initramfs switches control to main rootfs mounted over storage device, after EVM is successfully enabled on the system.

For secure boot process see Secure boot process.

6.2.2 Secure key and its blob

Secure key is generated using CAAM security engine which constitutes of random bytes. The key contents are stored in kernel space and not visible to user. User space will only be able to see the key blob.

Blobs are special data structures for holding encrypted data, along with an encrypted version of the key used to decrypt the data. Typically, blobs are used to hold data to be stored in external storage (such as flash memory or in an external file system), making the contents of a blob a semi-persistent secret. The secrecy of the blob depends on the device-specific 256-bit master key, which is derived from the OTMPK or ZMK on Layerscape trust architecture-based SoCs.

Secure key is loaded on the user keyring by the kernel keys framework and its blob is stored on the rootfs to be used during next bootup.

6.2.3 EVM Key on user keyrings

The EVM security attribute depends on an encrypted key (named evm-key) loaded on the user keyring. The encrypted key is derived by the kernel using the master key. The master key can be of following types:

- User-Key
- Secure-Key
- Trusted-Key

Secure and trusted keys are derived using a hardware security engine for greater security while the security of user-key depends upon the user-defined mechanisms irrespective of the hardware. The secure-key is derived using the Layerscape's SEC (aka CAAM). The trusted-key can be used on the platforms supporting TPM.

The encrypted key acts as an HMAC key, which is subsequently used to calculate the HMAC value (security.evm) over other security attributes. This key is stored internally by the kernel and user can only see its blob.

6.2.4 Enabling secure key and encrypted key in kernel config

The secure key is dependent on the Layerscape SoCs SEC (also known as: CAAM) hardware. To enable the secure key following SEC related configs needs to be enabled under **Hardware crypto devices** config option. See below specified hierarchy for the config options:

```
Cryptographic API
|___Hardware crypto devices
|   |___Freyscale CAAM-Multicore platform driver backend
|   |___Freyscale CAAM Job Ring driver backend
|   |___Register algorithm implementation with Crypto API
|   |___Queue Interface as Crypto API backend
|   |___Register hash algorithm implementations with Crypto API
|   |___Register CAAM device for hwrng API
```

After configuring basic CAAM options, enable the secure key and the encrypted key under Security options.

```
Security options
|___SECURE_KEYS
|___ENCRYPTED_KEYS
```

NOTE

1. To enable any option press Y.
2. If any option is enabled by default, there is no need to press Y.

6.2.5 Enabling IMA EVM integrity options in kernel image

Enable IMA EVM integrity options under Security options as shown below:

```
Security options
|___Integrity subsystem
|___Enables integrity auditing support
|___Integrity Measurement architecture (IMA)
|___Appraise integrity measurements
|___EVM support
|___FSUID (version 2)
```

6.2.6 Modes of operation in IMA EVM

IMA EVM is enabled in two modes: fix mode, enforce mode. To enable a system with IMA EVM, both modes must be implemented in a sequence as described below.

1. First, the system needs to be booted in fix mode with `ima_appraise=fix` and `evm=fix` bootargs. After loading the keys on the root keyring, the entire file system is labelled with security attributes. In fix mode any file with `INTEGRITY_UNKNOWN` is labelled with proper attribute values. This mode must be executed only once while preparing system for field deployment.

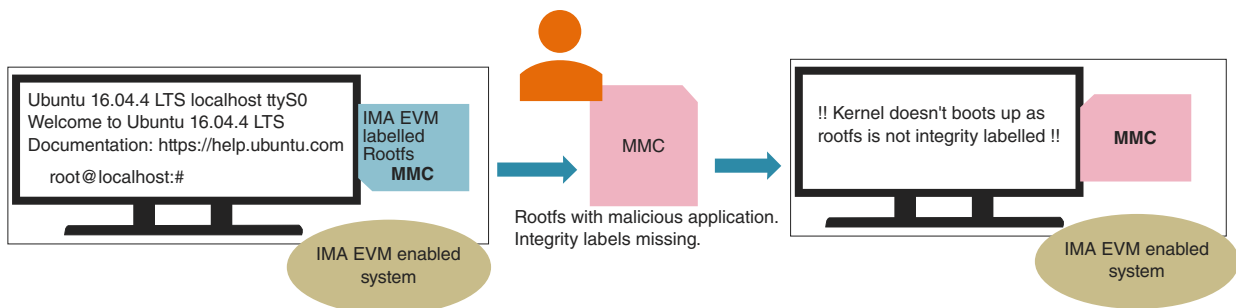
- Second, after the fix mode execution is completed successfully, system needs to be booted IMA EVM in enforce mode. Enforce mode is enabled by setting `ima_appraise=enforce` bootargs. In enforce mode the files are measured against their “good” values. In case there is a mismatch between calculated security attribute value and stored value, access to that file is denied. While in field the system boot is done in enforce mode only.

6.2.7 IMA EVM feature use cases

IMA EVM prevents system from offline attack

Scenario: If attacker tries to boot the system using the rootfs mounted over persistent storage device for which the files are wrongly labelled or not labelled with security labels (such as `security.ima`, `security.evm`).

System behavior: In enforce mode, the IMA EVM labelled kernel stops booting due to mismatch in calculated security attribute values and labelled values.



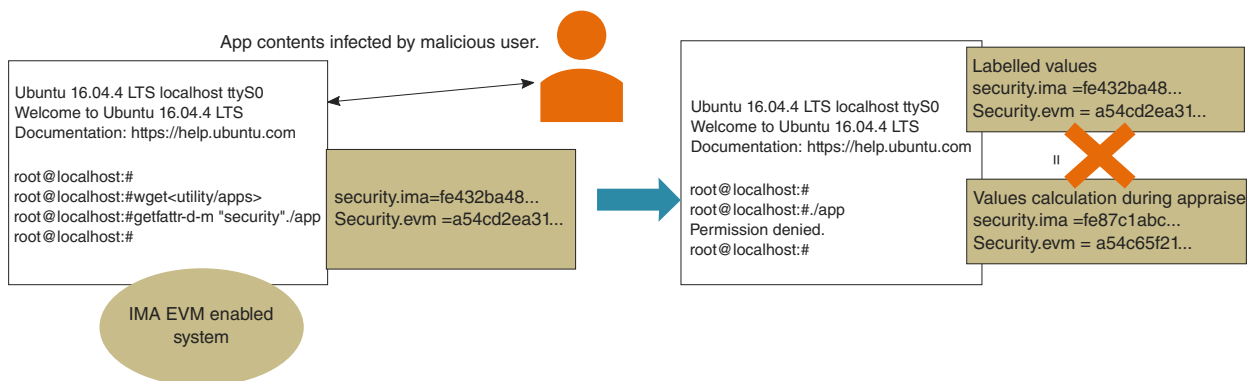
IMA EVM protect the system from offline attacks. System will not boot up if the files over rootfs are wrongly labelled or not labelled with security attributes.

Figure 25. IMA EVM protects system from offline attack

IMA EVM protects system against malicious file changes

Scenario: Any application loaded by the root user is changed maliciously. The labelled attribute values will be different from the calculated values on the next access.

System behavior: On next access, the file will not be accessible due to different security attribute values.



Any app content changed by the malicious user would disturb the ima and evm attribute values. On next access the app will not be accessible

Figure 26. IMA EVM protect system from malicious file changes

NOTE

IMA EVM is independent of rootfs version i.e. it works with ubuntu rootfs 16.04 as well as ubuntu rootfs 18.04

6.2.8 Appendix A: Steps to enable IMA EVM using flex builder

To boot the SoC with the IMA EVM feature, various boot images can be generated using flex build. Secure boot process is mandatory to run the images generated using flex build.

Steps to generate images for IMA EVM along with secure boot:

1. Execute the commands below to enable configuration options for secure key and IMA-EVM feature in kernel as specified in [Enabling secure key and encrypted key in kernel config](#) on page 182 and [Enabling IMA EVM integrity options in kernel image](#) on page 182 respectively.

```
$ flex-builder -c linux:custom -a <architecture>
$ flex-builder -c linux -a <architecture>
```

2. Prepare initramfs images for arm32 or arm 64 architecture.

```
$ flex-builder -i mkrfs -r buildroot:imaevm -a <architecture>
```

3. Prepare distro boot script images for the desired platform,

```
$ flex-builder -i mkdistroscr -t
```

4. Sign the images (initramfs image, boot scripts and others) for secure boot process.

```
$ flex-builder -i signing -m <platform> -b <boottype> -t
```

5. Create the firmware image to be used:

```
$ flex-builder -i mkfw -m <platform> -b <boottype> -t
```

6. Build the boot partition tar ball:

```
$ flex-builder -i mkbootpartition -m <platform> -a <architecture> -t
```

7. Use the flex installer to program the boot partition and rootfs on the storage device.
8. System can be booted to run in secure mode.
9. On first boot up system will run IMA EVM in fix mode, during this mode the boot script in boot partition will be replaced with the enforce mode boot script. Upon first successful boot up reboot the system. Now on every bootup system will run in enforce mode.

NOTE

- Platform to be used: LS1088ARDB, LS1021ATWR, LS1046ARDB, LS2088ARDB. (LS2088ARDB Rev F does not support SD card.)
- Architecture to be used is: ARM32 or ARM64
- Boot type to be used is: qspi, sd, etc.
- -t flag is specified to indicate trust using IMA EVM.

6.2.9 Appendix B: Standalone steps to enable IMA EVM

To boot the system with IMA EVM, follow the steps below:

1. Clone the kernel (dash-lts) repository and checkout to the latest tag.
2. To compile the kernel set arch and path for cross-compilation:

```
$: export CROSS_COMPILE=<aarch64-toolchain>
$: export ARCH=arm64
```

3. Make the default config:

```
$: make defconfig lsdk.config
```

NOTE

Above command is for ARM64 platform, use the necessary defconfigs required to build the default config.

4. Enable the config options for secure key and IMA-EVM feature in kernel as mentioned in [Enabling secure key and encrypted key in kernel config](#) on page 182 and [Enabling IMA EVM integrity options in kernel image](#) on page 182 respectively.
5. Build the kernel image using make command:

```
$:make -j<no. of jobs>
```

6. Prepare the SD card, ext4 type partition (say mmcblk0p3) with Ubuntu rootfs image extracted over it.
7. Boot the board to U-Boot prompt.
8. Prepare the initramfs for specific architecture using flex build as mentioned in step 2 at [Appendix A: Steps to enable IMA EVM using flex builder](#) on page 184.
9. Using tftp or by some other means, place the initramfs image on DDR (say at location 0xa0000000).
10. Enable the IMA EVM in fix mode by adding the following boot arguments to present bootargs.

rootwait ima_tcb ima_appraise=fix ima_appraise_tcb evm=fix enforcing=0

```
$: setenv bootargs=" console=ttyS0,115200 root=/dev/mmcblk0p3 rw rootwait ima_tcb
ima_appraise=fix ima_appraise_tcb evm=fix enforcing=0
```

NOTE

- a. The default mount partition is 3rd partition over mmc device (mmcblk0p3).
- b. Set the mount option accordingly if ROOTFS is not mounted on mmcblk0p3 partition e.g.
mount="usb or sata or mmcblkpX"
- c. This step to be executed only once while preparing the board in factory for field deployment.

11. Run bootm command:

```
$: bootm <uImage_addr> <initramfs_img_addr> <platform_dtb_img_addr>
```

12. Once the system gets booted up, immediately perform reboot. This ensure the system is ready to be used in enforce mode.
13. To start with enforce mode, boot the system to U-Boot Prompt and repeat the steps 10 and 11.
14. Enable the IMA EVM in enforce mode by adding the following boot arguments to present bootargs.

" rootwait ima_tcb ima_appraise=enforce ima_appraise_tcb enforcing=1 "

```
$: setenv bootargs="console=ttyS0,115200 root=/dev/mmcblk0p3 rw rootwait
ima_tcb ima_appraise=enforce ima_appraise_tcb enforcing=1
```

NOTE

- The default mount partition is 3rd partition over mmc device (mmcblk0p3).
- Set the mount option accordingly if ROOTFS is not mounted on mmcblk0p3 partition e.g.
mount="usb or sata or mmcblkpX"

15. Run bootm command

```
$: bootm <uImage_addr> <initramfs_img_addr> <platform_dtb_img_addr>
```

16. System is booted up in enforce mode and ready with IMA EVM feature.

NOTE

Bootargs as mentioned in step 10 and 14 can be set using the boot script along with the secure boot chain.

6.2.10 Appendix C: Steps to verify IMA EVM feature

Perform the following checks to ensure that IMA EVM is successfully enabled in enforce mode.

1. Secure keys and encrypted keys are enabled in kernel image: The Following kernel logs ensures that secure key and encrypted key is successful registered.

```
[ 3.887255] Key type secure registered
[ 3.892526] Key type encrypted registered
```

2. IMA EVM is enabled in kernel image. The following kernel logs ensures IMA EVM is enabled.

```
[ 3.896537] ima: No TPM chip found, activating TPM-bypass! (rc=-19)
[ 3.902804] ima: Allocated hash algorithm: sha256
[ 3.907542] evm: HMAC attrs: 0x1
```

3. System is up in enforce mode. The following logs from initramfs and kernel ensures enforce mode is enabled.

```
[ 31.945199] EXT4-fs (mmcblk0p3): mounted filesystem with ordered data mode. Opts: (null)
Loading blobs
[ 31.965094] evm: key initialized
```

4. EVM attributes over a file can be checked using getfattr utility.

```
root@localhost# getfattr -d -m "security" /path/to/file
```

5. Security attribute are appraised successfully upon changing any file contents. Below steps will verify the appraise functionality.

```
root@localhost# vim test_file // wrtite contents "abc"
root@localhost# getfattr -d -m "security" /path/to/test_file
root@localhost# vim test_file // write contents "abcd"
root@localhost# getfattr -d -m "security" /path/to/test_file
```

NOTE

Security attributes values fetched in step d will be different from root@localhost# getfattr -d -m "security" /path/to/file.

6.2.11 Appendix - D: Points to remember

When preparing a system in fix mode, if labelling of the file system is interrupted (perhaps due to system power failure), remount the rootfs over the SD card partition before running fix mode again.

6.3 Trusted Execution (OP-TEE)

6.3.1 Introduction

Trusted Execution Environment (TEE), for ARM-based chips supporting TrustZone technology.

NXP Platforms are enabled with Open Portable TEE (OP-TEE), which is an open source project which contains a full implementation to make up a complete Trusted Execution Environment. This component meets the Global Platform TEE System Architecture specification. It also provides the TEE Internal core API v1.1 as defined by the Global Platform TEE Standard for the development of Trusted Applications.

OP-TEE consists of three components.

- OP-TEE Client, which is the client API running in normal world user space.
- OP-TEE Linux Kernel driver, which is the driver that handles the communication between normal world user space and secure world.
- OP-TEE Trusted OS, which is the Trusted OS running in secure world.

OP-TEE OS is made of 2 main components: the OP-TEE core and a collection of libraries designed for being used by Trusted Applications. While OP-TEE core executes in the ARM CPU privileged level (also referred to as 'kernel land'), the Trusted Applications execute in the non-privileged level (also referred to as the 'userland'). The static libraries provided by the OP-TEE OS enable Trusted Applications to call secure services executing at a more privileged level.

6.3.1.1 Support Platform

OP-TEE is supported on following NXP boards:

- LS1046A-RDB
- LS1043A-RDB
- LS2088A-RDB
- LS1088A-RDB
- LS1012A-RDB
- LS1012A-FRWW

6.3.1.2 Test Sequence

Execute the test sequence specified below on target machine:

On the target NXP board:

- To check if the OP-TEE kernel driver is successfully initialized (after successfully communicating with [OP-TEE OS](#) running in OP-TEE), look for the following in Linux boot logs:

```
optee: probing for conduit method from DT.
optee: initialized driver
```

Note: TF-A FIP image must be compiled with OP-TEE binary.

```
node. Else, an error appears : optee: api uid mismatch
```

- Now, run the tee-supplciant (binary generated from [optee_client](#) repo) binary

```
$>: tee-supplciant & (press enter).
```

- Run the **xtest**(binary generated from [optee_test](#) repo) app as follows:

```
$>: xtest -l 15 (press enter and look for the below logs to verify app runs successfully):  
47123 subtests of which 0 failed  
79 test cases of which 0 failed  
0 test case was skipped  
OP-TEE test application done!
```

6.4 Fuse Provisioning User Guide

6.4.1 Introduction

NXP SoC's Trust Architecture provides non-volatile secure storage in form of on-chip fuse memory. Following information can be programmed into fuse memory via Security Fuse Processor (SFP):

- One Time Programmable Master Key Registers (OTPMKRs)
- Super Root Key Hash Registers (SRKHRs)
- Debug Challenge and Response Value Registers (DCVRs & DRVRs)
- OEM Security Policy Registers (OSPRs)
- OEM Unique ID/Scratch Pad Registers (OUIDRs)

6.4.2 Fuse Programming Scenarios

Phase	NXP Fuses	OEM Fuses	Software
NXP Manufacturing	FUID, FSV, CSFF, WP (+ On Trust 3.0, DPL) Can set RT & RDPL up to 4x before shipping part)		Fuse programming done on tester, no software involved
Ship to contract manufacturer			
OEM Manufacturing (Can be split into two stages if required)	Minimal Fuse Provisioning	SRKH, DP, CSFF, ITS Minimal OTPMK & Optional OEM, DRV UIDs, DCV	Fuse Provisioning Tool Doesn't need to pass secure boot to execute, but must set up the system so that the next boot passes secure boot.
	At contract manufacturer or in the field		
	Final Fuse Provisioning	Final OTPMK & DRV, WP Optional OEM UIDs, DCV	Fuse Provisioning Tool pass secure boot.
In field, later in lifecycle			
Lifecycle fuse update		Key Revocation, Monotonic Counter Era, OEM Scratchpad, Field Return	Currently no software utility available. Can be done by custom app.

6.4.2.1 Fuse Provisioning during OEM Manufacturing

This stage may be split into two stages:

Stage 1 (Non-secure boot) – Minimal Fuse Provisioning

The following few fuses (Minimal Fuse File) programmed for secure boot to run:

- SRKH
- DP
- CSFF
- Minimal OTPMK
- ITS.

This stage does not pass secure boot to execute, but must set up the system so that the next boot passes secure boot. If this step happens in a trusted environment, OEM can choose to blow all the fuses in this stage itself.

Stage 2 (Secure Boot) – Final Fuse Provisioning

Rest of the fuses can be programmed after secure boot is up and running. This step ends with OEM WP fuse getting blown which renders most of the fuses as un- writable.

6.4.3 Fuse Provisioning Utility

Secure firmware provides support to do the fuse provisioning. By default, the support is enabled and requires a built in. Steps to do so using flex build are available in [Steps to build fuse provisioning firmware image](#) on page 193.

The information about the fuse values to be blown to be provided via a fuse file. The fuse file is a binary file with bits to indicate what fuses to be blown and their corresponding values.

CST provides an input file where user can enter the required values. Tool generates a Fuse file which is parsed in BL2 image to do fuse provisioning.

Secure firmware would have the required checks to determine if the provided input values are correct or not.

For example, OTPMK, SRKH etc. cannot be programmed when OEM_WP is already set in SFP fuses.

6.4.3.1 Fuse file structure

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																																																
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																
Word0	Bariker Code																																																																																
Word1	Flags	Reserved										OTPMK Flag				Reserved				DB				G _{LVL}				OU _{D4}				OU _{D3}				OU _{ID1}				OU _{ID0}				DR _{V1}				DR _{V0}				DC _{V1}				DC _{V0}				MC				SR _{KH}				SYS _{CF}				PO _{VD}									
Word2	GPIO Pin Numbe	GPIO Pin to be set for raising POV DD																																																																															
Word3	OTPMK0	OTPMK0																																																																															
Word4	OTPMK1	OTPMK1																																																																															
Word5	OTPMK2	OTPMK2																																																																															
Word6	OTPMK3	OTPMK3																																																																															
Word7	OTPMK4	OTPMK4																																																																															
Word8	OTPMK5	OTPMK5																																																																															
Word9	OTPMK6	OTPMK6																																																																															
Word10	OTPMK7	OTPMK7																																																																															
Word11	SRKH0	SRKH0																																																																															
Word12	SRKH1	SRKH1																																																																															
Word13	SRKH2	SRKH2																																																																															
Word14	SRKH3	SRKH3																																																																															
Word15	SRKH4	SRKH4																																																																															
Word16	SRKH5	SRKH5																																																																															
Word17	SRKH6	SRKH6																																																																															
Word18	SRKH7	SRKH7																																																																															
Word19	OEM UID0	OEM UID0																																																																															
Word20	OEM UID1	OEM UID1																																																																															
Word21	OEM UID2	OEM UID2																																																																															
Word22	OEM UID3	OEM UID3																																																																															
Word23	OEM UID4	OEM UID4																																																																															
Word24	DCV0	DCV0																																																																															
Word25	DCV1	DCV1																																																																															
Word26	DRV0	DRV0																																																																															
Word27	DRV1	DRV1																																																																															
Word28	OSPR1-Monoton	MC ERA (16b)																Reserved												DBGL Level																																																			
Word29	OSPR0-Setting of any bit in this field controlled by Sys Cfg field	FR				FR				Res								Res								K0				K1				K2				K3				K4				K5				K6				Res				NS				EC				IT				S				W				P			
Word30		Reserved																																																																															
Word31		Reserved																																																																															
	Bariker Code	Some value to indicate a valid fuse file																																																																															
	OTPMK Flags	0 0 0 0 Program Minimal Value 0 0 0 1 Program random OTPMK value 0 0 1 0 Program user supplied OTPMK value 0 0 1 0 Program random OTPMK value with pre-programmed minimal value 0 1 1 1 Program user supplied OTPMK value with pre-programmed minimal value 1 x x x Don't blow OTPMK																																																																															
	Other flags	1 Blow the fuses as per value indicated in the corresponding word 0 The corresponding fuse is not supposed to be blown																																																																															

6.4.3.2 Sample input file for fuse provisioning tool

```
-----
# Specify the platform. [Mandatory]
# Choose Platform - LS1/LS1043/LS1012/LS1046
PLATFORM=LS1046
-----
# GPIO Pin to be set for raising POVDD [Optional]
POVDD_GPIO=
-----
# One time programmable master key flags in binary form.[Mandatory]
# 0000 -> Program default minimal OTPMK value
# 0001 -> Program random OTPMK value
# 0010 -> Program user supplied OTPMK value
```

```

# 0101 -> Program random OTPMK value with pre-programmed minimal value
# 0110 -> Program user supplied OTPMK value with pre-programmed minimal value
# 1xxx -> Don't blow OTPMK
OTPMK_FLAGS=0000
# One time programmable master key value.
# [Optional dependent on flags, Mandatory in case OTPMK_FLAGS="0010" or "0110"]
OTPMK_0=
OTPMK_1=
OTPMK_2=
OTPMK_3=
OTPMK_4=
OTPMK_5=
OTPMK_6=
OTPMK_7=
-----
# Super root key hash [Optional]
SRKH_0=
SRKH_1=
SRKH_2=
SRKH_3=
SRKH_4=
SRKH_5=
SRKH_6=
SRKH_7=
-----
# Specify OEM UIDs. [Optional]
# e.g OEM_UID_0=11111111
OEM_UID_0=
OEM_UID_1=
OEM_UID_2=
OEM_UID_3=
OEM_UID_4=
-----
# Specify Debug challenge and response values. [Optional]
# e.g DCV_0=11111111
DCV_0=
DCV_1=
DRV_0=
DRV_1=
-----
# Specify Debug Level in binary form. [Optional]
# 000 -> Wide open: Debug portals are enabled unconditionally.
# 001 -> Conditionally open via challenge response, without notification.
# 01x -> Conditionally open via challenge response, with notification.
# 1xx -> Closed. All debug portals are disabled.
DBG_LVL=
-----
# System Configuration register bits in binary form [Optional]
# WP (OEM write protect)
# ITS (Intent to Secure)
# NSEC (Non secure)
# ZD (ZUC Disable)
# K0,K1,K2 (Key revocation bits)
# FR0 (Field return 0)
# FR1 (Field return 1)
WP=
ITS=
NSEC=
ZD=
K0=

```

```
K1=
K2=
FR0=
FR1=
-----
# Specify the output fuse provisioning file name. (Default: fuse_scr.bin) [Optional]
OUTPUT_FUSE_FILENAME=fuse_scr.bin
-----
```

6.4.4 Steps to build fuse provisioning firmware image

Use following Flexbuild commands to build composite fuse provisioning firmware image. For detailed info regarding usage of Flexbuild, refer to Layerscape SDK user guide.

1. Command to build Code Signing Tool (CST):

```
$:> flex-builder -c cst
```

Note: Before running the following commands make CONFIG_FUSE_PROVISIONING=y in configs/build_lsdk_xx.cfg in flexbuild to enable fuse programming.

2. Command to generate BL31 image (part of fip image) with fuse provisioning support:

```
$:> flex-builder -c atf -m ls1046ardb -b sd
```

3. Command to generate firmware image for SD boot source:

```
$:> flex-builder -i mkfw -m ls1046ardb -b sd
```

4. Optional to edit input file used for fuse provisioning present here:

```
"<flexbuild_dir>/packages/apps/cst/input_files/gen_fusescr/ls104x_1012/input_fuse_file
```

" Again, repeat above steps 2 & 3 to generate composite image.

5. Composite firmware image present here:

```
<flexbuild_dir>/build/images/firmware_ls1046ardb_uboot_sdboot.img
```

NOTE

The above steps are explained for LS1046ARDB platform. Same steps to be followed for LS1088ARDB and LS2088ARDB, where cst input file to be used as *input_files/gen_fusescr/ls2088_1088/input_fuse_file*

6.4.5 Deploy and run fuse provisioning

6.4.5.1 Enable POVDD for SFP

1. LX2160A RDB board
 - Put J9 to enable PWR_PROG_SFP
2. LS2088A RDB Board
 - Put J12 to enable PWR_PROG_SFP
3. LS1088A RDB Board
 - Put J10 to enable PWR_PROG_SFP.

4. LS1046A RDB Board

- Put J21 to enable PWR_PROG_SFP

For more details to enable POVDD on board, refer to section [Prepare board for Secure Boot](#) on page 56 (point 1 only).

6.4.5.2 Deploy firmware image on board

1. Program composite firmware image (firmware_ls1046ardb_uboot_<boot-source>boot.img) built using [Steps to build fuse provisioning firmware image](#) on page 193, on corresponding boot-source using U-Boot commands as shown in below example for SD boot-source:

```
=> tftp a0000000 firmware_ls1046ardb_uboot_sdboot.img  
=> mmc write a0000000 8 1fff8
```

6.4.5.3 Run firmware image on board

1. Execute the following U-Boot command to switch to boot-source. Below is example to switch to SD boot-source on LS1046ARDB:

```
=> cpld reset sd
```

2. On U-Boot prompt, check for any error code in DCFG scratch 4 register for any [Error Codes](#) on page 194 as follows:

```
=> md 1ee020c 1
```

3. If above “md” command shows that no error as follows, then fuse provisioning is successful:

```
01ee020c: 00000000
```

6.4.6 Validation

The procedure specified above is fully validated and verified on LS1046ARDB, LS1088ARDB, and LS2088ARDB platforms.

6.4.7 Error Codes

Table 1: Error Codes

Error Code	Value	Description
ERROR_FUSE_BARKER	0x1	Occurs if fuse script not found.
ERROR_READFB_CMD	0x2	Occurs if SFP Read Fuse Box (READFB) command fails.
ERROR_PROGFB_CMD	0x3	Occurs if SFP Program Fuse Box (PROGFB) command fails.
ERROR_SRKH_ALREADY_BLOWN	0x4	Occurs if SRKH is already blown.
ERROR_SRKH_WRITE	0x5	Occurs if write to SRKH mirror registers fails.
ERROR_OEMUID_ALREADY_BLOWN	0x6	Occurs if OEMUID is already blown.
ERROR_OEMUID_WRITE	0x7	Occurs if write to OEMUID mirror registers fails.

Table continues on the next page...

Table continued from the previous page...

ERROR_DCV_ALREADY_BLOWN	0x8	Occurs if DCV is already blown.
ERROR_DCV_WRITE	0x9	Occurs if write to DCV mirror registers fails.
ERROR_DRV_ALREADY_BLOWN	0xa	Occurs if DRV is already blown.
ERROR_DRV_HAMMING_ERROR	0xb	Occurs if write to DRV mirror registers gives hamming error.
ERROR_OTPMK_ALREADY_BLOWN	0xc	Occurs if OTPMK is already blown.
ERROR_OTPMK_HAMMING_ERROR	0xd	Occurs if write to OTPMK mirror registers gives hamming error.
ERROR_OTPMK_USER_MIN	0xe	Occurs if user supplied OTPMK does not have minimal OTPMK bits set in case where OTPMK flags represents to program user supplied OTPMK value with pre-programmed minimal value.
ERROR_OSPR1_ALREADY_BLOWN	0xf	Occurs if OSPR1 is already blown.
ERROR_OSPR1_WRITE	0x10	Occurs if write to OSPR1 mirror register fails.
ERROR_SC_ALREADY_BLOWN	0x11	Occurs if SysCfg is already blown.
ERROR_SC_WRITE	0x12	Occurs if write to SysCfg mirror register fails.
ERROR_POVDD_GPIO_FAIL	0x13	Occurs if gpio number configured is incorrect.
ERROR_GPIO_SET_FAIL	0x14	Occurs if the gpio bit is not set correctly
ERROR_GPIO_RESET_FAIL	0x15	Occurs if the gpio bit reset is not reset to initial state.

Appendix A Manual steps to build fuse fip image

CST

1. Clone cst from LSDK components.
2. Now make.

```
$:> make
```

3. Default sample input file programs minimal OTPMK values only in fuse memory. Edit "*input_files/gen_fusescr/ls104x_1012/input_fuse_file*" file to select/change values to be programmed in fuses for LS1046 or LS1012. Edit "*input_files/gen_fusescr/ls2088_1088/input_fuse_file*" file to select/change values to be programmed in fuses for LS1088A and LS2088A.

4. To generate fuse_scr.bin, execute the following command:

```
$:> ./gen_fusescr input_files/gen_fusescr/<platform>/input_fuse_file
platform: ls104x_1012 for LS1046A or LS1012A
platform: ls2088_1088 for LS1088A or LS2088A
```

ATF

1. Clone ATF from LSDK components.

2. Set path for the following:

```
$:> export CROSS_COMPILE=<aarch64-toolchain-path->
```

3. Run the following make command in cloned ATF repository:

```
$:> make realclean; make all fip pbl PLAT=<platform> BOOT_MODE=<boot_mode> RCW=$path/rcw.bin  
BL33=$path/uboot.bin fip_fuse FUSE_PROG=1 FUSE_PROV_FILE=$path/fuse_scr.bin
```

NOTE

- <platform> such as ls1046ardb, ls1088ardb, ls2088ardb etc.
- <boot_mode> such as qspi, sd, nor, etc. as per boot mode supported by different platforms.
- Replace \$path with the locations of the respective images to be used to build the image.

4. *fuse_fip.bin* will be present at location `./build/release/<platform>/fuse_fip.bin`.

6.5 PKCS#11 and Secure Object Library

6.5.1 Introduction

NXP SoCs such as LS1046A can store keys securely using built-in SoC capabilities - virtual HSM. With such devices, sensitive private keys never leave the device and cryptographic operations are performed on this virtual HSM.

The PKCS#11 is a standard programming interface to communicate with HSMs. This standard specifies an application programming interface (API), called “Cryptoki” to devices which hold cryptographic information and perform cryptographic functions.

Proprietary interfaces using Secure Object Library are provided to interact with the HSM for:

- Generating key pair within the HSM.
- Installing existing key in the HSM.
- Manufacturing Protection key operations. (MPKey)

The private keys are never visible to normal world.

Sensitive Cryptographic operations using these keys can only be done using PKCS#11 cryptographic token standard.

An OpenSSL engine on Secure Object Library is also provided to interface directly with OpenSSL APIs

The PKCS#11 library release is compliant to v2.40. It is targeted for LS1046ARDB and supports

- RSA keys of size 1K and 2K.
- ECDSA keys curve prime256v1 and secp384r1.

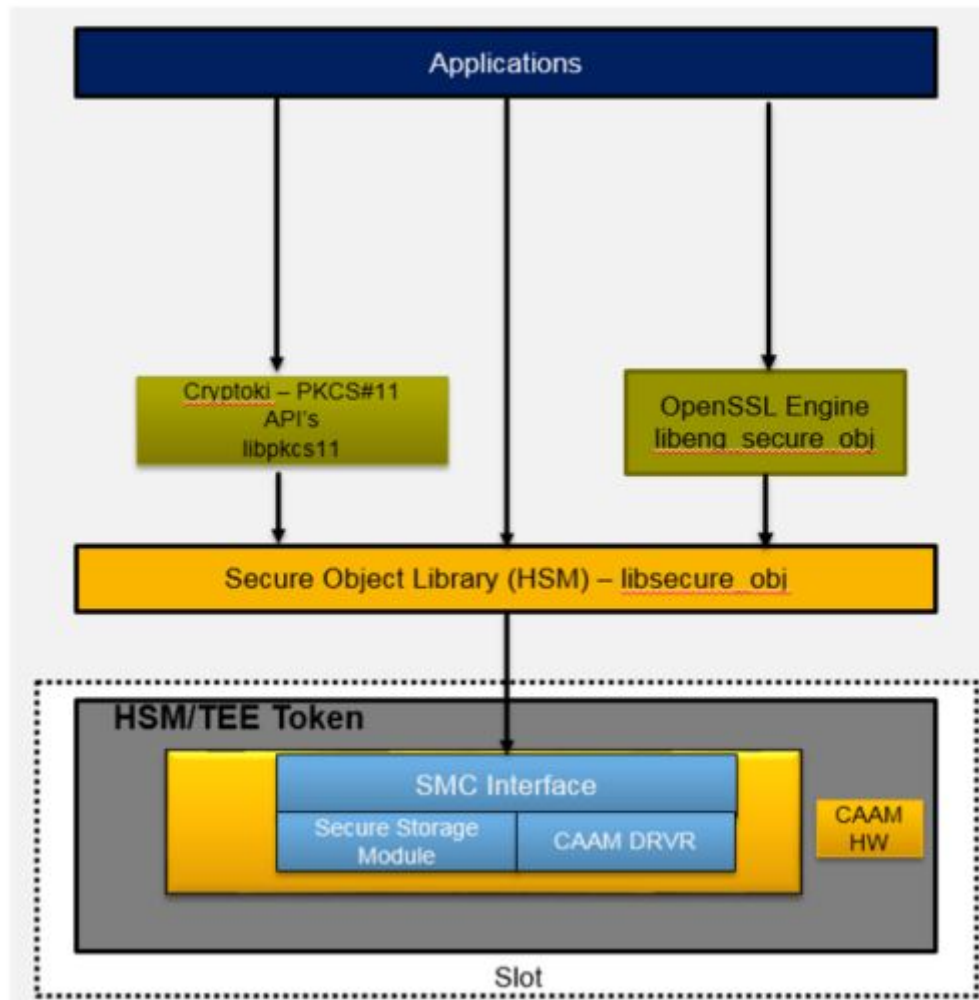


Figure 27. Block Diagram

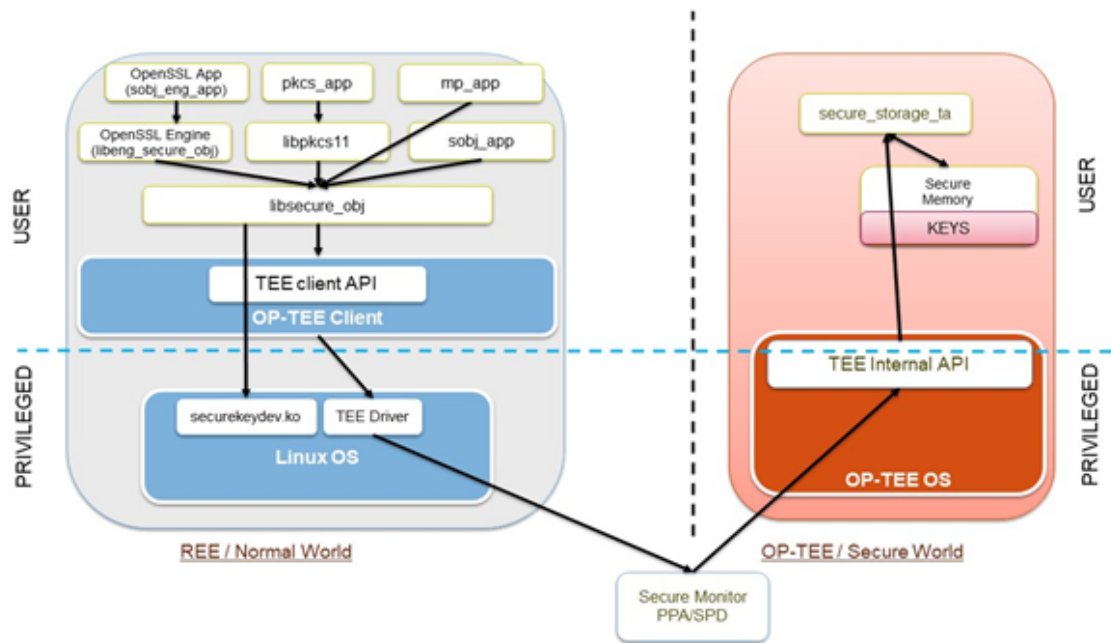


Figure 28. Details of HSM

6.5.2 Supported APIs

6.5.2.1 PKCS#11 Library – libpkcs11.so

The PKCS#11 interfaces are exposed and implemented via a shared library with a name called libpkcs11.so (Cryptoki Library). Any PKCS#11 library has a static CK_FUNCTION_LIST structure, and a pointer to it may be obtained by the C_GetFunctionList() function.

The table below, summarizes the list of supported PKCS#11 interfaces. The return values and API behaviors are compliant with the PKCS#11 standard v2.40. Library expects the caller to use them in a standard way.

API	Description
C_Initialize	Initialize Cryptoki library
C_Finalize	Clean up cryptoki related resources
C_GetFunctionList	Obtains entry points of Cryptoki library functions.
C_GetInfo	Obtains general information about Cryptoki
C_GetSlotInfo	Obtains information about a particular slot
C_GetTokenInfo	Obtains information about a particular token
C_GetSlotList	Obtain list of slots in the system. Only a fixed slot with fixed token is supported. Dynamic slot or token addition is not supported.

Table continues on the next page...

Table continued from the previous page...

C_OpenSession	Opens/Closes a session.
C_CloseSession	• All types of sessions are supported with Token.
C_CloseAllSessions	• Only Token Objects can be created/destroyed, Session Objects are not supported.
C_Login	Logs into a token.
C_Logout	Logs out from a token
C_CreateObject	Creates an object (RSA Keys of size up to 2048bits are supported)
C_DestroyObject	Destroys an object
C_FindObjectsInit	Objects search operations.
C_FindObjects	RSA Public and Private key objects of size up to 2048bits are supported.
C_FindObjectsFinal	ECDSA Public and Private key objects of size 256 & 384 bits are supported.
C_GetAttributeValue	Obtains the value of one or more attributes of the objects.
C_GetMechanismList	Obtains List of mechanism supported by token.
C_GetMechanismInfo	Obtains the information about a mechanism.
C_GenerateKeyPair	Generates a public-key/private-key pair (RSA Keys of size up to 2048bits are supported)
C_SignInit	Initialize a signature operation.
C_Sign	Signs single-part data.
C_SignUpdate	Continues a multiple-part signature operation.
C_SignFinal	Finishes a multiple-part signature operation. Mechanisms supported: • RSA-based Mechanisms • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA256_RSA_PKCS • CKM_SHA384_RSA_PKCS • CKM_SHA512_RSA_PKCS • ECDSA-based Mechanisms (Single Part Only) • CKM_ECDSA • CKM_ECDSA_SHA1

Table continues on the next page...

Table continued from the previous page...

C_DigestInit	Initializes a message-digesting operation.
C_Digest	Digests single-part data.
C_DigestUpdate	Continues a multiple-part digesting operation.
C_DigestFinal	Finishes a multiple-part digesting operation. Mechanisms supported: • CKM_MD5 • CKM_SHA1 • CKM_SHA256 • CKM_SHA384 • CKM_SHA512
C_DecryptInit	Initializes a decryption operation.
C_Decrypt	Decrypts single-part encrypted data. Mechanisms supported: • CKM_RSA_PKCS • CKM_RSA_PKCS_OAEP

6.5.2.2 Secure Object Library – libsecure_obj.so

The following are the details of the supported interfaces to generate/import keys using the Secure Object library.

1. Import Keys

SK_RET_CODE SK_CreateObject(SK_ATTRIBUTE *attr, uint16_t attrCount, SK_OBJECT_HANDLE *phObject);

The API creates an Object on the HSM, and returns a handle to it. API always succeeds even if an object with same attributes exists in HSM. Duplicate object is created. Application needs to take care that duplicate objects should not be created.

attr is an array of attributes that the object should be created with. Some of the attributes may be mandatory, such as SK_ATTR_OBJECT_TYPE and SK_ATTR_OBJECT_INDEX (the id of the object), and some are optional.

Application needs to take care that valid attributes are passed, library does not return any error on receiving inconsistent/incompatible attributes.

param[in] attr: The array of attributes to be used in creating the Object.

param[in] attrCount: The number of attributes in attr

param[in, out] phObjectIN: A pointer to a handle (must not be NULL);

OUT: The handle of the created Object

Return Values:

SKR_OK Successful execution, phObject filled with created object handle.

SKR_ERR_BAD_PARAMETERS Invalid function arguments

SKR_ERR_OUT_OF_MEMORY Memory allocation failed.

SKR_ERR_NOT_SUPPORTED The function and/or parameters are not supported by the library.

-- Some internal error code other than mentioned above can be returned. Refer to `securekey_api_types.h` for error code description.

2. Generate Key.

SK_RET_CODE SK_GenerateKeyPair(SK_MECHANISM_INFO *pMechanism, SK_ATTRIBUTE *attr, uint16_t attrCount, SK_OBJECT_HANDLE *phKey);

This API generates key pair on the HSM, and returns a handle to it. API always succeeds even if an object with same attributes exists in HSM. Duplicate object is created. Application needs to take care that duplicate objects should not be created.

`pMechanism` is mechanism for key pair generation. For example: `SKM_RSA_PKCS_KEY_PAIR_GEN`.

`attr` is an array of attributes that the object should be created with. Some of the attributes may be mandatory, such as `SK_ATTR_OBJECT_INDEX` (the id of the object), and some are optional.

Application needs to take care that valid attributes are passed, library does not return any error on receiving inconsistent/incompatible attributes.

param[in] pMechanism Mechanism for key pair generation

param[in] attr The array of attributes to be used in creating the Object.

param[in] attrCount The number of attributes in `attr`

param[in, out] phKey IN: A pointer to a handle (must not be NULL);

OUT: The handle of the created Object

Return Values:

`SKR_OK` Successful execution, `phObject` is filled with created object handle.

`SKR_ERR_BAD_PARAMETERS` Invalid function arguments

`SKR_ERR_OUT_OF_MEMORY` Memory allocation failed.

`SKR_ERR_NOT_SUPPORTED` The function and/or parameters are not supported by the library.

--Some internal error code other than mentioned above can be returned. Refer to `securekey_api_types.h` for error code description.

3. Erase Object.

SK_RET_CODE SK_EraseObject(SK_OBJECT_HANDLE hObject);

Erases an object from the HSM. This means that the object with the specified handle can no longer be used.

param[in] hObject

The handle of the Object to be erased.

Return Values:

`SKR_OK` Successful execution `SKR_ERR_BAD_PARAMETERS` Invalid function arguments -- Some internal error code other than mentioned above to be returned. Refer to `securekey_api_types.h` for error code description.

Further details of the APIs and its types are available in the files `<securekey_api.h>` and `<securekey_api_types.h>` in folder **secure_obj**.

NOTE

1. Maximum of 50 objects can be created/generated as of now.
2. Secure Object Library will not be throwing any error if multiple objects having same attributes are being created. It is applications responsibility to take care of attributes that are passed during creation/generation of objects.

Manufacturing Key APIs:

Following secure boot, the system runs the key generation routine producing an ECC Public and Private Key pair, referred to as Manufacturing Protection Key Pair(MPKey).

Key Generation is performed by BootRom. APIs for getting MP Public key, signing using MP Private key and for getting the MP Tag are described below.

For complete documentation on how to perform the key generation, public key export, and signing with the ECC private key, refer to the Manufacturing-protection chip authentication

process section in the SoC's Security (SEC) Reference Manual 5.6

NOTE: For this feature to work board must be booted in Secure Boot mode, with ITS bit set to 1.

1. **Get MP Public key.**

```
enum sk_status_code sk_mp_get_pub_key(struct sk_EC_point *pub_key);
```

Get Manufacturing Protection(MP) Public Key (ECC P256 Key).

param[in,out] pub_key: This is MP Public Key to be returned. Application needs to allocate memory for sk_EC_point. Each of the coordinate x & y needs to allocate sk_EC_point.len memory. sk_EC_point.len can be obtained using **sk_mp_get_pub_key_len()**.

Return Values:

SK_SUCCESS on success, error value otherwise.

2. **Sign using MP Private Key**

```
enum sk_status_code sk_mp_sign(unsigned char * msg, uint8_t msglen,  
struct sk_EC_sig * sig, uint8_t * digest, uint8_t digest_len)
```

Sign the msg using MP Priv Key. While signing MP Message, it will be prepended to message. Message over which signature will be calculated = MP message + msg.

param[in] msg: Pointer to the message to be signed.

param[in] msglen: Length of the message to be signed.

param[in,out] sig: This is Signature calculated. Application needs to allocate memory for sk_EC_sig. Each of the parts r & s needs to be allocated sk_EC_sig.len memory. sk_EC_sig.len can be obtained using **sk_mp_get_sig_len()**.

param[in, out] digest: Digest(SHA256) of the message to be signed. Digest is calculated by prepending MP Message to the msg.

param[in] digest_len: Length of digest. Application needs to allocate memory for sk_EC_point. Each of the coordinate x & y needs to allocate sk_EC_point.len memory. sk_EC_point.len can be obtained using **sk_mp_get_pub_key_len()**.

Return Values:

SK_SUCCESS on success, error value otherwise.

3. **Get MP Tag.**

```
enum sk_status_code sk_mp_get_mp_tag(uint8_t *mp_tag_ptr, uint8_t mp_tag_len);
```

Get the MP Message. While signing, MP Message is prepended to message automatically. User can call this function to get MP message tag during verification operation.

param[in, out] mp_tag_ptr: Pointer to the message to be signed. Application needs to allocate memory of length returned by **sk_mp_get_tag_len()**.

param[in] mp_tag_len: Length of the mp_tag_ptr buffer

Return Values:

SK_SUCCESS on success, error value otherwise.

The API definition can be found in file **securekey_mp.h**. Sample applications have also been provided which demonstrate how to use APIs.

6.5.3 Integrating applications with Secure Object

There are following ways in which applications can interact with Secure Objects stored in HSM/Token.

- Using PKCS#11 APIs.
- Using Secure Object APIs.
- Applications using OpenSSL APIs
 - Secure Object Library based OpenSSL Engine (libeng_secure_obj)
 - PKCS#11 based OpenSSL Engine (Third party OpenSC/libp11)

6.5.3.1 Using PKCS#11 APIs

Applications can directly use the PKCS#11 APIs to interact with the Secure Objects stored in HSM/Token. Currently we support PKCS#11 APIs mentioned in PKCS#11 APIs.

PKCS#11 library can also be used with any OpenSource PKCS#11 application such as p11tool, softhsm2-utils etc.

We have tested this library with p11tool for following operations:

- Listing tokens: **p11tool --list-tokens**
- Initializing token: **p11tool --initialize**
- Initializing User pin: **p11tool --initialize-pin**
- Initializing SO pin: **p11tool --initialize-so-pin**
- Generating RSA Key: **p11tool --generate-rsa**
- Importing RSA Key: **p11tool --write --load-privkey <rsa_key.pem>**

For more information on p11tool commands, please check [here](#)

We have also created a reference application pkcs11_app for showing how to use the PKCS#11 APIs for writing your own application.

Commands to run **pkcs11_app** are shown [here](#)

6.5.3.2 Using Secure Object APIs

Applications can directly use the Secure Object Library APIs to interact with the Secure Objects stored in HSM/Token. Currently we support APIs mentioned in Secure Object APIs.

We have also created a reference application sobj_app for showing how to use the Secure Object APIs.

Commands to run **sobj_app** are shown [here](#).

6.5.3.3 Applications using OpenSSL APIs

This topic provides examples of usage with OpenSSL. It is recommended that you should familiarize yourself with Open SSL.

Refer to the appropriate documents for Open SSL commands at the following location:

<http://www.openssl.org/docs/>

Open SSL provides the support of engine (basically hardware devices) to store the keys on hardware devices to make keys more secure.

There are 2 ways in which applications using the OpenSSL APIs can access the Secure Objects stored in HSM/Token.

- Secure Object Library based OpenSSL Engine (libeng_secure_obj).
- PKCS#11 based OpenSSL Engine (Third party OpenSC/libp11).

6.5.3.3.1 Secure Object Library based OpenSSL Engine (libeng_secure_obj)

NXP provides the Secure Object Library based OpenSSL Engine that is used to communicate with underlying HSM.

This engine is based on Secure Object Library, It does following things:

1. RSA Private Encryption.
2. RSA Private Decryption.
3. ECDSA Signing Operation.

All other RSA/ECDSA operations will be done by OpenSSL itself.

This engine does not support generation of RSA Keys. Keys will be generated via another app **sobj_app** and these keys are used in the applications using this OpenSSL Engine.

Refer **Running the sobj_eng_app** section for screenshots of app using OpenSSL engine.

Example Usage with OpenSSL

This topic provides examples of usage with OpenSSL:

- Using the engine from command Line. Change the following in `openssl.cnf` (often in `/etc/ssl/openssl.cnf`).

This line must be placed at the top, before any sections are defined:

```
openssl_conf = conf_section
```

Add following section at bottom of file:

```
[conf_section]
engines = engine_section
[engine_section]
secure_obj = sobj_section
[sobj_section]
engine_id = eng_secure_obj
dynamic_path = <path where lib_eng_secure_obj.so is placed>
default_algorithms = RSA
init = 1
```

Note: This sections shows only RSA examples, same can be done for EC by changing `default_algorithms` in `openssl.cnf` as shown below:

```
default_algorithms = RSA, EC
```

Testing the engine operation:

To verify that the engine is properly operating, you can use the following example:

```
root@Ubuntu:~#
root@Ubuntu:~# openssl engine
(dynamic) Dynamic engine loading support
(eng_secure_obj) secure object OpenSSL Engine.
root@Ubuntu:~#
root@Ubuntu:~#
```


If you do not update the OpenSSL configuration file, specify the engine configuration explicitly.

```
$: openssl engine -t dynamic -pre SO_PATH:<path-to-libeng_secure_obj.so> -pre ID:eng_secure_obj -pre LIST_ADD:1 -pre LOAD
```

```
root@Ubuntu:~#
root@Ubuntu:~# openssl engine -t dynamic -pre SO_PATH:/usr/lib/aarch64-linux-gnu/openssl-1.0.0/engines/libeng_secure_obj.so -pre ID:eng_secure_obj -pre LIST_ADD:1 -pre LOAD
(dynamic) Dynamic engine loading support
[Success] : SO_PATH:/usr/lib/aarch64-linux-gnu/openssl-1.0.0/engines/libeng_secure_obj.SO
[Success] : ID:eng_secure_obj
[Success] : LIST_ADD:1
[Success] : LOAD
LOADED: (eng_secure_obj) Secure object OpenSSL Engine.
[available]
root@Ubuntu:~#
```

- Using OpenSSL from the command line.

Following commands can be used to generate RSA/ECDSA key-pair and use them in signing any data and verifying the signatures generated.

```
$: sobj_app -G -m rsa-pair -s 2048 -l "rsa_gen_2048" -i 1 -w rsa_2048.pem ## Generating RSA key-pair ##
$: openssl rsa -in rsa_2048.pem -pubout -out rsa_pub_2048.pem ## Taking out Public Key for verifying signature ##
$: openssl dgst -sha1 -sign rsa_2048.pem -out sig.data data ## Generating Signature "sig.data" of "data" ##
$: openssl dgst -sha1 -verify rsa_pub_2048.pem -signature sig.data data ## Verifying the signature using Public Key ##
```

Same thing can be done for ECDSA keys of prime256v1 by using following commands:

```
$: sobj_app -G -m ec-pair -c prime256v1 -l "ecc_256" -i 2 -w ec256.pem
$: openssl ec -in ec256.pem -pubout -out ec_pub_256.pem
$: openssl dgst -sha1 -sign ec256.pem -out sig.data data
$: openssl dgst -sha1 -verify ec_pub_256.pem -signature sig.data data
```

For ECDSA secp384r1 curve us following commands:

```
$: sobj_app -G -m ec-pair -c secp384r1 -l "ecc_384" -i 3 -w ec384.pem
$: openssl ec -in ec384.pem -pubout -out ec_pub_384.pem
$: openssl dgst -sha1 -sign ec384.pem -out sig.data data
$: openssl dgst -sha1 -verify ec_pub_384.pem -signature sig.data data
```

- This section describes how to use the command line to create a self-signed certificate for "NXP Semiconductor". The key of the certificate is generated in the Secure Object HSM and will not exportable.

As per the following examples, generate a private key in the HSM with sobj_app, This will also create a fake PEM file “dev_key.pem” having information to get the required key from HSM.

Following command is generating RSA key-pair.

```
$: sobj_app -G -m rsa-pair -s 2048 -l "Test_Key" -i 1 -w dev_key.pem
```

ECDSA key-pair can also be generated using following command:

```
$: sobj_app -G -m ec-pair -c prime256v1 -l "ecc_256" -i 30 -w dev_key.pem
```

To generate a certificate with key in the Secure Object module, the following commands can be used:

```
$ openssl req -new -key dev_key.pem -out req.pem -text -x509 -subj "/CN=NXP Semiconductor"
$ openssl x509 -signkey dev_key.pem -in req.pem -out cert.pem
```

The first command creates a self-signed Certificate for "NXP Semiconductor". The signing is done using the key specified by the fake PEM file.

The second command creates a self-signed certificate for the request, the private key used to sign the certificate is the same private key used to create the request.

6.5.3.3.2 PKCS#11 based OpenSSL Engine (Third party OpenSC/libp11)

libp11 is a library implementing a thin layer on top of PKCS#11 API to make using PKCS#11 implementations easier.

You can get library from: <https://github.com/OpenSC/libp11>.

This code repository produces two libraries:

- libp11 provides a higher-level (compared to the PKCS#11 library) interface to access PKCS#11 objects. It is designed to integrate with applications that use OpenSSL.
- pkcs11 engine plugin for the OpenSSL library allows accessing PKCS#11 modules in a semi-transparent way.

pkcs11 engine for OpenSSL can be installed on board using command **sudo apt-get install libengine-pkcs11-openssl**

Above command will install the libpkcs11.so (pkcs11 engine) in /usr/lib/aarch64-linux-gnu/engines-1.1/libpkcs11.so and this will be *dynamic_path* in OpenSSL onfiguration file.

For running the PKCS#11 OpenSSL Engine with our PKCS#11 Library add following into your global OpenSSL configuration file (often in /etc/ssl/openssl.cnf). This line must be placed at the top, before any sections are defined:

```
openssl_conf = openssl_init
```

This should be added to the bottom of the file:

```
[openssl_init]
engines=engine_section

[engine_section]
pkcs11 = pkcs11_section

[pkcs11_section]
engine_id = pkcs11
dynamic_path = <path-to-pkcs11-engine>/libpkcs11.so
MODULE_PATH = <path-to-NXP-pkcs11-library>/libpkcs11.so
init = 0
```

The dynamic_path value is the pkcs11 engine plug-in, the MODULE_PATH value is the NXP PKCS#11 library. The engine_id value is an arbitrary identifier for OpenSSL applications to select the engine by the identifier.

Testing the engine operation

To verify that the engine is properly operating you can use the following example.

```
$ openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
[ available ]
```

Using p11tool and OpenSSL from the command line:

This section demonstrates how to use the command line to create a self-signed certificate for "NXP Semiconductor". The key of the certificate will be generated in the token and will not exportable.

p11tool from GnuTLS and this engine with OpenSSL work in combination.

p11tool is a tool that manipulate PKCS #11 tokens. Export/import data from PKCS #11 tokens. To use PKCS #11 tokens with gnutls the configuration file /etc/gnutls/pkcs11.conf must exist and contain number lines of the form "load=<pkcs-library-path>" or this PKCS#11 module can be provided directly as --provider in command line as argument.

p11tool can be installed by running command **sudo apt-get install gnutls-bin**

For more configuration options check: https://www.gnutls.org/manual/html_node/p11tool-Invocation.html.

Check for key which is already created from **sobj_app** via p11tool.

The following commands utilize p11tool for that.

```
$ p11tool --provider <path-to-NXP-PKCS-library>/libpkcs11.so --list-privkeys
```

```
root@localhost:~# p11tool --provider /root/libpkcs11.so --list-privkeys
Object 0:
  URL: pkcs11:model=;manufacturer=NXP;serial=1;token=TEE_BASED_TOKEN;
%01%00%00%00;object=Device_Key3;type=private
  Type: Private Key
  Label: Device Key3
  Flags: CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
  ID: 01:00:00:00
Object 1:
  URL: pkcs11:model=;manufacturer=NXP;serial=1;token=TEE_BASED_TOKEN;
%01%00%00%00;object=Device_Key2;type=private
  Type: Private Key
  Label: Device Key2
  Flags: CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
  ID: 01:00:00:00
Object 0:
  URL: pkcs11:model=;manufacturer=NXP;serial=1;token=TEE_BASED_TOKEN;
%01%00%00%00;object=Device_Key3;type=private
  Type: Private Key
  Label: Device Key
  Flags: CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
  ID: 01:00:00:00
root@localhost:~#
```

Note the PKCS #11 URL shown above and use it in the commands below.

To generate a certificate with its key in the PKCS #11 module, the following command can be used.

Following command creates a self-signed Certificate for "NXP Semiconductor". The signing is done using the key specified by the URL.

```
$ openssl req -engine pkcs11 -new -key
"pkcs11:model=;manufacturer=NXP;serial=1;token=TEE_BASED_TOKEN;id=%01%00%00%00;object=Device_Key3
;type=private" -keyform engine -out req.pem -text -x509 -subj "/CN=NXP Semiconductor"
```

6.5.4 Board Bootup & Running applications

6.5.4.1 Board Bootup

1. Prepare the images using the LSDK documentation and bootup the board with **secure-boot** and **ITS set to 1**. ITS = 1 is required for bootrom to generate the Manufacturing Protection Private Key.

For setting ITS bit to 1 run following command after programming SRK Hash and before removing the boot hold-off. The test is performed on LS1046ARDB.

#To do ITS=1

ccs::write_mem 32 0x1e80200 4 0 0x00000004

You can refer here for documentation - <https://lsdk.github.io/document.html>

- After booting up the board with LSDK1812 images, Check if following images are placed in corresponding places.

Binary	Place in rootfs
b05bcf48-9732-4efa-a9e0-141c7c888c34.ta	/lib/optee_armtz/
libsecure_obj.so	/usr/local/lib
sobj_app	/usr/local/bin
mp_app	/usr/local/bin
mp_verify	/usr/local/bin
libeng_secure_obj.so	/usr/lib/aarch64-linux-gnu/openssl-1.0.0/engines/
sobj_eng_app	/usr/local/bin
securekeydev.ko	This path depends on Linux Kernel Version: Linux Kernel 4.9 - /lib/modules/4.9.xx<commit-id>/extra/ Linux Kernel 4.14 - /lib/modules/4.14.xx<commit-id>/extra/
libpkcs11.so	/usr/local/lib
pkcs11_app	/usr/local/bin
thread_test	/usr/local/bin

For Compilation steps, refer **Appendix Section** at end of this document.

- Run “**tee-supplciant &**” command from linux prompt.
- Depending on linux kernel version used “**insmod securekeydev.ko**” from right folder.
- Run the applications as described in [Running the applications](#).

6.5.4.2 Running applications

Two applications are available with the package.

- **sobj_app** - Provides interface to generate/import key objects via Secure Object Library
- **pkcs11_app** – Provides interface to enumerate objects in the HSM and perform cryptographic operations.
- **mp_app** - This application demonstrates how to Get MP Public Key, sign a message using MP Private Key, Get Message tag.
- **mp_verify** - This app uses OpenSSL APIs to verify the signature obtained by using mp_app application.
- **sobj_eng_app** – This app uses OpenSSL APIs to show how to use Secure Object based OpenSSL Engine. This application is loading the private key and then doing cryptographic operations using this key.
- **thread_test** - PKCS#11 application to test multithreading feature of PKCS#11 library.

NOTE: These are reference applications to demonstrate the usage of APIs as described in [Supported APIs](#)

6.5.4.2.1 subj_app

To create/generate objects, run **subj_app** application.

- **subj_app** – This command shows help related to subj_app.

```
root@Ubuntu:~# subj_app
Only one of the below options are allowed per execution:-

-C - Create Object
-G - Generate Object
-A - Attributes of the Object
-L - List Object
-R - Remove/Erase Object

Use below Sub options along with Main options:-
-o - Object Type (Supported: pair, pub)
-k - Key Type (Supported: rsa, ec)
-s - RSA Key Size/Length (Supported: 1024, 2048).
-c - EC Curve (Supported: prime256v1, secp384r1).
-f - File Name (.pem) (Private Key).
-l - Object Label
-i - Object Id. (In Decimal)
-h - Object Handle (In Decimal)
-n - Number of Objects (Default = 5)
-m - Mechanism Id (Supported: rsa-pair, ec-pair)
-w - Fake .pem file. (Optional command while generating/creating RSA, ECDSA key-pair).

Usage:
Creation:
subj_app -C -f <private.pem> -k <key-type> -o <obj-type> -s <key-size> -l <obj-label> -i <obj-ID>
subj_app -C -f sk_private.pem -k rsa -o pair -s 2048 -l "Device_Key" -i 1
subj_app -C -f sk_private.pem -k ec -o pair -l "Device_Key" -i 1
subj_app -C -f sk_private.pem -k rsa -o pair -s 2048 -l "Device_Key" -i 1

Generation:
subj_app -G -m <mechanism-ID> -s <key-size> -l <key-label> -i <obj-ID>
subj_app -G -m rsa-pair -s 2048 -l "Device_Key" -i 1
subj_app -G -m ec-pair -c prime256v1 -l "Device_Key" -i 1
subj_app -G -m rsa-pair -s 2048 -l "Device_Key" -i 1 -w dev_key.

Attributes:
subj_app -A -h <obj-handle>
subj_app -A -h 1

List:
subj_app -L [-n <num-of-obj>] -k <key-type> -l <obj-label> -s <key-size>
Objects can be listed based on combination of any above criteria

Remove
subj_app -R -h <obj-handle>
subj_app -R -h 1
```

- **Importing an RSA key pair to HSM**

subj_app -C -f <private.pem> -k <key-type> -o <obj-type> -s <key-size> -l <obj-label> -i <obj-ID>

This command helps in importing a key to the HSM. It creates an object in HSM reading key from <private.pem> with object label <obj-label> and object ID <obj-ID>. This private.pem can be generated by openssl using the command below:

openssl genpkey -algorithm RSA -out sk_private.pem -pkeyopt rsa_keygen_bits:2048

Handle of the object created in the HSM is printed as an output to the command. This handle can be used for further operations on the created object (for example, delete, printing attributes and so on)

```
root@Ubuntu:~# subj_app -C -f sk_private.pem -k rsa -o pair -s 2048 -l "rsa_create_2048" -i 0
Creating the Object.
Import Key from sk_private.pem
Key Length = 2048
Object created successfully handle = 0
root@Ubuntu:~#
root@Ubuntu:~#
```

- **Importing an ECDSA key pair to HSM**

sobj_app -C -f <private.pem> -k <key-type> -o <obj-type> -l <obj-label> -i <obj-ID>

This command helps in importing a key to the HSM. It will create an object in HSM reading key from <private.pem> with object label <obj-label> and object ID <obj-ID>.

This private.pem can be generated by openssl using below command:

openssl ecparam -genkey -name prime256v1 -noout -out ec_key_256.pem

Handle of the object created in the HSM is printed as an output to the command. This handle can be used for further operations on the created object (eg delete, printing attributes etc)

```
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -C -f ec_key_256.pem -k ec -o pair -l "ecc_create_256" -i 1
Creating the Object.
Object created successfully handle = 1
root@Ubuntu:~#
```

- **Generating an RSA key pair in HSM**

sobj_app -G -m <mechanism-ID> -s <key-size> -l <key-label> -i <key-ID>

This command generates an object of type derived from mechanism-ID of size <key-size> with label <key-label> and ID <key-ID>

Handle of the object created is printed as an output to the command. This handle can be used for further operations on the created object (for example, delete, printing attributes and so on)

```
root@Ubuntu:~# sobj_app -G -m rsa-pair -s 2048 -l "rsa_gen_2048" -i 2
Generating the Object.
Object generated successfully handle = 2
root@Ubuntu:~#
root@Ubuntu:~#
```

- **Generating ECDSA key pair in HSM**

sobj_app -G -m <mechanism-ID> -c <curve> -l <key-label> -i <key-ID>

This command will generate an object of type derived from mechanism-ID of size <key-size> with label <key-label> and ID <key-ID>

Handle of the object created is printed as an output to the command. This handle can be used for further operations on the created object (eg delete, printing attributes etc)

```
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -G -m ec-pair -c prime256v1 -l "ecc_gen_256" -i 3
Generating the Object.
Object generated successfully handle = 3
root@Ubuntu:~#
```

- **Display attributes of an object in the HSM**

sobj_app -A -h <obj-handle>

This command shows some attributes related to object created. Pass the object handle <obj-handle> to the command. This <obj-handle> is printed during generation/import of objects to HSM.

```
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -A -h 0
Attributes of Object Handle: 0
    Object Label: rsa_create_2048
    Object Id: 0
    Object Type: KEY_PAIR[0x10000]
    Object Key Type: RSA[0x0]
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -A -h 1
Attributes of Object Handle: 1
    Object Label: ecc_create_256
    Object Id: 1
    Object Type: KEY_PAIR[0x10000]
    Object Key Type: EC[0x1]
root@Ubuntu:~#
```

- **List handles of the objects available in the HSM**

sobj_app -L [-n <num-of-obj> -k <key-type> -l <obj-label> -s <key-size> -i <obj-id>]

This command lists handles of the objects already created/generated based on some search criteria (if given). User can then use this handle to print the rest of the attributes. (See above command)

```
root@Ubuntu:~# sobj_app -L
None of the search option (-i -o -k -s -l) is provided. Listing all Object.
Missing Option [-n]. Listing max of 5 objects.
Following objects found:
Object[0] handle = 0
Object[1] handle = 1
Object[2] handle = 2
Object[3] handle = 3
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -L -l ecc_create_256
Missing Option [-n]. Listing max of 5 objects.
Following objects found:
Object[0] handle = 1
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -L -l ecc_create
Missing Option [-n]. Listing max of 5 objects.
No Object Found.
```

```
Following objects found:
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -L -s 2048
Missing Option [-n]. Listing max of 5 objects.
Following objects found:
Object[0] handle = 0
Object[1] handle = 2
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# sobj_app -L -k ec
Missing Option [-n]. Listing max of 5 objects.
Following objects found:
Object[0] handle = 1
Object[1] handle = 3
root@Ubuntu:~#
```

6.5.4.2.2 pkcs11_app

- **pkcs11_app** – This command shows commands available.

```

root@Ubuntu:/greengrass# pkcs11_app
Only one of the below option is allowed per execution:-

-I - Library Information.
-T - Token
-P - Slot
-M - Mechanism
-F - Find
-S - Sign
-V - Verify
-E - Encrypt
-D - Decrypt

Use below Sub options along with Main options:-
-i - Info.
-l - List.
-k - Key Type (Supported: rsa, ec)
-o - Object Type (Supported: pub, prv)
-b - Object Label.
-p - Slot Id.
-n - Number of Object to be Listed (Default n =10).
-m - Mechanism Id
Supported Mechanism: rsa, rsa-oaep, md5-rsa, sha1-rsa, sha256-rsa, sha384-rsa, sha512-rsa, ec, sha1-ec
EC/RSA Sign/Verify: rsa, md5-rsa, sha1-rsa, sha256-rsa, sha384-rsa, sha512-rsa, ec, sha1-ec
RSA Encrypt/Decrypt: rsa, rsa-oaep
-d - Plain Data
-s - Signature Data
-e - Encrypted Data

Usage:
Library Information:
pkcs11_app -I

Slot/Token Commands:
pkcs11_app -P -l
pkcs11_app -P -i -p <slot-ID>; (pkcs11_app -P -i -p 0)
pkcs11_app -T -i -p <slot-ID>; (pkcs11_app -T -i -p 0)

Mechanism:
pkcs11_app -M -l -p <slot-ID>; (pkcs11_app -M -l -p 0)
pkcs11_app -M -m <mech-ID> -i -p <slot-ID>; (pkcs11_app -M -m rsa -i -p 0)
pkcs11_app -M -i -p <slot-ID>; (pkcs11_app -M -i -p 0)

Object Search:
pkcs11_app -F -p <slot-ID> [-n <num-of-obj> -k <key-type> -b <obj-label> -o <obj-type>]
Objects can be listed based on combination of any above criteria.

Signature Generation
pkcs11_app -S -k <key-type> -b <key-label> -d <Data-to-be-signed> -m <mech-ID> -p <slot-ID>
pkcs11_app -S -k rsa -b Device_Key -d "PKCS11 TEST DATA" -m md5-rsa -p 0
pkcs11_app -S -k ec -b Device_Key -d "PKCS11 TEST DATA" -m sha1-ec -p 0

Signature Verification
pkcs11_app -V -k <key-type> -b <key-label> -d <Data-previously-signed> -s <signature-file> -m <mech-ID> -p <slot-ID>
pkcs11_app -V -k rsa -b Device_Key -d "PKCS11 TEST DATA" -s sig.data -m md5-rsa -p 0
pkcs11_app -V -k ec -b Device_Key -d "PKCS11 TEST DATA" -s sig.data -m sha1-ec -p 0

Public Key Encryption (RSA Only)
pkcs11_app -E -k <key-type> -b <key-label> -d <Data-to-be-encrypted> -m <mech-ID> -p <slot-ID>
pkcs11_app -E -k rsa -b Device_Key -d "PKCS11 TEST DATA" -m rsa -p 0

Private Key Decryption (RSA Only)
pkcs11_app -D -k <key-type> -b <key-label> -e enc.data -m <mech-ID> -p <slot-ID>
pkcs11_app -D -k rsa -b Device_Key -e enc.data -m rsa -p 0

```

- **pkcs11_app -I:** Library Information

pkcs11_app -P -I: List the all available slots

pkcs11_app -P -i -p <slot-ID> : Provides the information about Slot with <slot-ID>

pkcs11_app -T -i -p <slot-ID> : Provides the information about Token inserted in Slot <slot-ID>


```

root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -I
Getting Information about Cryptoki Library
Library Manufacturer = NXP
Library Description = libpkcs11
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -P -l
Slot List :
    Slot ID = 0
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -P -i -p 0
Slot info of in-use slot with ID = 0 :
    Slot Description: TEE_BASED_SLOT
    Slot Manufacturer = NXP
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -T -i -p 0
TokenInfo for Slot Id: 0.
    Token Label = TEE_BASED_TOKEN
    Token Manufacturer = NXP
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# _

```

- **pkcs11_app -M -l -p <slot-ID>** : Lists the Mechanism List supported by token in Slot <slot-ID>

pkcs11_app -M -m <mech-ID> -i -p <slot-ID> : Gives information about the mechanism with <mech-ID> for Slot <slot-ID>

```

root@Ubuntu:~/new# pkcs11_app -M -l -p 0
Mechanism listing from the Slot Id = 0:
    CKM_MD5 with mechanism ID[528].
    CKM_SHA_1 with mechanism ID[544].
    CKM_SHA256 with mechanism ID[592].
    CKM_SHA384 with mechanism ID[608].
    CKM_SHA512 with mechanism ID[624].
    CKM_RSA_PKCS with mechanism ID[1].
    CKM_MD5_RSA_PKCS with mechanism ID[5].
    CKM_SHA1_RSA_PKCS with mechanism ID[6].
    CKM_SHA256_RSA_PKCS with mechanism ID[64].
    CKM_SHA384_RSA_PKCS with mechanism ID[65].
    CKM_SHA512_RSA_PKCS with mechanism ID[66].
    CKM_ECDSA_SHA1 with mechanism ID[4162].
    CKM_ECDSA with mechanism ID[4161].
    CKM_RSA_PKCS_KEY_PAIR_GEN with mechanism ID[0].
    CKM_EC_KEY_PAIR_GEN with mechanism ID[4160].
    CKM_RSA_PKCS_OAEP with mechanism ID[9].
root@Ubuntu:~/new#
root@Ubuntu:~/new#
root@Ubuntu:~/new# pkcs11_app -M -m rsa -i -p 0
Mechanism Info for CKM_RSA_PKCS with mechanism ID[1].
    Minimum Key Size = 512
    Maximum Key Size = 2048
    Mechanism Capabilities: CKF_DECRYPT, CKF_SIGN,
root@Ubuntu:~/new#
root@Ubuntu:~/new# pkcs11_app -M -m rsa-oaep -i -p 0
Mechanism Info for CKM_RSA_PKCS_OAEP with mechanism ID[9].
    Minimum Key Size = 1024
    Maximum Key Size = 2048
    Mechanism Capabilities: CKF_DECRYPT,
root@Ubuntu:~/new#
root@Ubuntu:~/new#
root@Ubuntu:~/new# pkcs11_app -M -m ec -i -p 0
Mechanism Info for CKM_ECDSA with mechanism ID[4161].
    Minimum Key Size = 256
    Maximum Key Size = 384
    Mechanism Capabilities: CKF_SIGN,
root@Ubuntu:~/new#

```

- **pkcs11_app -F -p <slot-ID>**: List all objects associated with token present in slot <slot-ID>

We have 2 objects already created via the **soj_app**, which will be shown here through **pkcs11_app** find operation.

```

root@Ubuntu:~# pkcs11_app -F -p 0
None of the search option (-b -o -k) is provided. Listing all Object.
Missing option [-n]. Listing Object max upto Count = 10.
object[0] = aaab045aba60
    Label: ecc_gen_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x3
    Key Type: CKK_EC
object[1] = aaab045ab360
    Label: ecc_gen_256
    Class: CKO_PUBLIC_KEY
    Object ID: 0x3
    Key Type: CKK_EC
object[2] = aaab045aa9e0
    Label: rsa_gen_2048
    Class: CKO_PRIVATE_KEY
    Object ID: 0x2
    Key Type: CKK_RSA
object[3] = aaab045aale0
    Label: rsa_gen_2048
    Class: CKO_PUBLIC_KEY
    Object ID: 0x2
    Key Type: CKK_RSA
object[4] = aaab045a99d0
    Label: ecc_create_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x1
    Key Type: CKK_EC
object[5] = aaab045a91f0
    Label: ecc_create_256
    Class: CKO_PUBLIC_KEY
    Object ID: 0x1
    Key Type: CKK_EC
object[6] = aaab045a8870
    Label: rsa_create_2048
    Class: CKO_PRIVATE_KEY
    Object ID: 0x0
    Key Type: CKK_RSA
object[7] = aaab045a7e30
    Label: rsa_create_2048
    Class: CKO_PUBLIC_KEY
    Object ID: 0x0
    Key Type: CKK_RSA
root@Ubuntu:~#

```

• **Currently search can be made based on 3 criteria via this app:**

- o: Object type (Can be public key, private key, certificates and so on)(For now supports only Public and Private keys)
- k: Key type (Can be RSA, EC, AES and so on)(For now supports only RSA)
- b: Object Label associated with object while creating/generating.

pkcs11_app -F -o <obj-type> -k <key-type> -b <label> -p <slot-ID> : List all objects which are having object type <obj-type> of key type <key-type> and with label <label> on token present in slot <slot-ID>

```

root@Ubuntu:~# pkcs11_app -F -p 0 -o prv
Missing Option [-n]. Listing Object max upto Count = 10.
object[0] = aaaac6933a80
    Label: ecc_gen_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x3
    Key Type: CKK_EC
object[1] = aaaac6932a00
    Label: rsa_gen_2048
    Class: CKO_PRIVATE_KEY
    Object ID: 0x2
    Key Type: CKK_RSA
object[2] = aaaac69319f0
    Label: ecc_create_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x1
    Key Type: CKK_EC
object[3] = aaaac6930890
    Label: rsa_create_2048
    Class: CKO_PRIVATE_KEY
    Object ID: 0x0
    Key Type: CKK_RSA
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -F -p 0 -k ec
Missing Option [-n]. Listing Object max upto Count = 10.
object[0] = aaaac8a4ba80
    Label: ecc_gen_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x3
    Key Type: CKK_EC
object[1] = aaaac8a4b380
    Label: ecc_gen_256
    Class: CKO_PUBLIC_KEY
    Object ID: 0x3
    Key Type: CKK_EC
object[2] = aaaac8a499f0
    Label: ecc_create_256
    Class: CKO_PRIVATE_KEY
    Object ID: 0x1
    Key Type: CKK_EC
object[3] = aaaac8a49210
    Label: ecc_create_256
    Class: CKO_PUBLIC_KEY
    Object ID: 0x1
    Key Type: CKK_EC
root@Ubuntu:~#

```

- **pkcs11_app -S -k <key-type> -b <key-label> -d <Data-to-be-signed> -m <mech-ID> -p <slot-ID>**

This command will sign the <Data> with private key of type <key-type> having label <key-label> using mechanism specified by <mech-ID> with functions provided by token in slot <slot-ID>

After successful signing, the signature will be saved in file "sig.data"

RSA signing:

```
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -S -k rsa -b Device_Key -d "PKCS11 TEST DATA" -m md5-rsa -p 0
Signing...
Size of Unsigned data = 16
Signature size: 256
Signature is saved in the file sig.data:
root@Ubuntu:~#
```

ECDSA signing

```
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -S -k ec -b ecc_gen_256 -d "PKCS11 TEST DATA" -m sha1-ec -p 0
Signing...
Size of Unsigned data = 16
Signature size: 512
Signature is saved in the file sig.data:
```

- **pkcs11_app -V -k <key-type> -b <key-label> -d <Data-previously-signed> -s <signature-file> -m <mech-ID> -p <slot-ID>**

This command verifies the signature <signature-file> with public key of type <key-type> having label <key-label> using mechanism specified by <mech-ID> with functions provided by token in slot <slot-ID> by comparing the data recovered from signature to <Data-previously-signed>. This command uses openssl APIs to do the verification. Refer to the application code for more details.

<mech-ID> passed must match with the <mech-ID> passed during signature otherwise verification fails, as shown in following picture.

RSA Verification:

```
root@Ubuntu:~# pkcs11_app -V -k rsa -b Device_Key -d "PKCS11 TEST DATA" -s sig.data -m md5-rsa -p 0
Verifying...
CKM_MD5_RSA_PKCS verification success
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -V -k rsa -b Device_Key -d "PKCS11TEST DATA" -s sig.data -m md5-rsa -p 0
Verifying...
CKM_MD5_RSA_PKCS verification failure
root@Ubuntu:~#
```

ECDSA Verification:

```
root@Ubuntu:~# pkcs11_app -V -k ec -b ecc_gen_256 -d "PKCS11 TEST DAT" -s sig.data -m sha1-ec -p 0
Verifying...
sig_bytes = 64
ret = 0, CKM_ECDSA_SHA1 verification failed
root@Ubuntu:~#
root@Ubuntu:~# pkcs11_app -V -k ec -b ecc_gen_256 -d "PKCS11 TEST DATA" -s sig.data -m sha1-ec -p 0
Verifying...
sig_bytes = 64
CKM_ECDSA_SHA1 verification success
root@Ubuntu:~#
```

- **pkcs11_app -E -k <key-type> -b <key-label> -d <Data> -m <mech-ID> -p <slot-ID>**

This command will encrypt the <Data> with pulic key of type <key-type> having label <key-label> using mechanism specified by <mech-ID> with functions provided by token in slot <slot-ID>

After successful signing, the signature will be saved in file "enc.data"

```
root@Ubuntu:~/new#
root@Ubuntu:~/new# pkcs11_app -E -k rsa -b Device_Key -d "PKCS11 TEST DATA" -m rsa -p 0
Encrypting...
Encrypted data saved in enc.data
root@Ubuntu:~/new#
root@Ubuntu:~/new#
```

- **pkcs11_app -D -k <key-type> -b <key-label> -e enc.data -m <mech-ID> -p <slot-ID>**

This command will decrypt the encrypted data in "enc.data" with private key of type <key-type> having label <key-label> using mechanism specified by <mech-ID> with functions provided by token in slot <slot-ID>

After successful signing, the signature will be saved in file "enc.data"

```
root@Ubuntu:~/new#
root@Ubuntu:~/new# pkcs11_app -D -k rsa -b Device_Key -e enc.data -m rsa -p 0
Decrypting...
Decrypted Data: PKCS11 TEST DATA
root@Ubuntu:~/new#
root@Ubuntu:~/new#
```

6.5.4.2.3 mp_app

This application demonstrates how to use the following APIs:

- Get MP Public Key.
- Sign a message using MP Private Key.
- Get Message tag.

The application source code at location "secure_obj/securekey_lib/app/mp_app.c" can be used as reference for integration of these APIs.

mp_app - This application gives 3 options.

Usage:

- **mp_app -p**: Get the MP Public key and store it in a file "pub_key"
- **mp_app -s <MSG>**: Sign <MSG> with MP Priv key and store signature in file "signature"
- **mp_app -m**: Get the MP Message tag and store it in file "mtag"

```
root@Ubuntu:~#
root@Ubuntu:~# mp_app -p
Generating the MP Public Key
Public key x part = 94d9548963b8fd4f4ebd253320cc8a7ad3342b16288c18eccddc2fd1482a9b0b
Public key y part = e2d4bd22c40a7c2a41a8ca68204c0a346dde9721bb710bc3a5df31a50099c572
Public key in form of x followed by y is saved in pub_key file
root@Ubuntu:~#
root@Ubuntu:~# mp_app -s "HELLO"
```

```
Signing message 'HELLO' with MP Priv Key
HELLO in Hex = 48454c4c4f
Generated Hash = 504eb5a99c0ab4a6b27678cb6dd36bd2c365d9d5a666cc16ac5a2fbd5b5049a9
Signature part r = 37a3bac5c27a981f6826c6c3e6841fff687a838e24fc5fb9dd8809b567a5fc194
Signature part s = 2babeaa4da0192d2e6363eebc179e49dbcf933865e13a0215eb06acb380a6cdd
Signature in form of r followed by s is saved in signature file
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# mp_app -m
```

```
MP Tag = 9bac8c5220dcbec0114d651879e5d5891160fcf14e3d5a4d79403c5a1252ded9
MP Tag is saved in mptag file
root@Ubuntu:~#
root@Ubuntu:~#
```

6.5.4.2.4 mp_verify

This app uses OpenSSL APIs to verify the signature obtained by using "mp_app" application. The application source code at location "secure_obj/securekey_lib/app/mp_verify.c" can be used as reference.

mp_verify - This application verifies the signature generated by **mp_app -s**.

Usage:

mp_verify -p <pubkeyfile> -s <signaturefile> -m <mtagfile> -M <MSG>

This <MSG> must be same which is used in **mp_app -s <MSG>**

```
root@Ubuntu:~# mp_verify -s signature -p pub_key -m mtag -M "HELLO"
pub key file = pub_key, sign file = signature, mtag file = mtag, Message = HELLO
Pub Key read from file = 94d9548963b8fd4f4ebd253320cc8a7ad3342b16288c18eccddc2fd
1482a9b0be2d4bd22c40a7c2a41a8ca68204c0a346dde9721bb710bc3a5df31a50099c572
Signature read from file = 37a3bac5c27a981f6826c6c3e6841ff687a838e24fc5fb9dd8809
b567a5fc1942babeea4da0192d2e6363eebc179e49dbcf933865e13a0215eb06acb380a6cdd
Mtag read from file = 9bac8c5220dcbec0114d651879e5d5891160fcf14e3d5a4d79403c5a12
52ded9

Generated Hash = 504eb5a99c0ab4a6b27678cb6dd36bd2c365d9d5a666cc16ac5a2fbd5b5049a
9
Verified EC Signature
Verification successful
root@Ubuntu:~#
```

6.5.4.2.5 sobj_eng_app

This app uses OpenSSL APIs to show how to use Secure Object based OpenSSL Engine.

Code for this app is at “**secure_obj/secure_obj-openssl-engine/app/sobj_eng_app.c**”:

This application is internally loading RSA private key and then doing cryptographic operations using this key.

Private Key operations are offloaded to Secure Object via this engine, and Public Key operations are done through OpenSSL itself.

In Following screenshot, see creating a key via **sobj_app**. It will be used by **sobj_eng_app** (using OpenSSL APIs) to do the cryptographic operations.

This **sobj_eng_app** is internally offloading the cryptographic operation to Secure Object Library using the OpenSSL Engine based on Secure Object Library.

```

root@Ubuntu:~#
root@Ubuntu:~# sobj_app -G -m rsa-pair -s 2048 -l "Device_Key" -i 1 -w dev_key.pem
Generating the Object.
Object generated successfully handle = 3
root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~# sobj_eng_app test_key.pem
Device Key Label = test_key.pem
fopen failed.
root@Ubuntu:~#
root@Ubuntu:~# sobj_eng_app dev_key.pem
Device Key Label = dev_key.pem
Plain Text = This is test data to be tested
Starting RSA Public Encrypt....
Encryption Complete: Length of Encrypted Data = 256

Starting RSA Private Decryption....
Decryption Complete: Decrypted Text = This is test data to be tested

Starting RSA Private Encryption....
Plain Text = This is test data to be tested
Encryption Complete: Length of Encrypted Data = 256

Starting RSA Public Decryption....
Decryption Complete: Decrypted Text = This is test data to be tested

root@Ubuntu:~#
root@Ubuntu:~#
root@Ubuntu:~#

```

6.5.4.2.6 thread_test

PKCS#11 based application to test the multithreading support in PKCS#11 Library.

This application will be taking the number of threads to create as an argument, if not given by default it will create 10 threads.

thread_test <num-of-threads>

This application is making threads and each thread is doing the signing operation.

As part of signing operation each thread is doing following operations:

- Opening a R/O session with token.
- Find a RSA Private Key from token.
- Sign data using this RSA Private Key.
- Get the Public part of the RSA Private Key.
- Verify the generated signature using OpenSSL.

All threads try to do this in parallel, but if one of the thread finished its work and “Finalized” the library, all other threads will terminate, because library is not in initialized state now.

NOTE: This sequence of operations are used only for test purpose.

6.5.5 Validation

Above steps are fully validated and verified on LS1046ARDB platform.

6.5.6 Appendix

Appendix A: Steps to build the PKCS#11 Library

PKCS Library is using Secure Object Library. For steps compiling Secure Object Library, see section **Appendix B: Steps to build the Secure Object Library**.

From flexbuild environment:

```
flex-builder -c libpkcs11 -m ls1046ar db
```

Standalone Build:

1. Clone the libpkcs11 from: <https://source.codeaurora.org/external/qorik/qorik-components/libpkcs11>
2. Checkout tag “**LSDK-18.12**”.
3. Set path for cross-compile:

```
$:> export CROSS_COMPILE=<aarch64-toolchain>
```

4. Set path for Secure Object:

```
$:> export SECURE_OBJ_PATH=<path-to-secure_obj>/secure_obj/securekey_lib/out/export/
```

5. Set path for OpenSSL:

Note: For interoperability, we are verifying the signature generated by PKCS Library via OpenSSL, so reference application needs OpenSSL library, so exporting OPENSSL_PATH.

We have cloned and compiled the OpenSSL in “Steps to build the Secure Object Library,” hence only give path of that folder in OPENSSL_PATH.

```
$:> export OPENSSL_PATH=<openssl-folder>
```

6. Run make:

```
$:> make
```

This compiles the libpkcs11 and reference applications and put it into “images” folder in libpkcs11. Following images are generated:

- **libpkcs11.so** – PKCS#11 User space library.
- **pkcs11_app** – PKCS#11 Test App.
- **thread_test** - PKCS#11 application to test multithreading feature of PKCS#11 library

Appendix B: Steps to build the Secure Object Library

From flexbuild environment:

```
flex-builder -c secure_obj -m ls1046ar db
```

Standalone Build:

Order of repo compilation for Secure Object Library.

1. OP-TEE OS
 - a. Clone optee_os from: https://source.codeaurora.org/external/qorik/qorik-components/optee_os
 - b. Checkout tag “**LSDK-18.12**”
 - c. Set the path for the following:

```
$:> export CROSS_COMPILE64=<aarch64-toolchain>
```

- d. Now make.

```
$:> make CFG_ARM64_core=y PLATFORM=ls-ls1046ardb ARCH=arm
```

2. OP-TEE Client

- Clone optee_client from: https://source.codeaurora.org/external/qoriq/qoriq-components/optee_client
- Checkout tag “**LSDK-18.12**”
- Set path for the following:

```
$:> export CROSS_COMPILE=<aarch64-toolchain-path->
```

- d. Now make.

```
$:> make
```

3. OpenSSL:

- Clone openssl from: <https://source.codeaurora.org/external/qoriq/qoriq-components/openssl>
- Checkout tag “**LSDK-18.12**”
- Set path for the following:

```
$:> export CROSS_COMPILE=<aarch64-toolchain-path->
```

- d. Run configure as follows:

```
$:>. /Configure shared linux-aarch64
```

- e. Run make

```
$:> make
```

4. Secure Object:

- Clone secure_obj from: https://source.codeaurora.org/external/qoriq/qoriq-components/secure_obj
- Checkout tag “**LSDK-18.12**”

- Secure Object Library code - securekey_lib
- Secure Object Trusted Application code - secure_storage_ta
- Secure Key Dev Kernel Module - securekeydev
- Secure Object OpenSSL Engine - secure_obj-openssl-engine

There is script “compile.sh” which compiles all above components and put all binaries in “images.”

- c. Follow the below compilation steps:

- export CROSS_COMPILE path:

```
$:> export CROSS_COMPILE= <aarch64-toolchain-path->
```

- export ARCH path:

```
$:> export ARCH=arm64
```

- Set the paths from OP-TEE OS:

```
$:> export TA_DEV_KIT_DIR=<path-to-optee-os>/optee_os/out/arm-plat-ls/export-ta_arm64/
```

- Set path for OP-TEE Client

```
$:> export OPTEE_CLIENT_EXPORT=<path-to-optee-client>/optee_client/out/export/
```

- Set path for Secure Storage:

```
$:> export SECURE_STORAGE_PATH=<path-to-secure_obj>/secure_obj/secure_storage_ta/ta/
```

- Set path for OpenSSL:

```
$:> export OPENSSL_PATH=<openssl-folder-path>
```

- Set path for Linux Code (Used Flexbuild kernel for this):

```
$:> export KERNEL_SRC=<path-in-flexbuild-containing-kernel-source-code>
For example,
$:> export KERNEL_SRC=/home/b42224/flexbuild_1812/flexbuild/build/linux/linux/arm64/lib/
modules/4.9.62/source
```

- Set path for Linux Build Directory (Used Flexbuild kernel for this):

```
$:> export KERNEL_BUILD=<path-in-flexbuild-containing-kernel-build>
For example,
$:> export KERNEL_BUILD=/home/b42224/flexbuild_1812/flexbuild/build/linux/linux/arm64/lib/
modules/4.9.62/build
```

- Run “/compile.sh” (It will compile TA, library and Kernel Module).

```
$:> ./compile.sh
```

This will compile all the binaries and put them into the images folder in `secure_obj`. After compilation, images folder have the following:

- **b05bcf48-9732-4efa-a9e0-141c7c888c34.ta** - Trusted application for Secure Object library.
- **libsecure_obj.so** - User space Secure Object Library
- **sobj_app** - Application for creating and erasing objects.
- **mp_app** - Application for getting MP Public Key, signing using MP Private key and getting the MP tag.
- **mp_verify** - Application for verifying the signature generated through mp_app.
- **securekeydev.ko** - Kernel Module for offloading MP Key feature to CAAM. Binaries to be placed at following locations in rootfs.
- **libeng_secure_obj** - Secure Object based OpenSSL engine offloading Private Key Operations to the Secure Object Library.
- **sobj_eng_app** - This app uses OpenSSL APIs to show how to use Secure Object based OpenSSL Engine. This application is loading the private key and then doing cryptographic operations using this key.

Chapter 7

Linux kernel

Introduction

The Linux kernel is a monolithic Unix-like computer operating system kernel. It is the central part of Linux operating systems that are extensively used on PCs, servers, handheld devices and various embedded devices such as routers, switches, wireless access points, set-top boxes, smart TVs, DVRs, and NAS appliances. It manages tasks/applications running on the system and manages system hardware. A typical Linux system looks like this:

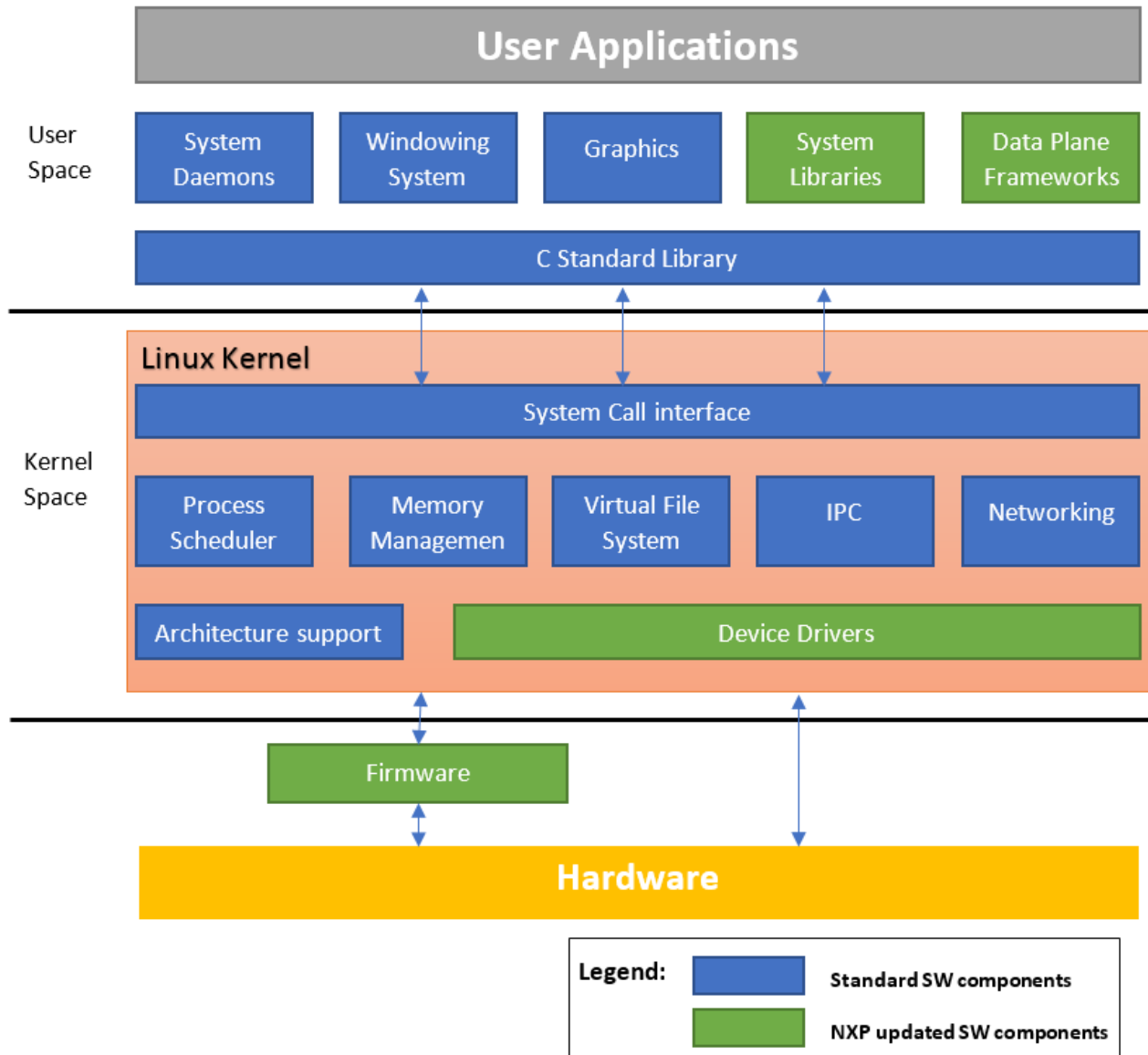


Figure 29. Typical Linux System

The Linux kernel was created in 1991 by Linus Torvalds and released as an open source project under GNU General Public License (GPL) version 2. It rapidly attracted developers around the world. In 2015 the Linux kernel has received contributions from nearly 12,000 programmers from more than 1,200 companies. The software is officially released on <http://www.kernel.org>

website through downloadable packages and GIT repositories. A general Linux kernel introduction from kernel.org can also be found at <https://www.kernel.org/doc/html/latest/admin-guide/README.html>.

Kernel Releases and relationship with Layerscape SDK

There are different Linux kernel releases coming from different sources. Below we listed the ones that are related to the LSDK kernel.

Kernel.org official kernel releases

- **Mainline**

Mainline tree is maintained by Linus Torvalds. It's the tree where all new features are introduced and where all the exciting new development happens. New mainline kernels are released every 2-3 months.

- **Longterm (LTS)**

There are usually several "longterm maintenance" kernel releases provided for the purposes of backporting bugfixes for older kernel trees. Only important bugfixes are applied to such kernels and they don't usually see very frequent releases, especially for older trees.

Refer to <https://www.kernel.org/category/releases.html> for the current maintained Longterm releases.

Linaro LSK kernel release

Linaro is an open organization focused on improving Linux on ARM. They are also providing a Linux kernel release called Linaro Stable Kernel (LSK). It is based on kernel.org Longterm kernel releases and included ARM related features developed by Linaro. Normally these features are generic kernel features for the ARM architecture. Please refer to <https://wiki.linaro.org/LSK> for more information about the LSK releases.

NXP Layerscape SDK kernel

NXP's SDK kernel often contains patches that are not upstream yet so essentially the LSDK kernel is an enhanced Linaro LSK which is in turn an enhanced kernel.org LTS. In order to fully utilize the ARM open source eco-system. The kernel versions provided in NXP LSDK will be chosen from the kernel.org Longterm releases to include the important bugfixes backported. It will also include generic ARM kernel features provided by the Linaro LSK release which could be important for some users.

Getting the LSDK kernel source code

With Layerscape SDK, NXP owned/updated software components are published on github. You can use git commands to get the latest kernel source code.

- Install git command if not there already. For example, on Ubuntu:

```
$ sudo apt-get install git
```

- Clone the Linux kernel source code with git.

```
$ git clone https://source.codeaurora.org/external/qoriq/qoriq-components/linux
```

- Checkout the desired kernel version. As we provide support to the two latest LTS kernel versions in the SDK, it is possible that the default one is not your desired kernel version

```
$ cd linux
$ git branch
```

Check the name of the current branch. If it is not the Kernel version you want, use the following command to checkout your desired kernel version: x.y

```
$ git checkout -b linux-x.y origin/linux-x.y
```

7.1 Configuring and building

Configuring and building the Linux kernel is controlled by the Kbuild sub-system. You can find documents describing the internal of Kbuild sub-system under the Documentation/kbuild/ folder in the Linux source code tree if you are adding new files or new configure options to the kernel. Otherwise as a user of Linux kernel, you probably only want to know how to fine tune the kernel configuration base on your system requirements and build new kernel image with updated configuration. These are done through *make* commands, below we will talk about *make* commands you probably need to know as a kernel user.

Environment setting for cross-compiling

This following settings are applicable when you are configuring and building kernel on a different architecture from the target. For example, compiling an ARMv8 kernel on an X86 computer. If you are compiling the kernel natively on a machine of the same architecture as the target, you should skip this chapter.

- Install the cross compiler of your distribution
- Specify the target architecture in ARCH environment variable
- Specify the prefix (and path) of a cross compiler in CROSS_COMPILE environment variable

```
$ export CROSS_COMPILE=/path/to/dir/tool-chain-prefix-
```

Or just the prefix if the cross-compiler commands are already in the execution PATH.

```
$ export CROSS_COMPILE=tool-chain-prefix-
```

For example, the commands needed on Ubuntu Linux will be like:

- 64-bit ARM:

```
$ sudo apt-get install gcc-aarch64-linux-gnu
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ export ARCH=arm64
```

- 32-bit ARM (ARMv7 / 32-bit mode of ARMv8):

```
$ sudo apt-get install gcc-arm-linux-gnueabi
$ export CROSS_COMPILE=arm-linux-gnueabi-
$ export ARCH=arm
```

For the shell environment variables exported above, you can also include them directly in each make command you use. E.g. `$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make {targets}`. Exporting them will save effort if you are using make in kernel frequently.

Configuring kernel

The current kernel configuration for a kernel source tree will be kept in a hidden file named `.config` at the top level of the kernel source code after you changed the configuration with any of the `make config` command variants. You can copy it directly from one kernel source tree to another with the same kernel version to duplicate the configuration exactly. Also, you can edit it with a text editor, in which you can see a list of `CONFIG_*` symbols corresponding to each of the kernel configure option.

The following targets from the Linux kernel Kbuild framework are used to load the default kernel configuration for LSDK:

- `defconfig/${PLATFORM}_defconfig`

Create the `.config` file by using the default config options of the architecture or platform defined in the `arch/${ARCH}/configs/` directory. This normally includes all the device drivers needed for the architecture or platform.

- `${FRAGMENT}.config`

Merge a configuration fragment that enables certain features into the `.config` file.

Specific command to load the default configuration of different platforms for LSDK will be:

- For Layerscape ARMv8 platforms in 64bit mode:

```
$ make defconfig lsdk.config
```

- For Layerscape ARMv7 platforms:

```
$ make multi_v7_defconfig multi_v7_lpae.config lsdk.config
```

- For Layerscape ARMv8 platforms in 32bit mode:

```
$ make multi_v7_defconfig multi_v7_lpae.config multi_v8.config lsdk.config
```

To further fine tune the configuration base on your system need you can use the following make commands.

- `$ make menuconfig`

Choose configure options in text based color menus, radiolists & dialogs. It is a good way to navigate through all the selectable kernel configure options in a well-organized human-readable hierarchy and you can get a description of every option when it is highlighted by selecting the <Help> button. In the device driver part of this User's Manual we also provided the path to the configure options needed for a feature to work in the menuconfig.

- `$ make ${FRAGMENT}.config`

You can also utilize this capability to enable options for a specific feature in your custom kernel configuration quickly without selecting each one of them in the menuconfig. In the device driver part of this User's Manual, we listed the `CONFIG_*` symbols needed by a specific feature/driver. Put these symbols with “=y” or “=m” depending on if you want these features/drivers to be built-in or built as loadable kernel module into a `${FEATURE}.config` file under `arch/${ARCH}/configs/` directory. Run `$ make ${FEATURE}.config` command, it will enable all these listed kernel configure options together.

Building kernel

Building the kernel is simple.

- To build kernel images and device tree images.

```
make
```

- To build loadable kernel modules:

```
make modules
```

You can supply `-j <NUM>` option to the above make commands to spin `NUM` concurrent threads to reduce build time on multicore systems.

After a successful build:

- Compiled kernel images are in `arch/${ARCH}/boot/` folder.
- Compiled device trees (dtb files) are in `arch/${ARCH}/boot/dts` folder.
- Compiled kernel modules are spread out in driver folders. You can extract them to a specific folder (e.g. `/folder/to/install`) by using command:

```
$ make modules_install INSTALL_MOD_PATH=/folder/to/install
```

Install new kernel and modules

The path or naming convention of kernel images and modules are different for different Linux distributions. The following instructions are based on the convention of LSDK.

Using the flex-build scripts

- Copy kernel image, dtb and kernel modules from your kernel tree to the staging folder of the flexbuild script (Skip if you are using the *flexbuild -c linux* to build the kernel directly).

- **For 64-bit ARM:**

```
$ cp arch/arm64/boot/Image.gz ${path-to-flexbuild}/build/linux/kernel/arm64/  
$ cp arch/arm64/boot/dts/freescale/*.dtb ${path-to-flexbuild}/build/linux/kernel/arm64/  
$ make modules_install INSTALL_MOD_PATH=${path-to-flexbuild}/build/linux/kernel/arm64/
```

- **For 32-bit ARM:**

```
$ cp arch/arm/boot/Image.gz ${path-to-flexbuild}/build/linux/kernel/arm/  
$ cp arch/arm/boot/dts/ls*.dtb ${path-to-flexbuild}/build/linux/kernel/arm/  
$ make modules_install INSTALL_MOD_PATH=${path-to-flexbuild}/build/linux/kernel/arm/
```

- Regenerate the boot partition and rootfs (for commands below: \${ARCH} = arm32 | arm64)

```
$ flex-builder -i mkbootpartition -a ${ARCH}  
$ flex-builder -i merge-component -a ${ARCH}  
$ flex-builder -i compressrfs -a ${ARCH}
```

- Use flex-installer to deploy the updated boot partition and rootfs to the device following the [Layerscape SDK user guide](#).

Update the target filesystem directly

This can be more convenient if you are compiling the kernel on the target device locally or you can easily update the filesystem of target device remotely (e.g. using scp, tftp, or etc.).

- Copy your Image file to /boot folder on the target using cp if compiled locally; Use any available remote update approach if compiled remotely.
- Copy dtb files to /boot folder on the target using cp if compiled locally; Use any available remote update approach to do the same if compiled remotely.
- Update kernel modules. (Note: kernel modules are required to be updated when you updated the kernel image).
 - If you compiled the kernel on the target device locally. Use the command below:

```
$ make modules_install
```

- If you compiled the kernel remotely. Do the following:
 - Install the modules into a temporary folder (e.g. /tmp/lsdk/).

```
$ make modules_install INSTALL_MOD_PATH=/tmp/lsdk/
```

- Transfer the lib/ directory from the temporary location above to the target device using any file transfer approach and put it in the / path of the filesystem.

7.2 Device Drivers

7.2.1 CAAM Direct Memory Access (DMA)

Description

The CAAM DMA module implements a DMA driver that uses the CAAM DMA controller to provide both SG and MEMCPY DMA capability to be used by the platform. It is based on the CAAM JR interface that must be enabled in the *kernel config* as a prerequisite for the CAAM DMA driver.

The driver is based on the DMA engine framework and it is located under the DMA Engine support category in the kernel config menu.

Kernel Configure Options

Tree Overview

To enable the CAAM DMA module, set the following options for `make menuconfig`:

```

- *- Cryptographic API --->
  [*] Hardware crypto devices --->
    <*> Freescale CAAM-Multicore driver backend
    <*> Freescale CAAM Job Ring driver backend
Device Drivers --->
  <*> DMA Engine support --->
    <*> CAAM DMA engine support

```

NOTE

Be aware that the CAAM DMA driver depends on the CAAM and CAAM JR drivers, which also have to be enabled.

Identifier

The following configure identifier is used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_CRYPTODEV_FSL_CAAM_DMA	y/m/n	n	CAAM DMA engine support

Device Tree Node

Below is an example device tree node required by this feature.

```

caam_dma {
    compatible = "fsl,sec-v5.4-dma";
};

```

Source Files

The following source file is related to this feature in the Linux kernel.

Source File	Description
drivers/dma/caam_dma.c	The CAAM DMA driver

Verification in Linux

On a successful probing, the driver will print the following message in `dmesg`:

```
[ 1.443940] caam-dma 1700000.crypto:caam_dma: caam dma support with 4 job rings
```

Additionally, you can also run the following commands:

```
ls -l /sys/class/dma/
total 0
lrwxrwxrwx 1 root root 0 Jan 1 1970 dma0chan0 -> ../../devices/platform/soc/1700000.crypto/1700000.crypto:caam_dma/dma/dma0chan0
lrwxrwxrwx 1 root root 0 Jan 1 1970 dma0chan1 -> ../../devices/platform/soc/1700000.crypto/1700000.crypto:caam_dma/dma/dma0chan1
lrwxrwxrwx 1 root root 0 Jan 1 1970 dma0chan2 -> ../../devices/platform/soc/1700000.crypto/1700000.crypto:caam_dma/dma/dma0chan2
lrwxrwxrwx 1 root root 0 Jan 1 1970 dma0chan3 -> ../../devices/platform/soc/1700000.crypto/1700000.crypto:caam_dma/dma/dma0chan3
```

Component Testing

To test both the SG and memcpy capability of the CAAM DMA driver use the `dmatest` module provided by the kernel.

Build dmatest

Build the `dmatest` utility as a module by running the command:

```
$ make menuconfig
```

Then select from the kernel `menuconfig` to build the `dmatest.ko` as a module:

```
Device Drivers --->
  <*> DMA Engine support --->
    <M>   DMA Test client
```

Configure dmatest

Before testing insert the module:

```
$ insmod dmatest.ko
```

The configure the `dmatest`. There is a general configuration that applies for both the sg and memcpy functionality:

```
$ echo 1 > /sys/module/dmatest/parameters/max_channels
$ echo 2000 > /sys/module/dmatest/parameters/timeout
$ echo 0 > /sys/module/dmatest/parameters/noverify
$ echo 4 > /sys/module/dmatest/parameters/threads_per_chan
$ echo 0 > /sys/module/dmatest/parameters/dmatest
$ echo 1 > /sys/module/dmatest/parameters/iterations
$ echo 2000 > /sys/module/dmatest/parameters/test_buf_size
```

The above configuration is self explanatory except a few:

If you set the 'noverify' parameter to 0 it will not perform check of the copied buffer at the end of each testing round. This should be used for performance testing. Set the 'noverify' parameter to 1 for functional testing.

Set the 'dmatest' parameter to 0 to test the memcpy functionality and to 1 to test the sg functionality.

Perform the test

To perform the test simply run the command:

```
$ echo 1 > /sys/module/dmatest/parameters/run
```

Depending on the type of test performed (sg/memcpy) the output may vary. Here is an example of output obtained with the above parameters:

```
[ 72.113769] dmatest: Started 4 threads using dma0chan0
[ 72.105334] dmatest: dma0chan0-copy0: summary 1 tests, 0 failures 9009 iops 9009 KB/s (0)
[ 72.113649] dmatest: dma0chan0-copy1: summary 1 tests, 0 failures 119 iops 119 KB/s (0)
[ 72.114927] dmatest: dma0chan0-copy2: summary 1 tests, 0 failures 24390 iops 0 KB/s (0)
[ 72.115098] dmatest: dma0chan0-copy3: summary 1 tests, 0 failures 37037 iops 0 KB/s (0)
```

7.2.2 Enhanced Secured Digital Host Controller (eSDHC)

Description

The enhanced secured host controller (eSDHC) provides an interface between the host system and the SD/SDIO cards and eMMC devices.

The eSDHC device driver supports either kernel built-in or module.

Kernel Configure Options

Tree View

Kernel Configure Options Tree View	Description
<pre>Device Drivers ---> <*> MMC/SD/SDIO card support ---> <*> MMC block device driver (8) Number of minors per block device [*] Use bounce buffer for simple hosts</pre>	Enables SD/MMC block device driver support
<pre>*** MMC/SD/SDIO Host Controller Drivers *** <*> Secure Digital Host Controller Interface support <*> SDHCI platform and OF driver helper [*] SDHCI OF support for the NXP eSDHC controller</pre>	Enables NXP eSDHC driver support

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_MMC	y/n	y	Enable SD/MMC bus protocol
CONFIG_MMC_BLOCK	y/n	y	Enable SD/MMC block device driver support
CONFIG_MMC_BLOCK_MINORS	integer	8	Number of minors per block device

Table continues on the next page...

Table continued from the previous page...

Option	Values	Default Value	Description
CONFIG_MMC_BLOCK_BOUNCE	y/n	y	Enable continuous physical memory for transmit
CONFIG_MMC_SDHCI	y/n	y	Enable generic sdhc interface
CONFIG_MMC_SDHCI_PLTFM	y/n	y	Enable common helper function support for sdhci platform and OF drivers
CONFIG_MMC_SDHCI_OF_ESDHC	y/n	y	Enable NXP eSDHC support

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/mmc/host/sdhci.c	Linux SDHCI driver support
drivers/mmc/host/sdhci-pltfm.c	Linux SDHCI platform devices support driver
drivers/mmc/host/sdhci-of-esdhc.c	Linux eSDHC driver

Device Tree Binding

Property	Type	Status	Description
compatible	String	Required	Should be 'fsl,esdhc'
reg	integer	Required	Register map

example:

```
esdhc: esdhc@1560000 {
    compatible = "fsl,ls1046a-esdhc", "fsl,esdhc";
    reg = <0x0 0x1560000 0x0 0x10000>;
    interrupts = <GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&clockgen 2 1>;
    voltage-ranges = <1800 1800 3300 3300>;
    sdhci,auto-cmd12;
    big-endian;
    bus-width = <4>;
};
```

Verification in U-Boot

The U-Boot log

```
=> mmcinfo
Device: FSL_SDHC
Manufacturer ID: 74
OEM: 4a45
Name: SDC
Tran Speed: 50000000
```

```
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.5 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
=> mw.l 81000000 11111111 100
=> mw.l 82000000 22222222 100
=> cmp.l 81000000 82000000 100
word at 0x0000000081000000 (0x11111111) != word at 0x0000000082000000 (0x22222222)
Total of 0 word(s) were the same
=> mmc write 81000000 0 2

MMC write: dev # 0, block # 0, count 2 ... 2 blocks written: OK
=> mmc read 82000000 0 2

MMC read: dev # 0, block # 0, count 2 ... 2 blocks read: OK
=> cmp.l 81000000 82000000 100
Total of 256 word(s) were the same
=>
```

Verification in Linux

Set U-Boot environment

```
=> setenv hwconfig sdhc
```

The linux booting log

```
.....
[ 3.913163] sdhci: Secure Digital Host Controller Interface driver
[ 3.919339] sdhci: Copyright(c) Pierre Ossman
[ 3.931467] sdhci-pltfm: SDHCI platform and OF driver helper
[ 3.938900] sdhci-esdhc 1560000.esdhc: No vmmc regulator found
[ 3.944728] sdhci-esdhc 1560000.esdhc: No vqmmc regulator found
[ 3.978676] mmc0: SDHCI controller on 1560000.esdhc [1560000.esdhc] using ADMA 64-bit
[ 4.197784] mmc0: new high speed SDHC card at address b368
[ 4.203502] mmcblk0: mmc0:b368 SDC 7.45 GiB
```

Partition the card with fdisk

```
~# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.26.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x5a5f34b3.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (1-4, default 1):
```

```
First sector (2048-15628287, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-15628287, default 15628287):
```

```
Created a new partition 1 of type 'Linux' and of size 7.5 GiB.
```

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() [ 410.501876] mmcblk0: p1
to re-read partition table.
Syncing disks.
```

```
~#
~# fdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 7.5 GiB, 8001683456 bytes, 15628288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5a5f34b3
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/mmcblk0p1		2048	15628287	15626240	7.5G	83	Linux

Format the card with mkfs

```
~# mkfs.ext2 /dev/mmcblk0p1
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: [ 37.611042] random: nonblocking pool is initialized
done
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
488640 inodes, 1953280 blocks
97664 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2000683008
60 block groups
32768 blocks per group, 32768 fragments per group
8144 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

~#
```

Mount, read, and write

```
~# mount /dev/mmcblk0p1 /mnt/
~# ls /mnt/
lost+found
~# cp -r /lib /mnt/
~# sync
~# ls /mnt/
lib lost+found
```

```
~# umount /dev/mmcblk0p1
~# ls /mnt/
~#
```

Known Bugs, Limitations, or Technical Issues

1. Call trace of more than 120 seconds task blocking when running iozone to test card performance. This is not issue and use below command to disable the warning.

```
echo 0 > /proc/sys/kernel/hung_task_timeout_secs
```

2. Layerscape boards could not provide a power cycle to SD card but according to SD specification, only a power cycle could reset the SD card working on UHS-I speed mode. When the card is on UHS-I speed mode, this hardware problem may cause unexpected result after board reset. The workaround is using power off/on instead of reset when using SD UHS-I card.
3. Transcend 8G class 10 SDHC card has some compatibility issue. It is observed it could not work on 50 MHz high-speed mode on LS2 boards, but other brand SD cards (Sandisk, Kingston, Sony ...) worked fine. Reducing SD clock frequency could also resolve the issue. The workaround is using other kind SD cards instead.
4. After sleep of LS1046ARDB, the card will get below interrupt timeout issue. This is hardware issue. CMD18 (multiple blocks read) has hardware interrupt timeout issue.

```
mmc0: Timeout waiting for hardware interrupt.
```

5. Linux MMC stack does not have SD UHS-II support currently. It could not handle SD UHS-II card well. If UHS-I support is enabled in eSDHC dts node, the driver may make SD UHS-II card enter 1.8v mode. Only a power cycle could reset the card, so use power off/on instead of reset for SD UHS-II card if UHS-I support is enabled in eSDHC dts node.
6. For LS1012ARDB RevD and later versions, I2C reading for DIP switch setting is not reliable so U-Boot could not enable/disable SDHC2 automatically. If SDHC2 is used, "esdhc1" should be set in U-Boot hwconfig environment to enable it manually.

7.2.3 IEEE 1588

Description

From IEEE 1588 perspective, the components required are:

1. IEEE 1588 extensions to the gianfar driver or DPAA/DPAA2 driver.
2. A stack application for IEEE 1588 protocol.

IEEE 1588 device driver supports either kernel built-in or module.

Kernel Configure Options

Tree View

1. eTSEC

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> PTP clock support ---> <*> Freescale QorIQ 1588 timer as PTP clock</pre>	Enable QorIQ PTP clock driver

2. DPAA

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> PTP clock support ---> <*> Freescale QorIQ 1588 timer as PTP clock</pre>	Enable QorIQ PTP clock driver
<pre>Device Drivers ---> Network device support ---> Ethernet driver support ---> DPAA Ethernet ---> [*] Linux compliant timestamping</pre>	Enable HW timestamping support for DPAA ethernet

3. DPAA2

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> Staging drivers ---> <*> Freescale DPAA2 devices <*> Freescale DPAA2 Ethernet <*> Freescale DPAA2 as PTP clock</pre>	Enable QorIQ DPAA2 PTP clock driver

Compile-time Configuration Options

1. eTSEC

Option	Values	Default Value	Description
CONFIG_GIANFAR	y/n/m	y	eTSEC ethernet driver
CONFIG_PTP_1588_CLOCK_QORIQ	y/n/m	y	QorIQ PTP clock driver

2. DPAA

Option	Values	Default Value	Description
CONFIG_FSL_SDK_DPAA_ETH	y/n/m	y	DPAA ethernet driver
CONFIG_PTP_1588_CLOCK_QORIQ	y/n/m	y	QorIQ PTP clock driver
CONFIG_FSL_DPAA_TS	y/n	n	DPAA HW timestamping support

3. DPAA2

Option	Values	Default Value	Description
CONFIG_FSL_DPAA2_ETH	y/n/m	y	DPAA2 ethernet driver
CONFIG_FSL_DPAA2_PTP_CLOCK	y/n/m	y	DPAA2 PTP clock driver

Source Files

The driver source is maintained in the Linux kernel source tree.

1. eTSEC

Source File	Description
drivers/net/ethernet/freescale/gianfar.c	eTSEC Ethernet driver
drivers/ptp/ptp_qorIQ.c	QorIQ PTP clock driver

2. DPAA

Source File	Description
drivers/net/ethernet/freescale/dpaa/dpaa_eth.c	DPAA ethernet driver
drivers/ptp/ptp_qorIQ.c	QorIQ PTP clock driver

3. DPAA2

Source File	Description
drivers/staging/fsl-dpaa2/ethernet/dpaa2-eth.c	DPAA2 ethernet driver
drivers/staging/fsl-dpaa2/rtc/rtc.c	DPAA2 PTP clock driver

Device Tree Binding

1. eTSEC / DPAA

Property	Type	Status	Description
compatible	String	Required	"fsl,etsec-ptp" for eTSEC PTP, "fsl,fman-ptp-timer" for DPAA PTP
reg	integer	Required	Register map

2. DPAA2

NA.

Verification in Linux

Connect Ethernet interfaces of two boards with back-to-back method (for example, eth0 to eth0).

One board runs as master and the other one runs as slave.

• The linux booting log

```
...
[ 3.568756] pps pps0: new PPS source ptp1
...
```

• Check PTP clock and timestamping capability

```
# ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
    hardware-transmit      (SOF_TIMESTAMPING_TX_HARDWARE)
```

```
hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
  off                  (HWTSTAMP_TX_OFF)
  on                   (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
  none                 (HWTSTAMP_FILTER_NONE)
  all                  (HWTSTAMP_FILTER_ALL)
```

• On the master side

```
# ifconfig eth0 10.10.10.1
# ptpd2 -i eth0 -o /dev/ptp1 -MV
```

• On the slave side

```
# ifconfig eth0 10.10.10.2
# ptpd2 -i eth0 -o /dev/ptp1 -sV --servo:kp=0.32 --servo:ki=0.05
```

The slave side would print synchronization messages.

Known Bugs, Limitations, or Technical Issues
No

7.2.4 Integrated Flash Controller (IFC)

7.2.4.1 Integrated Flash Controller NOR Flash User Manual

Description
NXP’s Integrated Flash Controller can be used to connect various types of flashes e.g. NOR/NAND on board for boot functionality as well as data storage.

U-Boot Configuration
Compile time options

Below are major u-boot configuration options related to this feature defined in platform specific config files under include/configs/ directory.

Option Identifier	Description
CONFIG_FSL_IFC	Enable IFC support
CONFIG_FLASH_CFI_DRIVER	Enable CFI Driver for NOR Flash devices
CONFIG_SYS_FLASH_CFI	
CONFIG_SYS_FLASH_EMPTY_INFO	

Source Files
The following source files are related to this feature in u-boot.

Source File	Description
./drivers/misc/fsl_ifc.c	Set up the different chip select parameters from board header file
drivers/mtd/cfi_flash.c	CFI driver support for NOR flash devices

Kernel Configure Options

Tree View

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel

Kernel Configure Tree View Options	Description
<pre> Device Drivers ---> <*> Memory Technology Device (MTD) support ---> [*] MTD partitioning support [*] Command line partition table parsing <*> Flash partition map based on OF description <*> Direct char device access to MTD devices *- Common interface to block layer for MTD 'translation layers' <*> Caching block device access to MTD devices < > FTL (Flash Translation Layer) support RAM/ROM/Flash chip drivers ---> <*> Detect flash chips by Common Flash Interface (CFI) probe <*> Support for Intel/Sharp flash chips <*> Support for AMD/Fujitsu/Spansion flash chips Mapping drivers for chip access ---> <*> Flash device in physical memory map based on OF description </pre>	These options enable CFI support for NOR Flash under MTD subsystem and Integrated Flash Controller support on Linux
<pre> File systems ---> [*] Miscellaneous filesystems ---> <*> Journalling Flash File System v2 (JFFS2) support </pre>	This option enables JFFS2 file system support for MTD Devices

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Special Configure needs to be enabled("Y") for LS1021. Please find in below table with default value as "N"

Option	Values	Default Value	Description
CONFIG_FSL_IFC	Y/N	Y	Integrated Flash Controller support
CONFIG_MTD	Y/N	Y	Memory Technology Device (MTD) support
CONFIG_MTD_PARTITIONS	Y/N	Y	MTD partitioning support
CONFIG_MTD_CMDLINE_PARTS	Y/N	Y	Allow generic configuration of the MTD partition tables via the kernel command line.
CONFIG_MTD_OF_PARTS	Y/N	Y	This provides a partition parsing function which derives the partition map from the children of the flash nodes described in Documentation/powerpc/booting-without-of.txt
CONFIG_MTD_CHAR	Y/N	Y	Direct char device access to MTD devices
CONFIG_MTD_BLOCK	Y/N	Y	Caching block device access to MTD devices
CONFIG_MTD_CFI	Y/N	Y	Detect flash chips by Common Flash Interface (CFI) probe
CONFIG_MTD_GEN_PROBE	Y/N	Y	NA
CONFIG_MTD_MAP_BANK_WIDTH_1	Y/N	Y	Support 8-bit buswidth
CONFIG_MTD_MAP_BANK_WIDTH_2	Y/N	Y	Support 16-bit buswidth
CONFIG_MTD_MAP_BANK_WIDTH_4	Y/N	Y	Support 32-bit buswidth
CONFIG_MTD_PHYSMAP_OF	Y/N	Y	Flash device in physical memory map based on OF description
CONFIG_FTL	Y/N	N	FTL (Flash Translation Layer) support
CONFIG_MTD_CFI_INTELEXT	Y/N	Y	Support for Intel/Sharp flash chips
CONFIG_MTD_CFI_AMDSTD	Y/N	Y	Support for AMD/Fujitsu/Spansion flash chips

Device Tree Binding

Documentation/devicetree/bindings/memory-controllers/fsl/ifc.txt

Flash partitions are specified by platform device tree.

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/memory/fsl_ifc.c	Integrated Flash Controller driver to handle error interrupts
drivers/mtd/mtdpart.c	Simple MTD partitioning layer
Table continues on the next page...	

Table continued from the previous page...

Source File	Description
drivers/mtd/mtdblock.c	Direct MTD block device access
drivers/mtd/mtdchar.c	Character-device access to raw MTD devices.
drivers/mtd/ofpart.c	Flash partitions described by the OF (or flattened) device tree
drivers/mtd/ftl.c	FTL (Flash Translation Layer) support
drivers/mtd/chips/cfi_probe.c	Common Flash Interface probe
drivers/mtd/chips/cfi_util.c	Common Flash Interface support
drivers/mtd/chips/cfi_cmdset_0001.c	Support for Intel/Sharp flash chips
drivers/mtd/chips/cfi_cmdset_0002.c	Support for AMD/Fujitsu/Spansion flash chips

Verification in U-Boot

Test the Read/Write/Erase functionality of NOR Flash

1. Boot the u-boot with above config options to get NOR Flash access enabled. Check this in boot log,

```
FLASH: * MiB
```

where * is the size of NOR Flash

2. Erase NOR Flash
3. Make test pattern on memory e.g. DDR
4. Write test pattern on NOR Flash
5. Read the test pattern from NOR Flash to memory e.g. DDR
6. Compare the test pattern data to verify functionality.

Test Log :

Test log with initial u-boot log removed

```
--
--

FLASH: 128 MiB

--
--
/* u-boot prompt */
=> mw.b 80000000 0xa5 10000
=> md 80000000
80000000: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000010: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000020: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000030: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
=> protect off all
Un-Protect Flash Bank # 1
=> erase 0x58410000 +0x10000

. done
Erased 1 sectors
```

```
=> cp.b 80000000 0x58410000 10000
Copy to Flash... 9....8....7....6....5....4....3....2....1....done
=> cmp.b 80000000 0x58410000 10000
Total of 65536 bytes were the same
=>
```

Verification in Linux

To cross check whether IFC NOR driver has been configured in the kernel or not, see the kernel boot log with following entries. Please note mtd partition number can be changed depending upon device tree.

```
[ 2.368207] 60000000.nor: Found 1 x16 devices at 0x0 in 16-bit bank. Manufacturer ID 0x000001
Chip ID 0x002801
[ 2.378219] Amd/Fujitsu Extended Query Table at 0x0040
[ 2.383374] Amd/Fujitsu Extended Query version 1.3.
[ 2.388427] number of CFI chips: 1
[ 2.391835] 8 cmdlinepart partitions found on MTD device 60000000.nor
[ 2.398277] Creating 8 MTD partitions on "60000000.nor":
[ 2.403591] 0x0000000000000-0x0000001000000 : "nor_bank0_rcw"
[ 2.409553] 0x0000001000000-0x0000001000000 : "nor_bank0_uboot"
[ 2.415653] 0x0000001000000-0x0000002000000 : "nor_bank0_kernel"
[ 2.421839] 0x0000002000000-0x0000004000000 : "nor_bank0_rootfs"
[ 2.428027] 0x0000004000000-0x0000004100000 : "nor_bank4_rcw"
[ 2.433948] 0x0000004100000-0x0000005000000 : "nor_bank4_uboot"
[ 2.440043] 0x0000005000000-0x0000006000000 : "nor_bank4_kernel"
[ 2.446228] 0x0000006000000-0x0000008000000 : "nor_bank4_rootfs"
```

Note: NOR address and number of partition will vary from SoC to SoC supported in LSDK

To verify NOR flash device accesses see the following test,

```
[root@ root]# cat /proc/mtd
dev:      size  erasesize  name
mtd0: 00100000 00020000 "nor_bank0_rcw"
mtd1: 00f00000 00020000 "nor_bank0_uboot"
mtd2: 01000000 00020000 "nor_bank0_kernel"
mtd3: 02000000 00020000 "nor_bank0_rootfs"
mtd4: 00100000 00020000 "nor_bank4_rcw"
mtd5: 00f00000 00020000 "nor_bank4_uboot"
mtd6: 01000000 00020000 "nor_bank4_kernel"
mtd7: 02000000 00020000 "nor_bank4_rootfs"
mtd8: 01000000 00040000 "nand_uboot"
mtd9: 01000000 00040000 "nand_kernel"
mtd10: 02000000 00040000 "nand_free"
mtd11: 00600000 00001000 "uboot"
mtd12: 00a00000 00001000 "free"
mtd13: 00080000 00001000 "spi0.1"
mtd14: 00800000 00001000 "spi0.2"

[root@ root]# flash_eraseall -j /dev/mtd2

Erasing 128 Kibyte @ 1400000 -- 100% complete. Cleanmarker written at 13e0000.

[root@P1010RDB root]# mount -t jffs2 /dev/mtdblock2 /mnt/

JFFS2 notice: (1202) jffs2_build_xattr_subsystem: complete building xattr subsystem, 0 of xdatum
(0 unchecked, 0 orphan) and 0 of xref (0 dead, 0 orphan) found.
```

```

[root@ root]# cd /mnt/

[root@ mnt]# ls -l

[root@ mnt]# touch flash_file

[root@ root]# umount mnt
//ls must list local_file
[root@ root]# ls mnt
//mount again
[root@ root]# mount -t jffs2 /dev/mtdblock2 /mnt/
JFFS2 notice: (1219) jffs2_build_xattr_subsystem: complete building xattr subsystem, 0 of xdatum
(0 unchecked, 0 orphan) and 0 of xref (0 dead, 0 orphan) found.
//use ls ; it must show the created file
[root@ root]# ls /mnt/
flash_file
//unmount
[root@ root]# umount /mnt/

```

7.2.4.2 Integrated Flash Controller NAND Flash User Manual

Description

NXP's Integrated Flash Controller can be used to connect various types of flashes (e.g. NOR/NAND) on board for boot functionality as well as data storage.

U-Boot Configuration

Compile time options

Below are major U-Boot configuration options related to this feature defined in platform specific config files under include/configs/ directory.

Option Identifier	Description
CONFIG_FSL_IFC	Enable IFC support
CONFIG_NAND_FSL_IFC	Enable NAND Machine support on IFC
CONFIG_SYS_MAX_NAND_DEVICE	No of NAND Flash chips on platform
CONFIG_MTD_NAND_VERIFY_WRITE	Verify NAND flash writes
CONFIG_CMD_NAND	Enable various commands support for NAND Flash
CONFIG_SYS_NAND_BLOCK_SIZE	Block size of the NAND flash connected on Platform

Source Files

The following source files are related to this feature in u-boot.

Source File	Description
./drivers/misc/fsl_ifc.c	Set up the different chip select parameters from board header file
drivers/mtd/nand/fsl_ifc_nand.c	IFC nand flash machine driver file

Kernel Configure Options

Tree View

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel

Kernel Configure Tree View Options	Description
<pre> Device Drivers ---> <*> Memory Technology Device (MTD) support ---> [*] MTD partitioning support [*] Command line partition table parsing= <*> Flash partition map based on OF description <*> Direct char device access to MTD devices *- Common interface to block layer for MTD 'translation layers' <*> Caching block device access to MTD devices <*> NAND Device Support ---> <*> NAND support for Freescale IFC controller Enable UBIFS filesystem in linux configuration Device Drivers ---> <*> Memory Technology Device (MTD) support ---> UBI - Unsorted block images ---> <*> Enable UBI (4096) UBI wear-leveling threshold (1) Percentage of reserved eraseblocks for bad eraseblocks handling </pre>	These options enable Integrated Flash Controller NAND support to work with MTD subsystem available on Linux. Also UBIFS support needs to be enabled.

Kernel Configure Tree View Options	Description
<pre> < > MTD devices emulation driver (gluebi) *** UBI debugging options *** [] UBI debugging File systems ---> [*] Miscellaneous filesystems ---> <*> UBIFS file system support [*] Extended attributes support [] Advanced compression options [] Enable debugging </pre>	

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_FSL_IFC	y/n	y	Enable Integrated Flash Controller support
CONFIG_MTD_NAND_FSL_IFC	y/n	Y	Enable Integrated Flash Controller NAND Machine support
CONFIG_MTD_PARTITIONS	y/n	Y	MTD partitioning support
CONFIG_MTD_CMDLINE_PARTS	y/n	Y	Allow generic configuration of the MTD partition tables via the kernel command line.
CONFIG_MTD_OF_PARTS	y/n	Y	This provides a partition parsing function which derives the partition map from the children of the flash nodes described in Documentation/powerpc/booting-without-of.txt
CONFIG_MTD_CHAR	y/n	Y	Direct char device access to MTD devices
CONFIG_MTD_BLOCK	y/n	Y	Caching block device access to MTD devices
CONFIG_MTD_GEN_PROBE	y/n	Y	NA
CONFIG_MTD_PHYSMAP_OF	y/n	Y	Flash device in physical memory map based on OF description

Device Tree Binding

Documentation/devicetree/bindings/memory-controllers/fsl/ifc.txt

Flash partitions are specified by platform device tree.

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/memory/fsl_ifc.c	Integrated Flash Controller driver to handle error interrupts
drivers/mtd/nand/fsl_ifc_nand.c	Integrated Flash Controller NAND Machine driver
include/linux/fsl_ifc.h	IFC Memory Mapped Registers

Verification in U-Boot

Test the Read/Write/Erase functionality of NAND Flash

1. Boot the u-boot with above config options to get NAND Flash driver enabled. Check this in boot log,

```
NAND: * MiB  
  
Where * is NAND flash size
```

2. Erase NAND Flash
3. Make test pattern on memory e.g. DDR
4. Write test pattern on NAND Flash
5. Read the test pattern from NAND Flash to memory e.g DDR
6. Compare the test pattern data to verify functionality.

Test Log :

```
...  
...  
  
NAND:  512 MiB  
  
...  
...  
  
/* U-boot prompt */  
=> nand erase.chip  
  
NAND erase.chip: device 0 whole chip  
  
Bad block table found at page 65504, version 0x01 Bad block table found at page 65472, version  
0x01  
  
Skipping bad block at  0x01ff0000  
  
Skipping bad block at  0x01ff4000  
  
Skipping bad block at  0x01ff8000  
  
Skipping bad block at  0x01ffc000
```

```

OK

=> mw.b 80000000 0xa5 100000

=> md 80000000

80000000: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000010: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000020: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....
80000030: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 .....

=> nand write 80000000 0 100000

NAND write: device 0 offset 0x0, size 0x100000

1048576 bytes written: OK

=> nand read 90000000 0 100000

NAND read: device 0 offset 0x0, size 0x100000

1048576 bytes read: OK

=> cmp.b 80000000 90000000 100000

Total of 1048576 bytes were the same

```

Verification in Linux

To cross check whether IFC NAND driver has been configured in the kernel or not, check the following. Please note mtd partition numbers can be changed depending upon board device tree

```

[root@(none) root]# cat /proc/mtd
dev:   size  erasesize  name
mtd0: 00100000 00020000 "nor_bank0_rcw"
mtd1: 00f00000 00020000 "nor_bank0_uboot"
mtd2: 01000000 00020000 "nor_bank0_kernel"
mtd3: 02000000 00020000 "nor_bank0_rootfs"
mtd4: 01000000 00040000 "nand_uboot"
mtd5: 01000000 00040000 "nand_kernel"
mtd6: 02000000 00040000 "nand_free"

[root@(none) root]# flash_eraseall /dev/mtd4 Erasing 16 Kibyte @ f00000 -- 100% complete.

[root@(none) root]# ubiattach /dev/ubi_ctrl -m 4

```

```
UBI: attaching mtd4 to ubi0

UBI: physical eraseblock size:   16384 bytes (16 KiB)

UBI: logical eraseblock size:    15360 bytes

UBI: smallest flash I/O unit:    512

UBI: VID header offset:          512 (aligned 512)

UBI: data offset:                1024

UBI: empty MTD device detected

UBI: create volume table (copy #1)

UBI: create volume table (copy #2)

UBI: attached mtd4 to ubi0

UBI: MTD device name:            "NAND Root File System"

UBI: MTD device size:            15 MiB

UBI: number of good PEBs:        960

UBI: number of bad PEBs:         0

UBI: max. allowed volumes:       89

UBI: wear-leveling threshold:    4096

UBI: number of internal volumes: 1

UBI: number of user volumes:     0

UBI: available PEBs:             947

UBI: total number of reserved PEBs: 13

UBI: number of PEBs reserved for bad PEB handling: 9

UBI: max/mean erase counter: 0/0

UBI: image sequence number: 0

UBI: background thread "ubi_bgt0d" started, PID 7541 UBI device number 0, total 960 LEBs
(14745600 bytes, 14.1 MiB), available 947 LEBs (14545920 bytes, 13.9 MiB), LEB size 15360 bytes
(15.0 KiB)

[root@(none) root]# ubimkvol /dev/ubi0 -N rootfs -s 14205KiB Volume ID 0, size 947 LEBs (14545920
bytes, 13.9 MiB), LEB size 15360 bytes (15.0 KiB), dynamic, name "rootfs", alignment 1

[root@(none) root]# mount -t ubifs /dev/ubi0_0 /mnt/

UBIFS: default file-system created
```

```
UBIFS: mounted UBI device 0, volume 0, name "rootfs"

UBIFS: file system size: 14361600 bytes (14025 KiB, 13 MiB, 935 LEBs)
UBIFS: journal size: 721920 bytes (705 KiB, 0 MiB, 47 LEBs)

UBIFS: media format: w4/r0 (latest is w4/r0)

UBIFS: default compressor: lzo

UBIFS: reserved for root: 678333 bytes (662 KiB)


[root@(none) root]# cd /mnt/

[root@(none) mnt]# ls

[root@(none) mnt]# touch flash_file

[root@(none) mnt]# ls -l

total 0

-rw-r--r-- 1 root root 0 Jul  6 14:45 flash_file

[root@(none) mnt]# cd

[root@(none) root]# umount /mnt/

UBIFS: un-mount UBI device 0, volume 0


[root@(none) root]# mount -t ubifs /dev/ubi0_0 /mnt/

UBIFS: mounted UBI device 0, volume 0, name "rootfs"

UBIFS: file system size: 14361600 bytes (14025 KiB, 13 MiB, 935 LEBs)
UBIFS: journal size: 721920 bytes (705 KiB, 0 MiB, 47 LEBs)

UBIFS: media format: w4/r0 (latest is w4/r0)

UBIFS: default compressor: lzo

UBIFS: reserved for root: 678333 bytes (662 KiB)


[root@(none) root]# ls -l /mnt/

total 0

-rw-r--r-- 1 root root 0 Jul  6 14:45 flash_file
```

Known Bugs, Limitations, or Technical Issues

Boards which have NAND Flash with 512byte page size, JFFS2 cannot be supported using H/W ECC support of IFC , as there is not enough remaining space in the OOB area.

To use JFFS2 use SOFT ECC.

7.2.5 PL011 Universal Asynchronous Receiver/Transmitter (UART)

Description

The PL011 Universal asynchronous receiver/transmitter (UART) is Advanced Microcontroller Bus Architecture(AMBA) compliant peripheral. Refer to below table for the NXP SoC UART support.

SOC	Num of UART module
LX2160A	4

U-boot Configuration Compile time options

Tree View

Below are major u-boot configuration options related to be set while compiling the u-boot

Option Identifier	Description
-> Device Drivers -> Generic Driver Options [*] Enable Driver Model	Enables Driver Model
-> Device Drivers -> Serial drivers [*] Enable Driver Model for serial drivers	Enable DM model UART driver support

Below are major u-boot configuration options related to this feature defined in platform specific config files under include/configs/ directory.

Option Identifier	Description
CONFIG_PL01X_SERIAL	Enable PL011 UART support

Device Tree Binding

Below is an example device tree node required by this feature. Note that it may has differences among platforms.

```
uart0: serial@21c0000 {
    compatible = "arm,pl011";
    reg = <0x0 0x21c0000 0x0 0x1000>;
    clocks = <&clockgen 4 0>;
    status = "okay";
};
```

Configure stdout-path to select the UART for boot console

```
chosen {
    stdout-path = &uart0;
};
```

Choosing predefined u-boot board configs:

Please make the defconfig like: lx2160ardb_defconfig. This support PL011 UART.

Runtime options

Env Variable	Env Description	Sub option	Option Description
bootargs	Kernel command line argument passed to kernel	console= ttyAMA0,115200	Select UART0 as the system console

Kernel Configure Options

Tree View

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> Character devices ---> Serial drivers ---> Symbol: SERIAL_AMBA_PL011 Symbol: SERIAL_AMBA_PL011_CONSOLE</pre>	PL011 UART driver and enable console support

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_SERIAL_AMBA_PL011	y/m/n	y	AMBA PL011 support
CONFIG_SERIAL_AMBA_PL011_CONSOLE	y/n	y	Console on AMBA Serial Port

Device Tree Binding

Below is an example device tree node required by this feature. Note that it may has differences among platforms.

```
uart0: serial@21c0000 {
    device_type = "serial";
    compatible = "arm,pl011", "arm,sbsa-uart";
    reg = <0x0 0x21c0000 0x0 0x1000>;
```

```

interrupts = <0 32 0x4>; /* Level high type */
current-speed = <115200>;
status = "okay";
};
chosen {
    stdout-path = "serial0:115200n8";
};

```

Source Files

The following source file are related the this feature in u-boot.

Table 43.

Source File	Description
drivers/serial/serial_pl01x.c	The PL011 UART driver file

The following source file are related the this feature in Linux kernel.

Source File	Description
drivers/tty/serial/amba-pl011.c	The PL011 UART driver file

Verification in U-Boot

1. Boot up U-Boot from bank0, and update rcw and u-boot for Uart support to bank4, first copy the rcw and U-Boot binary(with stdout-path changes) to the tftp directory.
2. Please refer to the platform depoly document to update the rcw and uboot.
3. After all is updated, run u-boot command to switch to alt bank, then will bring up the new U-Boot to the uart console.

```

U-Boot 2017.07-03498-gbcbf1e0-dirty (May 24 2018 - 16:09:22 +0530)

SoC: LX2160ACE Rev1.0 (0x87360010)
Clock Configuration:
  CPU0 (A72):1900 MHz   CPU1 (A72):1900 MHz   CPU2 (A72):1900 MHz
  CPU3 (A72):1900 MHz   CPU4 (A72):1900 MHz   CPU5 (A72):1900 MHz
  CPU6 (A72):1900 MHz   CPU7 (A72):1900 MHz   CPU8 (A72):1900 MHz
  CPU9 (A72):1900 MHz   CPU10 (A72):1900 MHz  CPU11 (A72):1900 MHz
  CPU12 (A72):1900 MHz  CPU13 (A72):1900 MHz  CPU14 (A72):1900 MHz
  CPU15 (A72):1900 MHz
  Bus:      600 MHz   DDR:      1600 MT/s
Reset Configuration Word (RCW):
  00000000: 4c434330 304c004c 00000000 00000000
  00000010: 00000000 06010000 00000000 00000000
  00000020: 00e001a0 00002580 00000000 00000096
  00000030: 00000000 00000000 00000000 00000000
  00000040: 00000000 00000000 00000000 00000000
  00000050: 00000000 00000000 00000000 00000000
  00000060: 00000000 00000000 00027000 00000000
  00000070: 08b30000 003f0020
Model: NXP Layerscape LX2160ARDB Board
Board: LX2160ACE Rev1.0-RDB, Board version: B, boot from FlexSPI DEV#1
FPGA: v1.7
SERDES1 Reference: Clock1 = 161.13MHz Clock2 = 161.13MHz
SERDES2 Reference: Clock1 = 100MHz Clock2 = 100MHz
SERDES3 Reference: Clock1 = 100MHz Clock2 = 100Hz
I2C:  ready

```



```

DRAM:  Initializing DDR...using SPD
Detected UDIMM 18ASF1G72AZ-2G6B1
Detected UDIMM 18ASF1G72AZ-2G6B1
DDR4(0) UDIMM with 2-rank 64-bit bus (x8)
DDR4(1) UDIMM with 2-rank 64-bit bus (x8)
15.9 GiB
DDR      15.9 GiB (DDR4, 64-bit, CL=11, ECC on)
        DDR Controller Interleaving Mode: 256B
        DDR Chip-Select Interleaving Mode: CS0+CS1
PPA Firmware: Version fsl-sdk-v2.0-1703-137-gb0a07cf
Using SERDES1 Protocol: 19 (0x13)
Using SERDES2 Protocol: 5 (0x5)
Using SERDES3 Protocol: 2 (0x2)
MMC:  FSL_SDHC: 0, FSL_SDHC: 1
SF: Detected mt35xu512g with page size 256 Bytes, erase size 128 KiB, total 64 MiB
EEPROM: NXID v1
In:  serial@21c0000
Out: serial@21c0000
Err: serial@21c0000
SATA link 0 timeout.
AHCI 0001.0301 32 slots 1 ports 6 Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc apst
Found 0 device(s).
SCSI: Net:  IN112525 phy init: Unable to TX PLL lock
FSL_MDIO1:0 is connected to DPMAC5@25g-aui. Reconnecting to DPMAC6@25g-aui
IN112525 phy init: Unable to TX PLL lock
PCIe0: pcie@3400000 disabled
PCIe1: pcie@3500000 disabled
PCIe2: pcie@3600000 Root Complex: x1 gen1
PCIe3: pcie@3700000 disabled
PCIe4: pcie@3800000 Root Complex: no link
PCIe5: pcie@3900000 disabled
e1000: 68:05:ca:2c:ed:b2
        DPMAC2@xlaui4, DPMAC3@xgmii, DPMAC4@xgmii, DPMAC5@25g-aui, DPMAC6@25g-aui, DPMAC17@rgmii-
id, DPMAC18@rgmii-id,
e1000#0

fsl-mc: ERR: Bad firmware image (not a FIT
image)
Hit any key to stop
autoboot: 0

=> edit
bootargs
edit: console=ttyAMA0,115200 root=/dev/ram0 earlycon=pl011,mmio32,0x21c0000
ramdisk_size=0x2000000 default_hugepagesz=2m hugepagesz=2m hugepages=256

[...]
```

Verification in Linux

1. After uboot startup, set the command line set the command line parameter to pass to the linux kernel including in bootargs. For deploying the ramdisk as rootfs,

```

=> edit
bootargs
edit: console=ttyAMA0,115200 root=/dev/ram0 earlycon=pl011,mmio32,0x21c0000
ramdisk_size=0x2000000 default_hugepagesz=2m hugepagesz=2m hugepages=256
=> tftpboot 0xa0000000 <kernel.itb>;fsl_mc apply dpl 0x20d00000; bootm 0xa0000000

[...]
```

```
[...]
[ 0.000000] earlycon: pl111 at MMIO32 0x00000000021c0000 (options '')
[ 0.000000] bootconsole [pl111] enabled
[...]
[ 1.336581] Serial: AMBA PL011 UART driver
[ 1.341369] Machine: NXP Layerscape LX2160ARDB
[ 1.345859] SoC family: QorIQ
[ 1.348850] SoC ID: svr:0x87360010, Revision: 1.0
[ 1.354854] 21c0000.serial: ttyAMA0 at MMIO 0x21c0000 (irq = 93, base_baud = 0) is a SBSA
[ 1.363146] console [ttyAMA0] enabled
[ 1.363146] console [ttyAMA0] enabled
[...]
QorIQ SDK (FSL Reference Distro) 2.4.2 lx2160ardb /dev/ttyAMA0
lx2160ardb login: root
root@lx2160ardb:~#
```

After the kernel boot up to the console, You can type any shell command in the UART TERMINAL.

Steps to test UART2 on u-boot/linux

U-boot

(1)Make following changes in u-boot to update stdout-path property to uart2 as boot console(Please make below changes in fsl-lx2160a-qds.dts if you are using LX2160A QDS board)

```
+++ b/arch/arm/dts/fsl-lx2160a-rdb.dts
@@ -22,7 +22,7 @@
     };

     chosen {

-         stdout-path = &uart0;
+         stdout-path = &uart1;

     };
};
```

- (2)Compile and build u-boot image
- (3)Connect/Open UART1 and UART2 blocks console
- (4)Boot LX2160ARDB primary bank on UART1 console
- (5)Flash u-boot image including above (1) changes to alternate bank
- (6)Switch to alternate bank(qixis_reset altbank)
- (7)U-boot alternate bank logs on UART2 console

Linux

(8)On UART2 console,boot linux image with example command as:

```
tftpboot 0xa0000000 nxf28358/test/kernel.itb;fsl_mc apply dpl 0x20d00000; bootm 0xa0000000
```

- (9)Linux boot log comes on UART1 console
- (10)On linux prompt,Type following command to switch UART2 console

```
sh -c "getty 115200 /dev/ttyAMA1;true"
```

(11)Following prompt on UART2 console

```
QorIQ SDK (FSL Reference Distro) 2.4.2 lx2160ardb /dev/ttyAMA1

lx2160ardb login: root
Last login: Thu May 31 08:40:06 UTC 2018 on ttyAMA0
root@lx2160ardb:~#
```

7.2.6 Flex Serial Peripheral Interface (FSPI)

U-Boot Configuration

Make sure your boot mode support FlexSPI.

Use FlexSPI boot mode to boot on board, please check the board user manual and boot from FlexSPI. (or some other boot mode decide by your board.)

Following Config options needs to be enabled for FlexSPI.

- CONFIG_NXP_FSPI=y
- CONFIG_FSPI_AHB_EN_4BYTE=y
- CONFIG_SYS_FSPI_AHB_INIT=y

Kernel Configure Tree View Options

```
Device Drivers --->
  Memory Technology Device (MTD) support
  RAM/ROM/Flash chip drivers --->
    < > Detect flash chips by Common Flash Interface (CFI) probe
    < > Detect non-CFI AMD/JEDEC-compatible flash chips
    < > Support for RAM chips in bus mapping
    < > Support for ROM chips in bus mapping
    < > Support for absent chips in bus mapping
  Self-contained MTD device drivers --->
    <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
  < > NAND Device Support ---
  <*> SPI-NOR device support --->
    the framework for SPI-NOR support
  <*> NXP Flex SPI controller
```

```
CONFIG_SPI_NXP_FLEXPSP:
This enabled support for the FlexSPI controller in master mode.
```

```
Symbol: SPI_NXP_FLEXPSP [=y]
Type : tristate
Prompt: NXP Flex SPI controller
Location:
  -> Device Drivers
    -> Memory Technology Device (MTD) support (MTD [=y])
      -> SPI-NOR device support (MTD_SPI_NOR [=y])
Depends on: MTD [=y] && MTD_SPI_NOR [=y]
```

Compile-time Configuration Options

Config	Values	Default Value	Description
CONFIG_SPI_NXP_FLEXSPI	y/n	y	Enable FlexSPI module
CONFIG_MTD_SPI_NOR_BASE	y/n	y	Enables the framework for SPI-NOR

Verification in U-Boot

```
=> sf probe 0:0
SF: Detected mt35xu512g with page size 256 Bytes, erase size 128 KiB, total 64 MiB
=> sf erase 0 100000
SF: 1048576 bytes @ 0x0 Erased: OK
=> sf write 82000000 0 1000
SF: 4096 bytes @ 0x0 Written: OK
=> sf read 81100000 0 1000
SF: 4096 bytes @ 0x0 Read: OK
=> cm.b 81100000 82000000 1000
Total of 4096 byte(s) were the same
```

Verification in Linux:

```
The booting log

.....
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
.....

Erase the FlexSPI flash

~ # mtd_debug erase /dev/mtd0 0x00000000 1048576
Erased 1048576 bytes from address 0x00000000 in flash

Write the FlexSPI flash

~ # dd if=/bin/ls.coreutils of=tp bs=4096 count=1
~ # mtd_debug write /dev/mtd0 0 4096 tp
Copied 4096 bytes from tp to address 0x00000000 in flash

Read the FlexSPI flash

~ # mtd_debug read /dev/mtd0 0 4096 dump_file

Copied 4096 bytes from address 0x00000000 in flash to dump_file

Check Read and Write
```

```
Use compare tools(yacto has tools named diff).
~ # diff tp dump_file
~ #
If diff command has no print log, the FlexSPI verification is passed.
```

7.2.7 PCI Express Interface Controller

7.2.7.1 PCIe Linux Driver

Module Loading

The MPC85xx/Layerscape PCIe host bridge support code is compiled into the kernel. It is not available as a module.

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description
<pre>Bus support ---> [*] PCI support [*] Message Signaled Interrupts (MSI and MSI-X)</pre>	Enable PCI host bridge and message support
<pre>Bus support ---> PCI host controller drivers ---> [*] Freescale Layerscape PCIe controller</pre>	Enable NXP Layerscape PCIe controller
<pre>Device Drivers ---> [*]Network device support ---> [*]Ethernet device support ---> [*] Intel devices ---> <*> Intel (R) PRO/1000 PCI-Express Gigabit Ethernet support</pre>	Intel PRO/1000 PCI-Express support
<pre>Device Drivers ---> <*> Serial ATA and Parallel ATA drivers (libata) ---> <*> Silicon Image 3124/3132 SATA support</pre>	Enable support for Silicon Image 3124/3132 Serial ATA.

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_PCI	y/n	y	Enable PCI host bridge
CONFIG_PCI_MSI	y/n	y	Message support
CONFIG_PCI_LAYERSCAPE	y/n	y	Enable PCI for Layerscape
CONFIG_E1000E	y/m/n	y	Enable Intel Pro/1000 driver
CONFIG_SATA_SIL	y/m/n	y	Silicon Image SATA support

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
arch/powerpc/sysdev/fsl_pci.c	The MPC85XX platform PCIE host bridge support source
drivers/pci/host/pci-layerscape.c	The Layerscape platform PCIE host bridge support source
drivers/net/ethernet/intel/e1000e/	Intel Pro/1000 driver source code
drivers/ata/sata_sil.c	Silicon Image source code

SATA Card Test Procedure

```
the user can use command
fdisk, mke2fs mount to operate the ide disk.
After kernel boots up, please follow the log to operate:

[root@pX0XX /root]# fdisk -l

Disk /dev/sda: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Disk /dev/sda doesn't contain a valid partition table

[root@pX0XX /root]# fdisk /dev/sda
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that the previous content
won't be recoverable.

The number of cylinders for this disk is set to 10443.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-10443, default 1): Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-10443, default 10443): 100

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table
sd 0:0:0:0: [sda] 167772160 512-byte hardware sectors (85899 MB)
sd 0:0:0:0: [sda] Write Protect is off
```

```
sd 0:0:0:0: [sda] Asking for cache data failed
sd 0:0:0:0: [sda] Assuming drive cache: write through
sda: sda1

[root@pX0XX /root]# mke2fs /dev/sda1
mke2fs 1.34 (25-Jul-2003)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
100576 inodes, 200804 blocks
10040 blocks (5.00%) reserved for the super user
First data block=0
7 block groups
32768 blocks per group, 32768 fragments per group
14368 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 31 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.

[root@pX0XX /root]# mkdir sda1_test
[root@pX0XX /root]# mount /dev/sda1 sda1_test/
[root@pX0XX /root]# cp /bin/tar sda1_test/
[root@pX0XX /root]#
```

Ethernet Card Test Procedure

- plug Intel Pro/1000e network card into standard PCI-E slot on a board. After linux bootup, ifconfig ethx ip address and netmask, then do ping testing.

Tips: x ethernet interface number, an example is as the following for Intel e1000 network card is eth0.

For example:

After kernel boot up, bring up with the pci Ethernet card

```
ifconfig ethx 192.168.20.100
```

ip address should not be conflicted with other Ethernet port.

In Linux window, run ping 192.168.20.101

Known Bugs, Limitations, or Technical Issues

- LSI-SAS card cannot be used on the second PCIe controller when system enables more than one PCIe controller. Use code modification below to workaround this issue:

```
--- a/arch/powerpc/sysdev/fsl_pci.c
+++ b/arch/powerpc/sysdev/fsl_pci.c
@@ -511,7 +511,7 @@ int __init fsl_add_bridge(struct platform_device *pdev, int is_primary)
     printk(KERN_WARNING "Can't get bus-range for %s, assume"
            " bus 0\n", dev->full_name);

-    pci_add_flags(PCI_REASSIGN_ALL_BUS);
```

```
+ pci_add_flags(PCI_ENABLE_PROC_DOMAINS);  
hose = pcibios_alloc_controller(dev);  
if (!hose)  
    return -ENOMEM;  
@@ -846,7 +846,7 @@ int __init mpc83xx_add_bridge(struct device_node *dev)  
    " bus 0\n", dev->full_name);  
}  
  
- pci_add_flags(PCI_REASSIGN_ALL_BUS);  
+ pci_add_flags(PCI_ENABLE_PROC_DOMAINS);  
hose = pcibios_alloc_controller(dev);  
if (!hose)  
    return -ENOMEM;
```

7.2.7.2 PCIe Advanced Error Reporting User Manual

Description

How to test the PCI Express Advanced Error Reporting (AER) function.

Testing the PCIe AER error recovery code in actual environment is quite difficult because it is hard to trigger real hardware errors. So we use a software tool based error injection to fake various kinds of PCIe errors.

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description
Bus options ---> [*] PCI Express support [*] Root Port Advanced Error Reporting support <*> PCIe AER error injector support	enable PCI-Express AER and AER-INJECTOR in kernel

Kernel compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_PCIEAER	y/n	y	Enable AER
CONFIG_PCIEAER_INJECT	y/n	n	Enables AER INJECT

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/pci/pcie/aer/*.c	AER driver support

• Prepare aer-inject test tool

```
1, Download aer-inject test utility.  
  
2, Write a test config file  
e.g. $ vi aer-cfg  
AER
```



```

DOMAIN 0001
BUS 1
DEV 0
FN 0
COR_STATUS BAD_TLP
HEADER_LOG 0 1 2 3

```

NOTE:

error type can be ["COR_STATUS", "UNCOR_STATUS"]

Corrected error can be:

```
["BAD_TLP", "RCVR", "BAD_DLLP", "REP_ROLL", "REP_TIMER"]
```

Uncorrected non-fatal error can be:

```
["POISON_TLP", "COMP_TIME", "COMP_ABORT", "UNX_COMP", "ECRC", "UNSUP"]
```

Uncorrected fatal error can be:

```
["TRAIN", "DLP", "FCP", "RX_OVER", "MALF_TLP"]
```

• Test Steps

1, insert a pcie device in PCI slot of board, ensure the pcie device has AER capability, e.g. e1000e PCIe NIC network card.

2, In u-boot prompt, add "pcie_ports=native" in bootargs command-line.
=> setenv othbootargs pcie_ports=native

3, boot the kernel and filesystem.

4, check AER device and config
zcat /proc/config.gz|grep -i CONFIG_PCIEAER_INJECT
CONFIG_PCIEAER_INJECT=y

cat /proc/cmdline
root=/dev/ram rw console=ttyS0,115200 pcie_ports=native
check "pcie_ports=native" has been set.

ls /dev/aer_inject
Check if the aer injector device is created.

lspci
00:00.0 Class 0604: 1957:0410
01:00.0 Class 0200: 8086:10d3
e.g. here device "01:00.0" is the PCIe NIC e1000 network card in the test scenario.

5, Download aer-inject and aer-cfg from host to test-board
\$ scp aer-inject aer-cfg root@test-board-ip:~

6, ensure the pcie device domain-number/bus-number/device-number/function-number in aer-cfg is accordant to those in the output of lspci

7, Run aer-inject, corresponding error information will be reported as below and AER will recover PCIe device according to the type of errors.

```

# ./aer-inject aer-cfg
example of error report as below:
pcieport 0000:00:00.0: AER: Corrected error received: id=0100
e1000e 0000:01:00.0: PCIe Bus Error: severity=Corrected, type=Data Link Layer, id=0100(Receiver ID)
e1000e 0000:01:00.0: device [8086:10d3] error status/mask=00000040/00002000

```

```
e1000e 0000:01:00.0: [ 6] Bad TLP
root@lsxxx:~#

8, The pcie device(e1000e PCIE NIC) should still work after AER error recovery.
# ping 192.168.1.1 -c 2 -s 64
PING 192.168.1.1 (192.168.1.1): 64 data bytes
72 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=0.272 ms
72 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.210 ms
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.210/0.241/0.272/0.031 ms
```

Note:

On some legacy platforms with legacy PCI controller(e.g. some non-DPAA platforms), hardware doesn't support Fatal error type for AER, just support Non-Fatal error.

Generally, DPAA platforms with new PCIE controller can support both Fatal error and Non-Fatal error.

LS1088A and LS1012A fail to recover from fatal errors.

7.2.7.3 PCI-e Remove and Rescan User Manual

Description

Describes how to remove and rescan a PCI-e device under runtime Linux system.

U-boot Configuration

Use the default configurations.

Kernel Configure Options

Use the default configurations, make sure the configure option is set while doing "make menuconfig" for kernel.

Kernel Configure Tree View Options	Description
Device Drivers ---> [*] Network device support---> [*] Ethernet (1000 Mbit) ---> [*] Intel(R) PRO/1000 PCI-Express Gigabit Ethernet support	This option enables kernel support for Intel PCI-e e1000e network card

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_E1000E	y/n	y	Intel PCI-e e1000e network card driver

Device Tree Binding

Use the default dtb file.

Verification in Linux

Make sure the PCI-e controller which you add the PCI-e e1000e network card to works as RC mode. Use the kernel, dtb and ramdisk rootfs to boot the board.

```

1. Suppose the PCI-e device under /sys/bus/pci/devices/0001\:03\:00.0 is the Intel PCI-e e1000e
network card, recognized as eth0. The
/sys/bus/pci/devices/0001\:02\:00.0 is the bus of network card. Configure an ip and ping another
host which is in the same subnet, make sure the network card works well.

# ls /sys/bus/pci/devices/0001\:03\:00.0/net
eth0
# ifconfig eth0 10.193.20.100
# ping -I eth0 10.193.20.31

2. Remove the PCI-e network card from system.
# echo 1 > /sys/bus/pci/devices/0001\:03\:00.0/remove
e1000e 0001:03:00.0 eth0: removed PHY

3. Check whether the PCI-e network card still exist in system. All should fail.
# ifconfig eth0
# ls /sys/bus/pci/devices/0001\:03\:00.0

4. Rescan it from the bus.
# echo 1 > /sys/bus/pci/devices/0001\:02\:00.0/rescan

5. Check whether the device is rescanned and works well.
# ls /sys/bus/pci/devices/0001\:03\:00.0
# ifconfig eth0 10.193.20.100
# ping -I eth0 10.193.20.31

6. All the commands of step 5 should success.

```

Known Bugs, Limitations, or Technical Issues

None

7.2.8 Flex Serial Peripheral Interface (FSPI)

U-Boot Configuration

Make sure your boot mode support FlexSPI.

Use FlexSPI boot mode to boot on board, please check the board user manual and boot from FlexSPI. (or some other boot mode decide by your board.)

Following Config options needs to be enabled for FlexSPI.

- CONFIG_NXP_FSPI=y
- CONFIG_FSPI_AHB_EN_4BYTE=y
- CONFIG_SYS_FSPI_AHB_INIT=y

Kernel Configure Tree View Options

```

Device Drivers --->
  Memory Technology Device (MTD) support
  RAM/ROM/Flash chip drivers --->

```

```

    < > Detect flash chips by Common Flash Interface (CFI) probe
    < > Detect non-CFI AMD/JEDEC-compatible flash chips
    < > Support for RAM chips in bus mapping
    < > Support for ROM chips in bus mapping
    < > Support for absent chips in bus mapping
Self-contained MTD device drivers --->
    <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
< > NAND Device Support ---
<*> SPI-NOR device support --->
    the framework for SPI-NOR support
<*> NXP Flex SPI controller

```

CONFIG_SPI_NXP_FLEXPSPi:
This enabled support for the FlexSPI controller in master mode.

Symbol: SPI_NXP_FLEXPSPi [=y]
Type : tristate
Prompt: NXP Flex SPI controller
Location:
-> Device Drivers
-> Memory Technology Device (MTD) support (MTD [=y])
-> SPI-NOR device support (MTD_SPI_NOR [=y])
Depends on: MTD [=y] && MTD_SPI_NOR [=y]

Compile-time Configuration Options

Config	Values	Default Value	Description
CONFIG_SPI_NXP_FLEXPSPi	y/n	y	Enable FlexSPI module
CONFIG_MTD_SPI_NOR_BASE	y/n	y	Enables the framework for SPI-NOR

Verification in U-Boot

```

=> sf probe 0:0
SF: Detected mt35xu512g with page size 256 Bytes, erase size 128 KiB, total 64 MiB
=> sf erase 0 100000
SF: 1048576 bytes @ 0x0 Erased: OK
=> sf write 82000000 0 1000
SF: 4096 bytes @ 0x0 Written: OK
=> sf read 81100000 0 1000
SF: 4096 bytes @ 0x0 Read: OK
=> cm.b 81100000 82000000 1000
Total of 4096 byte(s) were the same

```

Verification in Linux:

The booting log

```

.....
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
.....

Erase the FlexSPI flash

~ # mtd_debug erase /dev/mtd0 0x00000000 1048576
Erased 1048576 bytes from address 0x00000000 in flash

Write the FlexSPI flash

~ # dd if=/bin/ls.coreutils of=tp bs=4096 count=1
~ # mtd_debug write /dev/mtd0 0 4096 tp
Copied 4096 bytes from tp to address 0x00000000 in flash

Read the FlexSPI flash

~ # mtd_debug read /dev/mtd0 0 4096 dump_file

Copied 4096 bytes from address 0x00000000 in flash to dump_file

Check Read and Write

Use compare tools(yacto has tools named diff).
~ # diff tp dump_file
~ #
If diff command has no print log, the FlexSPI verification is passed.

```

7.2.9 Queue Direct Memory Access Controller (qDMA)

The qDMA controller transfers blocks of data between one source and one destination. The blocks of data transferred can be represented in memory as contiguous or noncontiguous using scatter/gather table(s). Channel virtualization is supported through enqueueing of DMA jobs to, or dequeuing DMA jobs from, different work queues.

QDMA can support Layerscape platform with DPAA1 or DPAA2.

QDMA for platform with DPAA1

Kernel Configure Options

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel.

Kernel Configure Tree View Options	Description
<pre> Device Drivers ---> [*] DMA Engine support ---> ---> <*> Freescale qDMA engine support </pre>	Support the Freescale qDMA engine with command queue and legacy mode. Channel virtualization is supported through enqueueing of DMA jobs to, or dequeuing DMA jobs from, different work queues. This module can be found on Freescale LS SoCs.

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_FSL_QDMA	y/m/n	n	qDMA driver

Device Tree Binding

Device Tree Node

Below is an example device tree node required by this feature. Note that it may has differences among platforms.

```
qdma: qdma@8380000 {
    compatible = "fsl,ls1046a-qdma", "fsl,ls1021a-qdma";
    reg = <0x0 0x8380000 0x0 0x1000>, /* Controller regs */
        <0x0 0x8390000 0x0 0x10000>, /* Status regs */
        <0x0 0x83a0000 0x0 0x40000>; /* Block regs */
    interrupts = <0 153 0x4>,
        <0 39 0x4>;
    interrupt-names = "qdma-error", "qdma-queue";
    channels = <8>;
    queues = <2>;
    status-sizes = <64>;
    queue-sizes = <64 64>;
    big-endian;
};
```

Source File

The following source files are related the feature in Linux kernel.

Source File	Description
drivers/dma/fsl-qdma.c	The qDMA driver file

Verification in Linux

```
root@ls1043ardb:~# echo 1024 > /sys/module/dmatest/parameters/test_buf_size;
root@ls1043ardb:~# echo 4 > /sys/module/dmatest/parameters/threads_per_chan;
root@ls1043ardb:~# echo 2 > /sys/module/dmatest/parameters/max_channels;
root@ls1043ardb:~# echo 100 > /sys/module/dmatest/parameters/iterations;
root@ls1043ardb:~# echo 1 > /sys/module/dmatest/parameters/run

[ 32.498138] dmatest: Started 4 threads using dma0chan0
[ 32.503430] dmatest: Started 4 threads using dma0chan1
[ 32.508939] dmatest: Started 4 threads using dma0chan2
[ 32.520073] dmatest: dma0chan0-copy0: summary 100 tests, 0 failures 4904 iops 2452 KB/s (0)
[ 32.520076] dmatest: dma0chan0-copy2: summary 100 tests, 0 failures 4923 iops 2461 KB/s (0)
[ 32.520079] dmatest: dma0chan0-copy3: summary 100 tests, 0 failures 4928 iops 2661 KB/s (0)
[ 32.520176] dmatest: dma0chan0-copy1: summary 100 tests, 0 failures 4892 iops 2446 KB/s (0)
[ 32.526438] dmatest: dma0chan1-copy0: summary 100 tests, 0 failures 4666 iops 2240 KB/s (0)
[ 32.526441] dmatest: dma0chan1-copy2: summary 100 tests, 0 failures 4675 iops 2291 KB/s (0)
[ 32.526469] dmatest: dma0chan1-copy3: summary 100 tests, 0 failures 4674 iops 2197 KB/s (0)
[ 32.529610] dmatest: dma0chan2-copy1: summary 100 tests, 0 failures 5168 iops 2791 KB/s (0)
[ 32.529613] dmatest: dma0chan2-copy0: summary 100 tests, 0 failures 5164 iops 2478 KB/s (0)
[ 32.529754] dmatest: dma0chan2-copy3: summary 100 tests, 0 failures 5215 iops 2555 KB/s (0)
[ 32.529756] dmatest: dma0chan2-copy2: summary 100 tests, 0 failures 5211 iops 2709 KB/s (0)
[ 32.537881] dmatest: dma0chan1-copy1: summary 100 tests, 0 failures 3044 iops 1461 KB/s (0) (0)
```

```
dmatest: dma0chan0-copy3: summary 1000 tests, 0 failures 4078 iops 33474 KB/s (0)
dmatest: dma0chan0-copy0: summary 1000 tests, 0 failures 3024 iops 24486 KB/s (0)
dmatest: dma0chan0-copy2: summary 1000 tests, 0 failures 2881 iops 23588 KB/s (0)
```

QDMA for platform with DPAA1

Kernel Configure Options

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel.

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> [*] DMA Engine support ---> ---> <*> NXP DPAA2 QDMA</pre>	NXP Data Path Acceleration Architecture 2 QDMA driver, using the NXP MC bus driver.

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_FSL_DPAA2_QDMA	y/m/n	n	qDMA driver

Source Files

The following source files are related the feature in Linux kernel.

Source File	Description
drivers/dma/dpaa2-qdma/*	The qDMA driver file

Verification in Linux

Create DPDMAI object using restool:

```
restool dpdmai create --priorities=2,5
restool dprc assign dprc.1 --object=dpdmai.0 --plugged=1
```

Configure parameters for dmatest and run it:

```
echo 8 > /sys/module/dmatest/parameters/test_flag
echo 100 > /sys/module/dmatest/parameters/sq_size
echo 10000 > /sys/module/dmatest/parameters/iterations
echo 1 > /sys/module/dmatest/parameters/threads_per_chan
echo 8 > /sys/module/dmatest/parameters/max_channels
echo 64 > /sys/module/dmatest/parameters/test_buf_size
echo 1 > /sys/module/dmatest/parameters/run
```

Example log:

```
root@ls2085ardb:~# echo 8 > /sys/module/dmatest/parameters/test_flag
root@ls2085ardb:~# echo 10 > /sys/module/dmatest/parameters/iterations
root@ls2085ardb:~# echo 2 > /sys/module/dmatest/parameters/threads_per_chan
root@ls2085ardb:~# echo 32384 > /sys/module/dmatest/parameters/test_buf_size
root@ls2085ardb:~# echo 4 > /sys/module/dmatest/parameters/max_channels
root@ls2085ardb:~# echo 1 > /sys/module/dmatest/parameters/run
```

```
[ 68.460353] dmatest: Started 2 threads using dma0chan0
[ 68.465549] dmatest: Started 2 threads using dma0chan1
[ 68.465755] dmatest: dma0chan0-sg0: summary 10 tests, 0 failures 1847 iops 422686 KB/s (0)
[ 68.465963] dmatest: dma0chan0-sg1: summary 10 tests, 0 failures 1786 iops 367095 KB/s (0)
[ 68.470694] dmatest: dma0chan1-sg0: summary 10 tests, 0 failures 1938 iops 608838 KB/s (0)
[ 68.470987] dmatest: dma0chan1-sg1: summary 10 tests, 0 failures 1843 iops 517419 KB/s (0)
[ 68.503858] dmatest: Started 2 threads using dma0chan2
[ 68.509042] dmatest: Started 2 threads using dma0chan3
[ 68.509255] dmatest: dma0chan2-sg0: summary 10 tests, 0 failures 1849 iops 549944 KB/s (0)
[ 68.509454] dmatest: dma0chan2-sg1: summary 10 tests, 0 failures 1789 iops 473514 KB/s (0)
[ 68.514518] dmatest: dma0chan3-sg1: summary 10 tests, 0 failures 1830 iops 414714 KB/s (0)
[ 68.515016] dmatest: dma0chan3-sg0: summary 10 tests, 0 failures 1670 iops 512859 KB/s (0)
```

7.2.10 Real Time Clock (RTC)

Linux SDK for QorIQ Processors

Description

Provides the RTC function.

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description
<pre>Device Drivers-> Real Time Clock--> [*] Set system time from RTC on startup and resume (new) (rtc0) RTC used to set the system time (new) <[*] /sys/class/rtc/rtcN (sysfs) <[*] /proc/driver/rtc (procfs for rtc0) <[*] /dev/rtcN (character devices)</pre>	Enable RTC driver

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_RTC_LIB	y/m/n	y	Enable RTC lib
CONFIG_RTC_CLASS	y/m/n	y	Enable generic RTC class support
CONFIG_RTC_HCTOSYS	y/n	y	Set the system time from RTC when startup and resume
CONFIG_RTC_HCTOSYS_DEVICE		"rtc0"	RTC used to set the system time
CONFIG_RTC_INTF_SYSFS	y/m/n	y	Enable RTC to use sysfs
CONFIG_RTC_INTF_PROC	y/m/n	y	Use RTC through the proc interface
CONFIG_RTC_INTF_DEV	y/m/n	y	Enable RTC to use /dev interface

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/rtc/	Linux RTC driver

Device Tree Binding

Preferred node name: rtc

Property	Type	Status	Description
compatible	string	Required	Should be "dallas,ds3232"

Default node:

```
i2c@3000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl-i2c";
    reg = <0x3000 0x100>;
    interrupts = <43 2>;
    interrupt-parent = <&mpic>;
    dfsrr;
    rtc@68 {
        compatible = "dallas,ds3232";
        reg = <0x68>;
    };
};
```

Verification in Linux

Here is the rtc booting log

```
...
rtc-ds3232 1-0068: rtc core: registered ds3232 as rtc0
MC object device driver dpaa2_rtc registered
rtc-ds3232 0-0068: setting system clock to 2000-01-01 00:00:51 UTC (946684851)
...
```

NOTE: Please refer to the related DTS file to enable the RTC driver before building.
For example, LS2080AQDS board, should enable the below option:
<*> Dallas/Maxim DS3232

Change the RTC time in Linux Kernel

```
~ # ls /dev/rtc -l
lrwxrwxrwx    1 root    root          4 Jan 11 17:55 /dev/rtc -> rtc0
~ # date
Sat Jan  1 00:01:38 UTC 2000
~ # hwclock
Sat Jan  1 00:01:41 2000  0.000000 seconds
~ # date 011115502011
Tue Jan 11 15:50:00 UTC 2011
~ # hwclock -w
~ # hwclock
```

```
Tue Jan 11 15:50:36 2011 0.000000 seconds
~ # date 011115502010
Mon Jan 11 15:50:00 UTC 2010
~ # hwclock -s
~ # date
Tue Jan 11 15:50:49 UTC 2011
~ #
```

NOTE: Before using the rtc driver, make sure the /dev/rtc node in your file system is correct. Otherwise, you need to make correct node for /dev/rtc

7.2.11 Serial Advanced Technology Attachment (SATA)

Description

The driver supports NXP native SATA controller.

Module Loading

SATA driver supports either kernel built-in or module.

Kernel Configure Tree View Options	Description
<pre>Device Drivers---> <*> Serial ATA and Parallel ATA drivers ---> --- Serial ATA and Parallel ATA drivers <*> AHCI SATA support <*> Freescale QorIQ AHCI SATA support</pre>	Enables SATA controller support on ARM-based SoCs

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_SATA_AHCI=y	y/m/n	y	Enables SATA controller
CONFIG_SATA_AHCI_QORIQ=y	y/m/n	y	Enables SATA controller

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/ata/ahci_qoriq.c	Platform AHCI SATA support

Test Procedure

```
Please follow the following steps to use USB in Simics
(1) Boot up the kernel
...
fsl-sata ffe18000.sata: Sata FSL Platform/CSB Driver init
scsi0 : sata_fsl
```

```

ata1: SATA max UDMA/133 irq 74
fsl-sata ffe19000.sata: Sata FSL Platform/CSB Driver init
scsi1 : sata_fsl
ata2: SATA max UDMA/133 irq 41
...
(2) The disk will be found by kernel.
...
ata1: Signature Update detected @ 504 msecs
ata2: No Device OR PHYRDY change,Hstatus = 0xa0000000
ata2: SATA link down (SStatus 0 SControl 300)
ata1: SATA link up 1.5 Gbps (SStatus 113 SControl 300)
ata1.00: ATA-8: WDC WD1600AAJS-22WAA0, 58.01D58, max UDMA/133
ata1.00: 312581808 sectors, multi 0: LBA48 NCQ (depth 16/32)
ata1.00: configured for UDMA/133
scsi 0:0:0:0: Direct-Access      ATA              WDC WD1600AAJS-2 58.0 PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 312581808 512-byte logical blocks: (160 GB/149 GiB)
sd 0:0:0:0: Attached scsi generic sg0 type 0
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO
or FUA
sda: sda1 sda2 sda3 sda4 < sda5 sda6 >
sd 0:0:0:0: [sda] Attached SCSI disk

(3)play with the disk according to the following log.
[root@ls1046 root]# fdisk -l /dev/sda
Disk /dev/sda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

    Device Boot      Start         End      Blocks   Id  System
/dev/sda1             1           237       1903671   83  Linux
/dev/sda2           238           480       1951897+   82  Linux swap
/dev/sda3           481          9852       75280590   83  Linux
/dev/sda4          9853         19457       77152162+   f  Win95 Ext'd (LBA)
/dev/sda5          9853         14655       38580066   83  Linux
/dev/sda6         14656         19457       38572033+   83  Linux
[root@ls1046 root]#
[root@ls1046 root]# mke2fs /dev/sda1
mke2fs 1.41.4 (27-Jan-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
65280 inodes, 261048 blocks
13052 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8160 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@ls1046 root]#
[root@ls1046 root]# mkdir sata

```

```
[root@ls1046 root]# mount /dev/sda1 sata
[root@ls1046 root]# ls sata/
lost+found
[root@ls1046 root]# cp /bin/busybox sata/
[root@ls1046 root]# umount sata/
[root@ls1046 root]# mount /dev/sda1 sata/
[root@ls1046 root]# ls sata/
busybox      lost+found
[root@ls1046 root]# umount sata/
[root@ls1046 root]# mount /dev/sda3 /mnt
[root@ls1046 root]# df
Filesystem            1K-blocks      Used Available Use% Mounted on
rootfs                 852019676 794801552  13937948  99% /
/dev/root              852019676 794801552  13937948  99% /
tmpfs                  1036480         52   1036428    1% /dev
shm                    1036480          0   1036480    0% /dev/shm
/dev/sda3              74098076   4033092  66300956    6% /mnt
```

Known Bugs, Limitations, or Technical Issues

- CDROM is not supported due to the silicon limitation

7.2.12 Security Engine (SEC)

SEC Device Drivers

Introduction and Terminology

The Linux kernel contains a Scatterlist Crypto API driver for the NXP SEC v4.x, v5.x security hardware blocks.

It integrates seamlessly with in-kernel crypto users, such as IPsec, in a way that any IPsec suite that configures IPsec tunnels with the kernel will automatically use the hardware to do the crypto.

SEC v5.x is backward compatible with SEC v4.x hardware, so one can assume that subsequent SEC v4.x references include SEC v5.x hardware, unless explicitly mentioned otherwise.

SEC v4.x hardware is known in Linux kernel as 'caam', after its internal block name: Cryptographic Accelerator and Assurance Module.

There are several HW interfaces ("backends") that can be used to communicate (i.e. submit requests) with the engine, their availability depends on the SoC:

- Register Interface (RI) - available on all SoCs (though access from kernel is restricted on DPAA2 SoCs)
Its main purpose is debugging (for e.g. single-stepping through descriptor commands), though it is used also for RNG initialization.
- Job Ring Interface (JRI) - legacy interface, available on all SoCs; on most SoCs there are 4 rings
Note: there are cases when fewer rings are accessible / visible in the kernel - for e.g. when firmware like Trusted Firmware-A (TF-A) reserves one of the rings.
- Queue Interface (QI) - available on SoCs implementing DPAA v1.x (Data Path Acceleration Architecture)
Requests are submitted indirectly via Queue Manager (QMan) HW block that is part of DPAA1.
- Data Path SEC Interface (DPSECI) - available on SoCs implementing DPAA v2.x
Similar to QI, requests are submitted via Queue Manager (QMan) HW block; however, the architecture is different - instead of using the platform bus, the Management Complex (MC) bus is used, MC firmware performing needed configuration to link DP* objects - see DPAA2 Linux Software chapter for more details.

NXP provides device drivers for all these interfaces. Current chapter is focused on JRI, though some general / common topics are also covered. For QI and DPSECI backends and compatible frontends, please refer to the dedicated chapters: for DPAA1, Security Engine for DPAA2.

On top of these backends, there are the "frontends" - drivers that sit between the Linux Crypto API and backend drivers. Their main tasks are to:

- register supported crypto algorithms
- process crypto requests coming from users (via the Linux Crypto API) and translate them into the proper format understood by the backend being used
- forward the CAAM engine responses from the backend being used to the users

Note: It is obvious that QI and DPSECI backends cannot co-exist (they can be compiled in the same "multi-platform" kernel image, however run-time detection will make sure only the proper one is active). However, JRI + QI and JRI + DPSECI are valid combinations, and both backends will be active if enabled; if a crypto algorithm is supported by both corresponding frontends (for e.g. both *caamalg* and *caamalg_qi* register cbc(aes)), a user requesting cbc(aes) will be bound to the implementation having the highest "crypto algorithm priority". If the user wants to use a specific implementation:

- it is possible to ask for it explicitly by using the specific (unique) "driver name" instead of the generic "algorithm name" - please see official Linux kernel Crypto API documentation (section [Crypto API Cipher References And Priority](#)); currently default priorities are: 3000 for JRI frontend and 2000 for QI and DPSECI frontends
- crypto algorithm priority could be changed dynamically using the "Crypto use configuration API" (provided that CONFIG_CRYPTO_USER is enabled); one of the tools available that is capable to do this is "[Linux crypto layer configuration tool](#)" and an example of increasing the priority of QI frontend based implementation of echainiv(authenc(hmac(sha1),cbc(aes))) algorithm is:

```
$ ./crconf update driver "echainiv-authenc-hmac-sha1-cbc-aes-caam-qi" type 3 priority 5000
```

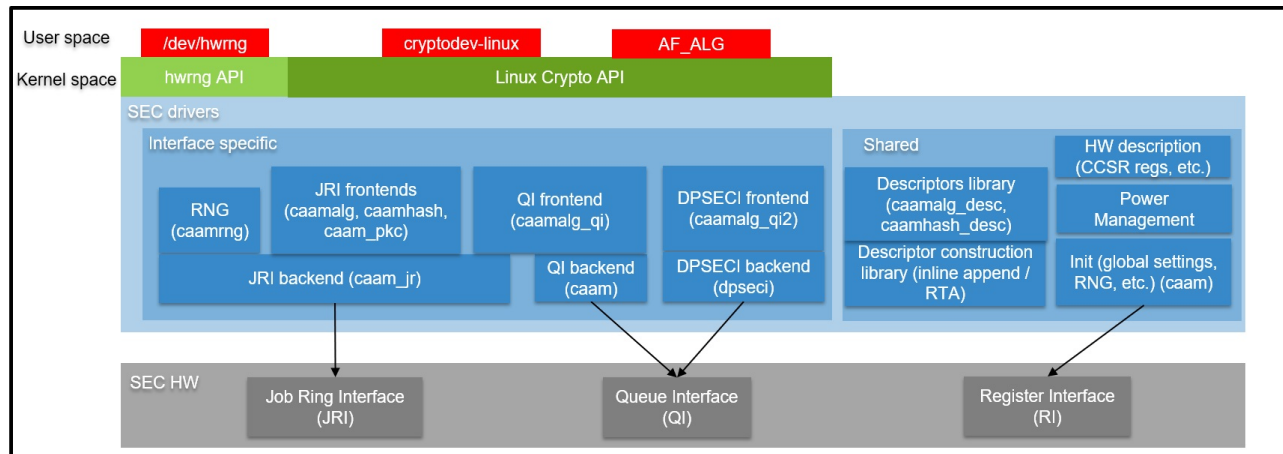


Figure 30. Linux kernel - SEC device drivers overview

Source Files

The drivers source code is maintained in the Linux kernel source tree, under *drivers/crypto/caam*. Below is a non-exhaustive list of files, mapping to Security Engine (SEC)(some files have been omitted since their existence is justified only by driver logic / design):

Source File(s)	Description	Module name
ctrl.[c,h]	Init (global settings, RNG, power management etc.)	caam
Table continues on the next page...		

Table continued from the previous page...

Source File(s)	Description	Module name
desc.h	HW description (CCSR registers etc.)	N/A
desc_constr.h	Inline append - descriptor construction library	N/A
caamalg_desc.[c,h]	(Shared) Descriptors library (symmetric encryption, AEAD)	caamalg_desc
caamrng.c	RNG (runtime)	caamrng
jr.[c,h]	JRI backend	caam_jr
qi.[c,h]	QI backend	caam
dpseci.[c,h], dpseci_cmd.h	DPSECI backend	N/A (built-in)
caamalg.c	JRI frontend (symmetric encryption, AEAD)	caamalg
caamhash.c	JRI frontend (hashing)	caamhash
caampkc.c, pkc_desc.c	JRI frontend (public key cryptography)	caam_pkc
caamalg_qi.c	QI frontend (symmetric encryption, AEAD)	caamalg_qi
caamalg_qi2.[c,h]	DPSECI frontend (symmetric encryption, AEAD)	caamalg_qi2

Module loading

CAAM device drivers can be compiled either built-in or as modules (with the exception of DPSECI backend, which is always built-in). See section [Source Files](#) on page 273 for the list of module names and section [Kernel Configuration](#) on page 274 for how kernel configuration looks like and a mapping between menu entries and modules and / or functionalities enabled.

Kernel Configuration

CAAM device drivers are located in the "Cryptographic API" -> "Hardware crypto devices" sub-menu in the kernel configuration. Depending on the target platform and / or configuration file(s) used, the output will be different; below is an example taken from NXP Layerscape SDK for ARMv8 platforms with default options:

Kernel Configure Tree View Options	Description
<pre> Cryptographic API ---> [*] Hardware crypto devices ---> <*> Freescale CAAM-Multicore platform driver backend (SEC) [] Enable debug output in CAAM driver <*> Freescale CAAM Job Ring driver backend (SEC) (9) Job Ring size [] Job Ring interrupt coalescing <*> Register algorithm implementations with the Crypto API <*> Queue Interface as Crypto API backend <*> Register hash algorithm implementations with Crypto API </pre>	Enable CAAM device drivers, options: • basic platform driver: <i>Freescale CAAM-Multicore platform driver backend (SEC)</i>; all non-DPAA2 sub-options depend on it • backends / interfaces: • <i>Freescale CAAM Job Ring driver backend (SEC)</i> - JRI; this also enables QI (QI depends on JRI) • <i>QorIQ DPAA2 CAAM (DPSECI) driver</i> - DPSECI • frontends / crypto algorithms: • symmetric encryption, AEAD, "stitched" AEAD, TLS; <i>Register algorithm implementations with the Crypto API</i> -

Table continues on the next page...

Table continued from the previous page...

Kernel Configure Tree View Options	Description
<pre> <*> Register public key cryptography implementations with Crypto API <*> Register caam device for hwrng API <M> QorIQ DPAA2 CAAM (DPSECI) driver </pre>	via JRI (<i>caamalg</i> driver) or <i>Queue Interface as Crypto API backend</i> - via QI (<i>caamalg_qi</i> drive) - Register hash algorithm implementations with Crypto API - hashing (only via JRI - <i>caamhash</i> driver) - Register public key cryptography implementations with Crypto API - asymmetric / public key (only via JRI - <i>caam_pkc</i> driver) - Register caam device for hwrng API - HW RNG (only via JRI - <i>caamrng</i> driver) - QorIQ DPAA2 CAAM (DPSECI) driver - DPSECI - options: debugging, JRI ring size, JRI interrupt coalescing
<pre> Networking support ---> Network option ---> <*> TCP/IP networking <*> IP: AH transformation <*> IP: ESP transformation <*> IP: IPsec transport mode <*> IP: IPsec tunnel mode </pre>	For IPsec support the TCP/IP networking option and corresponding sub-options should be enabled.

Device Tree binding

Property	Type	Status	Description
compatible	String	Required	fsl,sec-vX.Y (preferred) OR fsl,secX.Y

Sample Device Tree crypto node

```

crypto@30000 {
    compatible = "fsl,sec-v4.0";
    fsl,sec-era = <2>;
    #address-cells = <1>;
    #size-cells = <1>;
    reg = <0x300000 0x10000>;
    ranges = <0 0x300000 0x10000>;
    interrupt-parent = <&mpic>;
    interrupts = <92 2>;
    clocks = <&clks IMX6QDL_CLK_CAAM_MEM>,
            <&clks IMX6QDL_CLK_CAAM_ACLK>,
            <&clks IMX6QDL_CLK_CAAM_IPG>,
            <&clks IMX6QDL_CLK_EIM_SLOW>;
    clock-names = "mem", "aclk", "ipg", "emi_slow";
};

```

NOTE

See linux/Documentation/devicetree/bindings/crypto/fsl-sec4.txt file in the Linux kernel tree for more info.

How to test the drivers

To test the drivers, under the "Cryptographic API -> Cryptographic algorithm manager" kernel configuration sub-menu, ensure that run-time self tests are not disabled, i.e. the "Disable run-time self tests" entry is not set (`CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=n`). This will run standard test vectors against the drivers after they register supported algorithms with the kernel crypto API, usually at boot time. Then run test on the target system. Below is a snippet extracted from the boot log of ARMv8-based LS1046A platform, with JRI and QI enabled:

```
[...]
platform caam_qi: Linux CAAM Queue I/F driver initialised
caam 1700000.crypto: Instantiated RNG4 SH1
caam 1700000.crypto: device ID = 0x0a11030100000000 (Era 8)
caam 1700000.crypto: job rings = 4, qi = 1, dpaa2 = no
alg: No test for authenc(hmac(sha224),ecb(cipher_null)) (authenc-hmac-sha224-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha256),ecb(cipher_null)) (authenc-hmac-sha256-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha384),ecb(cipher_null)) (authenc-hmac-sha384-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha512),ecb(cipher_null)) (authenc-hmac-sha512-ecb-cipher_null-caam)
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam)
alg: No test for authenc(hmac(sha224),cbc(aes)) (authenc-hmac-sha224-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(aes))) (echainiv-authenc-hmac-sha224-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(aes))) (echainiv-authenc-hmac-sha256-cbc-aes-caam)
alg: No test for authenc(hmac(sha384),cbc(aes)) (authenc-hmac-sha384-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(aes))) (echainiv-authenc-hmac-sha384-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(aes))) (echainiv-authenc-hmac-sha512-cbc-aes-caam)
alg: No test for authenc(hmac(md5),cbc(des3_ede)) (authenc-hmac-md5-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(des3_ede))) (echainiv-authenc-hmac-md5-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des3_ede))) (echainiv-authenc-hmac-sha1-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des3_ede))) (echainiv-authenc-hmac-sha224-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des3_ede))) (echainiv-authenc-hmac-sha256-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des3_ede))) (echainiv-authenc-hmac-sha384-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des3_ede))) (echainiv-authenc-hmac-sha512-cbc-des3_ede-caam)
alg: No test for authenc(hmac(md5),cbc(des)) (authenc-hmac-md5-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(des))) (echainiv-authenc-hmac-md5-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des))) (echainiv-authenc-hmac-sha1-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des))) (echainiv-authenc-hmac-sha224-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des))) (echainiv-authenc-hmac-sha256-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des))) (echainiv-authenc-hmac-sha384-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des))) (echainiv-authenc-hmac-sha512-cbc-des-caam)
alg: No test for authenc(hmac(md5),rfc3686(ctr(aes))) (authenc-hmac-md5-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(md5),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-md5-rfc3686-ctr-aes-caam)
alg: No test for authenc(hmac(sha1),rfc3686(ctr(aes))) (authenc-hmac-sha1-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha1),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha1-rfc3686-ctr-aes-caam)
```



```
alg: No test for authenc(hmac(sha224),rfc3686(ctr(aes))) (authenc-hmac-sha224-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha224),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha224-rfc3686-ctr-aes-caam)
alg: No test for authenc(hmac(sha256),rfc3686(ctr(aes))) (authenc-hmac-sha256-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha256),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha256-rfc3686-ctr-aes-caam)
alg: No test for authenc(hmac(sha384),rfc3686(ctr(aes))) (authenc-hmac-sha384-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha384),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha384-rfc3686-ctr-aes-caam)
alg: No test for authenc(hmac(sha512),rfc3686(ctr(aes))) (authenc-hmac-sha512-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha512),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha512-rfc3686-ctr-aes-caam)
caam algorithms registered in /proc/crypto
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam-qi)
alg: No test for authenc(hmac(sha224),cbc(aes)) (authenc-hmac-sha224-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(aes))) (echainiv-authenc-hmac-sha224-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(aes))) (echainiv-authenc-hmac-sha256-cbc-aes-caam-qi)
alg: No test for authenc(hmac(sha384),cbc(aes)) (authenc-hmac-sha384-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(aes))) (echainiv-authenc-hmac-sha384-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(aes))) (echainiv-authenc-hmac-sha512-cbc-aes-caam-qi)
alg: No test for authenc(hmac(md5),cbc(des3_ede)) (authenc-hmac-md5-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(des3_ede))) (echainiv-authenc-hmac-md5-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des3_ede))) (echainiv-authenc-hmac-sha1-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des3_ede))) (echainiv-authenc-hmac-sha224-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des3_ede))) (echainiv-authenc-hmac-sha256-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des3_ede))) (echainiv-authenc-hmac-sha384-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des3_ede))) (echainiv-authenc-hmac-sha512-cbc-des3_ede-caam-qi)
alg: No test for authenc(hmac(md5),cbc(des)) (authenc-hmac-md5-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(des))) (echainiv-authenc-hmac-md5-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des))) (echainiv-authenc-hmac-sha1-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des))) (echainiv-authenc-hmac-sha224-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des))) (echainiv-authenc-hmac-sha256-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des))) (echainiv-authenc-hmac-sha384-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des))) (echainiv-authenc-hmac-sha512-cbc-des-caam-qi)
platform caam_qi: algorithms registered in /proc/crypto
caam_jr 1710000.jr: registering rng-caam
```

```
caam 1700000.crypto: caam pkc algorithms registered in /proc/crypto  
[...]
```

Crypto algorithms support

Algorithms Supported in the linux kernel scatterlist Crypto API

The Linux kernel contains various users of the Scatterlist Crypto API, including its IPsec implementation, sometimes referred to as the NETKEY stack. The driver, after registering supported algorithms with the Crypto API, is therefore used to process per-packet symmetric crypto requests and forward them to the SEC hardware.

Since SEC hardware processes requests asynchronously, the driver registers asynchronous algorithm implementations with the crypto API: ahash, ablkcipher, and aead with CRYPTO_ALG_ASYNC set in .cra_flags.

Different combinations of hardware and driver software version support different sets of algorithms, so searching for the driver name in /proc/crypto on the desired target system will ensure the correct report of what algorithms are supported.

Authenticated Encryption with Associated Data (AEAD) algorithms

These algorithms are used in applications where the data to be encrypted overlaps, or partially overlaps, the data to be authenticated, as is the case with IPsec and TLS protocols. These algorithms are implemented in the driver such that the hardware makes a single pass over the input data, and both encryption and authentication data are written out simultaneously. The AEAD algorithms are mainly for use with IPsec ESP (however there is also support for TLS 1.0 record layer encryption).

CAAM drivers currently supports offloading the following AEAD algorithms:

- "stitched" AEAD: all combinations of { NULL, CBC-AES, CBC-DES, CBC-3DES-EDE, RFC3686-CTR-AES } x HMAC- {MD-5, SHA-1, -224, -256, -384, -512}
- "true" AEAD: generic GCM-AES, GCM-AES used in IPsec: RFC4543-GCM-AES and RFC4106-GCM-AES
- TLS 1.0 record layer encryption using the "stitched" AEAD cipher suite CBC-AES-HMAC-SHA1

Encryption algorithms

The CAAM driver currently supports offloading the following encryption algorithms.

Authentication algorithms

The CAAM driver's ahash support includes keyed (hmac) and unkeyed hashing algorithms.

Asymmetric (public key) algorithms

Currently, RSA is the only public key algorithm supported.

Random Number Generation

caamrng frontend driver supports random number generation services via the kernel's built-in hwrng interface when implemented in hardware. To enable:

1. verify that the hardware random device file, e.g., /dev/hwrng or /dev/hwrandom exists. If it doesn't exist, make it with:

```
$ mknod /dev/hwrng c 10 183
```

2. verify /dev/hwrng doesn't block indefinitely and produces random data:

```
$ rngtest -C 1000 < /dev/hwrng
```

3. verify the kernel gets entropy:

```
$ rngtest -C 1000 < /dev/random
```

If it blocks, a kernel entropy supplier daemon, such as rngd, may need to be run. See [linux/Documentation/hw_random.txt](#) for more info.

Table 44. Algorithms supported by each interface / backend

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
rsa	Yes	No	No
tls10(hmac(sha1),cbc(aes))	No	Yes	Yes
authenc(hmac(md5),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(md5),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(md5),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)

Table continues on the next page...

Table 44. Algorithms supported by each interface / backend (continued)

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
authenc(hmac(md5),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha1),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha224),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha256),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha384),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha512),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(md5),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha1),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha224),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha256),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha384),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha512),ecb(cipher_null))	Yes	No	No
rfc7539(chacha20,poly1305)	Yes (LX2160A only)	No	Yes (LX2160A only)
rfc7539esp(chacha20,poly1305)	Yes (LX2160A only)	No	Yes (LX2160A only)
gcm(aes)	Yes	Yes	Yes
rfc4543(gcm(aes))	Yes	Yes	Yes
rfc4106(gcm(aes))	Yes	Yes	Yes
cbc(aes)	Yes	Yes	Yes
cbc(des3_ede)	Yes	Yes	Yes
cbc(des)	Yes	Yes	Yes
ctr(aes)	Yes	Yes	Yes
rfc3686(ctr(aes))	Yes	Yes	Yes
chacha20	No	No	Yes (LX2160A only)
Table continues on the next page...			

Table 44. Algorithms supported by each interface / backend (continued)

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
xts(aes)	Yes	Yes	Yes
hmac(md5)	Yes	No	Yes
hmac(sha1)	Yes	No	Yes
hmac(sha224)	Yes	No	Yes
hmac(sha256)	Yes	No	Yes
hmac(sha384)	Yes	No	Yes
hmac(sha512)	Yes	No	Yes
md5	Yes	No	Yes
sha1	Yes	No	Yes
sha224	Yes	No	Yes
sha256	Yes	No	Yes
sha384	Yes	No	Yes
sha512	Yes	No	Yes

CAAM Job Ring backend driver specifics

CAAM Job Ring backend driver (*caam_jr*) implements and utilizes the job ring interface (JRI) for submitting crypto API service requests from the frontend drivers (*caamalg*, *caamhash*, *caam_pkc*, *caamrng*) to CAAM engine.

CAAM drivers have a few options, most notably hardware job ring size and interrupt coalescing. They can be used to fine-tune performance for a particular use case.

The option *Freescal CAAM-Multicore platform driver backend* enables the basic platform driver (*caam*). All (non-DPAA2) sub-options depend on this.

The option *Freescal CAAM Job Ring driver backend (SEC)* enables the Job Ring backend (*caam_jr*).

The sub-option *Job Ring Size* allows the user to select the size of the hardware job rings; if requests arrive at the driver enqueue entry point in a bursty nature, the bursts' maximum length can be approximated etc. One can set the greatest burst length to save performance and memory consumption.

The sub-option *Job Ring interrupt coalescing* allows the user to select the use of the hardware's interrupt coalescing feature. Note that the driver already performs IRQ coalescing in software, and zero-loss benchmarks have in fact produced better results with this option turned off. If selected, two additional options become effective:

- *Job Ring interrupt coalescing count threshold* (CRYPTO_DEV_FSL_CAAM_INTC_THLDD)

Selects the value of the descriptor completion threshold, in the range 1-256. A selection of 1 effectively defeats the coalescing feature, and any selection equal or greater than the selected ring size will force timeouts for each interrupt.

- *Job Ring interrupt coalescing timer threshold* (CRYPTO_DEV_FSL_CAAM_INTC_TIME_THLDD)

Selects the value of the completion timeout threshold in multiples of 64 SEC interface clocks, to which, if no new descriptor completions occur within this window (and at least one completed job is pending), then an interrupt will occur. This is selectable in the range 1-65535.

The options to register to Crypto API, hwrng API respectively, allow the frontend drivers to register their algorithm capabilities with the corresponding APIs. They should be deselected only when the purpose is to perform Crypto API requests in software (on the GPPs) instead of offloading them on SEC engine.

caamhash frontend (hash algorithms) may be individually turned off, since the nature of the application may be such that it prefers software (core) crypto latency due to many small-sized requests.

caam_pkc frontend (public key / asymmetric algorithms) can be turned off too, if needed.

caamrng frontend (Random Number Generation) may be turned off in case there is an alternate source of entropy available to the kernel.

Verifying driver operation and correctness

Other than noting the performance advantages due to the crypto offload, one can also ensure the hardware is doing the crypto by looking for driver messages in dmesg.

The driver emits console messages at initialization time:

```
caam algorithms registered in /proc/crypto
caam_jr 1710000.jr: registering rng-caam
caam 1700000.crypto: caam pkc algorithms registered in /proc/crypto
```

If the messages are not present in the logs, either the driver is not configured in the kernel, or no SEC compatible device tree node is present in the device tree.

Incrementing IRQs in /proc/interrupts

Given a time period when crypto requests are being made, the SEC hardware will fire completion notification interrupts on the corresponding Job Ring:

```
$ cat /proc/interrupts | grep jr
```

	CPU0	CPU1	CPU2	CPU3		
[...]						
78:	1007	0	0	0	GICv2 103 Level	1710000.jr
79:	7	0	0	0	GICv2 104 Level	1720000.jr
80:	0	0	0	0	GICv2 105 Level	1730000.jr
81:	0	0	0	0	GICv2 106 Level	1740000.jr

If the number of interrupts fired increment, then the hardware is being used to do the crypto.

If the numbers do not increment, then first check the algorithm being exercised is supported by the driver. If the algorithm is supported, there is a possibility that the driver is in polling mode (NAPI mechanism) and the hardware statistics in debugfs (inbound / outbound bytes encrypted / protected - see below) should be monitored.

Verifying the 'self test' fields say 'passed' in /proc/crypto

An entry such as the one below means the driver has successfully registered support for the algorithm with the kernel crypto API:

```
name      : cbc(aes)
driver    : cbc-aes-caam
module    : kernel
priority  : 3000
refcnt    : 1
selftest  : passed
internal  : no
type      : givcipher
async     : yes
blocksize : 16
min keysize : 16
max keysize : 32
ivsize    : 16
geniv     : <built-in>
```

Note that although a test vector may not exist for a particular algorithm supported by the driver, the kernel will emit messages saying which algorithms weren't tested, and mark them as 'passed' anyway:

```
[...]
alg: No test for authenc(hmac(sha224),ecb(cipher_null)) (authenc-hmac-sha224-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha256),ecb(cipher_null)) (authenc-hmac-sha256-ecb-cipher_null-caam)
[...]
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam)
[...]
alg: No test for authenc(hmac(sha512),rfc3686(ctr(aes))) (authenc-hmac-sha512-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha512),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha512-rfc3686-ctr-aes-caam)
[...]
```

Examining the hardware statistics registers in debugfs

When using the JRI or QI backend, performance monitor registers can be checked, provided CONFIG_DEBUG_FS is enabled in the kernel's configuration. If debugfs is not automatically mounted at boot time, then a manual mount must be performed in order to view these registers. This normally can be done with a superuser shell command:

```
$ mount -t debugfs none /sys/kernel/debug
```

Once done, the user can read controller registers in /sys/kernel/debug/1700000.crypto/ctl. It should be noted that debugfs will provide a decimal integer view of most accessible registers provided, with the exception of the KEK/TDSK/TKEK registers; those registers are long binary arrays, and should be filtered through a binary dump utility such as hexdump.

Specifically, the CAAM hardware statistics registers available are:

fault_addr, or FAR (Fault Address Register): - holds the value of the physical address where a read or write error occurred.

fault_detail, or FADR (Fault Address Detail Register): - holds details regarding the bus transaction where the error occurred.

fault_status, or CSTA (CAAM Status Register): - holds status information relevant to the entire CAAM block.

ib_bytes_decrypted: - holds contents of PC_IB_DECRYPT (Performance Counter Inbound Bytes Decrypted Register)

ib_bytes_validated: - holds contents of PC_IB_VALIDATED (Performance Counter Inbound Bytes Validated Register)

ib_rq_decrypted: - holds contents of PC_IB_DEC_REQ (Performance Counter Inbound Decrypt Requests Register)

kek: - holds contents of JDKEKR (Job Descriptor Key Encryption Key Register)

ob_bytes_encrypted: - holds contents of PC_OB_ENCRYPT (Performance Counter Outbound Bytes Encrypted Register)

ob_bytes_protected: - holds contents of PC_OB_PROTECT (Performance Counter Outbound Bytes Protected Register)

ob_rq_encrypted: - holds contents of PC_OB_ENC_REQ (Performance Counter Outbound Encrypt Requests Register)

rq_dequeued: - holds contents of PC_REQ_DEQ (Performance Counter Requests Dequeued Register)

tdsk: - holds contents of TDKEKR (Trusted Descriptor Key Encryption Key Register)

tkek: - holds contents of TDSKR (Trusted Descriptor Signing Key Register)

For more information see section "Performance Counter, Fault and Version ID Registers" in the Security (SEC) Reference Manual (SECRM) of each SoC (available on company's website).

Note: for QI backend there is also *qi_congested*: SW-based counter that shows how many times queues going to / from CAAM to QMan hit the congestion threshold.

Kernel configuration to support CAAM device drivers

Using the driver

Once enabled, the driver will forward kernel crypto API requests to the SEC hardware for processing.

Running IPsec

The IPsec stack built-in to the kernel (usually called NETKEY) will automatically use crypto drivers to offload crypto operations to the SEC hardware. Documentation regarding how to set up an IPsec tunnel can be found in corresponding open source IPsec suite packages, e.g. strongswan.org, openswan, setkey, etc. DPAA2-specific section contains a generic helper script to configure IPsec tunnels.

Running OpenSSL

Please see Hardware Offloading with OpenSSL for more details on how to offload OpenSSL cryptographic operations in the SEC crypto engine (via cryptodev).

Executing custom descriptors

SEC drivers have public descriptor submission interfaces corresponding to the following backends:

- JRI: `drivers/crypto/caam/jr.c:caam_jr_enqueue()`
- QI: `drivers/crypto/caam/qi.c:caam_qi_enqueue()`
- DPSECI: `drivers/crypto/caam/caamalg_qi2.c:dpaa2_caam_enqueue()`

caam_jr_enqueue()

Name

`caam_jr_enqueue` — Enqueue a job descriptor head. Returns 0 if OK, -EBUSY if the ring is full, -EIO if it cannot map the caller's descriptor.

Synopsis

```
int caam_jr_enqueue (struct device *dev, u32 *desc,  
    void (*cbk) (struct device *dev, u32 *desc, u32 status, void *areq),  
    void *areq);
```

Arguments

`dev`: contains the job ring device that is to process this request.

`desc`: descriptor that initiated the request, same as “desc” being argued to `caam_jr_enqueue`.

`cbk`: pointer to a callback function to be invoked upon completion of this request. This has the form: `callback(struct device *dev, u32 *desc, u32 stat, void *arg)`

`areq`: optional pointer to a user argument for use at callback time.

caam_qi_enqueue()

Name

`caam_qi_enqueue` — Enqueue a frame descriptor (FD) into a QMan frame queue. Returns 0 if OK, -EIO if it cannot map the caller's S/G array, -EBUSY if QMan driver fails to enqueue the FD for some reason.

Synopsis

```
int caam_qi_enqueue(struct device *qidev, struct caam_drv_req *req);
```

Arguments

`qidev`: contains the queue interface device that is to process this request.

`req`: pointer to the request structure the driver application should fill while submitting a job to driver, containing a callback function and its parameter, Queue Manager S/Gs for input and output, a per-context structure containing the CAAM shared descriptor etc.

dpaa2_caam_enqueue()

Name

`dpaa2_caam_enqueue` — Enqueue a frame descriptor (FD) into a QMan frame queue. Returns 0 if OK, `-EBUSY` if QMan driver fails to enqueue the FD for some reason or if congestion is detected.

Synopsis

```
int dpaa2_caam_enqueue(struct device *dev, struct caam_request *req);
```

Arguments

`dev`: DPSECI device.

`req`: pointer to the request structure the driver application should fill while submitting a job to driver, containing a callback function and its parameter, Queue Manager S/Gs for input and output, a per-context structure containing the CAAM shared descriptor etc.

Please refer to the source code for usage examples.

Supporting Documentation

DPAA1-specific SEC details - Queue Interface (QI)

DPAA2-specific SEC details - Data Path SEC Interface (DPSECI)

7.2.13 Universal Serial Bus Interfaces

See table below for USB controllers which are present on the SoCs:

SoC	No. of USB 3.0 controllers present	No. of USB 2.0 controllers present
LS1012A	1	1
LS1021A	1	1
LS1028A	2	0
LS1043A	3	0
LS1046A	3	0
LS1088A	2	0
LS2088A	2	0
LX2160A	2	0

Typical USB nodes on device trees:

- USB 3.0 controller

```
usb0: usb3@3100000 {
    compatible = "snps,dwc3";
    reg = <0x0 0x3100000 0x0 0x10000>;
    interrupts = <0 80 IRQ_TYPE_LEVEL_HIGH>;
    dr_mode = "host";
    snps,quirk-frame-length-adjustment = <0x20>;
    snps,dis_rxdet_inp3_quirk;
    status = "disabled";
    snps,incr-burst-type-adjustment = <1>, <4>, <8>, <16>;
};
```

- USB 2.0 controller

```
usb1: usb2@8600000 {
    compatible = "fsl-usb2-dr-v2.5", "fsl-usb2-dr";
    reg = <0x0 0x8600000 0x0 0x1000>;
    interrupts = <0 139 0x4>;
    dr_mode = "host";
    phy_type = "ulpi";
};
```

7.2.13.1 USB 3.0 Controller (DesignWare USB3)

Description

The U-Boot and Linux kernel driver support DWC3 USB 3.0 Dual-Role-Device (DRD) controller.

U-Boot

Host Mode

With default configuration of LSDK, host mode should be ready to use, below are related CONFIG files to select.

Configure Tree View Options

Configure Tree View Options	Description
<pre>U-Boot--> USB support --> [*] Enable driver model for USB [*] xHCI HCD (USB 3.0) support [*] Designware USB3 DRD Core Support ... [*] Support for NXP Layerscape on-chip xHCI USB controller ... [*] USB Mass Storage support</pre>	Enables USB host controller support

Device Tree (take arch/arm/dts/fsl-ls1012a.dtsi as example)

```
usb1: usb3@2f00000 {
    compatible = "fsl,layerscape-dwc3";
    reg = <0x0 0x2f00000 0x0 0x10000>;
    interrupts = <0 61 0x4>;
    dr_mode = "host";
};
```

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/usb/host/xhci.c	USB HOST xHCI Controller stack
Table continues on the next page...	

Table continued from the previous page...

Source File	Description
drivers/usb/host/xhci.c	USB HOST xHCI Controller stack
drivers/usb/host/xhci-fsl.c	FSL USB HOST xHCI Controller driver, basing on dwc3 driver
drivers/usb/host/xhci-dwc3.c	DWC3 controller driver
drivers/usb/host/usb-uhcd.c	USB host driver
common/usb.c	USB generic driver
common/usb_hub.c	USB hub driver
cmd/usb.c	USB command-line support

Verification

- Enumeration
 - Plug USB drive.
 - Boot RDB board to U-Boot console, type below commands to scan USB devices
- =USB

```
=> usb start
starting USB...
USB0: Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus 0 for devices... 2 USB Device(s) found
      scanning usb for storage devices... 1 Storage Device(s) found
=> usb treed
USB device tree:
  1 Hub (5 Gb/s, 0mA)
    | U-Boot XHCI Host Controller
    |
+-2 Mass Storage (5 Gb/s, 224mA)
    SanDisk Extreme 4C530001060207103322
=> usb info
1: Hub, USB Revision 3.0
- U-Boot XHCI Host Controller
- Class: Hub
- PacketSize: 512 Configurations: 1
- Vendor: 0x0000 Product 0x0000 Version 1.0
Configuration: 1
- Interfaces: 1 Self Powered 0mA
Interface: 0
- Alternate Setting 0, Endpoints: 1
- Class Hub
- Endpoint 1 In Interrupt MaxPacket 8 Interval 255ms

2: Mass Storage, USB Revision 3.0
- SanDisk Extreme 4C530001060207103322
- Class: (from Interface) Mass Storage
- PacketSize: 512 Configurations: 1
```

```
- Vendor: 0x0781 Product 0x558b Version 1.0
  Configuration: 1
- Interfaces: 1 Bus Powered 224mA
  Interface: 0
- Alternate Setting 0, Endpoints: 2
- Class Mass Storage, Transp. SCSI, Bulk only
- Endpoint 1 In Bulk MaxPacket 1024
- Endpoint 2 Out Bulk MaxPacket 1024
```

Mass Storage device read write

```
=> md a0000000
a0000000: fefeff7d f3bfffff dffffeff bff77bf2 .....{..
a0000010: efefeffe 7b7f33ff 7dfeff7c 7effff77 .....3.{|..}w..~
a0000020: fdaefccf 737ffffb 75ffffdf febfbffa .....s...u...
a0000030: 7fccff4f f3ff7ffb fee6fcfc bffb3ff7 0.....?..
a0000040: dfdebfcf 37bf7b37 ffefdfcc 3337fff3 ....7{.7.....73
a0000050: ffeddeee 737333b7 fbefefdf bbf3f7f3 .....3ss.....
a0000060: defcffe f7bff7fb ffdfffce 3bbf77ff .....w.;
a0000070: dfcfffef b3fb7fb6 e2dfeede b7b3bff7 .....
a0000080: feffbfcf 73bf3fb3 dffaceff 3bb6b773 .....?.s....s.;
a0000090: fdcffece 7bbfbf7b fdeeffdc f3eff7f7 ....{..{.....
a00000a0: dfecdffe fb3733b7 d9deffdf 73f737bf .....37.....7.s
a00000b0: c76effde faf3bb3f deffdeeb 2f7fb37b ..n.?.....{../
a00000c0: fffcef5b 7bf333bf fedffefe 773f7377 [...3.{...ws?w
a00000d0: fbfdfdfd f7bb73f7 fffffedd ff37bf3e .....s.....>.7.
a00000e0: dfd9fecf 3f77fbb3 77cfdeee b3f77f73 .....w?...ws...
a00000f0: cfecffde bfff33fb ffe6ffdf fb73337f .....3.....3s.
=> mw a0000000 ffffaaaa 100
=> md a0000000
a0000000: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000010: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000020: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000030: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000040: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000050: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000060: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000070: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000080: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a0000090: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000a0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000b0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000c0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000d0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000e0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
a00000f0: ffffaaaa ffffaaaa ffffaaaa ffffaaaa .....
=> usb write a0000000 0 100
usb write: device 0 block # 0, count 256 ... 256 blocks written: OK
=> md b0000000
b0000000: 77fdff79 c97cefdb a7dffffb fffedddf y..w..|.....
b0000010: fb9ff7f3 fdfeedef feb7db9 cffbccef .....}.....
b0000020: ff7ebf7b fd6efffa 5efbbfbf cfffffff {..~..n....^....
b0000030: bbf7f7e7 fcfedcbd f7f3bff7 fedceded .....
b0000040: df7b3337 cfcefcf b7afb7f ddcddfce 73{.....
b0000050: ffb3bdf3 dedfefed ff3bfef3 fefffdff .....;.....
b0000060: 333f9b37 efccffef f7bbffff 5fceefff 7.?3....._
b0000070: f7bffa37 7edeefff ffff3ff3 fffedfee 7.....~.?.....
b0000080: 7b37fb3a dffefecf ffff93f5 eeceffcf :.7{.....
b0000090: ff3f1ffb fffcdcf f77bf77b ddefffeef ..?.....{.....
```

```

b00000a0: 52b77bba acfffcff bdfbf33 feffebff .{.R....3.....
b00000b0: ffffffff7f fe6eeddd 7ffbbb3b 6dffceff .....n.;.....m
b00000c0: 3bfbbd73 fd7fedef ff73f3ef fefaedde s.;.....s.....
b00000d0: 7f77ff73 4ffdcdee 7f3b7f72 ecfbedef s.w....Or.;....
b00000e0: f73b7f77 fffdfbfd f7f5fffb eddefefc w.;.....
b00000f0: bfb3bfa3 cdfdfcce 655fbfbb eeffcefd ....._e....
=> usb read b0000000 0 100
usb read: device 0 block # 0, count 256 ... 256 blocks read: OK
=> md b0000000
b0000000: ffffffff ffffffff ffffffff ffffffff .....
b0000010: ffffffff ffffffff ffffffff ffffffff .....
b0000020: ffffffff ffffffff ffffffff ffffffff .....
b0000030: ffffffff ffffffff ffffffff ffffffff .....
b0000040: ffffffff ffffffff ffffffff ffffffff .....
b0000050: ffffffff ffffffff ffffffff ffffffff .....
b0000060: ffffffff ffffffff ffffffff ffffffff .....
b0000070: ffffffff ffffffff ffffffff ffffffff .....
b0000080: ffffffff ffffffff ffffffff ffffffff .....
b0000090: ffffffff ffffffff ffffffff ffffffff .....
b00000a0: ffffffff ffffffff ffffffff ffffffff .....
b00000b0: ffffffff ffffffff ffffffff ffffffff .....
b00000c0: ffffffff ffffffff ffffffff ffffffff .....
b00000d0: ffffffff ffffffff ffffffff ffffffff .....
b00000e0: ffffffff ffffffff ffffffff ffffffff .....
b00000f0: ffffffff ffffffff ffffffff ffffffff .....
=>

```

Linux Kernel Host Mode

With default configuration of LSDK, host mode should be ready to use, below are related CONFIGs that should have been selected.

Configure Tree View Options

Configure Tree View Options	Description
USB support ---> [*] xHCI HCD (USB3.0) support	USB host controller support.
[*] USB Mass Storage support	USB mass storage support.
[*] DesignWare USB3 DRD Core support [*] Dwc3 Mode Selection [X] Dual Role mode	DesignWare USB3 DRD Core Support.
Device Drivers --> HID support --> USB HID support [*] USB HID transport layer USB HID support	USB HID support

Device Tree (take arch/arm/boot/dts/freescale/fsl-ls1012a.dtsi as example)

```
usb0: usb3@2f00000 {
    compatible =
    "snps,dwc3";          reg = <0x0
    0x2f00000 0x0 0x10000>;          interrupts = <0
    60 0x4>;          dr_mode = "host";
    snps,quirk-frame-length-adjustment = <0x20>;
    snps,dis_rxdet_inp3_quirk;
};
```

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/usb/core/*	USB subsystem/framework
drivers/usb/host/xhci.c xhci-mem.c xhci-ring.c xhci-hub.c	USB xHCI (host) driver
drivers/usb/storage/scsiglue.c protocol.c transport.c usb.c	USB Mass Storage (device) driver

Verification

Enumeration

- Plug USB drive
- Boot RDB board to Linux console, type below commands to list USB devices(s):

```
root@ls1012ardb:~# lsusb
Bus 002 Device 002: ID 0781:558b <-- Whose 'Device' is 002 should be a USB device we found
Bus 001 Device 001: ID 1d6b:0002
Bus 002 Device 001: ID 1d6b:0003
```

- Mass Storage device read write

```
root@ls1012ardb:~# ls /dev/sd*
/dev/sda /dev/sda1
root@ls1012ardb:~# udevadm info -q all -n /dev/sda | grep -e usb
P: /devices/platform/soc/2f00000.usb3/xhci-hcd.0.auto/usb2/2-1/2-1:1.0/host1/target1:0:0/1:0:0:0/block/sda
S: disk/by-id/usb-SanDisk_Extreme_4C530001020308102474-0:0
S: disk/by-path/platform-xhci-hcd.0.auto-usb-0:1:1.0-scsi-0:0:0:0
E: DEVLINKS=/dev/disk/by-id/usb-SanDisk_Extreme_4C530001020308102474-0:0 /dev/disk/by-path/platform-xhci-hcd.0.auto-usb-0:1:1.0-scsi-0:0:0:0 /dev/disk/by-uuid/928B-C6D2
E: DEVPATH=/devices/platform/soc/2f00000.usb3/xhci-hcd.0.auto/usb2/2-1/2-1:1.0/host1/target1:0:0/1:0:0:0/block/sda
E: ID_BUS=usb
E: ID_PATH=platform-xhci-hcd.0.auto-usb-0:1:1.0-scsi-0:0:0:0
E: ID_PATH_TAG=platform-xhci-hcd_0_auto-usb-0_1_1_0-scsi-0_0_0_0
E: ID_USB_DRIVER=usb-storage
root@ls1012ardb:~# mkfs.ext2 /dev/sda1 # Format USB drive partition 1 with EXT2
mke2fs 1.42.9 (28-Dec-2013)
[ 1032.401738] urandom_read: 1 callbacks suppressed
[ 1032.401745] random: mkfs.ext2: uninitialized urandom read (16 bytes read)
[ 1032.413812] random: mkfs.ext2: uninitialized urandom read (16 bytes read)
Filesystem label=
```

```

OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
3833856 inodes, 15318784 blocks
765939 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
468 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
root@ls1012ardb:~# mount /dev/sda1 /mnt      # Manually mount USB drive to file system
root@ls1012ardb:~# cd /mnt
root@ls1012ardb:/mnt# dd if=/dev/zero of=./test_400MB bs=1M count=400 # Write test
400+0 records in
400+0 records out
419430400 bytes (419 MB) copied, 4.54194 s, 92.3 MB/s
root@ls1012ardb:/mnt# sync      # Make sure ./test_400MB has been written to drive
root@ls1012ardb:/mnt# md5sum test_400MB      # Read file out, do MD5 checksum
61eabaf2bf278703738b433ff884c91f  test_400MB

```

HID use case

- Boot RDB board to Linux console,
- Plug USB mouse/keyboard, then below message appears:

```
root@ls1012ardb:/mnt#
[ 3415.406370] usb 1-1: new low-speed USB device number 2 using xhci-hcd
[ 3415.582798] input: PixArt Dell MS116 USB Optical Mouse as /devices/platform/soc/2f00000.usb3/xhci-hcd.0.auto/usb1/1-1/1-1:1.0/0003:413C:301A.0001/input/input0
[ 3415.600539] hid-generic 0003:413C:301A.0001: input: USB HID v1.11 Mouse [PixArt Dell MS116 USB Optical Mouse] on usb-xhci-hcd.0.auto-1/input0
```

- Type below commands to begin receiving data, then move mouse or press keys on keyboard, you see that some unreadable data popping up.

fYRY fYRY fYx fYx fYx fYx fYx
 fY fY fYQ fYQ fYQ fY fY fY fY fY

- Ethernet Use case

- Rebuild Linux kernel, make sure your USB network card-related driver has been included. For example: TP-LINK USB 3.0 to Gigabit Ethernet Network Adapter should select below CONFIG in menuconfig:

```
Symbol : USB_RTL8152 [=y]
Type   : tristate
Prompt : Realtek RTL8152/RTL8153 Based USB Ethernet Adapters
Location:
->Device Drivers
    ->Network device support (NETDEVICES [=Y])
(1)          -> USB Network Adapters (USB_NET_Drivers [=y])
Defined at drivers/net/usb/Kconfig:98
```

```
Depends on: NRYFRBOVRD [=y] && USB_NET_Drivers [=Y]  
Selects: MII [=y]
```

- Boot RDB board to Linux console,
- Plug USB network, then below log appears:

```
root@ls1012ardb:~# [ 18.677661] usb 1-1: new  
high-speed USB device number 2 using xhci-hcd[ 19.529741] usb 1-1: reset  
high-speed USB device number 2 using xhci-hcd[ 19.706274] random: fast init  
done[ 19.742956] r8152 1-1:1.0 eth1:  
v1.09.9
```

- Configure IP and do ping test

```
root@ls1012ardb:~# ifconfig eth1 192.168.0.2  
[ 110.365205] IPv6: ADDRCONF(NETDEV_UP): eth1: link is not ready  
root@ls1012ardb:~# [ 110.394378] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready  
[ 110.401079] r8152 1-1:1.0 eth1: carrier on  
  
root@ls1012ardb:~# ping 192.168.0.1  
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.  
64 bytes from 192.168.0.1: icmp_seq=2 ttl=63 time=10.876 ms  
64 bytes from 192.168.0.1: icmp_seq=3 ttl=63 time=10.829 ms  
64 bytes from 192.168.0.1: icmp_seq=4 ttl=63 time=10.900 ms  
64 bytes from 192.168.0.1: icmp_seq=5 ttl=63 time=10.844 ms  
64 bytes from 192.168.0.1: icmp_seq=6 ttl=63 time=10.908 ms
```

Speaker and Microphone

- A play utility can be used to list the available sound cards, e.g., Here Jabra 410 USB speaker is detected as a second sound card and can be addressed as `-D hw:1.0` OR `-c1`:

```
[root@freescale ~]$ aplay -l  
**** List of PLAYBACK Hardware Devices ****  
card 0: FSLVF610TWRBOAR [FSL-VF610-TWR-BOARD], device 0: HiFi sgtl5000-0 [ ]  
Subdevices: 1/1  
Subdevice #0: subdevice #0  
card 1: USB [Jabra SPEAK 410 USB], device 0: USB Audio [USB Audio] Subdevices: 1/1  
Subdevice #0: subdevice #0
```

- Sample wav file can be played using the below command:

```
[root@freescale ~]$ aplay -D hw:1,0  
LYNC_fsringing.wavPlaying WAVE 'LYNC_fsringing.wav' : Signed 16 bit Little Endian, Rate 48000  
Hz, Stereo
```

- Sample wav file can be recorded using the below command:

```
[root@freescale ~]$ arecord -f S16_LE -t wav -Dhw:1,0 -r 16000 foobar.wav -d 5  
Recording WAVE 'foobar.wav' : Signed 16 bit Little Endian, Rate 16000 Hz, Mono
```

NOTE

If recorded audio is not played, try to use `"-D plughw:1,0"` in above command.

- Audio controls can be checked using the below command, control details, and name of the controls can be checked from output of “amixer –c1” as below:

```
[root@freescale ~]$ amixer -c1 controls
numid=3,iface=MIXER,name='PCM Playback Switch'
numid=4,iface=MIXER,name='PCM Playback Volume'
numid=5,iface=MIXER,name='Headset Capture Switch'
numid=6,iface=MIXER,name='Headset Capture Volume'
numid=2,iface=PCM,name='Capture Channel Map'
numid=1,iface=PCM,name='Playback Channel Map'

[root@freescale ~]$ amixer -c1
Simple mixer control 'PCM',0  Capabilities: pvolume pvolume-joined pswitch pswitch-joined
penum
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 4 [36%] [-20.00dB] [on]

Simple mixer control 'Headset',0  Capabilities: cvolume cvolume-joined cswitch cswitch-joined
penum
  Capture channels: Mono
  Limits: Capture 0 - 7
  Mono: Capture 5 [71%] [0.00dB] [on]
```

For Example, in above output there are two controls named “PCM” and “Headset” for Speaker and microphone respectively. Sample Audio controls Usage: a. mute/unmute

```
[root@freescale ~]$ amixer -c1 set PCM  mute
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch pswitch-joined
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 2 [18%] [-28.00dB] [off]
[root@freescale ~]$ amixer -c1 set PCM  unmute
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch pswitch-joined
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 2 [18%] [-28.00dB] [on]
Aplay utility can be used to list the available sound cards e.g. Here Jabra 410 USB speaker is
detected as a second sound card and can be addressed as -D hw:1,0 OR -c1:
```

Volume up/down – Below commands are trying to set volume to 11 and 2 performing volume up and down respectively.

```
root@freescale ~]$ amixer -c1 set PCM 11
Simple mixer control 'PCM',0
Capabilities: pvolume pvolume-joined pswitch pswitch-joined
Playback channels: Mono
Limits: Playback 0 - 11
Mono: Playback 11 [100%] [8.00dB] [on]
[root@freescale ~]$ amixer -c1 set PCM 2
Simple mixer control 'PCM',0
Capabilities: pvolume pvolume-joined pswitch pswitch-joined
Playback channels: Mono
Limits: Playback 0 - 11
Mono: Playback 2 [18%] [-28.00dB] [on]
```

- Device mode (Gadget driver)

Important note: Device mode enabling requires manually **insmod** some ko files at runtime, make sure use the ko files which built together with that kernel image, otherwise you might encounter failures like below:

```
root@ls1043a:/run/media/mmcblk0p1 # insmod libcomposite.ko
[ 2748.620682] libcomposite: version magic '4.14.47-50925-gd677346-dirty SMP preempt mod_unload
aarch64' should be '4.14.47-50925-gd224085 SMP preempt mod_unload aarch64'
insmod: ERROR: could not insert module libcomposite.ko: Invalid module format
```

- **Mass Storage gadget**

Basing on default configuration of LSDK, also select below options in Linux kernel menuconfig (follow the highlighted choice)

Configure Tree View Options

Configure Tree View Options	Description
USB Gadget support ---> <M> USB Gadget functions configurable through configs [*] Mass storage	USB host controller support.
<M> USB Gadget precomposed configurations	USB configuration support.
<M> Mass Storage Gadget	Mass storage support.

Device Tree update, change property **dr_mode**'s data from “host” to “peripheral”, add property **maximum-speed** = “super-speed”; as below:

```
usb0: usb3@2f00000 {
    compatible = "snps,dwc3";
    reg = <0x0 0x2f00000 0x0 0x10000>;
    interrupts = <0 60 0x4>;
    dr_mode = "peripheral";
    snps,quirk-frame-length-adjustment = <0x20>;
    snps,dis_rxdet_inp3_quirk;
    maximum-speed = "super-speed";
};
```

NOTE

Make sure to modify the correct USB nodes that mapped to the physical USB port that your are verifying, and you can only change one USB node.

Source Files

Source File	Description
drivers/usb/gadget/function/storage_common.c	Common definitions for mass storage functionality
drivers/usb/gadget/function/f_mass_storage.c	Mass Storage USB Composite Function
drivers/gadget/legacy/mass_storage.c	Mass Storage USB Gadget

Verification (test with Win7 as host)

- Build kernel, then copy below ko files to an SD card.
 - ./drivers/usb/gadget/libcomposite.ko
 - ./drivers/usb/gadget/function/usb_f_mass_storage.ko
 - ./drivers/usb/gadget/legacy/g_mass_storage.ko
- Insert that SD card into RDB board SD slot.
- Boot RDB board with that Linux kernel
- In RDB board Linux console, execute below commands (assume that you copy those ko files at SD card root folder, and mount to /run/media/mmcblk0p1/)

```

root@ls1043a:/ # df
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/root        85352 65515    15430  81% /
devtmpfs         1940036    4    1940032    1% /dev
tmpfs            1961116   132    1960984    1% /run
tmpfs            1961116   172    1960944    1% /var/volatile
/dev/mmcblk0p1   3931136 32964   3898172    1% /run/media/mmcblk0p1
root@ls1043a:~#cd /run/media/mmcblk0p1/      # this is where you put your ko files
root@ls1043a:/run/media/mmcblk0p1/ # dd if=/dev/zero of=./test bs=1M count=500
root@ls1043a:/run/media/mmcblk0p1/ # insmod libcomposite.ko
root@ls1043a:/run/media/mmcblk0p1/ # insmod usb_f_mass_storage.ko
root@ls1043a:/run/media/mmcblk0p1/ # insmod g_mass_storage.ko file=/run/media/mmcblk0p1/test
[ 780.758286] Mass Storage Function, version: 2009/09/11
[ 780.763465] LUN: removable file: (no medium)
[ 780.767791] LUN: file: /run/media/mmcblk0p1/test
[ 780.772406] Number of LUNs=1
[ 780.775355] g_mass_storage gadget: Mass Storage Gadget, version: 2009/09/11
[ 780.782322] g_mass_storage gadget: userspace failed to provide iSerialNumber
[ 780.789371] g_mass_storage gadget: g_mass_storage ready

```

- Connect USB cable with PC and RDB board
 - You can see Windows Device Manager as Linux File-Stor Gadget USB Drive.

NOTE

Some times you need manually allocate a drive name/letter in My Computer. After that manually format that disk to keep it in ready status.

- **Ethernet gadget**
 - Basing on default configuration of LSDK, please also select below options in Linux kernel menuconfig (follow the highlighted choice)

Configure Tree View Options

Configure Tree View Options	Description
<pre> USB Gadget support ---> <M> USB Gadget functions configurable through configfs [*] Mass storage </pre>	USB host controller support.
Table continues on the next page...	

Table continued from the previous page...

Configure Tree View Options	Description
<M> SDB Gadget precomposed configurations	SDB configuration support.
<M> Ethernet Gadget (with CDC Ethernet support)	Ethernet gadget support.

Device Tree update, change property **dr_mode**'s data from **"host"** to **"peripheral"**, add property **maximum-speed = "super-speed"**; as below:

```
usb0: usb3@2f00000 {
    compatible = "snps,dwc3";
    reg = <0x0 0x2f00000 0x0 0x10000>;
    interrupts = <0 60 0x4>;
    dr_mode = "peripheral";
    snps,quirk-frame-length-adjustment = <0x20>;
    snps,dis_rxdet_inp3_quirk;
    maximum-speed = "super-speed";
};
```

NOTE

Make sure to modify the correct USB nodes that mapped to the physical USB port that you are verifying, and you can only change one USB node.

Source Files

Source File	Description
drivers/usb/gadget/function/u_ether.c	Ethernet-over-USB link layer utilities for Gadget stack
drivers/usb/gadget/function/f_ecm.c	USB CDC Ethernet (ECM) link function driver
drivers/usb/gadget/function/f_subset.c	"CDC Subset" Ethernet link function driver
drivers/usb/gadget/function/f_rndis.c	RNDIS link function driver
drivers/usb/gadget/function/rndis.c	RNDIS MSG parser
drivers/usb/gadget/legacy/ether.c	Ethernet gadget driver, with CDC and non-CDC options

Verification

- Build Linux Kernel, then copy ko files to a SD card.
- Insert that SD card into RDB board.
- Connect RDB board and Windows PC host port with USB cable.
- Boot RDB board with above kernel.

- Execute below shell commands to insmod related ko files on RDB board.

```
root@ls1043a:/run/media/mmcblk0p1# insmod libcomposite.ko
root@ls1043a:/run/media/mmcblk0p1# insmod u_ether.ko
root@ls1043a:/run/media/mmcblk0p1# insmod usb_f_ecm.ko
root@ls1043a:/run/media/mmcblk0p1# insmod usb_f_ecm_subset.ko
root@ls1043a:/run/media/mmcblk0p1# insmod usb_f_rndis.ko
root@ls1043a:/run/media/mmcblk0p1# insmod g_ether.ko
[ 138.046732] using random self ethernet address
[ 138.051188] using random host ethernet address
[ 138.055884] usb0: HOST MAC 5e:4a:86:d0:dc:b6
[ 138.060219] usb0: MAC c2:53:e1:5b:d0:d9
[ 138.064100] using random self ethernet address
[ 138.068549] using random host ethernet address
[ 138.073041] g_ether gadget: Ethernet Gadget, version: Memorial Day 2008
[ 138.079653] g_ether gadget: g_ether ready
```

- Install Microsoft RNDIS driver on Windows 7 for ping test

1. Right click Computer and select Manage. From System Tools, select Device Manager.

It displays a list of devices currently connected with the development PC. In the list, you can see RNDIS Kitl with an exclamation mark implying that driver has not been installed.

2. Right click RNDIS Kitl and select Update Driver Software when prompted to choose how to search for device driver software, choose Browse my computer for driver software.

Browse for driver software on your computer appears.

3. Select Let me pick from a list of device drivers on My Computer.

The Update Driver Software - RNDIS Kitl window appears.

4. Select the device type as Select Network adapters, as RNDIS emulates a network connection.

5. Select Microsoft Corporation from the Manufacturer list in the Select Network Adapter window.

6. Select Remote NDIS compatible device from the Network Adapter frame and click Next.

After, several minutes of installation you can see a message as "Windows has successfully updated your driver software." and the RNDIS device is ready to use.

After, successful installation you can see RNDIS/Ethernet Gadget under Network adapters.

7. Allocate IP for USB interface to the ping test.

On RDB board Linux console configure the network interface as shown below:

```
root@ls1043a:/run/media/mmcblk0p1 # ifconfig -a
..... # <snip>
usb0      Link encap:Ethernet  HWaddr c2:53:e1:5b:d0:d9
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ls1043a:/run/media/mmcblk0p1 # ifconfig usb0 192.168.5.3
root@ls1043a:/run/media/mmcblk0p1 # ifconfig usb0
usb0      Link encap:Ethernet  HWaddr c2:53:e1:5b:d0:d9
          inet addr:192.168.5.3  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:1000  
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

- Configuring Network interface on Windows 7 PC host

1. Open Network and Sharing Center in Control Panel, click the Local Area Connection <number> .

NOTE

The number might be different in your ENV.

2. On the Local Area Connection <number> status pop-up window, click Properties.

The Local Area Connection Properties window opens.

3. Double-click TCP/IPv4 Version (TCP/IPv4).

The Internet Protocol Version 4 (TCP/IPv4) Properties window appears.

4. Enter the IP address, Subnet mask and click OK.

On the RDB board Linux console, the following ping test begins:

```
root@ls1043a:/run/media/mmcblk0p1/ # ping 192.168.5.2  
PING 192.168.5.2 (192.168.5.2) 56(84) bytes of data.  
64 bytes from 192.168.5.2: icmp_seq=1 ttl=128 time=3.17 ms  
64 bytes from 192.168.5.2: icmp_seq=2 ttl=128 time=1.93 ms  
64 bytes from 192.168.5.2: icmp_seq=3 ttl=128 time=1.04 ms  
64 bytes from 192.168.5.2: icmp_seq=4 ttl=128 time=1.22 ms  
64 bytes from 192.168.5.2: icmp_seq=5 ttl=128 time=1.81 ms  
64 bytes from 192.168.5.2: icmp_seq=6 ttl=128 time=1.54 ms  
64 bytes from 192.168.5.2: icmp_seq=7 ttl=128 time=1.84 ms  
64 bytes from 192.168.5.2: icmp_seq=8 ttl=128 time=1.49 ms  
64 bytes from 192.168.5.2: icmp_seq=9 ttl=128 time=0.633 ms  
64 bytes from 192.168.5.2: icmp_seq=10 ttl=128 time=0.915 ms
```

Known Bugs, Limitations, or Technical Issues

- Linux only allows one peripheral at one time. Make sure that when one of DWC3 controllers is set as peripheral, then the others should not be set to the same mode.
- For USB host mode, some Pen drives such as Kingston / Transcend / SiliconPower / Samtec might have a compatibility issue.
- Some USB micro ports might have OTG3.0 cable compatibility issue. An OTG 2.0 cable and USB standard port works fine.

7.2.13.2 USB 2.0 Controller

(Freescale multi-port host and/or dual-role USB controller)

U-Boot

Host Mode

Basing on default configuration of LSDK, make sure to select the configs below:

Configure Tree View Options

Configure Tree View Options	Description
<pre> USB support ---> [*] EHCI HCD (usb 2.0) Support </pre>	Enables USB host controller support
<pre> [*] USB Mass Storage Support </pre>	Enables USB host controller support.

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/usb/host/ehci-hcd	USB HOST xHCI Controller stack
drivers/usb/host/ehci-fsl.c	FSL USB HOST xHCI Controller driver, basing on dwc3 driver
drivers/usb/host/usb-uhcd.c	USB host driver
common/usb.c	USB generic driver
common/usb_hub.c	USB hub driver
cmd/usb.c	USB command-line support

Verification

- Enumeration
 - Refer to USB 3.0 controller test steps.

Mass Storage

- Refer to USB 3.0 controller test steps.

Linux Kernel

Host Mode

Basing on default LSDK config, make sure to select CONFIGs below:

Configure Tree View Options

Configure Tree View Options	Description
<pre> USB support ---> [*] Support for Host-side USB </pre>	USB host controller support.
<pre> [*] EHCI HCD (USB 2.0) support </pre>	USB HCD support.

Table continues on the next page...

Table continued from the previous page...

Configure Tree View Options	Description
[*] USB Mass Storage support	USB mass storage support.
<*> Support for Freescale PPC on-chip EHCI USB controller [*] EHCI support for PPC USB controller on OF platform bus	USB EHCI support

Device Tree

```
usb@22000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl-usb2-<controller-type>-v<controller version>",
        "fsl-usb2-<controller-type>";
    reg = <0x22000 0x1000>;
    interrupt-parent = <&mpic>;
    interrupts = <28 0x2>;
    phy_type = "ulpi";
    dr_mode = "host"
};
```

NOTE

controller-type: dr(dual-role), mph(multi-port-host) controller-version: 1.6, 2.2, or earlier default mode is always host.

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/usb/core/*	USB subsystem/framework
drivers/usb/host/ehci-hcd.c	USB EHCI (host) driver
drivers/usb/host/ehci-fsl.c fsl-mph-dr-of.c	Freescale multi-port host and/or dual-role USB2.0 controller driver
drivers/usb/storage/scsiglue.c protocol.c transport.c usb.c	USB Mass Storage (device) driver

Verification

- Refer to USB 3.0 controller test steps.
- **Device mode (Gadget driver)**
- **Ethernet gadget**

Basing on default configuration of LSDK config, make sure to select config files below:

Configure Tree View Options

Configure Tree View Options	Description
<*> Freescale Highspeed USB DR Peripheral Controller	Freescale USB host controller support.
<M> USB Gadget functions configurable through configs	Configuration support.
<M> USB Gadget precomposed configurations	USG gadget support
<M> Ethernet Gadget (with CDC Ethernet support)	USB ethernet support
[*] RNDIS support	USB RNDIS gadget support

Device tree

```
usb@22000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl-usb2-<controller-type>-v<controller version>",
        "fsl-usb2-<controller-type>";
    reg = <0x22000 0x1000>; /* specifies register base addr, soc dependent */
    interrupt-parent = <&mpic>;
    interrupts = <28 0x2>; /* specifies usb interrupt line, soc dependent */
    phy_type = "ulpi"; /* phy can be ulpi(external)/utmi(internal) */
    dr_mode = "peripheral" /* this entry specifies usb mode */
};
```

NOTE

Controller-type: dr(dual-role), mph(multi-port-host) controller-version: 1.6, 2.2, or earlier default mode is always host. It can be either changed to peripheral inside the dts entry like above. In this case re-compilation of dts is required. DR mode can also be changed to peripheral via U-Boot command line. This will not require DTS recompilation, and can work with default DTS For USB1 controller.

```
=> setenv hwconfig 'usb1:dr_mode=peripheral,phy_type=<ulpi/utmi>
```

Source Files

Source File	Description
drivers/usb/host/fsl-mph-dr-of.c	Freescale dual-role USB2.0 controller driver
drivers/usb/gadget/function/u_ether.c	Ethernet-over-USB link layer utilities for Gadget stack
drivers/usb/gadget/function/f_ecm.c	USB CDC Ethernet (ECM) link function driver
drivers/usb/gadget/function/f_subset.c	"CDC Subset" Ethernet link function driver
Table continues on the next page...	

Table continued from the previous page...

Source File	Description
drivers/usb/gadget/function/f_rndis.c	RNDIS link function driver
drivers/usb/gadget/function/rndis.c	RNDIS MSG parser
drivers/usb/gadget/legacy/ether.c	Ethernet gadget driver, with CDC and non-CDC options

Verification

- Refer to USB 3.0 controller test steps.

7.2.14 Watchdog

Module Loading

Watchdog device driver support kernel built-in mode.

U-Boot Configuration

Runtime options

Env Variable	Env Description	Sub option	Option Description
bootargs	Kernel command line argument passed to kernel	setenv othbootargs wdt_period=35	Sets the watchdog timer period timeout

Kernel Configure Options

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description
<pre>Device Drivers ---> [*] Watchdog Timer Support ---> [*] Disable watchdog shutdown on close [*] IMX2+ Watchdog</pre>	IMX2 Watchdog Timer

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_IMX2_WDT	y/n	y	IMX2 Watchdog Timer

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/watchdog/imx2_wdt.c	IMX2 Watchdog Timer

User Space Application

The following applications will be used during functional or performance testing. Please refer to the SDK UM document for the detailed build procedure.

Command Name	Description	Package Name
watch	watchdog is a daemon for watchdog feeding	watchdog

Verification in Linux

- Set NFS rootfs. Build a rootfs image which includes watchdog daemon.
- Set boot parameter. On the U-Boot prompt, set following parameter:

Set nfsargs:

```
setenv bootargs wdt_period=35 root=/dev/nfs rw nfsroot=$serverip:$rootpath ip=$ipaddr:$serverip:
$gatewayip:$netmask:$hostname:$netdev:off
console=$consoledev,$baudrate $othbootargs
```

Set nfsboot

```
run nfsargs;tftp $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr - $fdtaddr
run nfsboot
```

NOTE

wdt_period is a watchdog timeout period. Set this parameter with the proper value depending on your board bus frequency.

wdt_period is inversely proportional to watchdog expiry time ie. the higher the wdt_period, the lower the watchdog expiry time. So if wdt_period is increased to high, watchdog will expiry early.

NOTE

When using watchdog as wake-up source with the default Ubuntu root filesystem, add watchdog-device = /dev/watchdog to /etc/watchdog.conf

Chapter 8

QorIQ networking technologies

8.1 Summary of networking technologies

NXP provides several different hardware networking architectures. Each SoC incorporates one of them. The hardware architectures are:

HW networking architectures	Blocks
DPAA1	QMan, BMan, and FMan
DPAA2	QBMan, WRIOP, and optionally AIOP
DPAA2 and DPAA1 are relatives in that they both use generic hardware-based queues. Also, each supports additional accelerators such as SEC via these queues.	
PFE	PFE package engine block
veTSEC	veTSEC traditional Ethernet controller block

The following table shows which SoCs supported by LSDK use which networking hardware architecture.

HW networking architectures	SoCs
DPAA1	LS1023A, LS1043A, LS1026A, LS1046A
DPAA2	LS1044A, LS1048A, LS1084A, LS1088A, LS2044A, LS2048A, LS2084A, LS2088A
PFE	LS1012A
veTSEC	LS1021A

8.2 DPAA2-specific Software

8.2.1 DPAA2 Software Overview

8.2.1.1 Introduction

The following section provides an overview of the software and tools for the DPAA2 networking hardware that is provided on NXP SoCs such as LS2088A and LS1088A. These SoCs are called "DPAA2 SoCs" because they contain the hardware that is required to support the DPAA2 networking architecture. This hardware includes Queue Manager/Buffer Manager (QBMan), the Wire Rate I/O Processor (WRIOP), and the Management Complex (MC).

DPAA2 is an architecture in which some facilities (and thus the hardware that supports them) are optional. For this reason, this document may describe features that are not available on all DPAA2 SoCs.

DPAA2 in the NXP Layerscape SDK

NXP provides a Linux-based software development kit (SDK) for SoCs. The core of the SDK is an embedded-oriented Linux distribution containing components such as:

- U-Boot boot loader
- Linux kernel with networking support
- GNU tool chain for ARMv8™
- Large set of standard Linux user space packages including shells, initialization scripts, servers, etc.
- Yocto-based package management in an embedded-style source-based Linux distribution

NXP supports and builds upon standard Linux with drivers and additional packages and capabilities including support for the DPAA2 networking hardware such as:

- Management complex firmware for the DPAA2 architecture. DPAA2 is a networking peripheral subsystem architecture and will be discussed at length in later sections.
- Restool: a DPAA2 object management tool
- A DPAA2 Linux Ethernet driver
- Linux kernel support for treating DPAA2 containers as plug-and-play buses with VFIO support
- Integrated kernel-based control of DPAA2 L2 switch objects
- Kernel support for DPAA2 acceleration objects including cryptographic offload
- And more

8.2.1.2 DPAA2 Hardware

8.2.1.2.1 Introduction

This section introduces the DPAA2 hardware components and explains their relationship to the DPAA hardware found on previous NXP SoCs. Finally, it shows the DPAA2 hardware blocks in the context of a specific SoC, LS2088A, LS1088A.

Note that the DPAA2 hardware is configured via DPAA2 objects as will be described below. This section on hardware provides background information to give context to the discussion of the DPAA2 objects. Most developers will deal with the DPAA2 objects and not directly with all aspects of the DPAA2 hardware blocks.

8.2.1.2.2 DPAA2 hardware

The DPAA2 hardware provides network interfaces, hardware-based queuing, layer 2 switching, more general switching, networking-related accelerators, and also memory dedicated to packet processing.

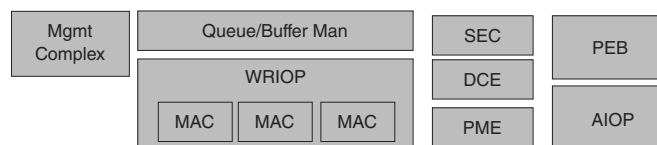


Figure 31. DPAA2 hardware components

The DPAA2 hardware contains the following components:

Management Complex (MC)

The DPAA2 hardware is abstracted by DPAA2 objects with the help of the Management Complex. This means that users need not study the details of the DPAA2 hardware blocks in order to develop drivers for or use DPAA2 capabilities. This software and solution oriented focus is one of the key differences between the first DPAA and DPAA2.

Queue and Buffer Manager (QBMan)

QBMan provides hardware-based buffer and queue management.

WRIOP

WRIOP provides hardware that serves as the basis for network interfaces. It includes Ethernet MACs, packet header key generators, parsers, table look up units, and an interface to the buffer and queue managers.

Accelerators (optional)

Accelerators that interface to QBMan are a key part of DPAA2. They include a cryptographic and security accelerator (SEC), a pattern matching accelerator (PME), a data compression/decompression accelerator (DCE), and a generic DMA engine. The set of accelerators may vary from SoC to SoC and new types of accelerators may be added.

PEB (optional)

PEB is a memory devoted to high-performance packet processing. It can be used to store in-flight packets and other items.

AIOP (optional)

AIOP is a fully programmable multicore engine with tightly coupled hardware accelerators that is specialized for efficient packet processing. It uses techniques somewhat similar to hardware multithreading to provide multiple "tasks" per core. The hardware supports efficient task switching to hide latencies associated with using accelerators and other hardware. The AIOP supports C language programming. It is optional in the DPAA2 architecture and thus is not available on all DPAA2 SoCs.

DPAA2 versus DPAA

DPAA2 is the latest generation of the Datapath Acceleration Architecture (DPAA) hardware. It is an evolution of the DPAA present in previous SoCs.

DPAA2 changes relative to DPAA include:

- DPAA2 contains a hardware block called the Management Complex. It facilitates and simplifies hardware resource allocation and hardware configuration.
- The hardware buffer and queue managers (QMan and BMan) are integrated into a single hardware block called QBMan.
- DPAA2 session context can be maintained per frame, rather than per frame queue, which allows multiple accelerator sessions to share a single frame queue pair. This single frame queue pair then reduces the number of frame queues needed, making session establishment more efficient because frame queues do not need to be initialized per session.
- Software portals are enhanced to make it easier and more efficient for General Purpose Processing (GPP) core software to share them.
- WRIOP in DPAA2 replaces FMan as the hardware block that provides Ethernet interfaces. WRIOP is designed to be more partitionable, in that it allows GPP software to more independently manage separate network interfaces.
- WRIOP and QBMan contain new features that support autonomous L2 switching functionality:
 - WRIOP: L2 address learning and forwarding unit.
 - QBMan: packet replication facility.
- WRIOP does not contain a generic programmable engine like the one present in FMan, and instead DPAA2 has a new hardware block called AIOP that is specifically designed to perform this function.

8.2.1.2.3 LS2088A block diagram

The LS2088A is an ARMv8-A 64-bit SoC. It contains eight ARM Cortex-A57 cores and numerous peripherals. The LS2088A is an example of a DPAA2 SoC because it contains the required DPAA2 hardware blocks: WRIOP, QBMan, and MC.

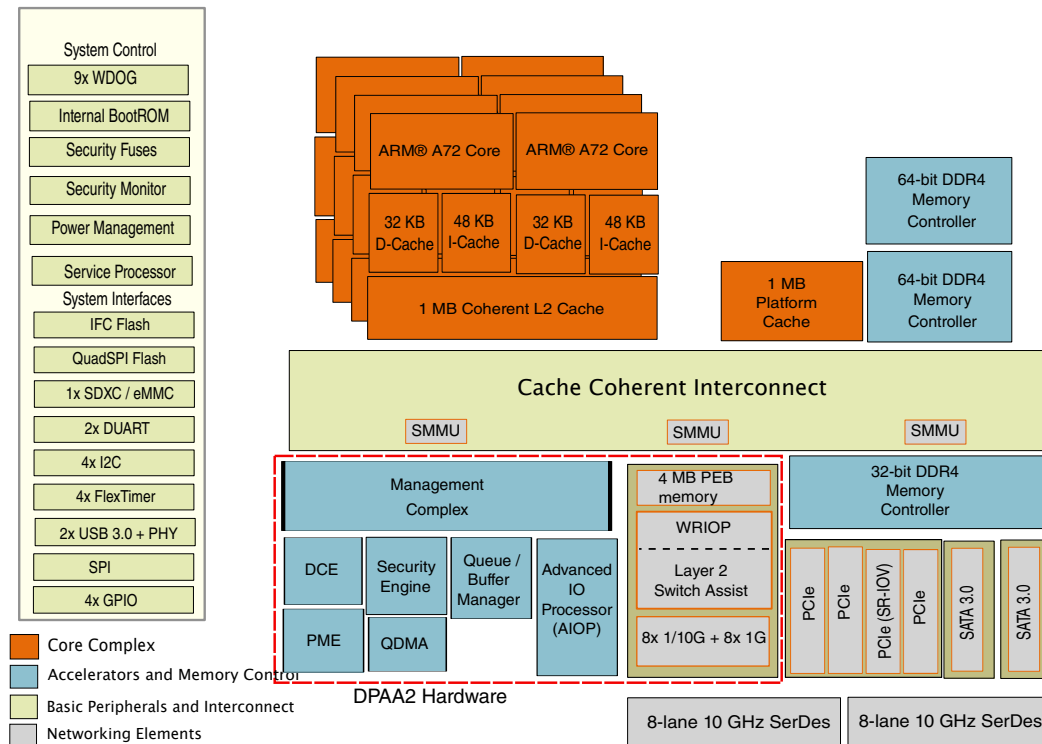


Figure 32. LS2088A SoC

The LS2088A contains standard-ARM components in addition to the cores, such as:

- ARM generic timer
- GIC-500 interrupt controller
- MMU-500 System Memory Management Unit (I/O MMU)

It also contains conventional hardware blocks including:

- DDR controllers
- Flash controller
- SDxC/eMMC controller
- USB controller
- PCIe controller
- SATA controller
- Other blocks visible in the diagram.

Finally, the following DPAA2 components are highlighted in the figure:

- QMan/BMan: hardware queue and buffer management
- WRIOP: Ethernet interfaces
- Management complex: DPAA2 objects and their management
- Accelerators: SEC, PME, and DCE

8.2.1.3 DPAA2 Linux Software

8.2.1.3.1 Introduction

This section provides a high-level summary of the most important DPAA2 software associated with the Linux operating system.

8.2.1.3.2 Linux and DPAA2

This section summarizes major Linux DPAA2 software. See [Linux DPAA2 software](#) which shows the software in relation to some standard Linux software.

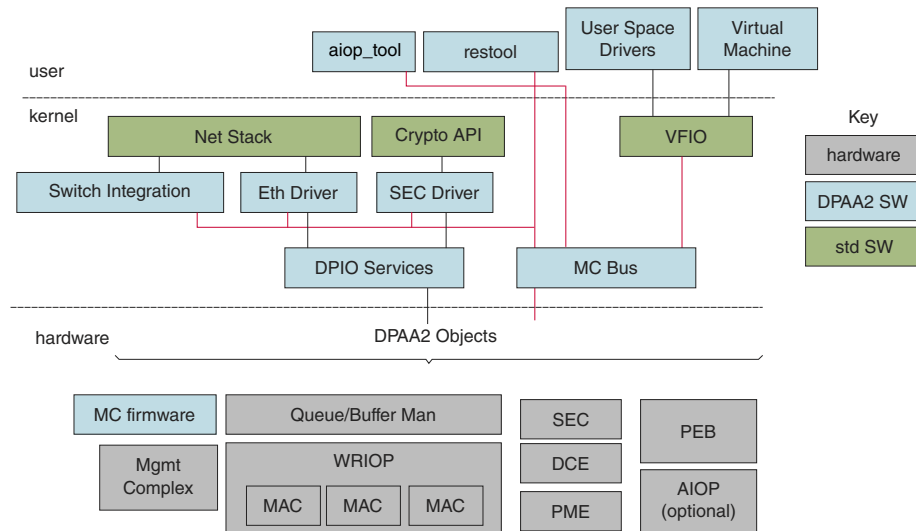


Figure 33. Linux DPAA2 software

Ethernet Driver

DPAA2 software includes a conventional Ethernet driver for use by the Linux network stack. This driver is controlled via standard Linux means such as the "ifconfig" or "ip" commands and also "ethtool". It operates in a manner that will be familiar to Linux users. Drivers in DPAA2 manage DPAA2 "objects" as will be described below. These objects are best regarded as hardware. They are formed from hardware resources.

DPIO Services

DPAA2 drivers such as the Ethernet driver in the Linux kernel use the DPIO services Linux component to do I/O. The DPIO services layer manages the kernel's DPIO objects. DPIO objects contain DPAA2 software portals (which are hardware components). The software portals can be shared by multiple higher-level drivers.

DPAA2 Objects and Management Complex (MC) Firmware

The DPAA2 hardware is presented to software in terms of DPAA2 objects that are realized by means of firmware running on the Management Complex. This will be explained in depth in [DPAA2 Networking Subsystem Deeper Dive](#) on page 310 and also immediately below.

MC Bus and Restool

The DPAA2 objects appear as devices on a special software-defined bus called the MC bus. Linux has a driver for this bus (and interactions with VFIO). This software is analogous to PCIe bus software. Like PCIe, the MC bus supports plug and play.

The "restool" utility is a Linux user space command that allows DPAA2 objects to be managed: created, destroyed, queried for status, etc.

SEC Driver

The SEC driver provides the standard Linux kernel cryptographic API but implemented by the SEC hardware by means of a special DPAA2 object. Other accelerators can be handled in the same way, but Linux tends to not provide standard (hardware-independent) kernel-level APIs for them so they are not discussed here.

Switch Integration

Finally, DPAA2 objects exist that perform L2 and more general network switching. These hardware elements can be configured using standard Linux mechanisms such as "bridge". As will be discussed later, there are two types of switch-related DPAA2 objects: DPSW and DPDMUX. There is kernel-based management support for both.

AIOP Tool

AIOP Tool (aiop_tool) is a user space command line utility that allows programs (images) to be loaded onto the AIOP and started. It also supports stopping and resetting the AIOP. Of course, aiop_tool is used only on DPAA2 SoCs that have an AIOP.

It is also possible for user space programs to manage these aspects of AIOP via programmatic means. The aiop_tool utility is most useful during development of AIOP software and for AIOP applications that happen not to be tightly integrated with control software running in user space on the GPP cores.

From the point of view of software running on the GPP cores, AIOP programs may be thought of as firmware that defines the functionality that the AIOP will provide to an overall system.

8.2.1.3.3 DPAA2, Management Complex, and drivers

DPAA2 is the architecture that describes network interfaces and other networking services for an SoC with DPAA2 hardware. It is discussed in depth in [DPAA2 Networking Subsystem Deeper Dive](#) on page 310. For now, think of DPAA2 as hardware for networking that is presented in terms of DPAA2 objects. The objects provide specific high-level features or services such as network interfaces or L2 switches.

The objects are managed by means of firmware running on a hardware block called the Management Complex. Software on general purpose cores must load firmware onto the Management Complex before networking can be done using DPAA2 hardware.

Normally, the MC firmware is loaded early in the boot process so that boot loaders can make use of DPAA2 objects and perform networking operations such as network-based booting.

Since the objects represent hardware, they require driver software on general purpose cores. NXP provides drivers for U-Boot and standard Linux and thus both support Ethernet networking out of the box. For example, one can use Linux networking without delving into the details of DPAA2 and its objects just as one can use Linux networking via a PCIe Ethernet card (whose manufacturer provides a driver) without delving into the design of the card.

DPAA2 and its objects are fully documented so it is possible to write drivers for other operating systems, applications, or boot loaders, e.g. DPDK, UEFI firmware, etc. Many of these drivers exist or are roadmap items.

8.2.1.3.4 DPAA2 and plug-and-play

There is another analogy between DPAA2 objects and PCIe devices. PCIe devices appear to operating systems as plug-and-play devices on a bus. The operating system can scan the bus to discover and identify the devices on it. It can then use the device identities to associate drivers with devices and bring them into service.

DPAA2 objects work in a similar way. They are placed into datapath containers (DPRC) that can be scanned in an analogous manner. Then objects are associated with drivers and placed into service.

The Linux kernel is provided with a container with its DPAA2 objects. Containers can also be provided to other software including virtual machines and even arbitrary user space processes. This is how the hardware that objects encapsulate can be directly assigned to virtual machines and user space processes. This allows them highly efficient access to hardware but in a secure fashion due to the involvement of the SoC IO-MMU.

This, also, is analogous to PCIe devices in standard Linux; DPAA2 objects can be directly assigned to virtual machines and user space processes using a standard Linux architecture called VFIO which allows devices to be mapped into the address space of user space processes and also enables IO-MMU configuration to constrain the memory to which devices can read and write data via their DMA engines.

Like PCIe devices, DPAA2 objects are also mapped using VFIO. NXP supplies the extensions to VFIO in Linux that makes this possible.

8.2.1.3.5 Datapath layout files and restool

As mentioned elsewhere, DPAA2 containers are like PCIe busses in that they can be scanned for objects/devices. But containers and PCIe busses are populated very differently. PCIe busses are populated physically, e.g. by plugging a card into a slot.

Objects are encapsulations of DPAA2 hardware resources that must be created via management complex firmware and then assigned to a container. There are several ways to do this:

1. the datapath layout file
2. restool
3. Management Complex commands

Datapath layout (DPL) file

Containers and objects can be defined statically in a file called a datapath layout file (DPL) that is passed to the management complex when it is initially booted. The DPL can specify containers, objects, and connections between objects. When an OS such as Linux boots, it will discover the populated containers.

restool

The utility called “restool” is a NXP-created Linux user space command that allows inspection and dynamic management of containers and objects. With it, one can

- Display the current set of containers and objects
- Create and destroy containers
- Create and destroy objects
- Assign objects to containers
- Create links among objects

One can use a sequence of restool command invocations to create the same container and object state that a DPL might specify. The difference is that restool is dynamic.

Management Complex commands

Finally, objects and containers can be manipulated by software running on general purpose cores by sending commands to the Management Complex. This is, in fact, what restool does. Command line arguments to restool define an operation. The restool utility simply forms a command and passes it to the Management Complex. Other drivers can also do this.

8.2.1.4 DPAA2 Networking Subsystem Deeper Dive

This section provides additional detail on the DPAA2 architecture and the DPAA2 object services paradigm.

This paradigm simplifies using the DPAA2 hardware IP blocks through abstraction and encapsulation. DPAA2 objects are objects in the sense that they:

- Encapsulate specific abstract functionality, e.g. L2 switching.

- Are composed of allocated hardware sub-components of the DPAA2 hardware peripherals, and then mostly abstract their functionality .
- Present functionality in terms of specific attributes and methods, meaning operations on the objects.

NOTE

DPAA2 objects are not associated with object-oriented programming languages, instead they are collections of hardware resources allocated for a specific purpose. General purpose processing (GPP) core software can configure objects by sending them commands expressed in terms of hardware-level descriptors. GPP software can also include C language functions that prepare and interpret the descriptors. No use of object-oriented programming languages is required. For the most part, Linux drivers are written in C as usual

This section:

- Presents the DPAA2 object model at a concept level and describes how objects are created, destroyed, conveyed, configured, and used
- Lists the objects types and their purposes
- Outlines how the Management Complex implements and provides the objects
- Explains what software components use the various DPAA2 object types, and how they use them. The users are often application software running on general purpose processors (cores) or on the optional AIOP.

Driver-level software on GPPs works with the abstracted objects, rather than directly with the hardware. For example, the GPP software deals with L2 switch and network interface objects rather than WRIOPs .

DPAA2 objects express and abstract the DPAA2 hardware into software-managed objects that are:

- Application-oriented in terminology and use, rather than hardware-oriented
- Based on concepts that are generally familiar to programmers and system architects
- Simpler than direct management of the hardware
- Indicate the architectural intent of the hardware blocks

DPAA2 object services are provided by software that runs as firmware on a DPAA2 hardware block called the Management Complex. Users do not need to program the Management Complex in order to use the Network Object Services; they simply use the NXP-supplied firmware. This firmware runs on the Management Complex instead of a general purpose core in order to simplify the integration of the NXP software with customer software. [DPAA2 object concept](#) below shows at a concept level how the Management Complex provides objects that perform specific services; the objects have attributes and interfaces that appear as hardware.

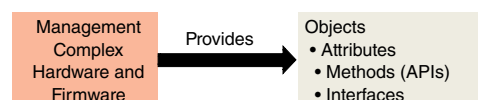


Figure 34. DPAA2 object concept

8.2.1.4.1 DPAA2 hardware abstraction example

This section introduces the DPAA2 objects and the abstractions they provide by means of an example. [Example scenario](#) shows a system constructed using the DPAA2 hardware on a DPAA2 SoC such as the LS2088A. The goal is to run two KVM virtual machines (VMs) on the SoC. The two virtual machines each have a hardware network interface that they can directly access (i.e. a dedicated interface) connected to a DPAA2 L2 switch. These VMs can communicate with each other via the L2 switch, and they can communicate externally via the MAC on the L2 switch. So, the L2 switch has three ports, one for an off-SoC connection (connected to a MAC), and two for the VMs.

In addition, there are two network interfaces with MAC addresses for off-SoC communication that are used by the host Linux. The host Linux instance and the virtual machines all run on the Cortex-A72 cores on the LS2088A. In this example, each network interface is associated with an Ethernet driver working with Linux.

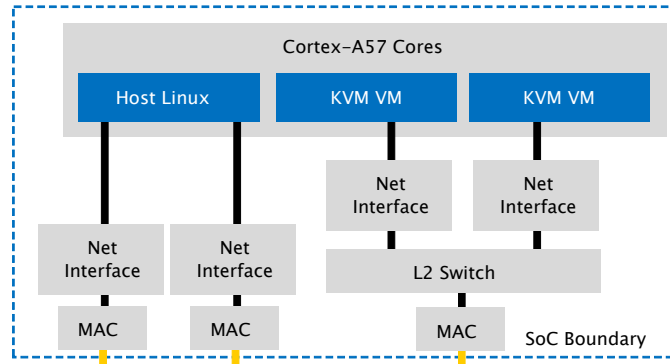


Figure 35. Example scenario

[DPAA2 hardware](#) shows the DPAA2 hardware blocks. This figure bears little resemblance to [Example scenario](#). It provides little guidance to how the example scenario could be realized because the hardware blocks are conceptually distant from a natural statement of what is desired in the example. The DPAA2 objects are much closer, as will be seen below.

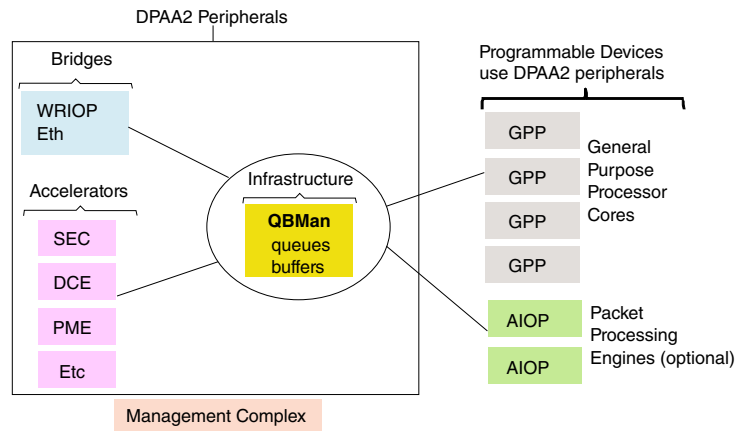


Figure 36. DPAA2 hardware

[Example scenario based on DPAA2 objects](#) shows how the example can be realized using the DPAA2 object abstractions of the DPAA2 hardware; this figure is much closer to the goal expressed in [Example scenario](#) and its components are described below:

- The host Linux is shown in more detail on the left. The network stack and two instances of the Ethernet drivers appear in the figure above the hardware boundary. Also, the figure shows the stacks and drivers for the two virtual machines.
- The DPAA2 objects appear below the hardware boundary
- The DPNI (Datapath Network Interface) objects correspond directly to the network interfaces in [Example scenario](#). The DPSW (Datapath Switch) object corresponds to the L2 switch.
- The DPMAC (Datapath MAC) objects represent Ethernet MACs within WRIOP. These are hardware components that connect to PHY hardware, and provide Ethernet physical layer termination, i.e. Ethernet connections to the SoC.
- The DPIO (Datapath I/O) objects include QBMan software portals, and they allow GPP core software to read and write packets from the DPNI. DPIOs are described in more detail later in this document.

See [Object summary](#) on page 315 for a summary of the DPAA2 objects.

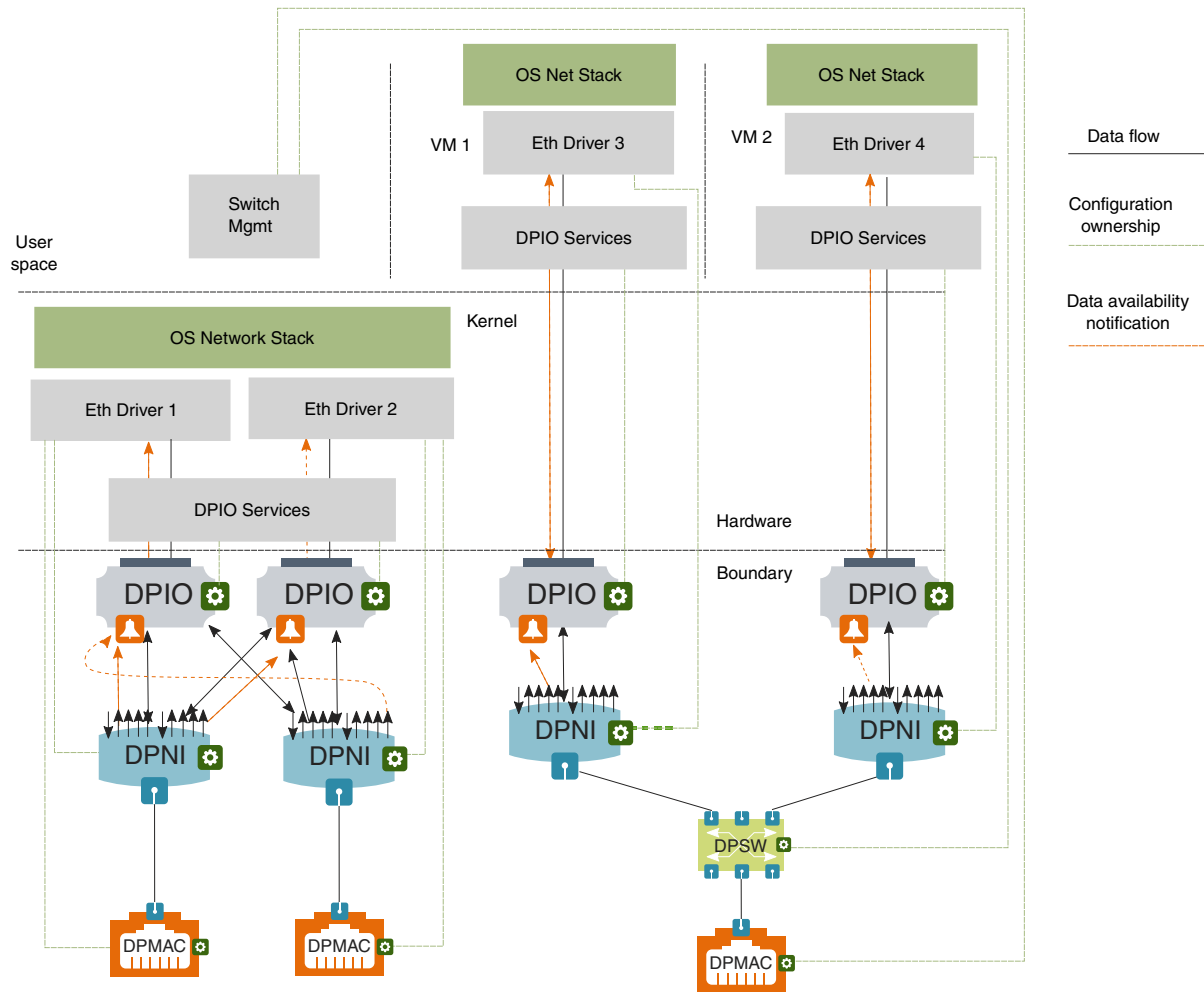


Figure 37. Example scenario based on DPAA2 objects

Objects are partitioned among software owners

Software management of DPAA2 objects is distributed. Software components that use a particular set of objects independently manage the objects in their set. The green boxes on the object icons in [Example scenario based on DPAA2 objects](#) represent management interfaces, and the green dashed lines show what software component owns the management of each object. For example, the DPSW is shown as managed by switch management software running on the general purpose processing cores.

Objects can be directly assigned

The virtual machines directly access and manage the objects their software uses, and they do this with minimal host kernel involvement; this enhances efficiency while preserving access isolation. In the figure, the virtual machines have directly assigned hardware-based network interfaces.

DPNI objects provide network interfaces

DPNI objects interact with drivers to allow software to send and receive network frames, usually Ethernet frames. DPNI are central to DPAA2's concept of network interfaces, but they do not act alone. In general, network drivers manage several

objects as part of managing network interfaces. [DPNI ingress](#) shows a high-level outline of DPNI ingress frame processing, and the following steps give insight into how objects work together.

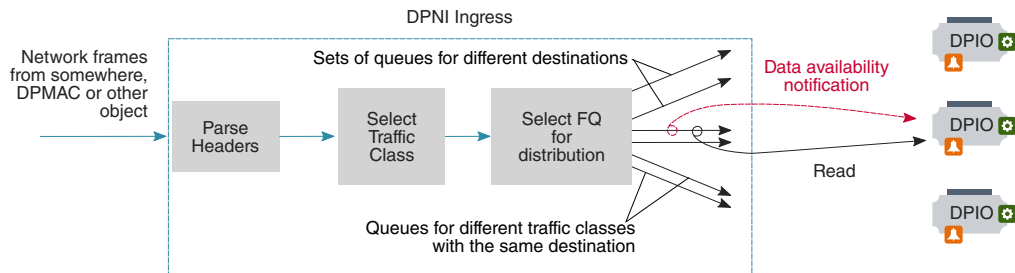


Figure 38. DPNI ingress

1. A frame arrives at DPNI from another object, a MAC (DPMAC), a switch (DPSW) or other object.
2. DPNI parses the packet to locate the header from which lookup keys can be generated.
3. A lookup selects a traffic class (priority) for the frame; this priority causes a specific set of queues (implemented as QMan frame queues) to be selected.
4. DPNI must select a destination for the frame, using either another lookup or an RSS-style hashing operation; this lookup causes a specific queue within the previously selected set to be selected.
5. The frame is enqueued onto the queue, and the queue represents the destination indirectly. At this point, DPIO objects enter the process.
6. Every queue is configured to deliver data availability notifications to a specific DPIO, and these notifications tell the driver software using the DPIO that one or more frames are available to read from a specific queue.
7. Driver software responds by using a DPIO (actually any of its DPIOs) to read a burst of one or more frames from the queue.

Egress is simpler. The driver software uses a DPIO to enqueue a frame to a specific egress queue within DPNI; the queue is selected based on the desired traffic class.

Multiple DPIOs provide parallelism

It is common to assign queues in network interfaces to specific cores, and then to distribute the traffic between them using techniques like RSS or explicit flow steering. DPAA2 supports this process by using multiple DPIOs. See [DPIO parallelism](#) for an example involving a single network interface and two cores.

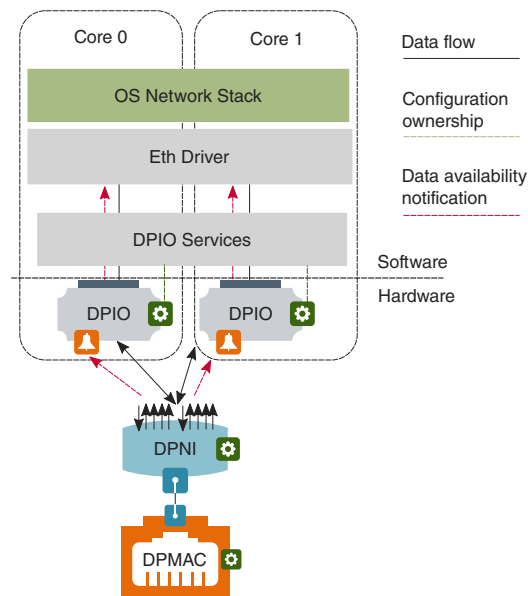


Figure 39. DPIO parallelism

The DPNI is configured so that each of its egress queues send its data availability notifications to one DPIO or another in a balanced way. A core receives an interrupt from its DPIO telling it to read a data availability notification, and then it then uses its DPIO to read a burst of one or more frames. In Linux terms, it starts a NAPI burst.

DPIO services

Notice in [Example scenario based on DPAA2 objects](#) that the host operating system on the left has two network interfaces. It has two DPIOs also, but either DPIO can be used for I/O to either of the interfaces. DPIOs are designed to be shared across network interfaces that belong to the same software component, such as the Linux kernel. For this reason, the Linux kernel contains a software layer called DPIO Services that facilitates driver instances performing I/O from a resource that might be shared across a network interface, and also might be shared across cores or software threads. Giving more DPIOs to the DPIO Services layer can increase performance, and using the same DPIO on a core for more than one network interface need not decrease performance because each core is physically able to do only one thing at a time.

8.2.1.4.1.1 Object summary

This section summarizes the DPAA2 objects and shows a standard icon for each used in the illustrations that follow. See [DPAA2 object summary and icons](#).

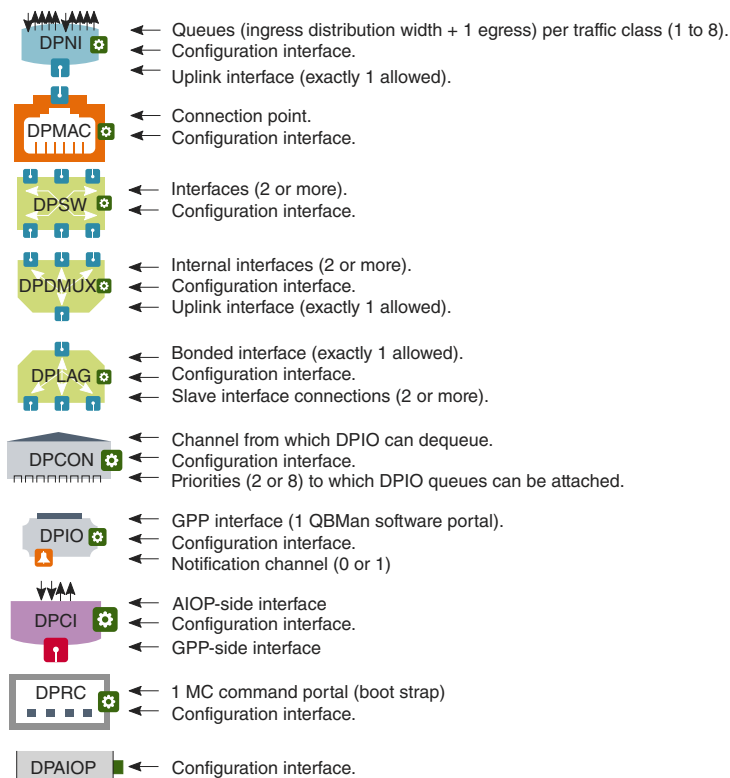


Figure 40. DPAA2 object summary and icons

DPNI

A DPNI object is the key to network interfaces. On ingress, it receives frames from a DPMAC or another object such as a DPSW, parses headers, determines the frame's traffic class, and enqueues the frame onto a frame queue selected based on the traffic class and other header values. This supports both hash-based distribution of frames to multiple cores, and also direct flow steering of frames to specific cores.

DPNI can generate a per-queue data availability notification when a frame is enqueued. On egress, the DPNI dequeues frames from frame queues and transmits them to an external port using a DPMAC, or to another DPAA2 object such as a DPSW.

DPMAC

The DPMAC object represents an Ethernet MAC, a hardware device that connects to a PHY and allows physical transmission and reception of Ethernet frames.

DPSW

The DPSW object provides the functionality of a general layer 2 switch. It receives packets on one port and sends them on another. It can also send packets out on multiple ports for the purposes of broadcast, multi-cast, or mirroring.

DPDMUX

The DPDMUX is another type of switch. It differs from a DPSW in several ways. A DPDMUX may have only a single uplink port. Also, it can be programmed to direct packets based on header values above layer 2.

DPLAG

The DPLAG object provides link aggregation. It combines two or more uplinks into a single downlink.

DPCON

The DPCON object allows multiple DPNIs to be aggregated into a single device that appears to a GPP core or AIOP software as single interface that carries frames from multiple DPNIs; it combines two or more network interfaces into one. It provides a hardware-based scheduling off load because the hardware selects the order based on the priority in which frames from the multiple DPNIs are provided to software on GPP cores or AIOP.

DPCON is also useful for software that polls for input frames; it allows a single interface to be polled instead of multiple interfaces.

DPCON objects are also used by Linux Ethernet drivers for priority-based frame delivery.

DPIO

General purpose processing core software uses a DPIO object to perform hardware queuing operations, such as enqueue and dequeue, and hardware buffer management operations, such as acquire and release. It also allows data availability notifications to be received. DPIOs can generate interrupts. The DPIO object is unusual in that GPP core software is expected to directly access portions of the DPIO's hardware (QBMan software portals) for run time operations, in addition to supporting configuration operations from the management complex.

Note that AIOP software does not rely on DPIO objects; they are used only by software on the general purpose processing cores.

DPBP

The Datapath Buffer Pool object represents a QBMan buffer pool. It is used mainly as a resource by network drivers, but it is an active entity because it can send buffer pool depletion notifications to GPP core software.

DPCI

The Datapath Communication Interface provides general purpose processing core software with a transport mechanism typically for control and configuration command interfaces to AIOP applications. The AIOP service layer implements the AIOP-side of the transport, but the commands are application-specific. Note that AIOP is optional in DPAA2 and is not present on some SoCs.

DPRC

The DPRC object allows the Management Complex to track sets of objects in use by the same software component. The objects in the set are said to be in the same container. It also facilitates the assignment of sets of objects to specific software components, such as a virtual machine or a user space application using user space drivers. The software component can query containers in order to discover objects at run time, and this enables plug-and-play drivers that interface to objects.

Some objects include DMA-capable hardware. All objects in the same DPRC share a common ICID, and a common set of IO-MMU mappings. A number of key features of DPRCs include:

- **Direct access.** All the objects and resources in a container are private to the container, and software components get direct access to the registers (as abstracted by the Management Complex) of the hardware objects.
- **Dynamic discovery.** A software context that is given a DPRC can dynamically discover the objects and resources placed in the container using MC commands.
- **Hot plug/unplug.** Objects can be dynamically plugged and unplugged into DPRCs.
- **Security.** A software context can only see the objects in its DPRC, and cannot affect other containers or the proper operation of other software contexts. DMA transactions from MC objects are isolated using the system IOM-MU.

DPMCP

The DPMCP object represents a Management Complex command portal and is used by drivers to send commands to manage objects.

DPAIOP

DPAIOP is a configuration object that aids in loading programs onto the AIOP, running the AIOP, resetting the AIOP, and receiving status, error, and log information from AIOP programs. Note that the AIOP is optional in DPAA2 and is not present on some SoCs.

Objects for accelerators

There are also objects associated with accelerators such as SEC, PME, and DCE. These objects provide software with interfaces to the accelerator hardware. For this reason, the accelerator interface objects end in "I".

- DPSECI - SEC (security/cryptographic coprocessor) interface.
- DPDCEI - DCE (data compression engine) interface.
- DPDMAI - DMA engine interface.

Software uses queues associated with an object to send a buffer to an accelerator for processing and to receive the result.

New types of objects

NXP will create new types of objects over time to address new needs and use cases as they arise.

8.2.1.4.2 Management Complex: How DPAA2 objects are created and managed

This section outlines how the Management Complex creates and manages DPAA2 objects.

The best way to think of DPAA2 hardware, in particular WRIOP and QBMan, is that it provides many low-level resources ranging from Ethernet MACs to look up tables to frame queues and so on. Software's mission is to assemble the right set of these low-level resources, and configure them collectively to achieve a goal.

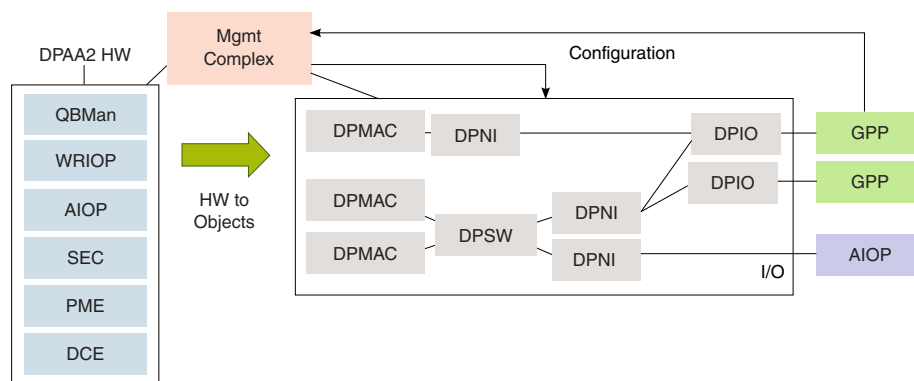


Figure 41. Management Complex creates objects from hardware sub-components

Think of the low-level resources as “atom resources” because they are always allocated as a unit. DPAA2 objects are then “composite resources,” or collections of atom resources that are then configured to achieve a common goal, like being an L2 switch as shown in [Realizing an L2 switch](#).

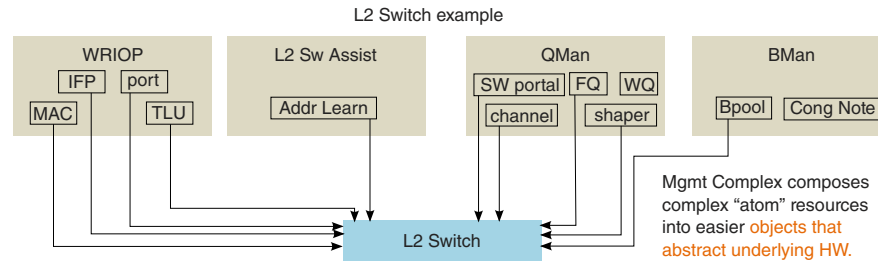


Figure 42. Realizing an L2 switch

The creation method for a DPAA2 object involves allocating the necessary atom resources and configuring them enough to place the object in an initial idle state. Object methods and other interfaces then allow it to be further configured and used. For example, forming an L2 switch from DPAA2 atom resources is quite complex. The NXP firmware running on the Management Complex implements the methods necessary, and hides this complexity from GPP (and AIOP) developers.

Continuing the example, an L2 switch object can also be shutdown and disassembled by its methods. Its atom-resources are then placed back into the pools of atom resources that the Management Complex firmware manages.

Hardware directly visible to software

Clearly, DPAA2 provides abstractions. The objects are best thought of as being hardware, and most actually are collections or encapsulations of hardware resources that are allocated and configured to achieve a higher-level and more abstract purpose than would be clear from a direct view of the hardware resources. An example of an abstract purpose is “be an L2 switch” (DPSW).

It can be helpful to focus on exactly what is visible to driver-level software running on the general purpose cores and AIOP, especially since what is visible is a mixture of direct access to hardware and indirect access to hardware via abstractions. This discussion will be biased towards the view of objects from drivers running on general purpose processing cores (such as in U-Boot and Linux).

Also the discussion will avoid details of individual objects since this is an overview with the purpose of clarifying objects in general.

[DPAA2 visibility boundary](#) describes in one diagram what is directly visible to the driver layer software.

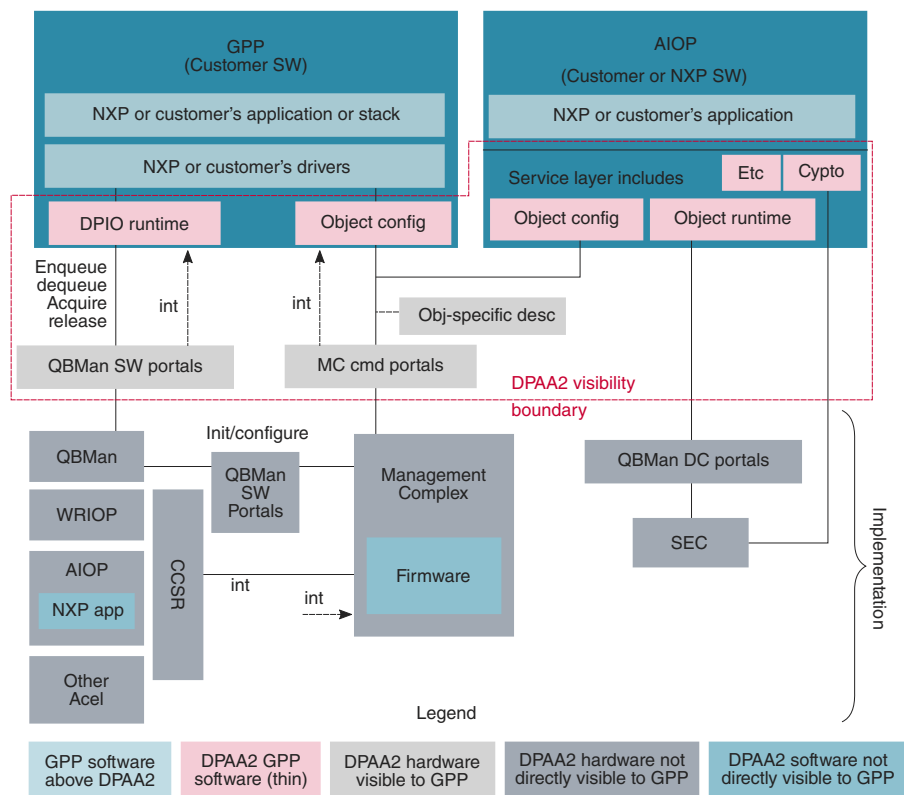


Figure 43. DPAA2 visibility boundary (AIOP not present on all DPAA2 SoCs)

There is quite a lot in the figure above, so it is best to break it down. What driver level software can see and do is dictated by its function.

This begins with the Management Complex (MC) itself. The discussion below will focus on the services that the MC provides to other software in the system. There will be no discussion of MC firmware's internal design.

See [Management complex visibility in DPAA2](#). The first step is that general purpose processing core software (usually a boot loader) must load the opaque firmware image onto the Management Complex and then start it running. This involves direct access to portions of the Management Complex hardware: registers defining the location of the Management Complex's portion of DDR, image location, address translation, and run state control.

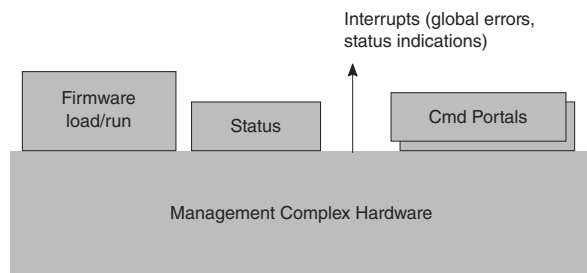


Figure 44. Management Complex visibility in DPAA2

The driver software also requires visibility to global status, particularly to status for global errors. Changes in the state of this status can be signaled by interrupts to the general purpose processing cores so the Management Complex can produce these interrupts.

Finally, the Management Complex exists to serve its masters, the general purpose processor core (and AIOP when present) software that “owns” objects, i.e. has been allocated access rights to them via container ownership and hierarchy. The service is provided by responding to commands so driver software needs a way to deliver commands to the Management Complex. In addition, this process must be secure in that the Management Complex must know, in a way that cannot be spoofed, an ID of the software sending the command. This is to allow the Management Complex to enforce object access rights.

Driver software delivers commands to the Management Complex via hardware called Management Complex command portals. SoC hardware provides significant numbers (at least 10s) of these portals because:

- They can be directly assigned to multiple different drivers, all of which independently use the Management Complex’s services. If they each have their own command portal, they do not have to coordinate with each other.
- Each independent driver instance has its own ID (ICID) that is securely associated with the command portal to prevent spoofing. This prevents a driver from being able to access for configuration an object that it does not “own”.

To send a command to the Management Complex, driver software creates a descriptor and enqueues a pointer to it to the command portal.

Next consider objects. See [DPAA2 objects](#).

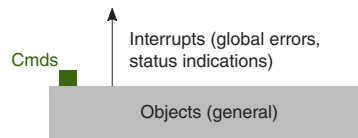


Figure 45. DPAA2 objects

Objects are created either via the DPL file or driver software sending a command to the Management Complex instructing it to create an object (as in restool). The Management Complex supplies a globally unique ID for the new object.

Object command interfaces are abstractions. There is no hardware that directly represents object command portals. Objects are usually hardware, but in most cases that hardware does not directly expose a hardware-level programming model to driver software. Instead, driver software configures objects via an indirect mechanism; it sends a command to the Management Complex. The command is a descriptor that includes the ID of the object as well as the definition of the operation to be performed.

The Management Complex automatically gets the ID of the requestor when it reads the command. The command portal securely adds it. The Management Complex then checks that the requestor is authorized to configure the object and, if so, performs the configuration on behalf of the requestor.

So, object configuration is a visible part of DPAA2, but the configuration of the individual hardware subcomponents that make up an object is not.

The fundamental programming model for object configuration is the commands that can be sent to the Management Complex to configure the object. Each object type has a different purpose so each object type’s configuration programming model is defined by the descriptor set that describes the commands to configure the particular type of object.

NXP also provides C callable APIs that basically allocate and populate descriptors and pass them as commands to the Management Complex. The APIs bear a close relationship to the more fundamental descriptors.

Many object types have nothing but a configuration space, but this is not always true. Some objects also provide I/O interfaces. The DPIO object is a prime example. See [DPIO object and I/O interfaces](#).

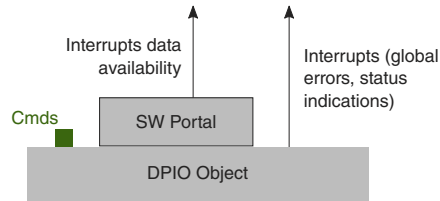


Figure 46. DPPIO object and I/O interfaces

As has been stated before, DPAA2 objects are usually opaque bundles of hardware sub-resources allocated and configured to achieve a more abstract purpose. A DPPIO object includes a hardware sub-resource called a QBMan software portal but this hardware is not opaque to the driver software running on the general purpose processing cores. The reason is performance. Software portals are the hardware mechanism for actually doing I/O with DPAA2 peripherals so driver software must directly access them. There are also data availability interrupts associated with DPPIOs. These indicate availability of data to read using the software portals.

NOTE

Software portals actually support more than I/O (enqueue onto queues and dequeue from them). They also support commands. The simplest examples are buffer acquires and releases. Without going into full detail, software portals actually support commands that require privilege (example: initialize a frame queue) and commands that do not (example: acquire a buffer). Driver software on general purpose processing cores (and AIOP) uses only the unprivileged commands. The privileged commands are not part of the visible architecture. They are used only by Management Complex firmware.

In summary, the visible architecture includes both hardware and abstractions as follows:

- Management Complex hardware associated with loading and running images
- Management Complex hardware associated with accessing global status
- Management Complex global interrupt
- Management Complex hardware command portals
- Objects themselves (abstraction):
 - Object configuration interface and command set as defined by descriptors (abstraction)
 - Object error interrupts
 - Some objects (like DPPIO) also have additional interfaces that are hardware directly accessed by driver software. DPPIO's QBMan software portals are an example. They can produce interrupts.

8.2.1.4.2.1 Object creation, the datapath layout file, and restool

DPAA2 objects can be created in multiple ways. First, they can be specified in a Datapath Layout (DPL) file that the Management Complex reads and applies before Linux boots. This file contains the specific list of objects that are to be automatically created as the system initializes.

DPAA2 objects also can be created and destroyed dynamically by sending commands to the Management Complex through its command portals via a kernel driver. For Linux, a user space command line tool called “restool” uses this interface to allow interactive and dynamic creation of objects. It also allows destruction and some additional configurations to be done.

Restool also shows information about objects and what they are connected to.

8.2.1.4.2.2 DPRC objects, plug and play, and the fsl-mc Linux “bus”

As mentioned previously, it is common for a GPP software component to manage multiple objects. The [DPPIO parallelism](#) diagram shows a simple example of the Linux kernel managing a set of objects to provide a pair of network interfaces. The DPRC (Datapath Resource Container) is a special object that serves to organize other objects, and also the hardware sub-

components from which objects can be dynamically created; the hardware sub-components include frame queues, channels, buffer pools, etc. Containers can be created and filled with objects and resources and then passed to the software component, such as a virtual machine, that will use them.

The software that was assigned a DPRC can enumerate the objects inside it; this is a form of dynamic hardware discovery that relates to plug-and-play. For example, an operating system can scan a DPRC and associate all DPNI objects found within with an Ethernet driver that will use them to form network interfaces. The Ethernet driver then uses a dynamic allocator within the kernel to acquire other objects such as DPBPs that it needs to operate.

The device discovery analogy is strong enough that Linux exposes DPRCs assigned to it as a bus in sysfs-- much like physical buses like PCI. The same sysfs mechanism that allow a physical PCI device to be assigned (bound) to virtual machines are also used to assign containers to virtual machines. Objects can even be dynamically added and removed from DPRCs. This is analogous to hot plug and unplug on a bus.

Many DPAA2 objects are DMA-capable so that they can autonomously read and write memory. SoCs like the LS2088A contain an IO-MMU, so objects must express an identifier (that they cannot control) when they perform DMA operations. This identifier is called an ICID in DPAA2, and it serves as a key for the IO-MMU to associate I/O virtual addresses with I/O physical addresses. In DPAA2, ICIDs are attributes of DPRCs, and all objects in a DPRC express the same ICID value.

A GPP software context (a virtual machine or application) will typically be assigned a single DPRC that contains all the fsl-mc resources that the software context can access or use. As mentioned elsewhere, there are two general types of resources that can be in a container:

- **Resources:** Resources are primitive resources that can't be further decomposed, and are uninitialized and unpurposed. Some examples are MC portals, QBman portals, frame queues, buffer pools, etc. Generally primitives are “fungible,” in that there is nothing distinctive among the same kind of primitives. However, some primitives may be non-fungible, such as an external port or MAC.
- **Objects:** Objects are created and configured with a purpose, typically constructed of multiple resources. Some examples of objects are network interfaces, an L2 switch, or a crypto instance. A DPRC is itself an fsl-mc object.

NOTE

See documentation of the Linux restool facility for more information related to this topic.

Management Complex (MC) initialization and boot

The MC is normally enabled and initialized by system boot firmware such as U-Boot. The boot firmware is responsible for reserving a region of memory (DDR) for the fsl-mc, and then loading the MC firmware into memory, loading a datapath layout file (see below for DPL overview info), and writing a bit to enable/start the MC. See [Management Complex initialization and boot](#).

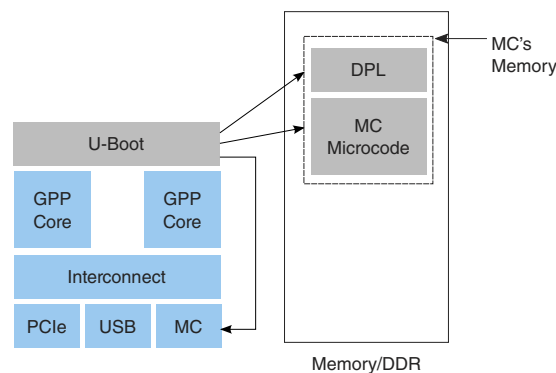


Figure 47. Management Complex initialization and boot

Management Complex datapath layout file (DPL)

As mentioned above, a datapath layout file (DPL) must be supplied to the Management Complex when it is booted. The DPL contains the definitions of initial objects and containers/DPRCs to create.

A DPL is defined in a text file in device tree syntax (DTS) format and then compiled into a standardized DTB binary format (used by ePAPR compliant device trees).

See the *DPAA2 User Manual* for more information and examples on the datapath layout file.

Boot loader use of the MC

In typical usage, the boot loader loads the MC firmware image and starts the MC running. At this time, it supplies a data path control (DPC) file that supplies the MC image with basic configuration information that allows it to operate.

The boot loader can now use the services of the MC in order to access network devices. It is a good approach to have the boot loader dynamically create the objects it needs and destroy them (releasing resources) before starting the operating system. This way, the operating system is not forced to operate with the constraints of objects and DPRCs established by the boot loader. The OS can see a "green field".

Optionally, the boot loader can apply the data path layout (DPL) file mentioned above just before starting this OS. This approach allows the DPL to be written only to serve the operating system's needs and not the boot loader's, which tend to be much simpler.

DPRCs are hierarchical

The MC manages DPRCs in a hierarchical relationship. There is a single root DPRC at the root of the hierarchy. That DPRC can have child DPRCs, children can have grandchildren, and so on. The root DPRC belongs to the root software context of the system, usually an OS or hypervisor and it should never be unbound from the corresponding driver. The root DPRC can further allocate its resources to its child DPRCs and assign them to other entities such as user space applications or virtual machines.

In this example there are 3 DPRCs/containers managed by the Management Complex: a root container "root" with 2 children "foo" and "bar". The DPRCs all contain 3 objects, a DPNI, DPBP, and DPIO. There are 3 software contexts: the host Linux, a user space application, and Linux in a KVM virtual machine. Each software context is assigned a DPRC that it can use and manage; see [DPRC hierarchy](#) for a figure that illustrates this example.

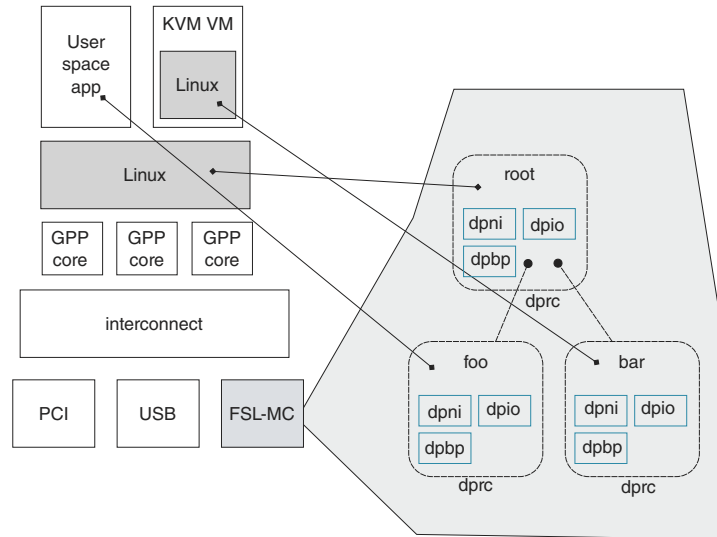


Figure 48. DPRC hierarchy

The container hierarchy allows the parent to manage the resources of the children. If the OS in the KVM VM crashes, the parent (Linux) can reset and clean up the VM's DPRC. If the user space application terminates, the parent (Linux) has the option of destroying the container.

8.2.1.4.3 Objects and topology

As mentioned elsewhere, objects have a topological relationship with each other. See [Object topology example](#) for an example.

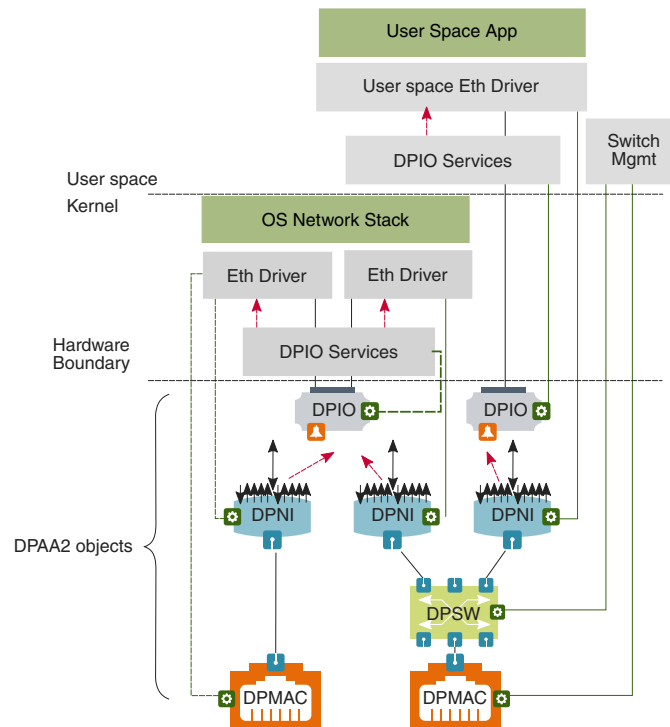


Figure 49. Object topology example

- There are three network interfaces managed by the DPAA2 Linux Ethernet driver. Each of the network interfaces uses a DPNI object.
- All of the Ethernet drivers happen to have a distribution width of one (an example), so they cannot load balance to multiple cores or threads; this was done to simplify the diagram and discussion. If a network interface has a distribution width greater than one, then many times it is connected to more than one DPIO but this is not required.
- Two of the network interfaces are connected to a switch; two DPNIs are connected to a DPSW. This allows both network interfaces to communicate outside of the SoC using the DPMAC that is also connected to the DPSW, and they can also communicate with each other using the DPSW.
- One of the network interfaces is directly assigned to a user space process, and has a user space Ethernet driver. This network interface could also be directly assigned to a KVM virtual machine under Linux.
- Two of the DPNIs have Linux network stack drivers; they interface to the Linux network stack. One of them has its own DPMAC, and a traditional type of controller represented by its DPNI being directly connected to a DPMAC.
- The two DPNIs connected to the Linux network stack share a single DPIO; this is possible when they can cooperatively use a layer of GPP software that provides DPIO services. The hardware that makes up a DPIO is a QMan software portal and, optionally, a QMan channel for data availability notifications. QMan software portals are a relatively scarce hardware resource, so they are designed to be sharable, in particular for NAPI-compliant Linux Ethernet drivers.
- It is a key assumption of DPAA2 that objects are managed (or “owned”) by a single software entity. Independent software entities can independently manage the objects they own, and this allows software to be decoupled from other entities.
- The management relationship between objects and software entities is not defined or imposed by DPAA2; DPAA2 defines the objects and what they do, and not what software uses them. Customer GPP core software is allowed to determine the management relationship; a single monolithic software entity that manages all of the objects can be created.

- The Linux DPAA2 Ethernet driver design defines the set of objects needed to provide a network interface. The green lines show the management relationships for Linux network interfaces and switches. Note that switches are managed independently from the network interfaces that connect to it.

The *DPAA2 User Manual* provides a complete description of the rules that govern object topology.

8.2.1.4.4 AIOP in DPAA2

This section describes AIOP in the DPAA2 architecture and how it uses DPAA2 objects. The figure below shows an example in which the Linux kernel network stack has a single Ethernet interface, a user space application has a single directly assigned Ethernet interface, and AIOP has two Ethernet interfaces.

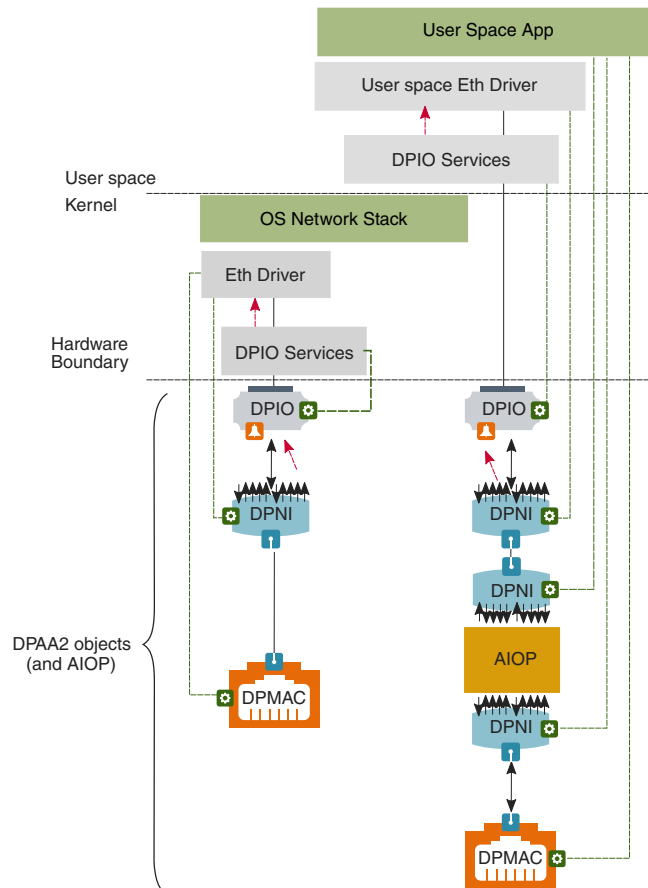


Figure 50. AIOP in DPAA2

AIOP runs software and uses objects in a manner that is similar to general purpose processing cores. In this example configuration, the AIOP's software has access to two Ethernet interfaces. One is connected to a MAC for an external connection to outside the SoC. The other is connected point-to-point to the general purpose processing core user space application's Ethernet interface.

So, there are two Ethernet interfaces involved on the path between the AIOP and the user space application. This is logical because there is software running in both places. Thus, each software component should see and control its own Ethernet Interface. Both software components do Ethernet I/O without being coupled to what their Ethernet interface is connected to.

One difference is that AIOP software is focused on packet processing. It does not actively manage or configure its own Ethernet interfaces. Usually, a control application on the general purpose processing cores takes that role. Also, the AIOP does not use DPIO objects.

Note that AIOP itself is not a DPAA2 object. It is an active entity that uses other DPAA2 objects. However, there is a DPAIOP object that general purpose processing core software can use to manage the AIOP, e.g. start it, stop it, load images onto it, get error status from it, etc.

In addition, there are DPAA2 objects (not shown) that facilitate passing commands (rather than packets) between general purpose processing core software and AIOP software.

AIOP Service Layer

NXP provides an AIOP software library called the "AIOP Service Layer". This library's main purpose is to provide very lightweight drivers for hardware components that AIOP software must access. These include components such as Ethernet interfaces, the QbMan buffer manager, timers, table lookup units, etc.

8.2.2 DPAA2 Quick Start Guide

8.2.2.1 Data Path Resource Containers

Many sections refer to Data Path Resource Containers (DPRC), so a brief introduction to the concept may be helpful. DPRCs are part of the DPAA2 object architecture that is described in the [DPAA2 Software Overview](#).

DPRCs are communicated to software entities as a part of their start up process; this is true for software entities such as:

- The host Linux kernel (that may provide KVM services to virtual machines)
- Linux kernel instances that run in virtual machines
- DPDK applications
- AIOP applications

DPRCs contain DPAA2 objects that are used by the software entity that owns the DPRC. For example, DPNI objects are used as network interfaces.

As an example, see [RDB DPL](#) on page 331 [RDB DPL](#) on page 330. The DPRC called "dprc@1" is supplied to the host Linux kernel. It contains one DPNI object. The DPNI object binds to the DPAA2 Linux kernel Ethernet driver, and causes two standard Linux Ethernet interfaces to exist and be visible using "ifconfig". See later sections in this document for additional details and explanations on the use of objects by various types of software entities.

As mentioned previously, DPRCs must be created and populated with the initial set of DPAA2 objects prior to the startup of the software entity that will use the DPRC.

8.2.2.1.1 Creating DPRCs

There are two ways to create and populate DPRCs:

1. Statically: by means of a control file called a datapath layout (DPL) file. [Key Release Files: RCW, DPC and DPL](#) on page 329 describes the DPL files that are supplied as examples with the Linux SDK.
2. Dynamically: by means of the Linux command line utility called restool. See [DPRCs and restool](#) section, and also the document titled *Standard Linux Documentation*.

A software entity behaves exactly the same way on startup regardless of whether its DPRC was created statically using a DPL or dynamically using restool. The DPL method is convenient for situations when the desired DPRCs are known in advance. DPRCs defined within the DPL are created and populated automatically with no need for a subsequent use of restool.

8.2.2.1.2 DPRCs and Hot Plug

A DPRC must be supplied to a software entity when the software entity is started; this implies prior creation of the DPRC. However, it is also possible to dynamically alter the contents of a DPRC *after* the software entity that is using it is already running; this is a form of hot plug.

For example, restool can be used to dynamically create a DPNI object and then assign it to a DPRC. If that DPRC is being used by a Linux kernel instance, this will cause that kernel to dynamically detect a new network interface and bind it to the Linux kernel Ethernet driver; the “ifconfig” command will now show a newly created network interface.

It is also possible to dynamically destroy or unassign objects within an in-use DPRC; this is a form of hot unplug. Hot plug and unplug are advanced topics, and not covered further here. The key take-away is that dynamically creating and populating a DPRC *before* supplying it to software entity when it is started is a very different use case than hot plug/unplug. The latter use case involves changing the contents of a DPRC *while it is in use* by a software entity.

8.2.2.2 Key Release Files: RCW, DPC and DPL

This section describes the key binaries that are available on the RDB. These include the reset configuration word (RCW), the data path configuration (DPC) and the data path layout (DPL) files.

8.2.2.2.1 RCW

The reset configuration word (RCW) resides in non-volatile memories (e.g. NOR, QSPI, SDHC). It gives flexibility to accommodate a large number of configuration parameters to support a high degree of configurability of the SoC. Configuration parameters generally include:

- Frequencies of various blocks including cores/DDR/interconnect.
- IP pin-muxing configurations
- Other SoC configurations

The RCW's provided with the release enable the following features:

- Boot location as NOR flash
- Enables 4 UART without flow control
- Enables I2C1, I2C2, I2C3, I2C4, SDHC, IFC, PCIe, SATA

The figure below shows the SERDES configuration supported for DPAA2 platforms. Note that each platform supports upto 5 ports out of the 8 available on the RDB at a time.

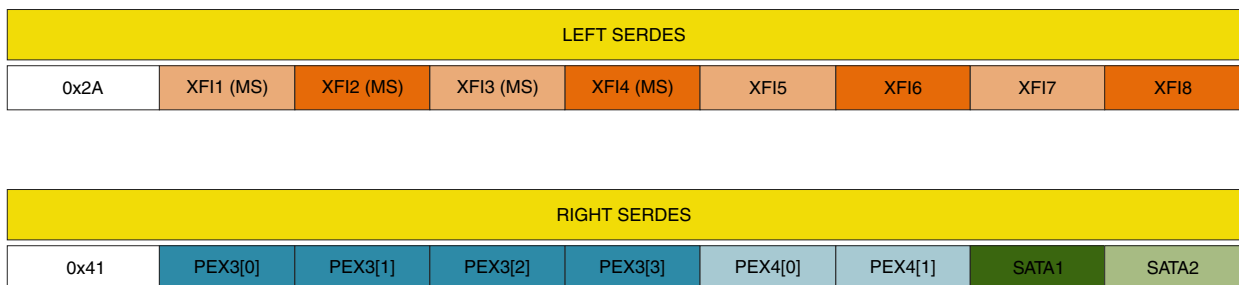


Figure 51. SERDES

8.2.2.2.2 Data path configuration file (DPC)

The data path configuration (DPC) contains board-specific and system-specific information that overrides the default DPAA hardware configuration.

The release provides one data path configuration (DPC) file per board type. This file specifies the following information:

- default logging mode for the Management Complex (MC)
- default board MACs
- default number of DPAA channels with 2 and 8 work-queues

The DPC is based on a text source file similar to a device tree source file (DTS) and should be compiled using the DTC utility to form a binary structure (blob, similar to DTB).

8.2.2.2.3 Data path layout file (DPL)

The data path layout file (DPL) defines the containers created during MC initialization. In order to compile the DPL, the device tree compiler (DTC) tool needs to be installed on the host system.

As described in [Creating DPRCs](#) on page 328, the example DPL source code is provided in the *dpl-examples* package.

The DPL file specifies the basic resources needed for a simple usecase; other resources are created and managed dynamically using restool capabilities. For each of the use cases included in this document, there is a diagram that depicts the objects that are necessary for that usecase.

The DPL is based on a text source file (similar to a device tree source file (DTS)) and compiled with the DTC utility to form a binary structure (blob, similar to DTB). The DPL file should be compiled to a binary blob using standard DTC tool.

Using a static DPL is not a requirement since restool can be used to dynamically create/manage objects and resources.

8.2.2.2.3.1 RDB DPL

The source for the RDB DPL is in the *dpl-examples* package:

- `dpl-examples/LS2088a/RDB/dpl-eth.0x2A_0x41.dts`

The figure below shows a graphical view of the container configuration:

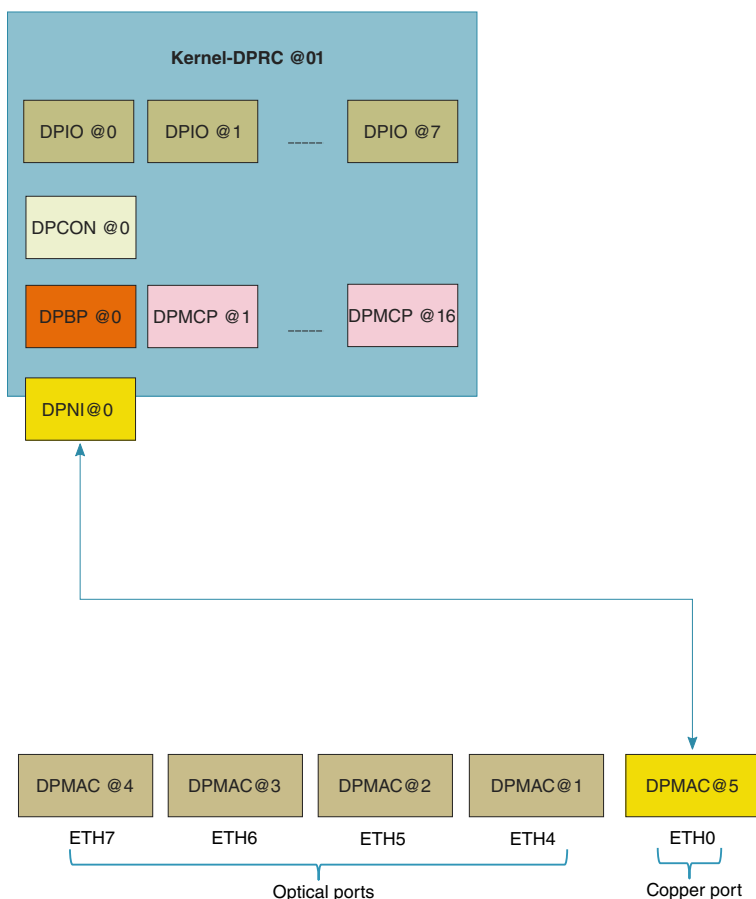


Figure 52. RDB DPL container configuration

8.2.2.2.3.2 RDB DPL

The source for the RDB DPL is in the *dpl-examples* package:

- `dpl-examples/ls1088a/RDB/dpl-eth.0x1D_0x0D.dts`

The figure below shows a graphical view of the container configuration:

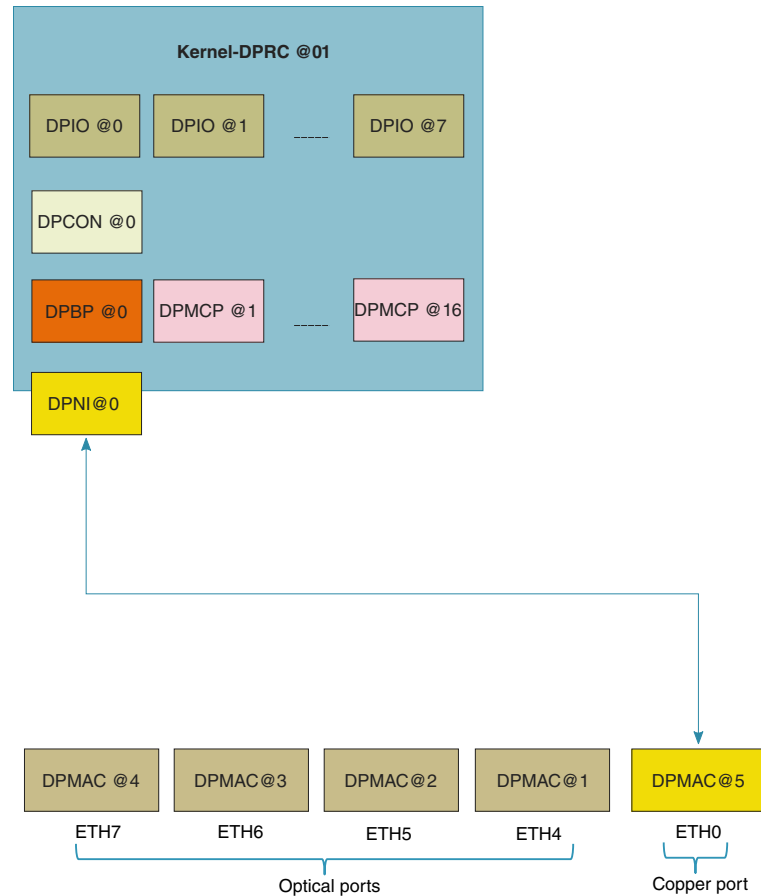


Figure 53. RDB DPL container configuration

8.2.2.2.3.3 DPRCs and restool

The release provides a Linux command-line tool called *restool* that can be used for examining the resource containers used for managing DPAA2 objects and resources. See the [DPAA2 Overview](#) for an overview of DPAA2 and the data path resource containers (DPRCs). Also see the [Standard Linux Documentation](#) for details about functionality and use of *restool*.

Given below is an example of using *restool*:

List dprc:

```
$ restool dprc list
dprc.1
```

List all objects in container dprc.1:

```
$ restool dprc show dprc.1
```

```
dprc.1 contains 33 objects:
object      label      plugged-state
dpio.7      label      plugged
dpio.6      label      plugged
dpio.5      label      plugged
dpio.4      label      plugged
dpio.3      label      plugged
dpio.2      label      plugged
dpio.1      label      plugged
dpio.0      label      plugged
dpni.0      label      plugged
dppbp.0     label      plugged
dpmac.5     label      plugged
dpmac.4     label      plugged
dpmac.3     label      plugged
dpmac.2     label      plugged
dpmac.1     label      plugged
dpcon.0     label      plugged
dpmcp.0     label      plugged
dpmcp.16    label      plugged
dpmcp.15    label      plugged
dpmcp.14    label      plugged
dpmcp.13    label      plugged
dpmcp.12    label      plugged
dpmcp.11    label      plugged
dpmcp.10    label      plugged
dpmcp.9     label      plugged
dpmcp.8     label      plugged
dpmcp.7     label      plugged
dpmcp.6     label      plugged
dpmcp.5     label      plugged
dpmcp.4     label      plugged
dpmcp.3     label      plugged
dpmcp.2     label      plugged
dpmcp.1     label      plugged
```

Get information about dpni.0:

```
dpni version: 7.1
dpni id: 0
plugged state: unplugged
endpoint state: -1
endpoint: No object associated
link status: 0 - down
mac address: 00:00:00:00:00:00
dpni_attr.options value is: 0
num_queues: 1
num_tcs: 1
mac_entries: 16
vlan_entries: 0
qos_entries: 0
fs_entries: 64
qos_key_size: 0
fs_key_size: 56
ingress_all_frames: 0
ingress_all_bytes: 0
ingress_multicast_frames: 0
ingress_multicast_bytes: 0
```



```

ingress_broadcast_frames: 0
ingress_broadcast_bytes: 0
egress_all_frames: 0
egress_all_bytes: 0
egress_multicast_frames: 0
egress_multicast_bytes: 0
egress_broadcast_frames: 0
egress_broadcast_bytes: 0
ingress_filtered_frames: 0
ingress_discarded_frames: 0
ingress_nobuffer_discards: 0
egress_discarded_frames: 0
egress_confirmed_frames: 0

```

8.2.2.3 Linux Ethernet

This chapter provides guidelines on exercising creation, functionality and statistics of Linux DPAA2 Ethernet interfaces.

8.2.2.3.1 Features overview

The following is an overview of the functionality of the Linux DPAA2 Ethernet driver that will be described in this chapter:

- Primary MAC address change
- Scatter-gather support
- Checksum offload
- MAC filtering
- Large frame support
- GRO – generic receive offload
- Egress traffic shaping
- Rx hashing
- Rx flow steering (LS2088A only)
- Flow control pause frames
- Interface statistics

8.2.2.3.2 Compiling and selecting Kconfig options

The DPAA2 Ethernet driver is by default selected by the kernel configuration shipped with the SDK, with a set of sensible compile-time defaults. The driver path in the kernel config file is: " Device Drivers -> Staging drivers -> Freescale DPAA2 devices -> Freescale DPAA2 Ethernet" (CONFIG_FSL_DPAA2_ETH).

The following Kconfig selects are also available, but not checked by default:

- "Enable Rx error queue" (CONFIG_FSL_DPAA2_ETH_USE_ERR_QUEUE) - This configures a separate queue for presenting Rx Error frames to the Ethernet driver running on the GPP. By default this option is disabled and Rx Error frames are dropped in hardware and counted for statistics inspection. Rx Error processing increases the GPP load and so it is only necessary in debugging situations when inspection of actual frames is required.
- "Enable debugfs support" (CONFIG_FSL_DPAA2_ETH_DEBUGFS). This option depends on "Debug Filesystem" (CONFIG_DEBUG_FS) being enabled. If selected with its dependencies, this option enables a number of Ethernet driver statistics under DebugFS nodes (/sys/kernel/debug/dpaa2-eth/*) that are not normally visible to the user.
- "Data Center Bridging (DCB) Support" (CONFIG_FSL_DPAA2_ETH_DCB). This option depends on "Data Center Bridging support" (CONFIG_DCB). It is required when configuring Priority-based Flow Control (PFC) scenarios.

- "DPAA2 Ethernet CEETM QoS" (`CONFIG_FSL_DPAA2_ETH_CEETM`). This option enables the use of a custom CEETM qdics to offload egress Qos support.

8.2.2.3.3 Creating a DPAA2 network interface

This section documents the resource utilization and the necessary steps for creating a DPAA2 network interface (DPNI) in Linux.

A DPNI can be created either statically through the DPL file or dynamically using the 'restool' utility.

8.2.2.3.3.1 DPAA2 objects dependencies

This section documents the steps to create a DPNI and related objects in order to have a fully functional network interface. It describes the dependencies a DPNI has on other DPAA2 objects.

This is of interest to anyone who changes a static DPL file or uses *restool* commands to dynamically create: *restool* to create a DPNI (For LS1088A see Using *restool* to create a DPNI).

The DPAA2 object definition allows for flexible software architectures. The Linux drivers, in particular the Ethernet driver, are additionally bound by requirements from the kernel architecture. This enforces a certain usage model of the DPAA2 objects that the DPNI interacts with; in particular, it affects the number of various other DPAA2 objects that a DPNI need s.

Generally, the Linux Ethernet driver requires private DPAA2 resources (e.g. Frame Queues) and objects (e.g. DPCON objects), distinct from other DPNIs. There are exceptions, such as the DPIO or DPMAC, which are not owned by the DPNI. To create a DPNI, either statically in DPL or dynamically using 'restool', the following types of objects may need to be instantiated in the current container (i.e., made available if they are not already):

- DPBP
- DPMCP
- DPCON
- DPIO
- DPMAC

The significance and number of these objects per DPNI are detailed in the following table:

Object	Private to DPNI?	Cardinality	Comments
DPBP	Yes	1 per DPNI	Each network interface (NI) has private buffer pools, not shared with other NIs.
DPMCP	Yes	1 per DPNI 1 per DPMAC	MC command portals (MCPs) are used to send commands to, and receive responses from, the MC firmware. One such example is configuring DPNI functionality like hashing or checksumming, but DPNI statistics are also queried via the MCPs. Like the DPNI, each DPMAC also requires one private DPMCP.

Table continues on the next page...

Table continued from the previous page...

Object	Private to DPNI?	Cardinality	Comments
DPCON	Yes	Rx hash size/ number of transmitter queues ("num_queues") of the DPNI.	DPCONs are used to distribute Rx or Tx Confirmation traffic to different GPPs, via affine DPIO objects. The implication is that one DPCON must be available for each GPP we want to distribute Rx or Tx Confirmation traffic to. Rx and Tx Confirmation share the same DPCONs if they are available. (If for example GPP #0 processes both types of traffic, one DPCON is enough. If in addition GPP #1 processes only Tx Confirmation traffic, then a second DPCON is necessary.) Since we must be able to distinguish between traffic from different NIs arriving on the same GPP, the DPCONs must be private to the DPNIs. These design constraints may cause a relatively large consumption of DPCONs by DPNI's with large Rx distribution width. The DPNI's Rx distribution width is implemented by the "num_queues" property (see Rx hashing on page 342 for extra information). Notes: DPCONs' main hardware resource are the Work Queues (WQs). The DPCONs come in 2 flavours: 2-WQ and 8-WQ DPCONs, depending on the number of traffic class priorities the object is going to support. (Note: the Ethernet driver only supports one traffic class at the moment, so using 2-WQ DPCONs is safe and enough.) The MC firmware can convert any number of 8-WQ DPCONs to four times as many 2-WQ DPCONs, depending on the static configuration provided at boot. Since WQs are a limited hardware resource, DPCONs tend to be limited, too, especially the 8-WQ flavor. The DPNI's being one of the major consumers of DPCONs, the current SDK ships with a default configuration where a number of the 8-WQ DPCONs are converted to 2-WQ DPCONs, thereby increasing their availability. Note that DPIO objects themselves transparently consume DPCONs (one per DPIO object), which therefore must be subtracted from the total number available to the DPNI's (they need not be explicitly declared in the DPL, but they are simply not available to the rest of the system). The system can provide up to 64 8-WQ DPCONs (and up to 256 2-WQ DPCONs and combinations thereof). So for a container with 8 DPIOs, only up to 56 8-WQ DPCONs will be in fact available for DPNI configuration.
DPIO	No	One per running GPP	DPIOs are used to provide data availability notifications to the GPPs. For each GPP that we want to distribute traffic to, there must be an affine DPIO. While DPIOs are the source of data availability interrupts, the DPCONs are used (among other things) to identify the NI that has produced ingress data to that GPP. Due to a known limitation, the number of DPIOs in a container must not be less than the number of running GPPs. The static DPL in this release defensively provides 8 DPIOs at boot-time, one for each running GPP.

Table continues on the next page...

Table continued from the previous page...

Object	Private to DPNI?	Cardinality	Comments
DPMAC	No	User-defined.	DPMACs are proxy objects which link DPNI to external PHYs on the board. DPMACs effectively decouple DPNI from the PHYs they are linked to (if they are indeed linked to an external PHY, which is in fact transparent to the DPNI). As such, the DPMACs are not "owned" by a DPNI, which is unaware of their presence, but they can be "connected" to the DPNI, via the DPL file or 'restool'. DPMACs can be connected to other types of objects, too, such as the EVB. Having DPMACs connected to external PHYs depends on the board wiring and is strictly confined to the SerDes configuration. See also: DPMAC configuration on page 337.

8.2.2.3.3.2 Static DPNI definition

The default DPL provides a simple DPNI object definition, under the dpni@0 node as follows:

```
dpni@0 {
    options = "";
    num_queues = <1>;
    num_tcs = <1>;
};
```

The DPNI object is linked to a DPMAC object, also created in the DPL, via the "connections" node as follows:

```
connections {
    connection@5 {
        endpoint1 = "dpni@0";
        endpoint2 = "dpmac@5";
    };
};
```

The DPNI object can be more complex, as in the following enhanced example of a DPNI node:

```
dpni@1 {
    options = "DPNI_OPT_HAS_KEY_MASKING";
    num_tcs = <1>;
    num_queues = <8>;
    mac_filter_entries = <64>;
};
```

In this example, dpni@1 has more options declared than dpni@0 in the previous example. In addition, it can distribute traffic to more GPPs than dpni@0, as declared by the "num_queues" attribute.

The following section describes the DPNI bindings in the DPL file.

8.2.2.3.3.2.1 DPNI bindings

- The `num_queues` attribute indicates the number of queues to be used for transmission as well as the number of Rx queues (hash distribution size). This also implicitly defines the number of queues used for Tx Confirmation, since each "sender" uses a dedicated queue for confirmations. This may impact the number of necessary DPCON objects - see "[DPAA2 objects dependencies](#)" chapter for details on resourcing the DPNI.

- `num_tcs` represents the number of traffic classes; maximum supported value is 8.
- `options` allows the creation of a DPNI object with non-default options
- Other possible attributes are listed below. Unless otherwise stated, attributes with value `<0>` receive a default, non-trivial, value from the MC firmware and can be skipped from the DPL altogether.
 - `fs_entries`
 - `vlan_filter_entries`
 - `mac_filter_entries`

Note: See MC documentation for all available options and supported values.

8.2.2.3.3.3 DPMAC configuration

This section is a brief introduction to DPMAC objects and their relation to the DPNI.

DPMACs are essentially proxies to external PHYs, which are board-level components and therefore not managed by the MC firmware.

DPMACs can be connected to other DPAA2 objects, such as DPNI, DPDMUX and DPSW. For example, to statically connect a DPMAC to a DPNI in the DPL file, the “connections” node is used:

```
connection@5{
    endpoint1 = "dpni@0";
    endpoint2 = "dpmac@5";
};
```

In this DPL example, DPNI0 is connected to DPMAC5, which the MC thereon connects to a lane depending on the current SerDes. Unlike most other DPAA2 objects, the id of the DPMAC (in this example, “5”) is relevant as the MC uses it to identify a physical MAC (at the moment, there is no other property of the DPMAC object to do that).

8.2.2.3.3.4 Dynamically creating a DPNI

This chapter documents the steps to create a DPNI using the *restool* utility and the *restool* wrapper scripts.

8.2.2.3.3.4.1 Using restool to create a DPNI

DPNIs can be dynamically created and plugged into the Linux container using the *restool* utility. Before creating a DPNI, one must create a number of DPAA2 objects (dependencies), for which multiple *restool* commands are needed. This section provides simple examples of commands that should be used to create a working DPNI (Linux network interface) and its dependencies. Usage of the Restool Wrapper Script bundled with the SDK is encouraged, because of their better ease-of-use.

In order to create an object, the “restool create” command must be executed and then the new object can be assigned to a container. For example, to create a DPBP object:

```
$ restool dpbp create
dpgp.1 is created under dprc.1
$ restool dprc assign dprc.1 --object=dpgp.1 --plugged=1
```

For automation purposes, the “--script” flag can be used, reducing the verbosity of the command output. Object properties can be specified at creation time as follows:

```
$ restool --script dpio create --channel-mode="DPIO_LOCAL_CHANNEL" --num-priorities=8
dpio.8
```

To create a DPNI, a number of DPMCP, DPBP and other dependencies are required, if they do not exist already in the container - refer to the [DPAA2 objects dependencies](#) on page 334 chapter for details on the types and number of DPNI

dependencies. The static DPL from the current release already defines 8 DPIO objects, one for each running GPP, so adding new DPIOs is not normally required. Also, the maximum number of DPMACs supported on LS2088A and LS1088A are already created in the static (default) DPLs, so adding new ones is not necessary in the default configuration.

The general steps to create and configure a DPNI using restool are:

1. Create DPAA2 object dependencies (DPBPs, DPCONs, DPMCPs, etc);
2. Create and parametrize the DPNI;
3. Connect the DPNI to another object (typically but not necessarily a DPMAC).

The Restool Wrapper Script automatically take care of the DPNI resourcing and parametrization, therefore we encourage their use instead of bare *restool* for complex objects like the DPNI.

8.2.2.3.3.4.2 Restool Wrapper Scripts

User-friendly scripts are provided in the release rootfs to assist dynamic creation of DPNI and associated dependencies. They also implement parameter restrictions and workarounds related to known limitations of the DPAA2 objects in the current SDK release.

The following scripts are available to interact with DPNI and DPMAC objects, respectively Linux network interfaces: *ls-addni*, *ls-listni*, *ls-listmac*.

1. *ls-addni*

This script creates a new DPNI object, required dependencies (potentially DPBP, DPMCP, DPCON, DPMAC, depending on the options being passed to the script) and an associated Linux network interface. The script can be used to connect the newly-created DPNI to another DPNI, DPMAC or DPDMUX, which must be already created and not currently connected.

The script supports a multitude of parameters to fine-tune configuration of the DPNI; in fact, it is intended to support every parameter as *restool* itself for creating DPNI. An empty list of options will choose sensible defaults for maximal performance of the new DPNI, such as Rx hashing to the maximum number of cores.

Adding a new DPNI has the effect of discovering the new object on the Linux mc-bus and probing it as a new Ethernet device. This results in a new Linux network interface becoming available. The new interface has the name *eth<X>*, where *x* depends on the order in which the interfaces are probed and on what other interfaces (e.g. PCIe NIC) are present. The mapping between the DPNI object and the interface name is shown by the *ls-listni* command.

Utilization examples:

- ```
ls-addni dpmac.6
[70218.813064] fsl_dpaa2_eth dpni.4: Probed interface eth2
Created interface: eth2 (object:dpni.4, endpoint: dpmac.6)
```

This is probably the most typical usage example. It creates a network interface (eth2) and the underlying dpni (dpni.4) and connects it to an external MAC (dpmac.6).

Connecting DPNI to DPMACs is not the only option, though:

- ```
# ls-addni -n
[70270.944458] fsl_dpaa2_eth dpni.5: Probed interface eth3
Created interface: eth3 (object:dpni.5, endpoint: none)
```

This command creates the unconnected object dpni.5 and the respective Linux interface eth3. Not being connected to anything, there is little practical use for this interface; therefore, a command such as the following would be used:

- ```
ls-addni dpni.5
[70312.255487] fsl_dpaa2_eth dpni.6: Probed interface eth4
Created interface: eth4 (object:dpni.6, endpoint: dpni.5)
```

This command creates another network interface eth4 (and the underlying dpni.6 object) and connects it with the previously created eth3 (dpni.5) interface.

**Notes:**

*ls-addni --help* list all supported options.

Although it is technically possible to connect a DPNI to itself, the wrapper scripts do not support this;

## 2. **ls-listni**

This script lists all the dpni objects available in the root and child containers, the associated network interface name, the end point and the label.

Output after running the above examples (dpni.0 through dpni.3 had been statically defined in the DPL):

```
ls-listni
dprc.1/dpni.6 (interface: eth4, end point: dpni.5)
dprc.1/dpni.5 (interface: eth3, end point: dpni.6)
dprc.1/dpni.4 (interface: eth2, end point: dpmac.6)
dprc.1/dpni.3
dprc.1/dpni.2
dprc.1/dpni.1
dprc.1/dpni.0 (interface: eth1, end point: dpmac.2)
```

## 3. **ls-listmac**

This script lists all the dpmac objects available in the root and child containers, the associated network interface name, the end point and the label.

Output after running the above examples (dpni.0 had been connected to dpmac.2 in the static DPL):

```
ls-listmac
dprc.1/dpmac.10
dprc.1/dpmac.9
dprc.1/dpmac.8
dprc.1/dpmac.7
dprc.1/dpmac.6 (end point: dpni.4)
dprc.1/dpmac.5
dprc.1/dpmac.4
dprc.1/dpmac.3
dprc.1/dpmac.2 (end point: dpni.0)
dprc.1/dpmac.1
```

## 8.2.2.3.4 DPAA2 Ethernet features

This section presents the individual functions of the Linux DPAA2 Ethernet driver.

### 8.2.2.3.4.1 Bring up the bootstrap DPNI interface

From Linux, interfaces are visible through the 'ifconfig' command. The DPAA2 interfaces are named as "eth<X>," where X depends on the order in which the interfaces are probed.

The default DPL file shipped with the current BSP release contains one statically-defined DPNI object (DPNI.0).

DPNI.0 is configured with a minimal set of resources – e.g. it can only receive traffic on GPP0 – and its intended uses are network boot and low-bandwidth traffic. For fully-featured DPNI objects, dynamic configuration is recommended (see [Dynamically creating a DPNI](#) on page 337).

For IP connectivity between the default interface and an external host, first assign a valid IP address to it as in the following example:

```
$ ifconfig eth1 192.168.1.2
```

Assuming the remote peer has address 192.168.1.1, ping to test as shown:

```
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=87.0 ms
```

### 8.2.2.3.4.2 Primary MAC address change

Changing the primary MAC address of a Linux Ethernet interface is supported without the need to bring down the interface.

For example:

```
$ ifconfig eth1 hw ether 02:00:C0:01:02:0a
```

```
$ ip link set dev eth1 address 02:00:C0:01:02:0b
```

### 8.2.2.3.4.3 Scatter/gather configuration

The Ethernet driver supports scatter/gather (S/G) on both the transmit and receive side. The S/G option can be configured through ethtool on Tx; on Rx, S/G support is always on. For example, in order to see the current state of the device features and hardware offloads for device `ni0` :

```
$ ethtool -k eth1
Features for eth1:
[...]
scatter-gather: on
 tx-scatter-gather: on
[...]
```

In order to change the S/G status of the Linux Ethernet interface `ni0`:

```
$ ethtool -K eth1 sg off
Actual changes:
scatter-gather: off
 tx-scatter-gather: off
generic-segmentation-offload: off [requested on]

$ ethtool -K eth1 sg on
Actual changes:
scatter-gather: on
 tx-scatter-gather: on
generic-segmentation-offload: on
```

#### Notes:

- S/G support on the *egress* path, together with High DMA, which is also supported, allows for efficiently transmitting TCP segments from user-space, without copying them to kernel-space first (“Tx zero-copy”).
- Egress S/G is necessary for other kernel features such as generic segmentation offload (GSO, implicitly turned on).
- The Ethernet driver support for S/G frames on the *ingress* path is transparent to the user.



#### 8.2.2.3.4.4 Checksum offload configuration

The Ethernet driver supports hardware offloading of both Rx checksum validation and Tx checksum generation for TCP and UDP over IPv4/IPv6. The hardware checksum offload can be configured through ethtool.

For viewing the current state of the feature use the “-k” flag:

```
$ ethtool -k eth1
Features for eth1:
[...]
rx-checksumming: on
tx-checksumming: on
 tx-checksum-ipv4: on
 tx-checksum-ipv6: on
[...]
```

The checksum offload can be controlled separately on Rx and Tx paths as follows:

```
$ ethtool -K eth1 rx on|off
$ ethtool -K eth1 tx on|off

$ ethtool -k eth1 | grep tx-checksumming
tx-checksumming: off
```

#### 8.2.2.3.4.5 MAC filtering

The DPAA2 hardware supports unicast and multicast MAC filters on the ingress path. In Linux, MAC unicast filtering can be accomplished with the help of MACVLAN interfaces. Kernel configuration and DPNI configuration are required to enable this feature, as follows:

- To enable support in the kernel, CONFIG\_MACVLAN must be selected at compile-time, from the kernel menuconfig:  
“Device Drivers -> Network device support -> Network core driver support -> MAC-VLAN support” .

The Linux Ethernet driver allows adding and deleting of MAC filters via the standard “ip” command. An example of adding/deleting a MAC unicast address is the following:

```
$ ip link add link eth1 address <macvlan_mac_addr> eth1.1 type macvlan
$ ifconfig eth1.1 up
[...]
$ ip link delete eth1.1 type macvlan
```

Adding a multicast address is also possible using the “ip” command as follows:

```
$ ip maddr add 01:00:00:00:00:01 dev eth1
```

#### 8.2.2.3.4.6 Large frame support

The DPAA2 hardware supports large frames. The Ethernet driver correlates between the Layer-2 maximum frame length (MFL) and Layer-3 MTUs. The maximum MTU that a Linux user can request on a DPAA2 Ethernet interface is 10222 bytes.

##### Notes:

- Outgoing packets larger than the current MTU are going to be fragmented by the kernel stack.
- All Ethernet devices on the same LAN must have the same MTU
- Ingress frames larger than MTU are accepted by the Ethernet driver

### 8.2.2.3.4.7 Generic receive offload

The DPAA2 Ethernet driver is integrated with the kernel's generic receive offload (GRO) support. GRO is enabled by default and is configurable via "ethtool":

```
$ ethtool -k eth1 | grep generic-receive-offload
generic-receive-offload: on
$ ethtool -K eth1 gro off
$ ethtool -k eth1 | grep generic-receive-offload
generic-receive-offload: off
```

#### NOTE

For better performance, GRO should be disabled on the receiving interfaces in certain scenarios such as IP Forwarding.

### 8.2.2.3.4.8 Egress traffic shaping

The DPAA2 Ethernet interface supports traffic shaping in the egress path. Egress shaping can be set or cleared via a per-interface entry in SysFS:

```
$ echo M N > /sys/class/net/eth1/tx_shaping
```

where:

M is the maximum throughput, expressed in Mbps.

N is the maximum burst size, expressed in bytes, at most 63487.

To remove shaping, use M=0, N=0.

### 8.2.2.3.4.9 Rx hashing

The DPAA2 Ethernet driver supports hash distribution of ingress flows, based on some of the common L2/L3/L4 fields. Configuration is done via standard "ethtool" support as follows:

```
$ ethtool -N <ethX> rx-flow-hash <proto_type> <header_fields>
```

The set of header fields from which the hash key is extracted is configured globally for all protocols and the protocol type parameter is ignored.

The following fields are supported:

- m - Ethernet destination address
- v - VLAN tag
- t - L3 protocol
- s - IPv4 source address
- d - IPv4 destination address
- f - L4 bytes 0 & 1 [TCP/UDP source port]
- n - L4 bytes 2 & 3 [TCP/UDP destination port]

The "r" flag (discard all packets of this flow type) is not supported.

For example, Rx hashing based on IP source and destination address can be configured with the following command:

```
$ ethtool -N <ethX> rx-flow-hash udp4 sd
```

The current hashing configuration can be viewed using the “-n” flag:

```
$ ethtool -n <ethX> rx-flow-hash udp4
```

The protocol type parameter is ignored; the configuration applies to both UDPv4 and TCPv4.

**Note:** By default, Ethernet interfaces start with hashing enabled on a 5-tuple key (IP proto, IP src/dst addresses, L4 src/dst ports). If an Ethernet interface is created with a single queue then hashing is not supported.

Interfaces created dynamically with “ls-addni” have a number of queues equal to the number of available CPUs, unless explicitly requested otherwise, so they'll have hashing enabled by default. For DPNI's statically defined inside a DPL file, in order to allow hashing the “num\_queues” property must have a value larger than 1 and there must also be a sufficient number of DPCON objects available.

For full details and examples on dynamic DPNI creation, refer to [this chapter](#).

### 8.2.2.3.4.10 Rx flow steering

The DPAA2 Ethernet driver supports steering of ingress traffic, directing flows to specific GPPs based on exact-match operations on some of the common L2/L3/L4 fields. The advantage versus [Rx hashing](#) on page 342 is cache locality of ingress data: the user-space applications that actually process the traffic make better use of the local GPP's cache than if the traffic were processed on another GPP. The disadvantage stems from the static configuration of flow affinity and from the fact that flow characteristics (e.g. L4 ports) must be known in advance, which is not always possible in real scenarios.

Configuration is done via standard “ethtool” support as follows:

```
$ ethtool -N eth1 flow-type <proto_type> <header_field> <value> [m <mask>] action <cpu_id>
```

Steering is supported for the following protocols:

- ethernet (flow-type `ether`)
- IPv4 (flow-type `ip4`)
- TCP, UDP over IPv4 (flow-type `tcp4`, `udp4`)

Supported fields are as follows:

- `src`, `dst` (L2 source/destination address; only for `ether` flow type)
- `dst-mac` (only for `ip4`, `udp4`, `tcp4` flow types)
- `vlan` (all flow types)
- `l4proto` (only for `ip4`)
- `src-ip`, `dst-ip`, `src-port`, `dst-port` (for `ip4`, `udp4`, `tcp4`)

Masking of header fields is also supported.

For example, in order to set up flow steering based on destination IP:

```
$ ethtool -N eth1 flow-type ip4 dst-ip 192.168.1.0 action 0
```

or subnet:

```
$ ethtool -N eth1 flow-type ip4 dst-ip 192.168.2.0 m 0.0.0.255 action 1
```

#### NOTE

The MC firmware and Linux Ethernet driver will only fully configure flow-steering if `DPNI_OPT_HAS_KEY_MASKING` is set in the "options" list of the DPNI object either via the DPL node or via `restool` - e.g.:

```
dpni@1 {
 [...]
 options = "DPNI_OPT_HAS_KEY_MASKING";
 [...]
};
```

respectively:

```
restool dpni create [...] --options=DPNI_OPT_HAS_KEY_MASKING
```

#### NOTE

On LS1088A only limited flow steering capabilities are offered. LS1088A does not support the `DPNI_OPT_HAS_KEY_MASKING` option, thus it won't allow rules with different keys in the classification table. In other words, all rules in the classification table must be based on the same header field(s). Also, `m` option is not supported and it will be silently ignored.

### 8.2.2.3.4.11 Flow Control Pause Frames

The DPAA2 Ethernet interfaces support sending and responding to pause frames, as part of the Ethernet flow control mechanism. The behavior of the pause frames is described in the IEEE 802.3x standard. In a nutshell, in a scenario involving a full duplex link, if the sender is sending at a higher rate than the receiver can process frames, the receiver can choose to send a special kind of frame, called pause frame, which asks the sender to halt the transmission of traffic for a specified period of time.

Pause frame control is integrated into `ethtool`. By default it's enabled for both Tx and Rx.

```
~# ethtool -a eth1
Pause parameters for eth1:
Autonegotiate: on
RX: on
TX: on

~# ethtool -A eth1 rx off
~# ethtool -a eth1
Pause parameters for eth1:
Autonegotiate: on
RX: off
TX: on
```

Currently, there's no support in configuring pause frame autonegotiation independently from link general autonegotiation. Hence, if link autonegotiation (e.g. rate and duplex) is on, it is on pause frames as well, and viceversa.

Pause frames are interpreted at the MAC level. Therefore, the counters for pause frames are visible when configuring netdevice objects for DPMAC objects using the `CONFIG_FSL_DPAA2_MAC_NETDEVS=y` kernel option. This option will bring up a set of additional Linux network interfaces named `macX`, one for each probed DPMAC object in the system.

```
~# ethtool -S mac1 | grep pause
rx pause: 106731474
tx b-pause: 15287253
```

If the driver is configured with Tx pause frames on, the hardware will start sending pause frames when the interface enters a congestion state on the Rx side.

If the driver is configured with Rx pause frames on, it will respond to any pause frames received on the line by reducing the send rate.

#### NOTE

Interrogating the current state of the flow control support and changing it through ethtool works when the interface is up.

### 8.2.2.3.4.12 Ethernet Priority-based Flow Control

The DPAA2 Ethernet interfaces support sending and responding to 802.1Qbb PFC (Priority-based Flow Control) frames, also known as CBFC (Class Based Flow Control) frames. PFC is a function of the 802.1 DCB (Data Center Bridging) standard, enabling lossless semantics at L2 on the Ethernet medium. Eight different classes of service (802.1p Ethernet priorities) are available as expressed through the 3-bit PCP field in an IEEE 802.1Q (VLAN) header added to the frame.

The DPAA2 Ethernet driver supports enabling PFC for a subset of the traffic classes. This configuration is done using a higher level protocol, LLDP - Link Layer Discovery Protocol. In Ubuntu, this protocol is implemented by the lldpad package, containing lldpad - the agent daemon - and lldptool - the client program.

Before attempting to configure PFC, make sure lldpad is installed and running:

```
~# apt-get update
~# apt-get install -y lldpad
~# service --status-all
~# service lldpad status
 lldpad.service - LSB: Start and stop the lldp agent daemon
Loaded: loaded (/etc/init.d/lldpad; bad; vendor preset: enabled)
Active: active (running) since Mon 2017-08-21 11:43:45 UTC; 2h 33min ago
Docs: man:systemd-sysv-generator(8)
CGroup: /system.slice/lldpad.service
-4542 /usr/sbin/lldpad -d
```

The LLDP agent daemon will register all the active interfaces in the system. In order to configure PFC for an interface (e.g. eth1) please run the following commands:

- Set the LLDP operation mode - in this case, to send and receive LLDP packets. This is required in order for PFC changes to take effect.

```
lldptool -L -i eth1 adminStatus=rxtx
```

- Enable PFC for priorities 1, 2, and 4 on ni0.

```
lldptool -T -i eth1 -V PFC -c enabled=1,2,4
```

- Display priorities enabled for PFC on ni0.

```
lldptool -t -i eth1 -V PFC -c enabled
```

In order to disable PFC, you can run the following command:

```
lldptool -T -i eth1 -V PFC -c enabled=none
```

When setting PFC for the first time since boot, the DPAA2 Ethernet driver will configure static ingress traffic classification based on VLAN PCP. In order for this to work, you need to configure the DPNI with a number of traffic classes that's greater than 1 - preferably `num_tcs=8`, since there are a total of 8 priorities handled by the 3-bit PCP VLAN field.

The LLDP interface configuration is persistent across reboot and stored in the lldpad configuration files.

It's not advised to change the PFC configuration when the interface is handling heavy traffic.

There's a current **known limitation** for PFC to work only with DPNIs created using the DPL. DPNIs created with restool will not behave as expected.

### 8.2.2.3.4.13 XDP support

The DPAA2 Ethernet driver offers support for XDP (eXpress Data Path) programs. Support is enabled by default and no special configuration is needed.

XDP is a high performance data path in the Linux kernel, which allows for fast and programmable frame processing.

XDP programs are based on eBPF (extended Berkeley Packet Filter) and some basic examples can be found in the kernel source tree, in `samples/bpf`. Samples of the associated userspace apps that load the XDP program and attach it to the desired network interface can also be found there. For example, the "xdp1" sample program can be loaded by running:

```
./samples/bpf/xdp1 -N <interface_index>
```

The userspace applications that load XDP programs have to be built for arm64. XDP programs are compiled using clang/llvm, with minimum required version being 6.0. We recommend building natively, following the steps described in `samples/bpf/README.rst`.

The currently supported actions are:

- *XDP\_DROP* : any frame for which this action is selected is dropped immediately by the driver
- *XDP\_PASS* : frame follows the standard processing path and is sent to the network stack
- *XDP\_TX* : frame is forwarded back to the same interface
- *XDP\_REDIRECT* : frame is forwarded to another interface; only available on the 4.14 kernel

The driver also supports header updates that change the frame header size, but only on the 4.14 kernel.

Scatter/gather frames are not handled by the XDP program and will go through the regular path to the stack.

#### 8.2.2.3.4.13.1 Building XDP Kernel Samples

In order to use XDP programs from the kernel `bpf/samples` folder, these are the steps for building them natively:

1. Prerequisites:

Use a Layerscape board with LSDK1806 images, that has external network connectivity at least 6GB of disk space.

2. Install dependent packages:

```
apt-get install git
git apt-get install make
apt-get install gcc
apt-get install bc
apt-get install elfutils
apt-get install libelf-dev
apt-get install bison
apt-get install flex
apt-get install cmake
```

3. Build the latest version of LLVM and clang (required to be >= 7.0):

```
git clone https://git.llvm.org/git/llvm.git/ LLVM
cd LLVM/tools
git clone https://git.llvm.org/git/clang.git/
cd ../..
mkdir <llvm-build-dir>; cd <llvm-build-dir>
cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD="BPF" ../LLVM/
make -j 8
```

4. Download the 4.14 kernel sources from the LSDK release
5. Build the bpf samples:

```
cd <kernel-src-dir>
make mrproper
make defconfig; make lsdh.config
make headers_install
make samples/bpf/ LLVM=<llvm-build-dir>/bin/llc CLANG=<llvm-build-dir>/bin/clang
```

The resulting binaries will be located in <kernel-src-dir>/samples/bpf.

### 8.2.2.3.4.14 MQPRIO qdisc support

The DPAA2 Ethernet driver supports the MQPRIO qdisc, configurable through the tc tool.

MQPRIO (Multiqueue Priority Qdisc) is a simple queuing discipline that allows mapping traffic flows to hardware queue ranges, using priorities and a configurable priority to traffic class mapping. When creating the qdisc, the user can pass the number of traffic classes handled by the netdevice, the skb priority to traffic class map, and the hardware offloading flag.

For example:

```
$ tc qdisc add dev <ethX> root handle 1: mqprio num_tc 2 map 0 0 1 1 hw 1
```

The above translates to:

- The mqprio qdisc has 2 traffic classes (num\_tc 2)
- The qdisc depends on hw offloading (hw 1)
- The skb prio to traffic class map is as follows:
  - skb prio 0 -> tc 0
  - skb prio 1 -> tc 0
  - skb prio 2 -> tc 1
  - skb prio 3 -> tc 1

**Note:** We only support the hardware offloading mode. Setting the "hw" param to 0 is not supported.

For setting the skb priority, the clsact qdisc can be used. Then we use the u32 filter to assign the skb priority based on traffic flow characteristics. This requires a recent iproute2-tc with clsact support compiled in:

```
$ tc qdisc add dev <ethX> clsact
$ tc filter add dev <ethX> egress prio 1 u32 match ip dport 7776 0xffff action skbedit priority 0
$ tc filter add dev <ethX> egress prio 1 u32 match ip dport 7777 0xffff action skbedit priority 1
$ tc filter add dev <ethX> egress prio 1 u32 match ip dport 7778 0xffff action skbedit priority 2
$ tc filter add dev <ethX> egress prio 1 u32 match ip dport 7779 0xffff action skbedit priority 3
```

In the above example, the destination port is used to assign an skb priority level. Outgoing IPv4 frames with port id 7778 and 7779 will be treated with highest priority.

**Note:** In order to use tc mqprio with the DPAA2 Ethernet driver, make sure the following kernel options are enabled:

```
CONFIG_NET_SCHED=y
CONFIG_NET_SCH_MQPRIO=y
```

Also, in order to run above examples, the following kernel options are also needed:

```
CONFIG_NET_CLS=y
CONFIG_NET_CLS_ACT=y
CONFIG_NET_ACT_SKBEDIT=y
```

```
CONFIG_NET_CLS_U32=y
CONFIG_CLS_U32_PERF=y
CONFIG_CLS_U32_MARK=y
CONFIG_NET_EMATCH_U32=y
```

**Note:** The DPNI has to be configured with a number of traffic classes greater than one - the maximum supported is `num_tcs=8`. Make sure the `num_tc` parameter passed at `mqprio` qdisc creation is not higher than the number of traffic classes supported by the DPNI.

### 8.2.2.3.4.15 CEETM support

DPAA2 platforms offer scheduling, shaping and prioritization capabilities through CEETM (Customer Edge Egress Traffic Management). The purpose of the CEETM block is to enhance networking performances by moving the egress QoS logic from software to hardware.

This section briefly describes what is supported and how CEETM can be configured through the Linux traffic control tool (`tc`) by using a custom queuing discipline.

#### 8.2.2.3.4.15.1 Features

Each network interface (DPNI) can be associated with a LNI (logical network interface) containing a class queue channel. We don't support more than one channel per LNI. The LNI channel allows dual-rate shaping, which can be configured by specifying the CIR (committed information rate) and/or EIR (excess information rate). CBS (committed burst size) and EBS (excess burst size) values can also be configured.

We also support scheduling of class queues inside the channel; the number of queues cannot be larger than the configured number of traffic classes ("`num_tcs`" DPNI option), with a maximum value of 8.

Queues can be independent or part of a group:

- inside a group, queues are selected based on the WBFS (weighted bandwidth fair scheduling) algorithm. We support at most two class queue groups (referred to as group A and group B) with up to 4 queues each; if a single group is used (group A), up to 8 queues can be configured to be part of it.
- independent queues have fixed priorities and are subject to a strict priority scheduling (i.e. queue 1 will always be higher prio than queue 2)

Weighted queues share the priority of the group they belong to. Groups have configurable strict priorities relative to the independent queues. See the next section for an example on how to configure both weighted and independent queues.

We consider 0 to be the highest priority level.

#### 8.2.2.3.4.15.2 Prerequisites

In order to use the CEETM feature, it must first be enabled in the kernel config file:

```
CONFIG_FSL_DPAA2_ETH_CEETM=y
```

Also, the following kernel option is needed:

```
CONFIG_NET_SCHED=y
```

The CEETM TC library (`q_ceetm.so`) should be located under `/usr/lib/tc`. It is built and deployed by default, without any user action needed.

#### 8.2.2.3.4.15.3 Usage

You can see the `ceetm` qdisc's help message by running the following command:

```
$ tc qdisc add ceetm help
Usage:
```



```
... qdisc add ... ceetm type root
... class add ... ceetm type root [cir CIR] [eir EIR] [cbs CBS] [ebs EBS] [coupled C]
... qdisc add ... ceetm type prio [prioA PRIO] [prioB PRIO] [separate SEPARATE]
... class add ... ceetm type prio [mode MODE] [weight W]
Update configurations:
... class change ... ceetm type root [cir CIR] [eir EIR] [cbs CBS] [ebs EBS] [coupled C]

Qdisc types:
root - associate a LNI to the DPNI
prio - configure the LNI channel's Priority Scheduler with up to eight classes

Class types:
root - configure the LNI channel
prio - configure an independent or weighted class queue

Options:
CIR - the committed information rate of the LNI channel
 dual-rate shaper (required for shaping scenarios)
EIR - the excess information rate of the LNI channel
 dual-rate shaper (optional for shaping scenarios, default 0)
CBS - the committed burst size of the LNI channel
 dual-rate shaper (required for shaping scenarios)
EBS - the excess of the LNI channel
 dual-rate shaper (optional for shaping scenarios, default 0)
C - shaper coupled, if both CIR and EIR are finite, once the
 CR token bucket is full, additional CR tokens are instead
 added to the ER token bucket
PRIO - priority of the weighted group A / B of queues
SEPARATE - groups A and B are separate
MODE - scheduling mode of class queue, can be:
 STRICT_PRIORITY
 WEIGHTED_A
 WEIGHTED_B
W - the weight of the class queue in the weighted group
```

#### 8.2.2.3.4.15.4 Example

We present here an example of how tc ceetm qdisc can be used to create a complex egress shaping and scheduling configuration.

We start by configuring the LNI channel to allow a maximum egress rate of 1Gbps:

```
tc qdisc add dev <ethX> root handle 1: ceetm type root
tc class add dev <ethX> parent 1: classid 1:1 ceetm type root cir 1000mibit
```

We configure queue\_1 and queue\_2 to be part of group A, with a group priority of 3, and queue\_4 and queue\_5 to be part of group B with prio 1. Independent queues queue\_0 and queue\_3 are also configured. The resulting order of priorities is as follows (highest to lowest): {queue\_0, group\_B, queue\_3, group\_A}

Inside group A, queue\_1 and queue\_2 have equal weights; inside group B, queue\_5 is given three times more bandwidth than queue\_4. The weights are not absolute values, the relevant information is the ratio between them; it's recommended to use the value 100 for the queue with the lowest bandwidth.

```
tc qdisc add dev <ethX> parent 1:1 handle 2: ceetm type prio prioA 3 prioB 1 separate 1
tc class add dev <ethX> parent 2: classid 2:1 ceetm type prio mode STRICT_PRIORITY
tc class add dev <ethX> parent 2: classid 2:2 ceetm type prio mode WEIGHTED_A weight 100
tc class add dev <ethX> parent 2: classid 2:3 ceetm type prio mode WEIGHTED_A weight 100
tc class add dev <ethX> parent 2: classid 2:4 ceetm type prio mode STRICT_PRIORITY
```

```
tc class add dev <ethX> parent 2: classid 2:5 ceetm type prio mode WEIGHTED_B weight 100
tc class add dev <ethX> parent 2: classid 2:6 ceetm type prio mode WEIGHTED_B weight 300
```

Additionally, we define flows based on IP destination address and match them to the class queues:

```
Flow 1 - queue 0
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.2/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.2/32 flowid 2:1

Flow 2 - queue 1
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.3/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.3/32 flowid 2:2

Flow 3 - queue 2
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.4/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.4/32 flowid 2:3

Flow 4 - queue 3
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.5/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.5/32 flowid 2:4

Flow 5 - queue 4
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.6/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.6/32 flowid 2:5

Flow 6 - queue 5
tc filter add dev <ethX> parent 1: protocol ip u32 match ip dst 192.85.2.7/32 flowid 1:1
tc filter add dev <ethX> parent 2: protocol ip u32 match ip dst 192.85.2.7/32 flowid 2:6
```

Assuming the initial throughput of each flow was 200Mbps, the final output is:

```
flow 1 (queue_0) - 200Mbps
flow 2 (queue_1) - 100Mbps
flow 3 (queue_2) - 100Mbps
flow 4 (queue_3) - 200Mbps
flow 5 (queue_4) - 200Mbps
flow 6 (queue_5) - 200Mbps
```

If initial throughput per flow was 600Mbps, the final output is:

```
flow 1 (queue_0) - 600Mbps
flow 2 (queue_1) - 0Mbps
flow 3 (queue_2) - 0Mbps
flow 4 (queue_3) - 0Mbps
flow 5 (queue_4) - 100Mbps
flow 6 (queue_5) - 300Mbps
```

Note: In order to run this example, the following kernel configs are also needed:

```
CONFIG_NET_CLS=y
CONFIG_NET_CLS_ACT=y
CONFIG_NET_CLS_U32=y
CONFIG_NET_EMATCH=y
CONFIG_NET_EMATCH_U32=y
```

## 8.2.2.3.4.16 Interface statistics

DPAA2 Ethernet interface counters can be read via either of two standard tools, but there is a subtle difference:

- “`ifconfig ethX`” counters reflect packets received by the Ethernet driver – i.e. those frames that have passed through the Rx filters (if any are active) and have been effectively processed by the Ethernet driver on the GPP, and possibly by the kernel stack. These are software counters, maintained by the Ethernet driver and the networking stack.
- “`ethtool -S ethX`” counters reflect more detailed counters, from two categories:
  1. Statistics maintained by the DPAA2 hardware. These largely correspond in meaning to the standard “ifconfig” counters, but the values may be different from the “ifconfig” counters - i.e. they may reflect frames that have not been received on the GPP, such as those dropped by the ingress policer or due to MAC filtering. Also noteworthy is that retrieving these counters requires a series of calls into the MC firmware, which could make the operation potentially slower.
  2. Advanced counters, specific to the DPAA2 Ethernet driver. These are software-maintained driver-specific counters which do not fit into the standard “ifconfig” set.

The following detailed counters are presented by the ‘`ethtool -S`’ command:

a. Hardware-maintained counters (prefixed by the “[hw]” tag):

- `rx frames`: number of valid frames received from the DPNI hardware;
- `rx bytes`: number of bytes comprised within the “rx frames” counter;
- `rx mcast frames`: number of valid multicast frames;
- `rx mcast bytes`: number of bytes included in “rx mcast frames”;
- `rx bcast frames`: number of valid broadcast frames;
- `rx bcast bytes`: number of bytes included in “rx bcast frames”;
- `tx frames`: number of valid frames presented for transmission;
- `tx bytes`: number of bytes included in “tx frames”;
- `tx mcast frames`: number of valid egress multicast frames;
- `tx mcast bytes`: number of bytes included in “tx mcast frames”;
- `tx bcast frames`: number of valid egress broadcast frames;
- `tx bcast bytes`: number of bytes included in “tx bcast frames”;
- `rx filtered frames`: number of valid frames but dropped because e.g. of MAC filtering;
- `rx discarded frames`: number of frames with various physical errors;
- `rx nobuffer discards`: number of frames discarded due to lack of buffers;
- `tx discarded frames`: number of frames with Tx errors;
- `tx confirmed frames`: number of Tx confirmed frames

b. Software-maintained, driver-specific counters (prefixed by the “[sw]” tag):

- `tx conf frames`: number of frames presented back to the Ethernet driver in the Tx confirmation queues. In an idle system, this counter should be equal to “tx frames”;
- `tx conf bytes`: number of bytes comprised by the “tx conf frames” counter;
- `tx sg frames`: number of egress frames in scatter-gather format; these are a subset of “tx frames”, the difference being contiguous frames;
- `tx sg bytes`: number of bytes comprised in “tx sg frames”;
- `tx realloc frames`: number of frames which had to be reallocated in the driver due to insufficient skb headroom; if a significant number of Tx frames are realloc’ed, it may be an indicator of suboptimal networking performance
- `rx sg frames`: number of frames received in scatter-gather format; typically this reflects frames larger than the largest buffer that can be used at the time of reception;

- `rx_sg_bytes`: number of bytes comprised in "rx sg frames";
- `enqueue_portal_busy`: number of times the Ethernet driver had to retry the frame enqueue command (on the egress path) due to QBMan portal being busy;
- `dequeue_portal_busy`: number of times the Ethernet driver had to retry the frame dequeue command (on the ingress path) due to QBMan portal being busy;
- `channel_pull_errors`: number of dequeue errors which are not due to the portal being busy;
- `cdan`: number of Channel Dequeue Available Notifications (CDANs) received by the Ethernet driver (Rx and Tx Conf paths). Each CDAN corresponds to one DPIO interrupt and triggers a NAPI processing cycle which can process Rx or Tx Conf frames (or both).
- `tx_congestion_state`: whether the Tx queues are currently congested or not; if congestion state is 1, it means one or more Tx queues have stopped and are waiting for the hardware to finish transmitting the frames already enqueued from the Ethernet driver;

## 8.2.2.4 Setting up Ethernet Switch Capability

### 8.2.2.4.1 Ethernet Switch overview

The following switch features are supported:

- Dynamic learning
- Adding/deleting static FDB entries
- Adding/deleting static multicast entries
- Configuring multicast groups
- Flooding of broadcast and multicast traffic
- Forwarding of unicast traffic that is both VLAN tagged and untagged
- Setting STP state of ports

**Important notes:**

- Learning is supported and enabled by default.
- Learned FDB entries cannot be displayed.
- There is no support to flush the FDB.

**Acronyms and abbreviations:**

- DPSW - DPAA2 object modelling an L2 Switch
- DPNI - DPAA2 object modelling a network interface

### 8.2.2.4.2 Switch object creation

A switch object can be created:

- Dynamically using the restool as described in [Using restool for dynamic object creation](#) on page 352 or
- Statically in a DPL file as described in [Using the data path layout file \(DPL\)](#) on page 354

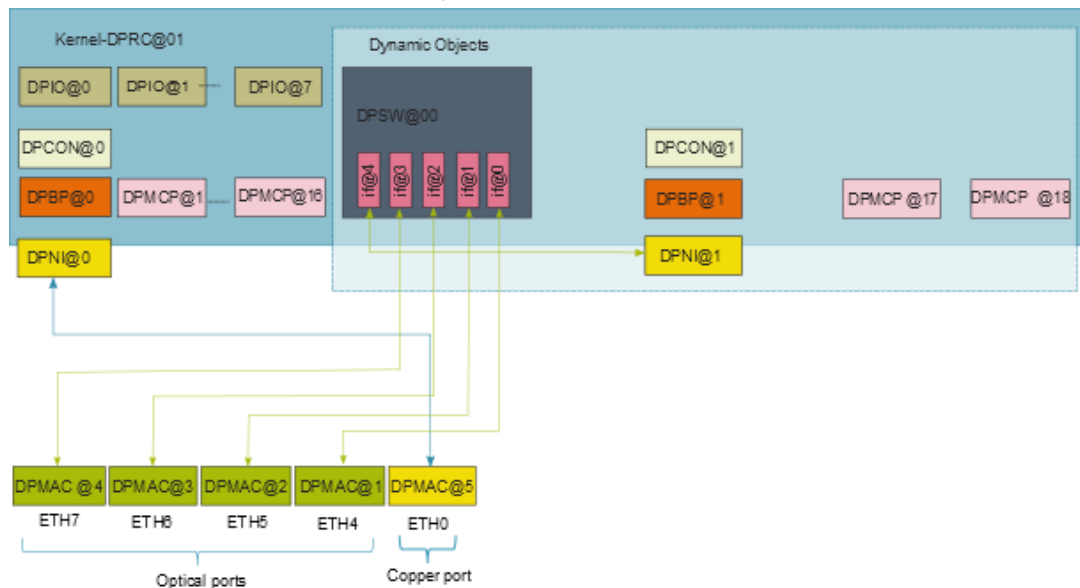
#### 8.2.2.4.2.1 Using restool for dynamic object creation

A switch can be created at run-time, using restool. Before creating the switch, a number of DPAA2 objects (dependencies) have to be added, for which multiple restool commands are needed. Switch requires at least a DPMCP object, which is created like:

```
$ restool dpmcp create
```

The following section describes the main commands to create a switch starting from the **dpl-eth.0x2A\_0x41.dtb** DPL file.

**Figure 54. Dynamic DPSW demo**



### NOTE

When a new object is created via `restool`, an object with the id of the first available resource will be returned.

## NOTE

Depending on the board type, DPMAC availability varies. For more details please refer to **Limitations and Known Issues**.

#### 8.2.2.4.2.1.1 *Creating a DPSW*

The switch object is created by this command:

```
$ restool dpsw create --num-ifs=5 --max-vlans=16 --max-fdbs=1 --options="DPSW_OPT_CTRL_IF_DIS"
```

The command specifies some configuration options, including the number of ports in the switch, the maximum number of VLANs that can be used on the switch – including VLAN1 used implicitly, plus the number of FDBs.

For all the configuration options and parameters see the help output:

```
$ restool dpsw create -h
```

Currently VLAN private FDBs are not supported; a single shared FDB is used. Also control traffic is not supported, so option `DPSW_OPT_CTRL_IF_DIS` has to be specified.

#### 8.2.2.4.2.1.2 Connecting the switch

Linking the switch ports to other objects is done with:

```
restool dprc connect "$RC" --endpoint1="$SW".0 --endpoint2=dpmac.1
restool dprc connect "$RC" --endpoint1="$SW".1 --endpoint2=dpmac.2
restool dprc connect "$RC" --endpoint1="$SW".2 --endpoint2=dpmac.3
restool dprc connect "$RC" --endpoint1="$SW".3 --endpoint2=dpmac.4
restool dprc connect "$RC" --endpoint1="$SW".4 --endpoint2="$NI"
```

In the context of the configuration script, these commands create the following layout:

- sw0p0 (dpsw@1/if@0) <-> dpmac@1 (SFP+ port, labeled ETH4 on RDB front panel)
- sw0p1 (dpsw@1/if@1) <-> dpmac@2 (SFP+ port, labeled ETH5 on RDB front panel)
- sw0p2 (dpsw@1/if@2) <-> dpmac@3 (SFP+ port, labeled ETH6 on RDB front panel)
- sw0p3 (dpsw@1/if@3) <-> dpmac@4 (SFP+ port, labeled ETH7 on RDB front panel)
- sw0p4 (dpsw@1/if@4) <-> dpni@1

For SerDes **0x2A\_0x41**, DPMACs 1-4 are mapped to the optical PHYs while DPMACs 5-8 are mapped to the copper PHYs. User can choose to connect any of the optical or copper ports and any NIs to the switch.

#### 8.2.2.4.2.1.3 Enabling the switch

This command plugs the switch object on the bus, in the Linux resource container. The switch driver probes the switch and presents the associated network interfaces in Linux.

```
$ restool dprc assign "$RC" --object="$SW" --plugged=1
```

After enabling the switch it can be configured from Linux using the commands specified in [Commands supported](#) on page 356.

#### 8.2.2.4.2.1.4 Restool wrapper scripts

For user convenience the **ls-addsw** script is provided to assist creation of a new DPSW object.

To replicate the setup described in section [Connecting the switch](#) on page 353 the following commands are required:

```
$ ls-addni -n
$ ls-addsw -i=5 dpmac.3 dpmac.4 dpmac.5 dpmac.6 dpni.1
```

The first command creates a new DPNI object and the second the DPSW object. DPMACs are already defined in the DPL file. To display the DPNI and DPMACs available one can use *ls-listni* or *ls-listmac* commands. The DPIO, DPBP, DPCON and DPMCP objects that are dependencies for the new objects are created by the script, without user intervention.

For all the script options and parameters see the help:

```
$ ls-addsw -h
```

The endpoints are connected in the specified order to switch ports. If there are less endpoints than the number of interfaces, the user can later add the rest using *ls-addni* or *restool* commands.

#### 8.2.2.4.2.2 Using the data path layout file (DPL)

A switch object may be defined statically in the DPL, allowing it to be created automatically during platform initialization. Below is an example of switch definition in the DPL:

```
dpsw@0 {
 compatible = "fsl,dpsw";
 options = "DPSW_OPT_CTRL_IF_DIS";
 max_vlans = <0x10>;
 max_fdb = <0x1>;
 num_fdb_entries = <0x400>;
 fdb_aging_time = <0x12c>;
 num_ifs = <0x5>;
 max_fdb_mc_groups = <32>;
};
```

This example is for a 5-port switch that includes support for up to 16 VLAN IDs, including VLAN 1 (that is internally used by the switch), up to 1024 FDB entries, and up to 32 multicast groups.

Links are defined in the DPL 'connections' section as follows:

```
connections {
 connection@1 {
 endpoint1 = "dpsw@0/if@0";
 endpoint2 = "dpmac@1";
 };
 connection@2 {
 endpoint1 = "dpsw@0/if@1";
 endpoint2 = "dpmac@2";
 };
 connection@3 {
 endpoint1 = "dpsw@0/if@2";
 endpoint2 = "dpmac@3";
 };
 connection@4 {
 endpoint1 = "dpsw@0/if@3";
 endpoint2 = "dpmac@4";
 };
 connection@5 {
 endpoint1 = "dpsw@0/if@4";
 endpoint2 = "dpni@1";
 };
};
```

The generated layout is the one described in [Connecting the switch](#) on page 353.

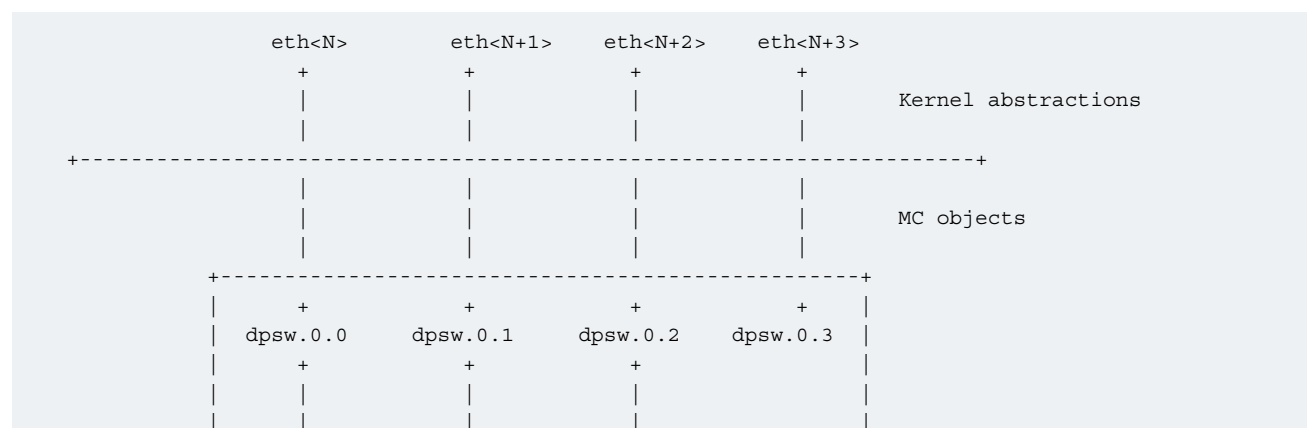
### 8.2.2.4.3 Setting up the driver

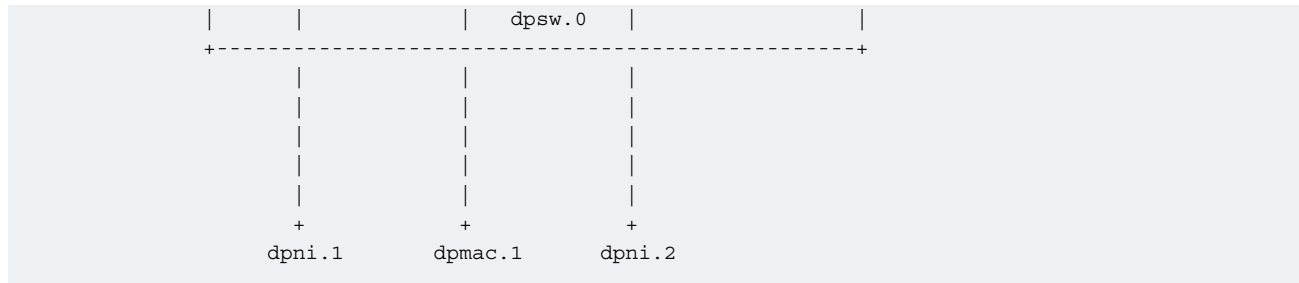
To compile the driver, you enable the **FSL\_DPAA2\_ETHSW** option in the kernel's config. It can be found in *menuconfig* under the following items:

```
| -> Device Drivers
| -> Staging drivers
| -> Freescale DPAA2 devices
| -> Freescale DPAA2 Ethernet Switch
```

The driver is enabled in the default kernel configuration file and uses the *switchdev* kernel support, thus it depends on the **NET\_SWITCHDEV** option. To have access to the kernel interfaces for configuring bridges, **BRIDGE** kernel option has to be enabled.

After deploying the driver and instantiating a DPSW, the system will present a network interface for each switch port. These are used for control, not for I/O. Any I/O through the switch must be performed using the interfaces linked to the switch.





To be able to use the switch, network interfaces have to be configured in a bridge using standard Linux tools like those in iproute2 suite.

### Configuring the switch in Linux

#### 1. Create a bridge interface

After the DPAA2 Ethernet Switch is created, network interfaces for each switch port are available. They need to be added to a bridge interface:

```
ip link add name <brX> type bridge
ip link set <brX> up
```

#### 2. Configuring MAC addresses

To be able to configure the ports, a unique MAC address has to be configured on each interface:

```
ip link set eth<N> down
ip link set eth<N> address xx:xx:xx:xx:xx:xx
ip link set eth<N> up
```

#### 3. Adding switch ports to bridge interface

Switch ports that are not added to the bridge interface cannot be used to forward traffic.

```
ip link set eth<N> master <brX>
```

## 8.2.2.4.4 Commands supported

The switch management commands supported are:

- ifup/ifdown (using ifconfig or similar)
- setting large frame size limit (using ifconfig or similar)
- retrieving statistics (using ifconfig or similar)
- configuring FDB (using bridge fdb)
- configuring multicast groups (using bridge fdb)
- configuring VLANs (using bridge vlan)
- configuring learning (using bridge link set)

### 8.2.2.4.4.1 Interface control

Any of the switch ports can be enabled/disabled using any of the following commands:

```
$ ifconfig eth<N> { up | down }
$ ip link set eth<N> { up | down }
```

Disabling the bridge device will also disable the assigned switch ports.



### 8.2.2.4.4.2 Maximum frame size configuration

The DPAA2 hardware supports large frames. Ethernet switch driver correlates between the Layer-2 maximum frame length (MFL) and Layer-3 MTUs. The maximum MTU that a Linux user can request on a DPAA2 Ethernet switch interface is 10218 bytes and has to be set on each port individually.

```
$ ifconfig eth<N> mtu <NN>
$ ip link set { eth<N> | dev eth<N> } mtu <NN>
```

#### Notes:

- Frames larger than the configured MTU will be dropped, so connected Ethernet devices need to have the same setting.
- All Ethernet devices on the same LAN must have the same MTU to avoid traffic loss.

### 8.2.2.4.4.3 Learning control

The switch is set by default to enable learning, and the learning can be controlled using this command:

```
$ bridge link set dev eth<N> learning { on | off }
```

#### NOTE

The command is executed on a switch port, although it does affect the learning function at switch level. Turning off learning does not remove the learned entries.

To establish a static topology, learning should be disabled before injecting any traffic.

### 8.2.2.4.4.4 FDB static entries

The default switch configuration does not include any static entries; these can be added using the *bridge fdb add* command, as shown below:

```
$ bridge fdb add xx:xx:xx:xx:xx:xx dev eth<N> master
```

### 8.2.2.4.4.5 Multicast entries

Multicast groups can be configured via *bridge fdb add|append|delete* commands:

```
$ bridge fdb add 01:00:05:00:00:13 dev eth6
$ bridge fdb append 01:00:05:00:00:13 dev eth7
$ bridge fdb append 01:00:05:00:00:13 dev eth8
$ bridge fdb append 01:00:05:00:00:13 dev eth9

$ bridge fdb del 01:00:05:00:00:13 dev eth9
```

The commands add all external ports, one by one, to the 01:00:05:00:00:13 multicast group. The last command removes the last port from the group.

### 8.2.2.4.4.6 VLAN configuration

All ports added to a bridge are added to the default VLAN. All untagged traffic received on switch ports is classified to VLAN 1, and all frames classified in VLAN 1 are sent out untagged on all ports. Additional VLANs can be added on the switch using these commands:

```
$ bridge vlan add vid 2 dev eth0
$ bridge vlan add vid 2 dev eth1
$ bridge vlan add vid 2 dev eth2
```

```
$ bridge vlan add vid 2 dev eth3
$ bridge vlan add vid 2 dev eth4
```

This example includes all five ports in VLAN 2. After running these commands the switch should allow frames tagged with vid 2 to pass through the switch; not all ports have to be included in the VLAN.

To remove a port from a given VLAN use the following command:

```
$ bridge vlan del vid 2 dev eth0
```

#### 8.2.2.4.4.7 Port statistics

Hardware counters for switch ports are available through `ethtool`:

```
$ ethtool -S eth<N>
NIC statistics:
 rx frames: 0
 rx bytes: 0
 rx filtered frames: 0
 rx discarded frames: 0
 rx b-cast frames: 0
 rx b-cast bytes: 0
 rx m-cast frames: 0
 rx m-cast bytes: 0
 tx frames: 0
 tx bytes: 0
 tx discarded frames: 0
```

No software statistics are available for switch ports.

### 8.2.2.5 Setting Up Edge Virtual Bridge Capability

#### 8.2.2.5.1 EVB overview

An edge virtual bridge allows the sharing of a physical connection between multiple entities (virtual hosts). It can act as a VEB or as a VEPA.

In VEB mode, traffic is forwarded between connected virtual hosts or between virtual hosts and uplink.

In VEPA mode, all traffic from virtual hosts is forwarded to uplink, bridging functions (including 'hairpin' forwarding) being performed by an external device.

##### Features supported:

- VEB/VEPA mode
- Traffic steering according to MAC, VLAN (in VEPA mode only) or MAC+VLAN
- Static FDB entries management (add/delete/show)
- Static multicast FDB entries management (add/delete/show)
- Flooding of broadcast and multicast traffic

#### 8.2.2.5.2 EVB object creation

EVB objects can be created as follows:

- Dynamically using the restool as described in [Using restool for dynamic object creation](#) on page 359
- Statically in a DPL file as described in [Using the data path layout file \(DPL\)](#) on page 360

### 8.2.2.5.2.1 Using restool for dynamic object creation

A DPDMUX can be instantiated at run-time, using restool.

The following section describes the main commands to create an EVB and its dependencies starting from the **dpl-eth.0x2A\_0x41.dtb** DPL file.

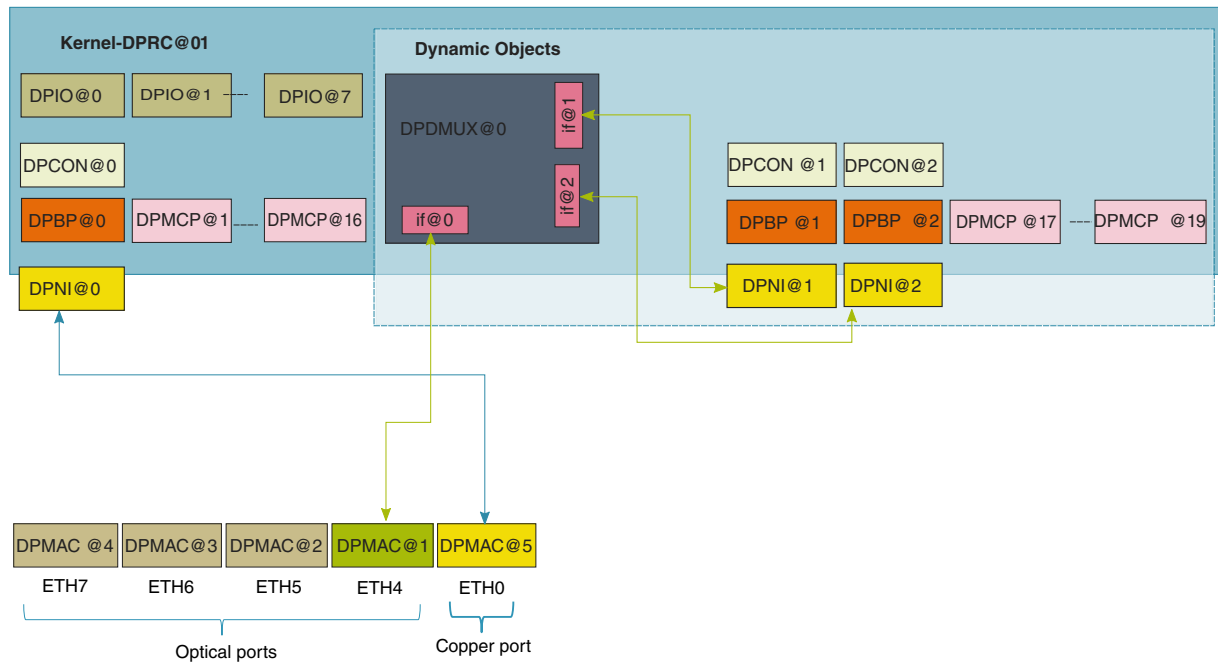


Figure 55. Dynamic DPDMUX demo

#### NOTE

When a new object is created using restool, an object with the ID of the first available resource is returned.

#### NOTE

Depending on the board type, DPMAC availability varies. For more details please refer to **Limitations and Known Issues**.

#### 8.2.2.5.2.1.1 Creating a DPDMUX

The EVB is created by this command:

```
$ restool dpdmux create --num-ifs=2 --control-if=0 \
 --options=DPDMUX_OPT_BRIDGE_EN --method=DPDMUX_METHOD_MAC \
 --max-dmat-entries=8 --max-mc-groups=8 --manip=DPDMUX_MANIP_NONE
```

The command must specify the number of downlinks and the ID of the uplink (ranges for 0 to [number of downlinks - 1]). The other parameters are optional. For more information about the available options see the output of the command:

```
$ restool dpdmux create -h
```

#### 8.2.2.5.2.1.2 **Connecting the EVB**

Linking the EVB ports to other objects is done with:

```
$ restool dprc connect "$RC" --endpoint1="$MUX".0 --endpoint2="$MAC"
$ restool dprc connect "$RC" --endpoint1="$MUX".1 --endpoint2="$NI1"
$ restool dprc connect "$RC" --endpoint1="$MUX".2 --endpoint2="$NI2"
```

\$RC represents the container for the objects, \$MUX is the object identified for the EVB. The uplink is the endpoint with the id specified by *control-if* parameter at creation time.

#### 8.2.2.5.2.1.3 **Enabling the EVB**

This command plugs the EVB object on the bus, in the Linux resource container. The EVB driver probes the switch and presents the associated network interfaces in Linux.

```
$ restool dprc assign "$RC" --object="$MUX" --plugged=1
```

#### 8.2.2.5.2.1.4 **Restool wrapper scripts**

For user convenience the **ls-addmux** script is provided to assist creation of a new DPDMUX object.

Example to replicate setup in section [Connecting the EVB](#) on page 360 :

```
ls-addmux -d=2 -u=0 dpmac.1
[4298.023745] dpaa2_evb dpdmux.0: probed evb device with 2 ports
Created EVB: evb0 (object: dpdmux.0, uplink: dpmac.1)
```

This command creates EVB evb0 (and the corresponding dpdmux.0 object) with two downlinks and the uplink connected to dpmac.1.

After creating the DPDMUX, its downlinks can be connected to DPNI's using **ls-addni** script:

```
ls-addni dpdmux.0.1
Will allocate 8 DPCON objects for this hash size
[5118.645253] fsl_dpaa2_eth dpni.1: Probed interface ni1
Created interface: ni1 (object:dpni.1, endpoint: dpdmux.0.1)
ls-addni dpdmux.0.2
Will allocate 8 DPCON objects for this hash size
[5122.169030] fsl_dpaa2_eth dpni.2: Probed interface ni2
Created interface: ni2 (object:dpni.2, endpoint: dpdmux.0.2)
```

#### 8.2.2.5.2.2 **Using the data path layout file (DPL)**

A DPDMUX instance can statically be defined in the DPL file:

```
dpdmux@0 {
 compatible = "fsl,dpdmux";
 options = "DPDMUX_OPT_BRIDGE_EN";
 method = "DPDMUX_METHOD_MAC";
 manip = "DPDMUX_MANIP_NONE";
 control_if = <0>;
 num_ifs = <2>;
}
```

```
max_dmat_entries = <8>;
max_mc_groups = <8>;
};
```

Links are defined in the DPL 'connections' section:

```
connection@1{
 endpoint1 = "dpdmux@0/if@0";
 endpoint2 = "dpmac@1";
};
connection@2{
 endpoint1 = "dpni@1";
 endpoint2 = "dpdmux@0/if@1";
};
connection@3{
 endpoint1 = "dpni@2";
 endpoint2 = "dpdmux@0/if@2";
};
```

Based on the above configuration the DPDMUX ports are linked to:

- evb0 (dpdmux@1/if@0) <-> dpmac@1
- evb0p0 (dpdmux@1/if@1) <-> dpni@1
- evb0p1 (dpdmux@1/if@2) <-> dpni@2

#### NOTE

DPDMUX ports connected to a DPMAC must be configured before the others (e.g. connected to DPNI's).

### 8.2.2.5.3 Setting up the EVB driver

Driver compilation is enabled by default and is controlled by the FSL\_DPAA2\_EVB option in the kernel's config. This can be found in *menuconfig* under the following items:

```
| -> Device Drivers
| -> Staging drivers
| -> Freescale Management Complex (MC) bus driver
| -> Freescale DPAA2 devices
| -> DPAA2 Edge Virtual Bridge
```

The kernel log will display a message when an EVB is probed as follows:

```
dpaa2_evb dpdmux.0: probed evb device with 2 ports
```

After deploying the driver and configuring an EVB (via DLP or restool), the system should present the following Linux interfaces after typing the 'ifconfig command':

```
evb0 Link encap:Ethernet HWaddr 00:00:00:00:00:00
 UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

evb0p0 Link encap:Ethernet HWaddr 00:00:00:00:00:00
 UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

evb0p1 Link encap:Ethernet HWaddr 00:00:00:00:00:00
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Interface **evb0** represents the uplink and is also the handler for the EVB. Each other EVB port has its own interface. They are used for management and cannot be used for I/O. Any I/O through the EVB must be performed using the connected interfaces.

An EVB only forwards traffic to links that are enabled (peer interface is up) and only if the filtering rules on the peer interface do not lead to the frame being discarded. One way to ensure that all traffic subject to forwarding rules is actually forwarded by the EVB is to set the peer interface in promiscuous mode, as follows:

```
$ ip link set ni0 up promisc on
```

## 8.2.2.5.4 EVB commands supported

EVB management can be performed using the following generic Linux networking tools:

- interface up/down (using ifconfig or similar)
- setting large frame size limit (using ifconfig or similar)
- configuring FDB (using bridge fdb)
- configuring VLANs (using bridge vlan)
- configuring multicast groups (using bridge fdb)
- port statistics retrieval (ethtool or similar)

### 8.2.2.5.4.1 EVB interface control

Any of the EVB ports, or the EVB as a whole, can be enabled/disabled using any of the following commands:

```
$ ifconfig { evb0pX | evb0 } { up | down }
$ ip link set { evb0pX | evb0 } { up | down }
```

### 8.2.2.5.4.2 Maximum frame size configuration

The DPAA2 hardware supports large frames. EVB driver correlates between the Layer-2 maximum frame length (MFL) and Layer-3 MTUs. The maximum MTU that a Linux user can request on a DPAA2 EVB interface is 10222 bytes. Setting a value on a downlink port or uplink will update the value for all EVB interfaces.

```
$ ifconfig { evbX | evbXpY } mtu <NN>
$ ip link set { evbX | evbXpY | dev evbXpY } mtu <NN>
```

#### Notes:

- Frames larger than the configured MTU will be dropped, so connected Ethernet devices need to have the same setting.
- All Ethernet devices on the same LAN must have the same MTU to avoid traffic loss.

### 8.2.2.5.4.3 EVB FDB entries

The EVB method DPDMUX\_METHOD\_MAC allows configuration of FDB entries via a bridge utility as follows:

```
$ bridge fdb add 02:00:c0:a8:50:01 dev evb0p0
$ bridge fdb show
02:00:c0:a8:50:01 self permanent
01:00:5e:00:00:01 self permanent
```

The EVB method DPDMUX\_METHOD\_C\_VLAN\_MAC also allows configuration of FDB entries via a bridge utility as follows:

```
$ bridge fdb add 02:00:c0:a8:50:02 vlan 10 dev evb0p0 vlan 10
$ bridge fdb show dev evb0p0
02:00:c0:a8:50:02 self permanent
01:00:5e:00:00:01 self permanent
```

### 8.2.2.5.4.4 EVB VLAN assignment

The EVB method DPDMUX\_METHOD\_C\_VLAN allows port VLAN assignment via a bridge utility as follows:

```
$ bridge vlan add vid 10 dev evb0p2
$ bridge vlan show dev evb0p2
port vlan ids
evb0p2 10
$ bridge vlan del vid 10 dev evb0p2
```

#### NOTE

This method is allowed only for VEPA mode.

### 8.2.2.5.4.5 EVB port statistics

EVB port statistics are available through *ip* or similar tools as follows:

```
$ ip -s link
[...]
9: evb0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN mode
DEFAULT group default qlen 1000
 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
 RX: bytes packets errors dropped overrun mcast
 0 0 0 0 0 0
 TX: bytes packets errors dropped carrier collsns
 384 6 0 0 0 0
10: evb0p0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master evb0 state
UNKNOWN mode DEFAULT group default qlen 1000
 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
 RX: bytes packets errors dropped overrun mcast
 252 6 0 0 0 0
 TX: bytes packets errors dropped carrier collsns
 0 0 0 0 0 0
[...]
```

### 8.2.2.5.5 Forwarding methods overview

A DPAA2 DPDMUX instance can forward traffic using information from various fields in the frame headers:

- Forwarding by destination MAC address

- Forwarding by VLAN tag
- Forwarding by VLAN tag and destination MAC address

### 8.2.2.5.5.1 Forwarding by destination MAC address

This method forwards frames according to the destination MAC address and the static rules added into the EVB forwarding database.

It is configured specifying `--method="DPDMUX_METHOD_MAC"` when the DPDMUX is created. It is the default value for the `ls-addmux` script.

Entries are configured in the FDB using `bridge fdb` command. See [EVB FDB entries](#) on page 363 section for more information.

#### Configuration example:

```
Create a MUX with 2 downlinks and uplink connected to dpmac.1;
forwarding method is by default DPDMUX_METHOD_MAC
$ ls-addmux -b -d=2 -u=0 dpmac.1

Create a ni (dpni.1) and links it to evb0p0
$ ls-addni dpdmux.0.1

Create a ni (dpni.2) and links it to evb0p1
$ ls-addni dpdmux.0.2

Check MUX configuration
$ restool dpdmux info dpdmux.0

Configure ni1
$ ip netns add ns1
$ ip link set ni1 netns ns1
$ ip netns exec ns1 ifconfig ni1 192.168.10.10/24 up
$ ip netns exec ns1 ip link set ni1 promisc on

Configure ni2
$ ip netns add ns2
$ ip link set ni2 netns ns2
$ ip netns exec ns2 ifconfig ni2 192.168.10.12/24 up
$ ip netns exec ns2 ip link set ni2 promisc on

Connectivity checks [downlink - uplink]
$ ip netns exec ns1 ping 192.168.10.13 -c 1
[...]
```

```
--- 192.168.10.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms

Check EVB port statistics
$ ip -s link
[...]
```

```
4: evb0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN mode
DEFAULT group default qlen 1000
 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
 RX: bytes packets errors dropped overrun mcast
 436 6 0 0 0 0
 TX: bytes packets errors dropped carrier collsns
 460 6 0 0 0 0

5: evb0p0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master evb0 state UP
mode DEFAULT group default qlen 1000
 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
 RX: bytes packets errors dropped overrun mcast
 364 6 0 0 0 0
```



```

TX: bytes packets errors dropped carrier collsns
376 5 0 0 0 0
6: evb0p1: <NO-CARRIER,BROADCAST,MULTICAST,SLAVE,UP> mtu 1500 qdisc pfifo_fast master evb0 state
DOWN mode DEFAULT group default qlen 1000
 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
RX: bytes packets errors dropped overrun mcast
0 0 0 0 0 0
TX: bytes packets errors dropped carrier collsns
0 0 0 0 0 0

```

### 8.2.2.5.2 Forwarding by VLAN tag

This method forwards frames according to the VLAN tag of the frame, as set into the customer tag of the double-tagged frames.

It is configured specifying `--method="DPDMUX_METHOD_C_VLAN"` when EVB is in VEPA mode (`--options="DPDMUX_OPT_BRIDGE_EN"` is not set).

EVB port VLAN assignment is done with `bridge vlan` command. See [EVB VLAN assignment](#) on page 363 section for more information.

#### Configuration example:

```

Create a MUX with DPDMUX_METHOD_C_VLAN forwarding method,
configured as a VEPA and with 2 downlinks and uplink connected
to dpmac.1
$ ls-addmux -v -m=DPDMUX_METHOD_C_VLAN -d=2 dpmac.1

Create a ni (dpni.1) and link it to evb0p0
$ ls-addni dpdmux.0.1

Create a ni (dpni.2) and link it to evb0p1
$ ls-addni dpdmux.0.2

Configure ni1
$ ip netns add ns1
$ ip link set ni1 netns ns1
$ ip netns exec ns1 ip link add link ni1 name ni1.6 type vlan id 6
$ ip netns exec ns1 ifconfig ni1.6 192.168.6.10
$ ip netns exec ns1 ip link set ni1 up
$ ip netns exec ns1 ip link set ni1 promisc on

Configure ni2
$ ip netns add ns2
$ ip link set ni2 netns ns2
$ ip netns exec ns2 ip link add link ni2 name ni2.7 type vlan id 7
$ ip netns exec ns2 ifconfig ni2.7 192.168.7.12
$ ip netns exec ns2 ip link set ni2 up
$ ip netns exec ns2 ip link set ni2 promisc on

For the downlinks interfaces also add the VLAN ids
$ bridge vlan add vid 6 dev evb0p0
$ bridge vlan add vid 7 dev evb0p1

Connectivity checkings [example for downlink - uplink]
$ ip netns exec ns1 ping -I ni1.6 192.168.6.13 -c 1

Check VLAN assignment
$ bridge vlan show

```

### 8.2.2.5.3 Forwarding by VLAN tag and destination MAC address

This method forwards frames according to the VLAN tag and the destination MAC address of the frame .

It is configured specifying `--method="DPDMUX_METHOD_C_VLAN_MAC"` when the DPDMUX is created.

Entries are configured in the FDB using *bridge fdb* command. See [EVB FDB entries](#) on page 363 section for more information.

#### Configuration example:

```
Create a MUX with DPDMUX_METHOD_C_VLAN_MAC forwarding method,
configured as a VEB and with 2 downlinks and uplink connected
to dpmac.1
$ ls-addmux -m=DPDMUX_METHOD_C_VLAN_MAC -d=2 dpmac.1

Create a ni (dpni.1) and link it to evb0p0
$ ls-addni dpdmux.0.1

Create a ni (dpni.2) and link it to evb0p1
$ ls-addni dpdmux.0.2

Configure ni1
$ ip netns add ns1
$ ip link set ni1 netns ns1
$ ip netns exec ns1 ip link add link ni1 name ni1.6 type vlan id 6
$ ip netns exec ns1 ifconfig ni1.6 192.168.6.10
$ ip netns exec ns1 ip link set ni1 up
$ ip netns exec ns1 ip link set ni1 promisc on

Configure ni2
$ ip netns add ns2
$ ip link set ni2 netns ns2
$ ip netns exec ns2 ip link add link ni2 name ni2.7 type vlan id 7
$ ip netns exec ns2 ifconfig ni2.7 192.168.7.12
$ ip netns exec ns2 ip link set ni2 up
$ ip netns exec ns2 ip link set ni2 promisc on

For the downlinks interfaces, you would also need to add
the downlinks MACs to fdb table
$ bridge fdb add 4a:64:0a:af:14:a2 dev evb0p0 vlan 6
$ bridge fdb add 62:9c:86:0f:f7:cf dev evb0p1 vlan 7

Connectivity checkings [example for downlink - uplink]
$ ip netns exec ns1 ping -I ni1.6 192.168.6.13 -c 1

Check EVB FDB entries
$ bridge fdb show
```

## 8.2.2.6 Security Engine (SEC)

This section describes the software for the SEC hardware block that is part of the DPAA2 family of SoCs.

### Introduction

Current chapter is focused on DPAA2-specific SEC details - Data Path SEC Interface (DPSECI) backend and frontend drivers.

- JRI - the common Job Ring Interface (on which QI is currently dependent)
- crypto algorithms supported by each backend (RI, JRI, QI, DPSECI)
- kernel configuration - how to build backend and frontend drivers
- how to make sure the algorithms registered successfully
- how to check that crypto requests are being offloaded on SEC engine

On SoCs with DPAA v2.x, DPSECI backend can be used to submit crypto API service requests from the frontend drivers. The corresponding frontend compatible with DPSECI backend is *caamalg\_qi2*, which supports symmetric encryption and AEAD algorithms-based crypto API service requests.

The Linux driver automatically sets the enable bit for the SEC hardware's Queue Interface (QI), depending on QI feature availability in the hardware. This enables the hardware to also operate as a DPAA component for use by e.g., USDPAA apps.

This behaviour does not conflict with normal in-kernel job ring operation, other than the potential performance-observable effects of internal SEC hardware resource contention, and vice-versa.

## Module loading

The DPSECI backend driver (*dpseci*) is compiled built-in, while the DPSECI frontend driver (*dpaa2\_caam*) is compiled, by default, as module (though it can also be compiled built-in). In this case, it has to be probed before dynamically creating *dpseci* objects with *restool*:

```
$ modprobe dpaa2_caam
```

Without any parameter, the *dpseci* object being created has 2 pairs of (rx,tx) queues.

```
$ restool dpseci create
$ restool dprc assign dprc.1 --object=dpseci.0 --plugged=1
```

To create 8 (maximum) number of queues:

```
$ restool dpseci create --num-queues=8 --priorities=1,2,3,4,5,6,7,8
$ restool dprc assign dprc.1 --object=dpseci.0 --plugged=1
```

More options can be displayed by using:

```
$ restool dpseci create --help
```

The list of algorithms registered by the *dpaa2\_caam* driver is available in */proc* filesystem:

```
$ grep caam-qi2 /proc/crypto
```

## Enabling congestion management

Congestion management can be enabled when working with a MC that has a DPSECI object version greater or equal to 5.1. The first MC firmware version that supports the congestion management feature is 10.2. Enabling congestion management is done when creating the DPSECI object:

```
$ restool dpseci create --num-queues=8 --priorities=1,2,3,4,5,6,7,8 --options="DPSECI_OPT_HAS_CG"
$ restool dprc assign dprc.1 --object=dpseci.0 --plugged=1
```

## Source files

The driver source files are maintained in the Linux kernel source tree: *drivers/crypto/caam*.

## How to test the driver

To test the driver, in the kernel configuration menu, under "Cryptographic API -> Cryptographic algorithm manager", ensure that run-time self-tests are not disabled, i.e. the "Disable run-time self tests" entry is not set (*CONFIG\_CRYPTO\_MANAGER\_DISABLE\_TESTS=n*). This will run standard test vectors against the driver after the driver registers its supported algorithms with the kernel crypto API. To verify if the 'selftest' fields have 'passed', the */proc/crypto* entries should be checked. An entry such as this:

```
name : cbc(aes)
driver : cbc-aes-caam-qi2
module : kernel
priority : 2000
refcnt : 1
selftest : passed
internal : no
```

```
type : givcipher
async : yes
blocksize : 16
min keysize : 16
max keysize : 32
```

means the driver has successfully registered support for the algorithm with the kernel crypto API. Note that although a test vector may not exist for a particular algorithm supported by the driver, the kernel will emit messages saying which algorithms weren't tested, and mark them as *passed* anyway. The driver's capabilities can also be tested with tcrypt testing framework available in linux kernel by selecting "Cryptographic API -> Testing module" (also Disable run-time self tests should be unchecked). A kernel module will be generated: *crypto/tcrypt.ko*. This has to be copied on the target. Then on target, after a dpseci object is registered:

```
$ insmod tcrypt.ko mode=10
```

Other ways to test with tcrypt:

- functional testing:
  - mode=3, 4, 35, 150, 152, 155, 181-191;
  - alg="algorithm\_name"
- speed (sec - seconds parameter is optional):
  - mode=500 [sec=1] - xxx(aes) acipher\_speed
  - mode=501 [sec=1] - xxx(3des) acipher\_speed
  - mode=502 [sec=1] - xxx(des) acipher\_speed etc.

There is no need to rmmod, tcrypt does not stay "resident", it exits after running the tests. That's why you'll see:

```
insmod: ERROR: could not insert module tcrypt.ko: Resource temporarily
```

For algorithms not supported, errors like below will be shown:

```
[2650.067737] failed to load transform for rmd128: -2
[2650.076480] failed to load transform for rmd160: -2
[2650.085099] failed to load transform for rmd256: -2
[2650.093739] failed to load transform for rmd320: -2
```

These are expected. Algorithm names registered by dpaa2\_caam frontend driver are ending in "-caam-qi2".

To verify the operation and correctness of the driver, other than noting the performance advantages due to the crypto offload, one can also ensure the h/w is doing the crypto by looking for driver messages in `dmesg`. The driver emits console messages at initialization time:

```
$ dmesg | grep dpaa2_caam
[1172.598591] dpaa2_caam dpseci.0: Opened dpseci object successfully
[1172.619979] dpaa2_caam dpseci.0: prio 0: rx queue 135, tx queue 119
[1172.626633] dpaa2_caam dpseci.0: prio 1: rx queue 136, tx queue 128
[1172.633278] dpaa2_caam dpseci.0: prio 2: rx queue 137, tx queue 129
[1172.639915] dpaa2_caam dpseci.0: prio 3: rx queue 138, tx queue 130
[1172.646555] dpaa2_caam dpseci.0: prio 4: rx queue 139, tx queue 131
[1172.653195] dpaa2_caam dpseci.0: prio 5: rx queue 140, tx queue 132
[1172.659831] dpaa2_caam dpseci.0: prio 6: rx queue 141, tx queue 133
[1172.666470] dpaa2_caam dpseci.0: prio 7: rx queue 142, tx queue 134
[1172.694319] dpaa2_caam dpseci.0: DPSECI version 3.0
[1172.700617] dpaa2_caam dpseci.0: algorithms registered in /proc/crypto
```

Given a time period when crypto requests are being made, the SEC h/w will fire completion notification interrupts:

```
$ cat /proc/interrupts | grep DP10
```

If the number of interrupts fired increment, then the h/w is being used to do the crypto. If the numbers do not increment, then check if the algorithm being exercised is supported by the driver.

## Running OpenSSL

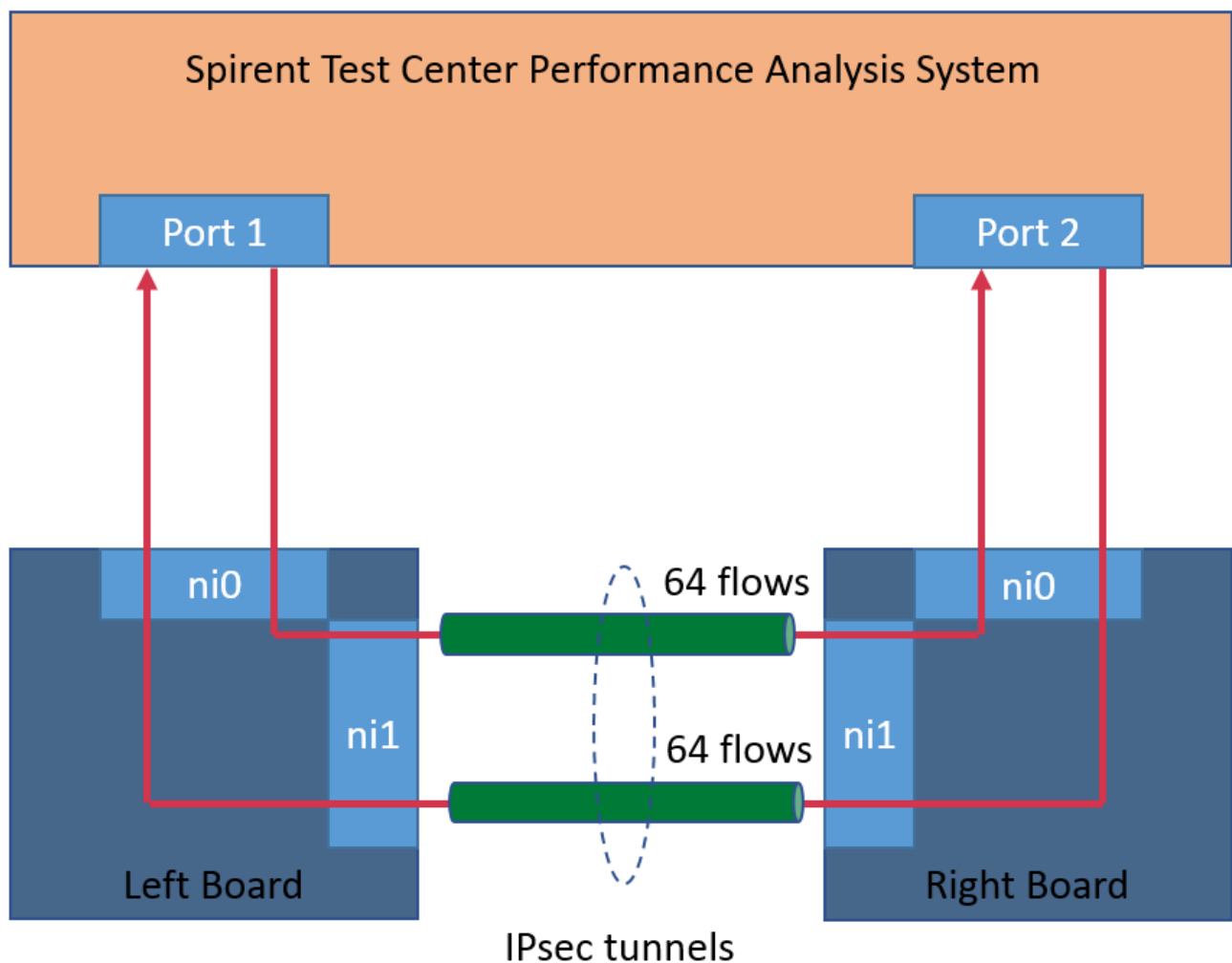
Some of the OpenSSL cryptographic operations (for e.g. TLS 1.0 record layer encryption, some non-protocol-specific crypto algorithms) can be offloaded to Linux kernel (and then further to SEC crypto engine) via cryptodev module.

Please refer to [Hardware Offloading with OpenSSL](#) on page 466 chapter for more details.

## >Running IPsec

### Setup Description

IPsec can be configured and used for NXP boards taking advantage of the cryptographic acceleration provided by the CAAM engine. Below is the description of the setup used to test IPsec traffic between two LS2088ARDB boards.



Traffic is generated from the Test Center on Port 1 as 64 flows. A flow is defined as a stream of packets that has a unique pair of values for IP source and IP destination. In our configuration the IP source ranges from 192.85.1.2 to 192.85.1.9 and the IP destination ranges from 192.86.1.2 to 192.86.1.9. The flows are received on the network interface ni0 of the left board, encapsulated and then sent over the ni1 network interface to the right board. Here the flows are decapsulated and routed to

the network interface ni0 towards the Port 2 of the Test Center. A similar traffic, of 64 flows, is sent from Port 2 to Port 1 of the Test Center.

## Board Bootup

Each LS2088ARDB board must be setup and configured properly. For more information about the booting process please see **NXP Soc Booting Principles**. For more details on the specifics of LS2088ARDB board boot please refer to LSDK Quick Start Guide for LS2088ARDB. This paragraph only provides some custom values and configuration files used for booting the LS2088ARDB board while testing IPsec. Although using the default configuration may work we strongly encourage using the values/configuration files chosen below.

- While in the U-Boot prompt make sure that the variable "mcmemsize" is set to 0x80000000. This will ensure that enough memory was allocated to MC. See DPAA2 specific Environment variables for more details.
- Use the files "dpl.dts" and "dpc.dts" provided under the title "Useful Resources". They provide a minimum viable MC configuration that will enable IPsec testing. Both are in the .dts file format. To use them to configure the MC you need to compile them using the "dtc" compiler to obtain the "dtb" files:

```
$ dtc -O dtb -I dts -o dpc.dtb dpc.dts
$ dtc -O dtb -I dts -o dpl.dtb dpl.dts
```

For more information about MC resource files refer to chapter [Key Release Files: RCW, DPC and DPL](#) on page 329.

## Linux setup

When the Linux console prompt is presented to the user (after inserting the username and password) the following actions must be taken:

- Create DPSECI object and assign them to a DPRC. Make sure to enable congestion management for the DPSECI object.

```
$ restool dpseci create --num-queues=8 --priorities=1,2,3,4,5,6,7,8 --options="DPSECI_OPT_HAS_CG"
$ restool dprc assign dprc.1 --object=dpseci.0 --plugged=1
```

- Create the IPsec tunnels for each board (left/right) using the script "iproute\_128tunnels.sh". The script takes as parameter the board position (left/write) and uses it to configure each board accordingly. The script can be found in the "Useful Resources" section. You can create your own copy on the board by copying and pasting the content to a local script file, preserving the name.

To create the tunnels on the left board run:

```
$./iproute_128tunnels.sh left
```

To create the tunnels on the right board run:

```
$./iproute_128tunnels.sh right
```

- Disable flow control on board for both ni0 and ni1.

```
$ ethtool -A ni0 rx off
$ ethtool -A ni0 tx off
$ ethtool -A ni1 rx off
$ ethtool -A ni1 tx off
```

### NOTE

The flow control must be either on or off but must match the settings of the Test Center. The situation where the Test Control and the boards don't match causes resource in the boards to be oversubscribed which in turn will lead to memory corruption.

## Running the Test

After Spirent Test Center application is configured and the testing Ethernet interfaces are connected to the traffic generator, start to generate traffic to measure IPv4 SEC & Forward throughput.

## Useful resources

- The "iproute\_128tunnels.sh" script

```
#!/bin/bash
eth0=ni0
eth1=ni1

make_esp_tunnel() {

echo "add $1 $2 esp 0x$3 -m tunnel
-E $4 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
-A hmac-sha1 0xe9c43acd5e8d779b6e09c87347852708ab49bdd3;" | setkey -c

echo "add $2 $1 esp 0x`expr $3 + 100` -m tunnel
-E $4 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
-A hmac-sha1 0xea6856479330dc9c17b8f6c37e2a895363d83f21;" | setkey -c

}

make_esp_policy() {

if [$1 == left]
then
 dir1=out
 dir2=in
 echo "spdadd $2 $3 any -P $dir1 ipsec
 esp/tunnel/$4-$5/require;" | setkey -c
 echo "spdadd $3 $2 any -P $dir2 ipsec
 esp/tunnel/$5-$4/require;" | setkey -c
else
 dir1=in
 dir2=out
 echo "spdadd $2 $3 any -P $dir1 ipsec
 esp/tunnel/$4-$5/require;" | setkey -c
 echo "spdadd $3 $2 any -P $dir2 ipsec
 esp/tunnel/$5-$4/require;" | setkey -c
fi

}

Flush the SAD and SPD
setkey -F
setkey -FP

set ip address
left_addr_ip=192.85.1.1
right_addr_ip=192.86.1.1
left_src_mac=00:10:94:00:00:01
right_src_mac=00:10:94:00:00:02
proto="aes-cbc"
base1=200
base2=200
echo 1 > /proc/sys/net/ipv4/ip_forward

case $1 in
 left)
```

```
 ifconfig $eth0 $left_addr_ip
 i=2
 for((j=2;j<10;j++))
 do
 arp -s 192.85.1.$j $left_src_mac -i $eth0
 for((k=2;k<10;k++))
 do
 if [$base2 == 256]
 then
 base2=`expr $base2 - 256`
 base1=`expr $base1 + 1`
 fi
 ip addr add 200.$base1.$base2.10/24 dev $eth1
 make_esp_policy $1 192.85.1.$j 192.86.1.$k 200.$base1.$base2.10
200.$base1.$base2.20
 make_esp_tunnel 200.$base1.$base2.10 200.$base1.$base2.20 `expr 200 +
$i` $proto
 ((base2++))
 ((i++))
 done
 done
 done
 ;;
right)
 ifconfig $eth0 $right_addr_ip
 i=2
 for((j=2;j<10;j++))
 do
 arp -s 192.86.1.$j $right_src_mac -i $eth0
 for((k=2;k<10;k++))
 do
 if [$base2 == 256]
 then
 base2=`expr $base2 - 256`
 base1=`expr $base1 + 1`
 fi
 ip addr add 200.$base1.$base2.20/24 dev $eth1
 make_esp_policy $1 192.85.1.$j 192.86.1.$k 200.$base1.$base2.10
200.$base1.$base2.20
 make_esp_tunnel 200.$base1.$base2.10 200.$base1.$base2.20 `expr 200 +
$i` $proto
 ((base2++))
 ((i++))
 done
 done
 done
 ;;
esac

ifconfig $eth1 up
route add default dev $eth1
```

- The "dpc.dts" configuration file

```
/dts-v1/;

/ {

 mc_general {

 log {
 mode = "LOG_MODE_ON";
```



```

 level = "LOG_LEVEL_WARNING";
 };

 console {
 mode = "CONSOLE_MODE_ON";
 uart_id = <3>;
 };
};

resources {

 icid_pools {

 icid_pool@1 {
 num = <0x64>;
 base_icid = <0x0>;
 };
 };

 controllers {

 gbman {
 total_bman_buffers = <0xe0000>;
 wq_ch_conversion = <32>;
 };
 };

 board_info {

 ports {
 };
 };
};

```

#### • The "dpl.dts" configuration file

```

/dts-v1/;
/ {
 dpl-version = <10>;
 /*****
 * Containers
 *****/
 containers {
 dprc@1 {
 parent = "none";
 options = "DPRC_CFG_OPT_SPAWN_ALLOWED" , "DPRC_CFG_OPT_ALLOC_ALLOWED",
"DPRC_CFG_OPT_IRQ_CFG_ALLOWED";
 objects {
 /* ----- MACs -----*/
 obj_set@dpmac {
 type = "dpmac";
 ids = <1 2 3 4 5 6 7 8>;
 };

 /* ----- DPNI's -----*/
 obj_set@dpni {
 type = "dpni";
 ids = <0 1>;
 };
 };
 };
 };
};

```

```

/* ----- DPBPs -----*/
obj_set@dpbp {
 type = "dpbp";
 ids = <0 1>;
};

/* ----- DPPIOs -----*/
obj_set@dpio {
 type = "dpio";
 ids = <0 1 2 3 4 5 6 7>;
};

/* ----- DPMCPs -----*/
obj_set@dpmcp {
 type = "dpmcp";
 ids = <1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16>;
};

/* ----- DPCON -----*/
obj_set@dpcon {
 type = "dpcon";
 ids = <0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15>;
};
};
};

/*****
 * Objects
 *****/
objects {

/* ----- DPNI -----*/
dpni@0 {
 type = "DPNI_TYPE_NIC";
 options = "";
 num_queues = <8>;
 num_tcs = <1>;
 mac_filter_entries = <16>;
 vlan_filter_entries = <0>;
 fs_entries = <0>;
 qos_entries = <0>;
};

dpni@1 {
 type = "DPNI_TYPE_NIC";
 options = "";
 num_queues = <8>;
 num_tcs = <1>;
 mac_filter_entries = <16>;
 vlan_filter_entries = <0>;
 fs_entries = <0>;
 qos_entries = <0>;
};

/* ----- DPBP -----*/
dpbp@0 {
};

```

```

dppb@1 {
};

/* ----- DPIO ----- */
dpio@0 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@1 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@2 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@3 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@4 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@5 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@6 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};
dpio@7 {
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <8>;
};

/* ----- DPMAC ----- */

dpmac@1 {
};
dpmac@2 {
};
dpmac@3 {
};
dpmac@4 {
};
dpmac@5 {
};
dpmac@6 {
};
dpmac@7 {
};
dpmac@8 {
};

/* ----- DPMCP ----- */
dpmcp@1 {
};
dpmcp@2 {

```

```
};
dpmcp@3 {
};
dpmcp@4 {
};
dpmcp@5 {
};
dpmcp@6 {
};
dpmcp@7 {
};
dpmcp@8 {
};
dpmcp@9 {
};
dpmcp@10 {
};
dpmcp@11 {
};
dpmcp@12 {
};
dpmcp@13 {
};
dpmcp@14 {
};
dpmcp@15 {
};
dpmcp@16 {
};

/* ----- DPCON -----*/
dpcon@0 {
 num_priorities=<2>;
};

dpcon@1 {
 num_priorities=<2>;
};

dpcon@2 {
 num_priorities=<2>;
};

dpcon@3 {
 num_priorities=<2>;
};

dpcon@4 {
 num_priorities=<2>;
};

dpcon@5 {
 num_priorities=<2>;
};

dpcon@6 {
 num_priorities=<2>;
};

dpcon@7 {
```

```

 num_priorities=<2>;
 };

 dpcon@8 {
 num_priorities=<2>;
 };

 dpcon@9 {
 num_priorities=<2>;
 };

 dpcon@10 {
 num_priorities=<2>;
 };

 dpcon@11 {
 num_priorities=<2>;
 };

 dpcon@12 {
 num_priorities=<2>;
 };

 dpcon@13 {
 num_priorities=<2>;
 };

 dpcon@14 {
 num_priorities=<2>;
 };

 dpcon@15 {
 num_priorities=<2>;
 };
};

/*****
 * Connections
 *****/
connections {
 connection@0{
 /* First copper port (ETH0 on the RDB chassis) */
 endpoint1 = "dpni@0";
 endpoint2 = "dpmac@1";
 };
 connection@1{
 /* Second copper port (ETH1 on the RDB chassis) */
 endpoint1 = "dpni@1";
 endpoint2 = "dpmac@2";
 };
};
};

```

## Supporting Documentation

Linux IPSec Benchmark Reproducibility Guide

General SEC information, Job Ring Interface (JRI)

DPAA1-specific SEC details - Queue Interface (QI)

## 8.2.2.7 Decompression Compression Engine (DCE)

### Introduction

This section describes the software interface to DCE (Decompression Compression Engine) accelerator available on the LS2088A SoC. The interface is designed to simplify interaction with DCE as much as possible without loss of flexibility and acceleration offered by DCE hardware.

### Hardware Overview

This section gives an overview of the operation of the DCE hardware to provide fundamentals for software developers using the driver API. More detailed information is available in the hardware reference manual.

The DCE is a hardware accelerator that is part of the Datapath Acceleration Architecture version 2 (DPAA2). The DCE is one of the DPAA2 accelerators that include others like security and pattern matching engines. These accelerators are connected using a queue manager (QMan) and buffer manager (BMan) that allows data to be exchanged between software and accelerators.

Software enqueues data to a TX frame queue leading to DCE. DCE receives the data and processes it. It then enqueues a response on an RX frame queue leading back to software. The software then provides a DCE object that abstracts the details of frame queue configuration and usage, as well as other hardware details. This object is called DPDCEI.

### Example Application

The LSDK contains an example application called `dce-api-perf-test.c`. This test can be run in multiple modes that simulate many of the interesting DCE use cases. The README in the `dce` directory has instructions on how to run the test.

### Software Details

#### User-space Interface

#### DCE Driver

Provides a set of user-space APIs that simplifies access to DCE. The driver provides accessor objects called sessions which maintain state in coordination with the hardware. Users can pull the responses from DCE. This DCE driver is meant to be entirely sufficient for a user to use DCE without having to read the related HW documents. It is documented in the `dce` repository included in `<PATH_TO_LSDK>/packages/apps/dce/dce.h`.

#### Linux

There is no DCE driver in the Kernel.

### Functionality

#### Configuration

The DCE configuration and setup is documented in the article DPDCEI Commands.

### Build Procedure

The procedure is a standard LSDK build.

### Test Procedure

Refer to <https://source.codeaurora.org/external/qoriq/qoriq-components/dce/tree/README?h=github.qoriq-os/integration> for detailed descriptions of sample DCE test procedure.

## 8.2.3 DPAA2 Standard Linux Documentation

Following is a summary of relevant documentation from standard Linux sources and formats. It provides links to these documents, provides a snapshot of the document, or both.

## 8.2.3.1 Kernel Documentation Directory

The Linux kernel source code contains a documentation directory, and there is some information there that is relevant to DPAA2. It is possible to see the upstream versions of these documents by going to <https://www.kernel.org/> and browsing the Linux source code trees.

- Kernel Management Complex (MC) bus driver: This document is in-flight to kernel.org so a copy is provided below rather than a link to kernel.org.

Copyright (C) 2016 Freescale Semiconductor Inc.

DPAA2 (Data Path Acceleration Architecture Gen2)

-----

This document provides an overview of the Freescale DPAA2 architecture and how it is integrated into the Linux kernel.

Contents summary

- DPAA2 overview
- Overview of DPAA2 objects
- DPAA2 Linux driver architecture overview
  - bus driver
  - dprc driver
  - allocator
  - dpio driver
  - Ethernet
  - mac

DPAA2 Overview

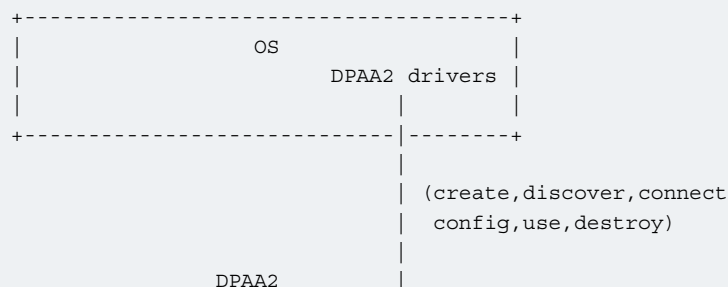
-----

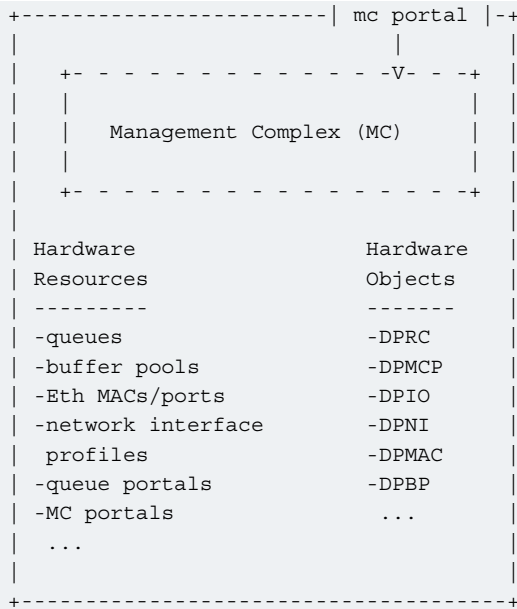
DPAA2 is a hardware architecture designed for high-speed network packet processing. DPAA2 consists of sophisticated mechanisms for processing Ethernet packets, queue management, buffer management, autonomous L2 switching, virtual Ethernet bridging, and accelerator (e.g. crypto) sharing.

A DPAA2 hardware component called the Management Complex (or MC) manages the DPAA2 hardware resources. The MC provides an object-based abstraction for software drivers to use the DPAA2 hardware.

The MC uses DPAA2 hardware resources such as queues, buffer pools, and network ports to create functional objects/devices such as network interfaces, an L2 switch, or accelerator instances.

The MC provides memory-mapped I/O command interfaces (MC portals) which DPAA2 software drivers use to operate on DPAA2 objects:





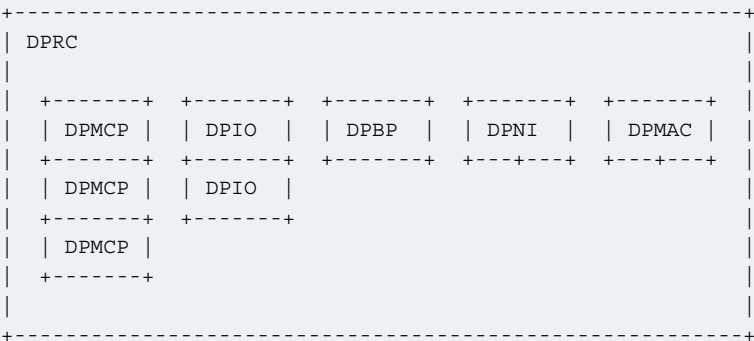
The MC mediates operations such as create, discover, connect, configuration, and destroy. Fast-path operations on data, such as packet transmit/receive, are not mediated by the MC and are done directly using memory mapped regions in DPIO objects.

Overview of DPAA2 Objects

The section provides a brief overview of some key objects in the DPAA2 hardware. A simple scenario is described illustrating the objects involved in creating a network interfaces.

-DPRC (Datapath Resource Container)

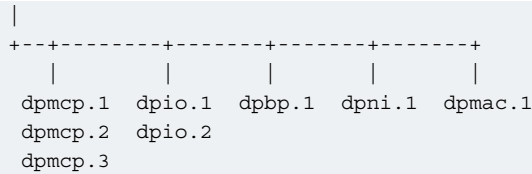
A DPRC is an container object that holds all the other types of DPAA2 objects. In the example diagram below there are 8 objects of 5 types (DPMCP, DPIO, DPBP, DPNI, and DPMAC) in the container.



From the point of view of an OS, a DPRC is bus-like. Like a plug-and-play bus, such as PCI, DPRC commands can be used to enumerate the contents of the DPRC, discover the hardware objects present (including mappable regions and interrupts).

dprc.1 (bus)





Hardware objects can be created and destroyed dynamically, providing the ability to hot plug/unplug objects in and out of the DPRC.

A DPRC has a mappable mmio region (an MC portal) that can be used to send MC commands. It has an interrupt for status events (like hotplug).

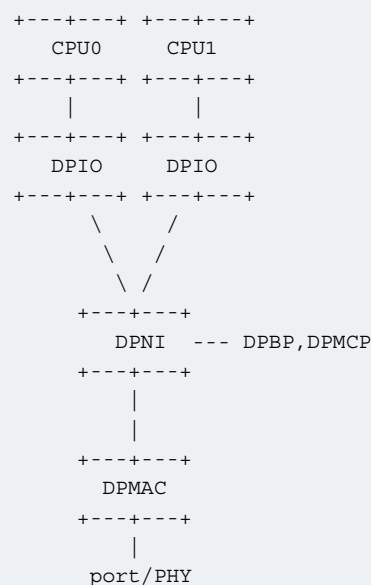
All objects in a container share the same hardware "isolation context". This means that with respect to an IOMMU the isolation granularity is at the DPRC (container) level, not at the individual object level.

DPRCs can be defined statically and populated with objects via a config file passed to the MC when firmware starts it. There is also a Linux user space tool called "restool" that can be used to create/destroy containers and objects dynamically.

#### -DPAA2 Objects for an Ethernet Network Interface

A typical Ethernet NIC is monolithic-- the NIC device contains TX/RX queuing mechanisms, configuration mechanisms, buffer management, physical ports, and interrupts. DPAA2 uses a more granular approach utilizing multiple hardware objects. Each object has specialized functions, and are used together by software to provide Ethernet network interface functionality. This approach provides efficient use of finite hardware resources, flexibility, and performance advantages.

The diagram below shows the objects needed for a simple network interface configuration on a system with 2 CPUs.



Below the objects are described. For each object a brief description is provided along with a summary of the kinds of operations the object supports and a summary of key resources of the object (mmio regions and irqs).

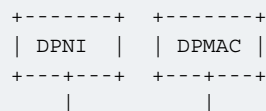
- DPMAC (Datapath Ethernet MAC): represents an Ethernet MAC, a hardware device that connects to an Ethernet PHY and allows physical transmission and reception of Ethernet frames.
  - mmio regions: none
  - irqs: dpni link change
  - commands: set link up/down, link config, get stats, irq config, enable, reset
- DPNI (Datapath Network Interface): contains TX/RX queues, network interface configuration, and rx buffer pool configuration mechanisms.
  - mmio regions: none
  - irqs: link state
  - commands: port config, offload config, queue config, parse/classify config, irq config, enable, reset
- DPIO (Datapath I/O): provides interfaces to enqueue and dequeue packets and do hardware buffer pool management operations. For optimum performance there is typically one DPIO per CPU. This allows each CPU to perform simultaneous enqueue/dequeue operations.
  - mmio regions: queue operations, buffer mgmt
  - irqs: data availability, congestion notification, buffer pool depletion
  - commands: irq config, enable, reset
- DPBP (Datapath Buffer Pool): represents a hardware buffer pool.
  - mmio regions: none
  - irqs: none
  - commands: enable, reset
- DPMCP (Datapath MC Portal): provides an MC command portal. Used by drivers to send commands to the MC to manage objects.
  - mmio regions: MC command portal
  - irqs: command completion
  - commands: irq config, enable, reset

#### Object Connections

-----

Some objects have explicit relationships that must be configured:

- DPNI <--> DPMAC
  - DPNI <--> DPNI
  - DPNI <--> L2-switch-port
- A DPNI must be connected to something such as a DPMAC, another DPNI, or L2 switch port. The DPNI connection is made via a DPRC command.



```
+=====+

-DPNI <--> DPBP
 A network interface requires a 'buffer pool' (DPBP
 object) which provides a list of pointers to memory
 where received Ethernet data is to be copied. The
 Ethernet driver configures the DPBPs associated with
 the network interface.
```

#### Interrupts

-----

All interrupts generated by DPAA2 objects are message interrupts. At the hardware level message interrupts generated by devices will normally have 3 components-- 1) a non-spoofable 'device-id' expressed on the hardware bus, 2) an address, 3) a data value.

In the case of DPAA2 devices/objects, all objects in the same container/DPRC share the same 'device-id'. For ARM-based SoC this is the same as the stream ID.

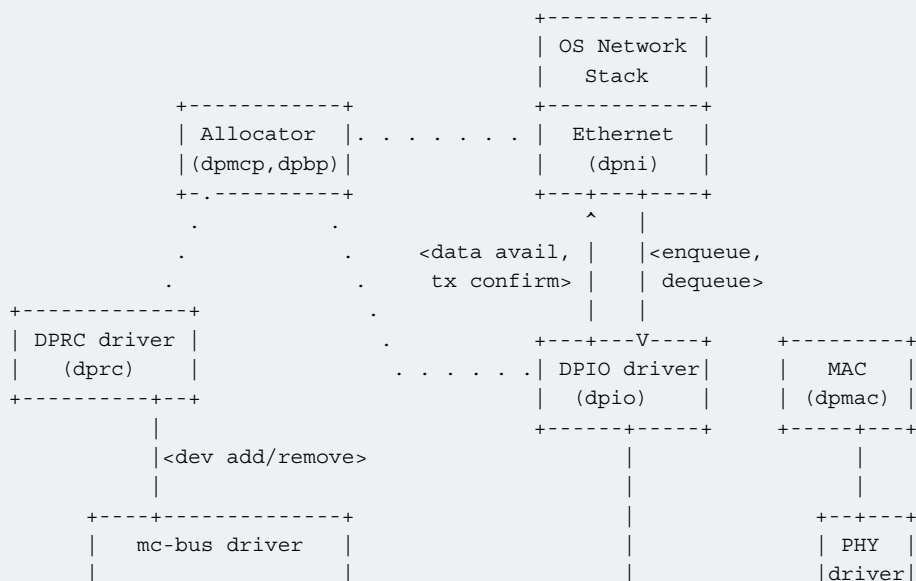
#### DPAA2 Linux Driver Overview

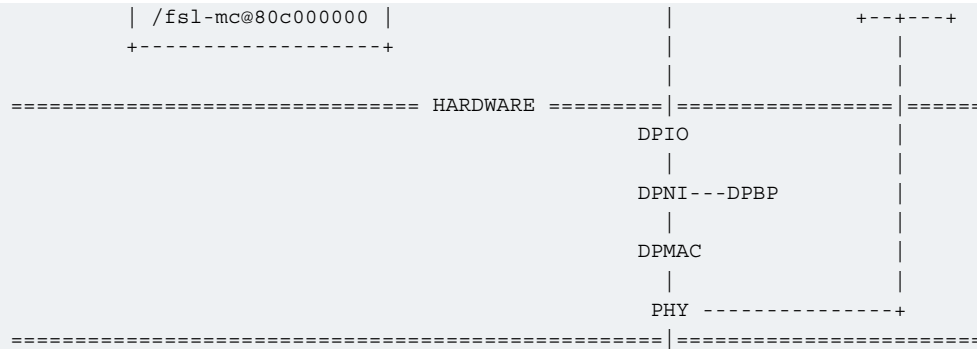
-----

This section provides an overview of the Linux kernel drivers for DPAA2-- 1) the bus driver and associated "DPAA2 infrastructure" drivers and 2) functional object drivers (such as Ethernet).

As described previously, a DPRC is a container that holds the other types of DPAA2 objects. It is functionally similar to a plug-and-play bus controller.

Each object in the DPRC is a Linux "device" and is bound to a driver. The diagram below shows the Linux drivers involved in a networking scenario and the objects bound to each driver. A brief description of each driver follows.





A brief description of each driver is provided below.

#### mc-bus driver

-----

The mc-bus driver is a platform driver and is probed from an "/fsl-mc@xxxx" node in the device tree passed in by boot firmware. It is responsible for bootstrapping the DPAA2 kernel infrastructure.

Key functions include:

- registering a new bus type named "fsl-mc" with the kernel, and implementing bus call-backs (e.g. match/uevent/dev\_groups)
- implementing APIs for DPAA2 driver registration and for device add/remove
- creates an MSI irq domain
- do a device add of the 'root' DPRC device, which is needed to bootstrap things

#### DPRC driver

-----

The dprc-driver is bound to DPRC objects and does runtime management of a bus instance. It performs the initial bus scan of the DPRC and handles interrupts for container events such as hot plug.

#### Allocator

-----

Certain objects such as DPMCP and DPBP are generic and fungible, and are intended to be used by other drivers. For example, the DPAA2 Ethernet driver needs:

- DPMCPs to send MC commands, to configure network interfaces
- DPBPs for network buffer pools

The allocator driver registers for these allocatable object types and those objects are bound to the allocator when the bus is probed. The allocator maintains a pool of objects that are available for allocation by other DPAA2 drivers.

#### DPIO driver

-----

The DPIO driver is bound to DPIO objects and provides services that allow other drivers such as the Ethernet driver to receive and transmit data.

Key services include:

- data availability notifications
- hardware queuing operations (enqueue and dequeue of data)
- hardware buffer pool management

There is typically one DPIO object per physical CPU for optimum performance, allowing each CPU to simultaneously enqueue and dequeue data.

The DPIO driver operates on behalf of all DPAA2 drivers active in the kernel-- Ethernet, crypto, compression, etc.

Ethernet

-----

The Ethernet driver is bound to a DPNI and implements the kernel interfaces needed to connect the DPAA2 network interface to the network stack.

Each DPNI corresponds to a Linux network interface.

MAC driver

-----

An Ethernet PHY is an off-chip, board specific component and is managed by the appropriate PHY driver via an mdio bus. The MAC driver plays a role of being a proxy between the PHY driver and the MC. It does this proxy via the MC commands to a DPMAC object.

## 8.2.3.2 DPAA2 Resource Management Tool (restool) User Manual

Restool is a Linux user space program that allows DPAA2 objects to be created, destroyed, and manipulated. Its primary documentation is in the style of a Linux man page.

The Management Complex architecture uses a hardware object called a “container” (or DPRC) to hold I/O resources and hardware objects for use by GPP software contexts.

DPRCs can be created and populated in two different ways:

- at MC initialization during system boot in a configuration file called a “DPL file”
- dynamically at runtime

This document describes how restool can be used to do dynamic management of MC resources in the context of Linux. Key resource management operations include:

- listing containers and their contents
- creating/destroying containers
- creating/destroying new MC objects
- move object between parent container and child container
- establishing connections between MC objects

The version of restool -restool v1.4 - included in this release is compatible with all MC firmware versions and will export different options based on the firmware found on the board. In the following pages it will be described the available options found running MC10.x on the board.

### 8.2.3.2.1 DPRC commands

#### 8.2.3.2.1.1 list command

The **list** command lists all containers in the system.

SYNTAX:

**restool dprc list**

ARGUMENTS:

none

#### EXAMPLE:

List all the containers in the system

```
$ restool dprc list
dprc.1
 dprc.2
 dprc.3
```

The container hierarchy (parent-child relationships) is shown by indentation.

### 8.2.3.2.1.2 show command

The **show** command displays the contents (objects and resources) of a DPRC/container.

#### SYNTAX:

**restool dprc show <container>**

**restool dprc show <container> --resources**

**restool dprc show <container> --resource-type=<resource-type>**

#### ARGUMENTS:

*<container>*

A string specifying the target dprc—e.g. “dprc.2”

The container argument value “mc.global” is special and refers to the global container of resource pools inside the Management Complex.

#### **--resources**

Display a container’s resource count for each resource (instead of displaying objects/resources)

**--resource-type <resource-type>**

Specifies the type of resource to list. The resource-type argument is a string specifying the resource name—e.g. “mcp”.

#### EXAMPLE :

Show all objects in dprc 2:

```
$ restool dprc show dprc.2
dprc.2 contains 6 objects:
object label plugged state
dpni.7 xyz plugged
dpni.8 abc plugged
dpio.2 plugged
dpio.3 unplugged
dpcon.9 plugged
dppb.1 plugged
```

Show all resources in dprc 2:

```
$ restool dprc show dprc.2 --resources
bpid: 16
```

```
fqid: 100
channel: 4
qpr: 2
cgid: 2
```

Show dprc with no objects in it:

```
$ restool dprc show dprc.4
(empty)
```

Show all buffer pool IDs in dprc 2:

```
$ restool dprc show dprc.2 --resource-type=bp
bp.35 - bp.36
bp.50
bp.52 - bp.63
```

Show all MC portal IDs in the global MC container:

```
$ restool dprc show mc.global --resource-type=mcp
mcp.30 - mcp.250
```

### 8.2.3.2.1.3 info command

The **info** command displays detailed information about a specific container.

SYNTAX:

**restool dprc info** *<dprc-object>* [**--verbose**]

ARGUMENTS :

*<dprc-object>*

Specifies which container to show detailed info for. The object argument is a string specifying the container name—e.g. “dprc.2”.

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dprc info dprc.2
container id: 2
icid: 2
portal id: 5
version: 0.0
dprc options: 0x3
 DPRC_CFG_OPT_SPAWN_ALLOWED
 DPRC_CFG_OPT_ALLOC_ALLOWED
object label: nadk's dprc

$ restool dprc info dprc.2 --verbose
container id: 2
icid: 2
portal id: 5
version: 0.0
dprc options: 0x3
 DPRC_CFG_OPT_SPAWN_ALLOWED
```

```
DPRC_CFG_OPT_ALLOC_ALLOWED
object label: nadk-usage-dprc
number of mappable regions: 1
number of interrupts: 1
interrupt 0's mask: 0
interrupt 0's status: 0x1
```

#### 8.2.3.2.1.4 create command

The **create** command creates a new child DPRC under the specified parent. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dprc create** <parent-container> [--options=<options-mask>] [--label=<object's-label>]

OPTIONS :

<parent-container>

--options=<options-mask>

Where <options-mask> is a comma separated list of DPRC options:

```
DPRC_CFG_OPT_SPAWN_ALLOWED
DPRC_CFG_OPT_ALLOC_ALLOWED
DPRC_CFG_OPT_OBJ_CREATE_ALLOWED
DPRC_CFG_OPT_TOPOLOGY_CHANGES_ALLOWED
DPRC_CFG_OPT_IOMMU_BYPASS
DPRC_CFG_OPT_AIOP
DPRC_CFG_OPT_IRQ_CFG_ALLOWED
```

--label=<object's-label>

Specify a label for the newly created object. It is kind of an alias for that object.

Length of the string is 15 characters maximum.

Say --label="nadk's dprc"

EXAMPLE:

Create a child DPRC under parent dprc.1 with default options:

```
$ restool dprc create dprc.1
dprc.9 is created under dprc.1
```

Create a child DPRC under parent dprc.1 with default options, with label "nadk's dprc":

```
$ restool dprc create dprc.1 --label="nadk's dprc"
dprc.11 is created under dprc.1
```

#### 8.2.3.2.1.5 create command

The **create** command creates a new child DPRC under the specified parent. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dprc create** <parent-container> [--options=<options-mask>] [--label=<object's-label>]



OPTIONS :

<parent-container>

--options=<options-mask>

Where <options-mask> is a comma separated list of DPRC options:

```
DPRC_CFG_OPT_SPAWN_ALLOWED
DPRC_CFG_OPT_ALLOC_ALLOWED
DPRC_CFG_OPT_OBJ_CREATE_ALLOWED
DPRC_CFG_OPT_TOPOLOGY_CHANGES_ALLOWED
DPRC_CFG_OPT_IOMMU_BYPASS
DPRC_CFG_OPT_AIOP
DPRC_CFG_OPT_IRQ_CFG_ALLOWED
```

--label=<object's-label>

Specify a label for the newly created object. It is kind of an alias for that object.

Length of the string is 15 characters maximum.

Say --label="nadk's dprc"

EXAMPLE:

Create a child DPRC under parent dprc.1 with default options:

```
$ restool dprc create dprc.1
dprc.9 is created under dprc.1
```

Create a child DPRC under parent dprc.1 with default options, with label "nadk's dprc":

```
$ restool dprc create dprc.1 --label="nadk's dprc"
dprc.11 is created under dprc.1
```

### 8.2.3.2.16 destroy command

The **destroy** command destroys the specified DPRC.

SYNTAX:

**restool dprc destroy** <container> **--help**

OPTIONS:

<container>

**--help**

Displays help for the command.

EXAMPLE:

Destroy a specified DPRC, say dprc.2:

```
$ restool dprc destroy dprc.2
dprc.2 is destroyed
```

### 8.2.3.2.17 assign command

The **assign** command moves an object or resource from a parent container to a child container. Object (dpni, dppb, etc) assignment is always explicit and the exact object id to be assigned must be specified. Resources (e.g. mcp, bp, fq, etc) are assigned by type and count.

SYNTAX:

**restool dprc assign** <parent-container> [--child=<child-container>] --object=<object> --plugged=<state>

This syntax changes the plugged state. The child-container must be the same as parent-container, or omit --target option. It is not possible to change the plugged state of a dprc.

**restool dprc assign** <parent-container> [--child=<child-container>] --object=<object>

This syntax moves one object from parent-container to another, so the target-container must be different from the parent-container. Limitation: cannot move dprc from one container to another.

**restool dprc assign** <parent-container> [--child=<child-container>] --resource-type=<type> -- count=<number>

This syntax moves a resource from parent-container to a child-container. If the childcontainer is the same as the parent-container, the resource will be taken from the parent of parent-container and will be assigned to the parent-container.

ARGUMENTS :

<container>

Specifies the parent container from which the object will be moved.

--object=<object>

Specifies the object to assign— value is a string specifying object name and ID (e.g. dpni.5)

--child=<child-container>

Specifies the destination container for the operation. Valid values are any child container. (The target container may be the same as the parent container, allowing “assign to self”)

--plugged=<state>

Specifies the plugged state of the object (valid values are 0 or 1)

--resource-type=<type>

String specifying the resource type to assign (e.g “mcp”, “fq”, “cg” etc). To see valid resources that may be assigned use the “dprc show <container> --resources” command.

--count=<number>

Number of resources to assign.

EXAMPLE:

Set the plugged state of dpni.5. Note source and destination containers are the same.

```
$ restool dprc assign dprc.1 --object=dpni.5 --child=dprc.1
--plugged=1
$ restool dprc assign dprc.1 --object=dpni.5 --plugged=1
```

Unset the plugged state of dpni.5. Note source and destination containers are the same.

```
$ restool dprc assign dprc.1 --object=dpni.5 --child=dprc.1
--plugged=0
$ restool dprc assign dprc.1 --object=dpni.5 --plugged=0
```

Move dpni.5 from dprc.1 (parent) to dprc.3 (child):

```
$ restool dprc assign dprc.1 --object=dpni.5 --child=dprc.3
```

Move 3 mcp resources from dprc.1 (parent) to dprc.2 (child):

```
$ restool dprc assign dprc.1 --resource-type=mcp --count=3
--child=dprc.2
```

### 8.2.3.2.18 unassign command

The **unassign** command moves an object or resource from a child container to a parent container

SYNTAX:

**restool dprc unassign** <container> **--object**=<object> [**--child**=<child-container>]

**restool dprc unassign** <container> **--resource-type**=<type> **--count** <number> [**--child**=<child-container>]

ARGUMENTS :

<container>

Specifies the container to which the object will be moved.

**--object**=<object>

Specifies the object to unassign— value is a string specifying object name and ID (e.g.

dpni.5)

**--child**=<child-container>

Specifies the container from which the object/resource will be moved from.

**--plugged**=<plugged-state>

Specifies the plugged state of the object (valid values are 0 or 1)

**--resource-type**=<type>

String specifying the resource type to assign (e.g “mcp”, “fq”, “cg”, etc)

**--count**=<number>

Number of resources to unassign.

EXAMPLE:

Unassign 3 mcp resources from dprc.2 (child) to dprc.1 (parent):

```
$ restool dprc unassign dprc.1 --resource-type=mcp --count=3
--child=dprc.2
```

Unassign dpni.5 from dprc.3 (child) to dprc.1 (parent):

```
$ restool dprc unassign dprc.1 --object=dpni.5 --child=dprc.3
```

### 8.2.3.2.1.9 set-quota command

The **set-quota** command sets quota policies for a child container, specifying the number of resources a child may take from its parent container. But remember a parent can assign any number of resource to its child if it wants to, and if it has enough resources to assign. So the quota is effective only when the child dprc does have enough resource and it wants to borrow resource from its parent. It could only “borrow” the quota number of resources from its parent.

SYNTAX:

```
restool dprc set-quota <parent-container> --resource-type=<type> --count=<number>
```

```
--child-container=<container>
```

ARGUMENTS :

<parent-container>

Specifies the parent container.

```
--resource-type=<type>
```

String specifying the resource type to set the quota for (e.g “mcp”, “fq”, “cg”, etc)

```
--count=<number>
```

Max number of resources the child is able to allocate.

```
--child-container=<container>
```

EXAMPLE:

Set a quota of 10 mcp resource that child container dprc.5 may take from parent dprc.1:

```
$ restool dprc set-quota dprc.1 --resource-type=mcp --count=10
--child-container=dprc.5
```

### 8.2.3.2.1.10 set-label command

The **set-label** command sets label for any objects excluding dprc.1

SYNTAX:

```
restool dprc set-label <object> --label=<label>
```

ARGUMENTS :

<object>

Specifies the object to be set.

```
--label=<label>
```

String specifying the label, maximum length is 15 characters.

EXAMPLE:

Set label of dprc.4 to “mountain view”:

```
$ restool dprc set-label dprc.4 --label="mountain view"
```

### 8.2.3.2.11 connect command

The **connect** command connects 2 objects, creating a link between them.

SYNTAX:

**restool dprc connect** <container> --endpoint1=<object> --endpoint2=<object>

ARGUMENTS :

<container>

A string specifying the target dprc—e.g. “dprc.2”

--endpoint1=<object>

Specifies the first endpoint object.

--endpoint2=<object>

Specifies the second endpoint object.

EXAMPLE:

The connect command connects a network object such as a DPNI to a peer object such as a DPMAC or DPSW port.

Connect dpni.2 to dpmac.5:

```
$ restool dprc connect dprc.2 --endpoint1=dpni.2 --endpoint2=dpmac.5
```

Connect dpni.2 to dpsw.1 interface 7:

```
$ restool dprc connect dprc.2 --endpoint1=dpni.2 --endpoint2=dpsw.1.7
```

### 8.2.3.2.12 disconnect command

The **disconnect** command removes the link between two objects. Either endpoint can be specified as the target of the operation.

SYNTAX:

**restool dprc disconnect** <container> --endpoint=<object>

ARGUMENTS:

<container>

A string specifying the target dprc—e.g. “dprc.2”

--endpoint=<object>

Specifies the first endpoint object.

EXAMPLE:

Remove the link between dpni.2 and dpmac.5

```
$ restool dprc disconnect dprc.2 --endpoint=dpni.2
```

### 8.2.3.2.13 generate-dpl command

The **generate-dpl** command prints to the standard output a DPL syntax file describing the specified container

SYNTAX:

**restool dprc generate-dpl** <container>

ARGUMENTS:

<container>

A string specifying the target dprc—e.g. “dprc.2”

EXAMPLE:

```
Generate a DPL for dprc.1
```

```
$ restool dprc generate-dpl dprc.1
```

## 8.2.3.2.2 DPNI Commands

### 8.2.3.2.2.1 help command

The **help** command displays usage information for the DPNI object

SYNTAX:

**restool dpni help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpni help
usage: restool dpni <command> [--help] [ARGS...]
Where <command> can be:
 info - displays detailed information about a DPNI object.
 create - creates a child DPNI under the root DPRC
 destroy - destroys a child DPNI under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.2.2 info command

The **info** command displays detailed information about a specific dpni object.

SYNTAX:

**restool dpni info** <dpni-object> [--verbose]

ARGUMENTS :

<dpni-object>

Specifies which dpni object to show detailed info for. The dpni-object argument is a string specifying the object name—e.g. “dpni.7”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpni info dpni.7
dpni version: 5.0
dpni id: 7
```

```

plugged state: plugged
endpoint: dpmac.2, link is down
link status: 0 - down
mac address: 00:00:00:00:00:07
dpni_attr.options value is: 0x190
 DPNI_OPT_DIST_HASH
 DPNI_OPT_UNICAST_FILTER
 DPNI_OPT_MULTICAST_FILTER
max senders: 8
max traffic classes: 1
max distribution's size per RX traffic class:
 class 0's size: 15
max unicast filters: 16
max multicast filters: 64
max vlan filters: 0
max QoS entries: 0
max QoS key size: 0
max distribution key size: 4

$ restool dpni info dpni.7 --verbose
dpni version: 5.0
dpni id: 7
plugged state: plugged
endpoint: dpmac.2, link is down
link status: 0 - down
mac address: 00:00:00:00:00:07
dpni_attr.options value is: 0x190
 DPNI_OPT_DIST_HASH
 DPNI_OPT_UNICAST_FILTER
 DPNI_OPT_MULTICAST_FILTER
max senders: 8
max traffic classes: 1
max distribution's size per RX traffic class:
 class 0's size: 15
max unicast filters: 16
max multicast filters: 64
max vlan filters: 0
max QoS entries: 0
max QoS key size: 0
max distribution key size: 4
number of mappable regions: 0
number of interrupts: 1
interrupt 0's mask: 0
interrupt 0's status: 0

```

### 8.2.3.2.3 create command

The **create** command creates a new DPNI. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpni create --mac-addr=<addr> [OPTIONS]**

ARGUMENTS :

**--mac-addr=<addr>**

String specifying primary MAC address (e.g., 00:00:05:00:00:05)

OPTIONS :

**--max-senders=<number>**

maximum number of different senders; will be used as the number of dedicated TX flows;

In case it isn't power-of-2 it will be ceiling to the next power-of-2 as HW demand it; 0 will be treated as 1

**--options=<options-mask>**

Where <options-mask> is a comma separated list of DPNI options:

```
DPNI_OPT_ALLOW_DIST_KEY_PER_TC
DPNI_OPT_TX_CONF_DISABLED
DPNI_OPT_PRIVATE_TX_CONF_ERR_DISABLED
DPNI_OPT_DIST_HASH
DPNI_OPT_DIST_FS
DPNI_OPT_UNICAST_FILTER
DPNI_OPT_MULTICAST_FILTER
DPNI_OPT_VLAN_FILTER
DPNI_OPT_IPR
DPNI_OPT_IPF
DPNI_OPT_VLAN_MANIPULATION
DPNI_OPT_QOS_MASK_SUPPORT
DPNI_OPT_FS_MASK_SUPPORT
```

**--max-tcs=<number>**

Specifies the maximum number of traffic-classes

**--max-dist-per-tc=<dist-size>,<dist-size>,...**

Comma separated list of counts specifying the maximum distribution's size per RX traffic-class

**--max-unicast-filters=<number>**

maximum number of unicast filters; 0 will be treated as 16

**--max-multicast-filters=<number>**

maximum number of multicast filters; 0 will be treated as 64

**--max-vlan-filters=<number>**

maximum number of vlan filters; '0' will be treated as '16'

**--max-qos-entries=<number>**

if max\_tcs > 1, declares the maximum entries for the QoS table; '0' will be treated as '64'

**--max-qos-key-size=<number>**

maximum key size for the QoS look-up; '0' will be treated as '24' which enough for IPv4 5-tuple

**--max-dist-key-size=<number>**

maximum key size for the distribution; '0' will be treated as '24' which enough for IPv4 5-tuple

EXAMPLE:



Create a DPNI, specifying MAC address, with all default options:

```
$ restool dpni create --mac-addr=00:00:05:00:00:05
dpni.9 is crated under dprc.1
```

Create a DPNI, specifying MAC address, and some options:

```
$ restool dpni create --mac-addr=00:00:05:00:00:05
--options=DPNI_OPT_MULTICAST_FILTER,DPNI_OPT_UNICAST_FILTER
dpni.11 is created under dprc.1
```

### 8.2.3.2.2.4 create command

The **create** command creates a new DPNI. The name/id of the object created is displayed to stdout.

In the following part are presented the options when creating a DPNI using MC10.x firmware version. Also, restool is compatible with older MC firmware versions and will export another set of options in these other cases.

SYNTAX:

**restool dpni create** [OPTIONS]

OPTIONS :

**--options=<options-mask>**

Where <options-mask> is a comma separated list of DPNI options:

```
DPNI_OPT_TX_FRM_RELEASE
DPNI_OPT_NO_MAC_FILTER
DPNI_OPT_HAS_POLICING
DPNI_OPT_SHARED_CONGESTION
DPNI_OPT_HAS_KEY_MASKING
DPNI_OPT_NO_FS
```

**--num-queues=<number>**

Number of TX/RX queues use for traffic distribution. Used to distribute traffic to multiple GPP cores.  
Defaults to one queue. Maximim supported value is 8

**--num-tcs=<number>**

Number of traffic classes (TCs), reserved for the DPNI. Defaults to one TC. Maximum supported value is 8

**--num-entries=<number>**

Number of entries in the MAC address filtering table. Allows both unicast and multicast entries.  
By default, there are 80 entries. Maximum supported value is 80.

**--vlan-entries=<number>**

Number of entries in the VLAN address filtering table. By default, VLAN filtering is disabled. Maximum values is 16.

**--qos-entries=<number>**

Number of entries in the QoS classification table. Ignored if DPNI has a single TC. By default, set to 64.

**--fs-entries=<number>**

Number of entries in the flow steering table. Defaults to 64. Maximum value is 1024.

**--container=<container-name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

#### EXAMPLE:

Create a DPNI with all default options:

```
$ restool dpni create
dpni.9 is created under dprc.1
```

Create a DPNI with some specific options as a child object of dprc.2 (dprc already created):

```
$ restool dpni create --options=DPNI_OPT_TX_FRM_RELEASE,DPNI_OPT_NO_FS --container=dprc.2
dpni.11 is created under dprc.2
```

### 8.2.3.2.2.5 destroy command

The **destroy** command destroys a DPNI.

SYNTAX:

**restool dpni destroy** *<dpni-object>*

ARGUMENTS :

*<dpni-object>*

Specifies which DPNI to destroy.

#### EXAMPLE:

```
$ restool dpni destroy dpni.9
dpni.9 is destroyed
```

### 8.2.3.2.3 DPIO Commands

#### 8.2.3.2.3.1 help command

The **help** command displays usage information for the DPIO object

SYNTAX:

**restool dpio help**

ARGUMENTS:

none

#### EXAMPLE:

```
$ restool dpio help
usage: restool dpio <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPIO object.
create - creates a DPIO under the root DPRC
destroy - destroys a DPIO under the root DPRC

For command-specific help, use the --help option available for each command.
```

#### 8.2.3.2.3.2 info command

The **info** command displays detailed information about a specific dpio object.

SYNTAX:

**restool dpio info** <dpio-object> [--verbose]

ARGUMENTS :

<dpio-object>

Specifies which dpio object to show detailed info for. The dpio-object argument is a string specifying the object name—e.g. “dpio.7”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
restool dpio info dpio.1
 dpio version: 3.0
 dpio id: 1
 plugged state: plugged
 offset of qbman software portal cache-enabled area: 0x20000
 offset of qbman software portal cache-inhibited area: 0x4020000
 qbman software portal id: 0x2
 dpio channel mode is: DPIO_LOCAL_CHANNEL
 number of priorities is: 0x8
restool dpio info dpio.1 --verbose
 dpio version: 3.0
 dpio id: 1
 plugged state: plugged
 offset of qbman software portal cache-enabled area: 0x20000
 offset of qbman software portal cache-inhibited area: 0x4020000
 qbman software portal id: 0x2
 dpio channel mode is: DPIO_LOCAL_CHANNEL
 number of priorities is: 0x8
 number of mappable regions: 2
 number of interrupts: 1
 interrupt 0's mask: 0
 interrupt 0's status: 0x8
```

### 8.2.3.2.3.3 create command

The **create** command creates a new DPIO. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpio create** [OPTIONS]

OPTIONS :

**--channel-mode=<mode>**

Where <mode> is one of:

```
DPIO_LOCAL_CHANNEL
DPIO_NO_CHANNEL
```

Default value is DPIO\_LOCAL\_CHANNEL .

**--num-priorities=<number>**

Valid values for <number> are 1-8. Default value is 8.

EXAMPLE:

Create a DPIO with all default options:

```
$ restool dpio create
```

dpio.10 is created under dprc.1

Create a DPIO, specifying number of priorities:

```
$ restool dpni create -num-priorities=4
dpio.2 is created under dprc.1
```

#### 8.2.3.2.3.4 create command

The **create** command creates a new DPIO. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpio create** [OPTIONS]

OPTIONS :

**--channel-mode=<mode>**

Where <mode> is one of:

```
DPIO_LOCAL_CHANNEL
DPIO_NO_CHANNEL
```

Default value is DPIO\_LOCAL\_CHANNEL .

**--num-priorities=<number>**

Valid values for <number> are 1-8. Default value is 8.

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPIO with all default options:

```
$ restool dpio create
dpio.8 is created under dprc.1
```

Create a DPIO, specifying number of priorities:

```
$ restool dpni create -num-priorities=4
dpio.2 is created under dprc.1
```

#### 8.2.3.2.3.5 destroy command

The **destroy** command destroys a DPIO.

SYNTAX:

**restool dpio destroy** <dpio-object>

ARGUMENTS :

<dpio-object>

Specifies which DPIO to destroy.

EXAMPLE:

```
$ restool dpio destroy dpio.9
dpio.9 is destroyed
```

## 8.2.3.2.4 DPSW Commands

text

### 8.2.3.2.4.1 help command

The **help** command displays usage information for the DPSW object

SYNTAX:

**restool dpsw help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpsw help
usage: restool dpsw <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPSW object.
create - creates a DPSW under the root DPRC
destroy - destroys a DPSW under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.4.2 info command

The **info** command displays detailed information about a specific dpsw object.

SYNTAX:

**restool dpsw info <dpsw-object> [--verbose]**

ARGUMENTS :

*<dpsw-object>*

Specifies which object to show detailed info for. The dpsw-object argument is a string specifying the object name—e.g. “dpsw.2”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpsw info dpsw.1
dpsw version: 6.0
dpsw id: 1
plugged state: unplugged
endpoints:
endpoint state: -1
 interface 0: No object associated
endpoint state: -1
 interface 1: No object associated
```

```
endpoint state: -1
 interface 2: No object associated
endpoint state: -1
 interface 3: No object associated
dpsw_attr.options value is: 0x1
 DPSW_OPT_FLOODING_DIS
max VLANs: 8
max FDBs: 8
DPSW frame storage memory size: 0
number of interfaces: 4
current number of VLANs: 1
current number of FDBs: 1
```

### 8.2.3.2.4.3 create command

The **create** command creates a new DPSW. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpsw create --num-ifs=<number> [OPTIONS]**

ARGUMENTS :

**--num-ifs=<number>**

Number of external and internal interfaces.

OPTIONS :

**--options=<options-mask>**

Where <options-mask> is a comma separated list of DPSW options:

DPSW\_OPT\_FLOODING\_DIS

DPSW\_OPT\_MULTICAST\_DIS

DPSW\_OPT\_CTRL\_IF\_DIS

DPSW\_OPT\_FLOODING\_METERING\_DIS

DPSW\_OPT\_METERING\_EN

**--max-vlans=<number>**

Maximum Number of VLAN's. Default is 16.

**--max-fdb=<number>**

Maximum Number of FDB's. Default is 16.

**--num-fdb-entries=<number>**

Number of FDB entries. Default is 1024.

**--fdb-aging-time=<number>**

Default FDB aging time in seconds. Default is 300 seconds.

**--max-fdb-mc-groups=<number>**

Number of multicast groups in each FDB table. Default is 32.

EXAMPLE:

Create a 4-port switch with all default options:

```
$ restool dpsw create --num-ifs=4
dpsw.8 is created under dprc.1
```

Create a 4-port switch with options:

```
$ restool dpsw create -num-ifs=4 -max-vlans=8 -max-fdb-mc-groups=300
--options=DPSW_OPT_TC_DIS,DPSW_OPT_FLOODING_DIS
dpsw.2 is created under dprc.1
```

## 8.2.3.2.4.4 create command

The **create** command creates a new DPSW. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpsw create --num-ifs=<number> [OPTIONS]**

ARGUMENTS :

**--num-ifs=<number>**

Number of external and internal interfaces.

OPTIONS :

**--options=<options-mask>**

Where <options-mask> is a comma separated list of DPSW options:

DPSW\_OPT\_FLOODING\_DIS

DPSW\_OPT\_MULTICAST\_DIS

DPSW\_OPT\_CTRL\_IF\_DIS

DPSW\_OPT\_FLOODING\_METERING\_DIS

DPSW\_OPT\_METERING\_EN

**--max-vlans=<number>**

Maximum Number of VLAN's. Default is 16.

**--max-fdbs=<number>**

Maximum Number of FDB's. Default is 16.

**--num-fdb-entries=<number>**

Number of FDB entries. Default is 1024.

**--fdb-aging-time=<number>**

Default FDB aging time in seconds. Default is 300 seconds.

**--max-fdb-mc-groups=<number>**

Number of multicast groups in each FDB table. Default is 32.

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a 4-port switch with all default options under dprc.2:

```
$ restool dpsw create --num-ifs=4 --container=dprc.2
dpsw.8 is created under dprc.2
```

Create a 4-port switch with options:

```
$ restool dpsw create -num-ifs=4 -max-vlans=8 -max-fdb-mc-groups=300
--options=DPSW_OPT_TC_DIS,DPSW_OPT_FLOODING_DIS
dpsw.2 is created under dprc.1
```

### 8.2.3.2.4.5 destroy command

The **destroy** command destroys a DPSW.

SYNTAX:

**restool dpsw destroy** <dpsw-object>

ARGUMENTS :

<dpsw-object>

Specifies which DPSW to destroy.

EXAMPLE:

```
$ restool dpsw destroy dpsw.8
dpsw.8 is destroyed
```

### 8.2.3.2.5 DPBP Commands

#### 8.2.3.2.5.1 help command

The **help** command displays usage information for the DPBP object

SYNTAX:

**restool dpbp help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpbp help
usage: restool dpbp <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPBP object.
create - creates a DPBP under the root DPRC
destroy - destroys a DPBP under the root DPRC

For command-specific help, use the --help option available for each command.
```

#### 8.2.3.2.5.2 info command

The **info** command displays detailed information about a specific dpbp object.



SYNTAX:

**restool dpbp info** <dpbp-object> [--verbose]

ARGUMENTS :

<dpbp-object>

Specifies which dpbp object to show detailed info for. The dpbp-object argument is a string specifying the object name—e.g. “dvpb.3”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpbp info dpbp.1
dpbp version: 2.0
dpbp id: 1
plugged state: plugged
buffer pool id: 0
```

### 8.2.3.2.5.3 create command

The **create** command creates a new DPBP. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpbp create** [OPTIONS]

OPTIONS:

**--container**=<container\_name>

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPBP:

```
$ restool dpbp create
dpbp.2 is created under dprc.1
```

### 8.2.3.2.5.4 create command

The **create** command creates a new DPBP. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpbp create** [OPTIONS]

OPTIONS:

**--container**=<container\_name>

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPBP under container dprc.3:

```
$ restool dpbp create --container=dprc.3
dpbp.2 is created under dprc.3
```

### 8.2.3.2.5.5 destroy command

The **destroy** command destroys a DPBP.

SYNTAX:

**restool dpbp destroy** <dpbp-object>

ARGUMENTS :

<dpbp-object>

Specifies which DPBP to destroy.

EXAMPLE:

```
$ restool dpbp destroy dpbp.2
dpbp.2 is destroyed
```

### 8.2.3.2.6 DPCON Commands

text

#### 8.2.3.2.6.1 help command

The **help** command displays usage information for the DPCON object.

SYNTAX:

**restool dpcon help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpcon help
usage: restool dpcon <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPCON object.
create - creates a DPCON under the root DPRC
destroy - destroys a DPCON under the root DPRC

For command-specific help, use the --help option available for each command.
```

#### 8.2.3.2.6.2 info command

The **info** command displays detailed information about a specific dpcon object.

SYNTAX:

**restool dpcon info** <dpcon-object> [--verbose]

ARGUMENTS :

<dpcon-object>

Specifies which dpcon object to show detailed info for. The dpcon-object argument is a string specifying the object name—e.g. “dpcon.8”

**--verbose**

Shows extended/verbose information about the object

**EXAMPLE:**

```
$ restool dpcon info dpcon.1
dpcon version: 2.0
dpcon id: 1
plugged state: plugged
qbman channel id to be used by dequeue operation: 40
number of priorities for the DPCON channel: 8
```

### 8.2.3.2.6.3 create command

The **create** command creates a new DPCON. The name/id of the object created is displayed to stdout.

**SYNTAX:**

**restool dpcon create** [OPTIONS]

**OPTIONS :**

**--num-priorities=<number>**

Specifies the number of priorities, valid values are 1-8. Default is 1.

**EXAMPLE:**

Create a DPCON with 4 priorities:

```
$ restool dpcon create --num-priorities=4
dpcon.8 is created under dprc.1
```

### 8.2.3.2.6.4 create command

The **create** command creates a new DPCON. The name/id of the object created is displayed to stdout.

**SYNTAX:**

**restool dpcon create** [OPTIONS]

**OPTIONS :**

**--num-priorities=<number>**

Specifies the number of priorities, valid values are 1-8. Default is 1.

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

**EXAMPLE:**

Create a DPCON with 4 priorities:

```
$ restool dpcon create --num-priorities=4
dpcon.8 is created under dprc.1
```

### 8.2.3.2.6.5 destroy command

The **destroy** command destroys a DPCON.

SYNTAX:

**restool dpcon destroy** *<dpcon-object>*

ARGUMENTS :

*<dpcon-object>*

Specifies which DPCON to destroy.

EXAMPLE:

```
$ restool dpcon destroy dpcon.9
```

dpcon.9 is destroyed

### 8.2.3.2.7 DPCI Commands

#### 8.2.3.2.7.1 help command

The **help** command displays usage information for the DPCI object.

SYNTAX:

**restool dpci help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpci help
usage: restool dpci <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPCI object.
create - creates a DPCI under the root DPRC
destroy - destroys a DPCI under the root DPRC

For command-specific help, use the --help option available for each command.
```

#### 8.2.3.2.7.2 info command

The **info** command displays detailed information about a specific dpci object.

SYNTAX:

**restool dpci info** *<dpci-object>* [--verbose]

ARGUMENTS :

*<dpci-object>*

Specifies which dpci object to show detailed info for. The dpci-object argument is a string specifying the object name—e.g. “dpci.8”

--verbose

Shows extended/verbose information about the object

#### EXAMPLE:

```
$ restool dpci info dpci.1
dpci version: 2.0
dpci id: 1
plugged state: plugged
num_of_priorities: 2
connected peer: dpci.4
peer's num_of_priorities: 2
link status: 0 - down
```

### 8.2.3.2.73 create command

The **create** command creates a new DPCI. The name/id of the object created is displayed to stdout.

#### SYNTAX:

**restool dpci create** [OPTIONS]

#### OPTIONS :

**--num-priorities=<number>**

Specifies the number of priorities, valid values are 1 or 2. Default is 1.

#### EXAMPLE:

Create a DPCI with 4 priorities:

```
$ restool dpci create --num-priorities=2
dpci.8 is created under dprc.1
```

### 8.2.3.2.74 create command

The **create** command creates a new DPCI. The name/id of the object created is displayed to stdout.

#### SYNTAX:

**restool dpci create** [OPTIONS]

#### OPTIONS :

**--num-priorities=<number>**

Specifies the number of priorities, valid values are 1 or 2. Default is 1.

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

#### EXAMPLE:

Create a DPCI with 4 priorities:

```
$ restool dpci create --num-priorities=2
dpci.8 is created under dprc.1
```

### 8.2.3.2.75 destroy command

The **destroy** command destroys a DPCI.

SYNTAX:

**restool dpci destroy** <dpqi-object>

ARGUMENTS :

<dpqi-object>

Specifies which DPCI to destroy.

EXAMPLE:

```
$ restool dpci destroy dpqi.9
```

dpqi.9 is destroyed

## 8.2.3.2.8 DPSECI Commands

### 8.2.3.2.8.1 help command

The **help** command displays usage information for the DPSECI object.

SYNTAX:

**restool dpseci help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpseci help
usage: restool dpseci <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPSECI object.
create - creates a DPSECI under the root DPCR
destroy - destroys a DPSECI under the root DPCR
For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.8.2 info command

The **info** command displays detailed information about a specific dpseci object.

SYNTAX:

**restool dpseci info** <dpseci-object> [--verbose]

ARGUMENTS :

<dpseci-object>

Specifies which dpseci object to show detailed info for. The dpseci-object argument is a string specifying the object name—e.g. “dpseci.8”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpseci info dpseci.1
dpseci version: 2.0
dpseci id: 1
```

```
plugged state: plugged
number of priorities: 1
dpcci id: 1
```

### 8.2.3.2.8.3 create command

The **create** command creates a new DPSECI. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpseci create** [OPTIONS]

OPTIONS :

**--priorities**=<priority1,priority2>

DPSEC support 2 priorities that can be individually set. Valid values for <priority1> and <priority2> are 1-8. Default is 1.

EXAMPLE:

Create a DPSECI with 4 priorities:

```
$ restool dpseci create --priorities=2,4
dpseci.9 is created under dprc.1
```

### 8.2.3.2.8.4 create command

The **create** command creates a new DPSECI. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpseci create** [OPTIONS]

OPTIONS :

**--priorities**=<priority1,priority2>

DPSEC support 2 priorities that can be individually set. Valid values for <priority1> and <priority2> are 1-8. Default is 1.

**--container**=<container\_name>

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPSECI with 4 priorities:

```
$ restool dpseci create --priorities=2,4
dpseci.9 is created under dprc.1
```

### 8.2.3.2.8.5 destroy command

The **destroy** command destroys a DPSECI.

SYNTAX:

**restool dpseci destroy** <dpseci-object>

ARGUMENTS :

<dpseci-object>

Specifies which DPSECI to destroy.

EXAMPLE:

```
$ restool dpseci destroy dpseci.9
```

dpseci.9 is destroyed

## 8.2.3.2.9 DPDMUX Commands

### 8.2.3.2.9.1 help command

The **help** command displays usage information for the DPDMUX object.

SYNTAX:

**restool dpdmux help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpdmux help
usage: restool dpdmux <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPDMUX object.
create - creates a DPDMUX under the root DPRC
destroy - destroys a DPDMUX under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.9.2 info command

The **info** command displays detailed information about a specific dpdmux object.

SYNTAX:

**restool dpdmux info <dpdmux-object> [--verbose]**

ARGUMENTS :

**<dpdmux-object>**

Specifies which dpdmux object to show detailed info for. The dpdmux-object

argument is a string specifying the object name—e.g. “dpdmux.2”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpdmux info dpdmux.0
dpdmux version: 4.1
dpdmux id: 0
plugged state: plugged
endpoints:
endpoint state: 0
 interface 0: dpmac.1, link is down
endpoint state: 0
```



```

 interface 1: dpni.0, link is down
endpoint state: 0
 interface 2: dpni.1, link is down
dpdmux_attr.options value is: 0x2
 DPDMUX_OPT_BRIDGE_EN
DPDMUX address table method: DPDMUX_METHOD_MAC
DPDMUX manipulation type: DPDMUX_MANIP_NONE
number of interfaces (excluding the uplink interface): 3
DPDMUX frame storage memory size: 0
control interface ID: 0

```

### 8.2.3.2.9.3 create command

The create command creates a new DPDMUX. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpdmux create** --num-ifs=<number> [OPTIONS]

ARGUMENTS :

**--num-ifs=<number>**

Number of virtual interfaces (excluding the uplink interface).

OPTIONS :

**--method=<dmatrix\_method>**

Where <dmatrix\_method> defines the method of the DPDMUX address table. A valid value is one of the following:

```

DPDMUX_METHOD_NONE
DPDMUX_METHOD_C_VLAN_MAC
DPDMUX_METHOD_MAC
DPDMUX_METHOD_C_VLAN
DPDMUX_METHOD_S_VLAN

```

Default is DPDMUX\_METHOD\_C\_VLAN\_MAC

**--manip=<manip>**

Where <manip> defines the DPDMUX required manipulation operation. A valid value is one of the following:

```

DPDMUX_MANIP_NONE

DPDMUX_MANIP_ADD_REMOVE_S_VLAN

```

Default is DPDMUX\_MANIP\_NONE

**--options=<options-mask>**

```

DPDMUX_OPT_BRIDGE_EN

```

Default is 0 (don't set any options)

**--max-dmat-entries=<number>**

max entries in DPDMUX address table. Default is 64.

**--max-mc-groups=<number>**

Number of multicast groups in DPDMUX table. Default is 32 groups.

#### EXAMPLE:

Create a DPDMUX with all default options:

```
$ restool dpdmux create --num-ifs=4
dpdmux.11 is created under dprc.1
```

### 8.2.3.2.9.4 create command

The create command creates a new DPDMUX. The name/id of the object created is displayed to stdout.

#### SYNTAX:

**restool dpdmux create** --num-ifs=<number> [OPTIONS]

#### ARGUMENTS :

**--num-ifs=<number>**

Number of virtual interfaces (excluding the uplink interface).

#### OPTIONS :

**--method=<dmatrix\_method>**

Where <dmatrix\_method> defines the method of the DPDMUX address table. A valid value is one of the following:

```
DPDMUX_METHOD_NONE
DPDMUX_METHOD_C_VLAN_MAC
DPDMUX_METHOD_MAC
DPDMUX_METHOD_C_VLAN
DPDMUX_METHOD_S_VLAN
```

Default is DPDMUX\_METHOD\_C\_VLAN\_MAC

**--manip=<manip>**

Where <manip> defines the DPDMUX required manipulation operation. A valid value is one of the following:

```
DPDMUX_MANIP_NONE
```

```
DPDMUX_MANIP_ADD_REMOVE_S_VLAN
```

Default is DPDMUX\_MANIP\_NONE

**--options=<options-mask>**

```
DPDMUX_OPT_BRIDGE_EN
```

Default is 0 (don't set any options)

**--max-dmat-entries=<number>**

max entries in DPDMUX address table. Default is 64.

**--max-mc-groups=<number>**

Number of multicast groups in DPDMUX table. Default is 32 groups.

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPDMUX with all default options under dprc.2:

```
$ restool dpdmux create --num-ifs=4 --container=dprc.2
dpdmux.11 is created under dprc.2
```

### 8.2.3.2.9.5 destroy command

The **destroy** command destroys a DPDMUX.

SYNTAX:

**restool dpdmux destroy <dpdmux-object>**

ARGUMENTS :

**<dpdmux-object>**

Specifies which DPDMUX to destroy.

EXAMPLE:

```
$ restool dpdmux destr
```

## 8.2.3.2.10 DPMCP Commands

### 8.2.3.2.10.1 help command

The **help** command displays usage information for the DPMCP object.

SYNTAX:

**restool dpmcp help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpmcp help
usage: restool dpmcp <command> [--help] [ARGS...]
Where <command> can be:
 info - displays detailed information about a DPMCP object.
 create - creates a DPMCP under the root DPRC
 destroy - destroys a DPMCP under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.10.2 info command

The **info** command displays detailed information about a specific dpmcp object.

SYNTAX:

**restool dpmcp info <dpmcp-object> [--verbose]**

ARGUMENTS :

#### <dpmcp-object>

Specifies which dpmcp object to show detailed info for. The dpmcp-object argument is a string specifying the object name—e.g. “dpmcp.8”

#### --verbose

Shows extended/verbose information about the object

#### EXAMPLE:

```
$ restool dpmcp info dpmcp.5
dpmcp version: 1.0
dpmcp object id/portal id: 5
plugged state: plugged
```

### 8.2.3.2.10.3 create command

The **create** command creates a new DPMCP. The name/id of the object created is displayed to stdout.

#### SYNTAX:

##### **restool dpmcp create**

#### EXAMPLE:

Create a DPMCP:

```
$ restool dpmcp create
dpmcp.15 is created under dprc.1
$ restool dpmcp create
MC error: No resource (status 0x8)
// when you see this error, it usually means no free portal available at this time.
```

### 8.2.3.2.10.4 create command

The **create** command creates a new DPMCP. The name/id of the object created is displayed to stdout.

#### SYNTAX:

##### **restool dpmcp create [OPTIONS]**

#### **--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

#### EXAMPLE:

Create a DPMCP:

```
$ restool dpmcp create
dpmcp.15 is created under dprc.1
$ restool dpmcp create
MC error: No resource (status 0x8)
// when you see this error, it usually means no free portal available at this time.
```

### 8.2.3.2.10.5 destroy command

The **destroy** command destroys a DPMCP.

#### SYNTAX:

**restool dpmcp destroy** <dpmcp-object>

ARGUMENTS :

<dpmcp-object>

Specifies which DPMCP to destroy.

EXAMPLE:

```
$ restool dpmcp destroy dpmcp.9
dpmcp.9 is destroyed
```

## 8.2.3.2.11 DPMAC Commands

### 8.2.3.2.11.1 help command

The **help** command displays usage information for the DPMAC object.

SYNTAX:

**restool dpmac help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpmac help
usage: restool dpmac <command> [--help] [ARGS...]
Where <command> can be:
 info - displays detailed information about a DPMAC object.
 create - creates a DPMAC under the root DPRC
 destroy - destroys a DPMAC under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.11.2 info command

The **info** command displays detailed information about a specific dpmac object.

SYNTAX:

**restool dpmac info** <dpmac-object> [--verbose]

ARGUMENTS:

<dpmac-object>

Specifies which dpmac object to show detailed info for. The dpmac-object argument is a string specifying the object name—e.g. “dpmac.8”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpmac info dpmac.5
dpmcp version: 2.0
dpmac object id/phy id: 5
plugged state: plugged
```

### 8.2.3.2.11.3 create command

The **create** command creates a new DPMAC. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpmac create --mac-id=<number>**

**--mac-id=<number>**

Specifies the mac id.

EXAMPLE:

Create a DPMAC with valid portal id:

```
$ restool dpmac create --mac-id=15
dpmac.15 is created under dprc.1
```

### 8.2.3.2.11.4 create command

The **create** command creates a new DPMAC. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpmac create --mac-id=<number> [OPTIONS]**

**--mac-id=<number>**

Specifies the mac id.

OPTIONS:

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPMAC with valid portal id:

```
$ restool dpmac create --mac-id=6
dpmac.6 is created under dprc.1
```

### 8.2.3.2.11.5 destroy command

The **destroy** command destroys a DPMAC.

SYNTAX:

**restool dpmac destroy <dpmac-object>**

ARGUMENTS :

**<dpmac-object>**

Specifies which DPMAC to destroy.

EXAMPLE:

```
$ restool dpmac destroy dpmac.9
dpmac.9 is destroyed
```

## 8.2.3.2.12 DPDCEI Commands

### 8.2.3.2.12.1 help command

The **help** command displays usage information for the DPDCEI object.

SYNTAX:

**restool dpdcei help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpdcei help
usage: restool dpdcei <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPDCEI object.
create - creates a DPDCEI under the root DPRC
destroy - destroys a DPDCEI under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.12.2 info command

The **info** command displays detailed information about a specific dpdcei object.

SYNTAX:

**restool dpdcei info <dpdcei-object> [--verbose]**

ARGUMENTS :

*<dpdcei-object>*

Specifies which dpdcei object to show detailed info for. The dpdcei-object argument

is a string specifying the object name, e.g. "dpdcei.2"

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpdcei info dpdcei.5
dpdcei version: 0.0
dpdcei id: 5
plugged state: plugged
DPDCEI engine: DPDCEI_ENGINE_COMPRESSION
```

### 8.2.3.2.12.3 create command

The **create** command creates a new DPDCEI. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpdcei create --engine=<engine> --priority=<number>**

**--engine=<engine>**

Compression or decompression engine to be selected.

A valid value is one of the following:

DPDCEI\_ENGINE\_COMPRESSION

DPDCEI\_ENGINE\_DECOMPRESSION

**--priority=<number>**

Priority for DCE hardware processing (valid values 1-8)

EXAMPLE:

Create a DPDCEI:

```
$ restool dpdcei create --engine=DPDCEI_ENGINE_COMPRESSION --priority=2
dpdcei.0 is created under dprc.1
$ restool dpdcei create --engine=DPDCEI_ENGINE_COMPRESSION --priority=3
dpdcei.1 is created under dprc.1
```

### 8.2.3.2.12.4 create command

The **create** command creates a new DPDCEI. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpdcei create --engine=<engine> --priority=<number> [OPTIONS]**

**--engine=<engine>**

Compression or decompression engine to be selected.

A valid value is one of the following:

DPDCEI\_ENGINE\_COMPRESSION

DPDCEI\_ENGINE\_DECOMPRESSION

**--priority=<number>**

Priority for DCE hardware processing (valid values 1-8)

OPTIONS:

**--container=<container\_name>**

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.

If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPDCEI:

```
$ restool dpdcei create --engine=DPDCEI_ENGINE_COMPRESSION --priority=2
dpdcei.0 is created under dprc.1
$ restool dpdcei create --engine=DPDCEI_ENGINE_COMPRESSION --priority=3
dpdcei.1 is created under dprc.1
```

### 8.2.3.2.12.5 destroy command

The **destroy** command destroys a DPDCEI.



SYNTAX:

**restool dpdcei destroy** <dpdcei-object>

ARGUMENTS :

<dpdcei-object>

Specifies which DPDCEI to destroy.

EXAMPLE:

```
$ restool dpdcei destroy dpdcei.9
dpdcei.9 is destroyed
```

## 8.2.3.2.13 DPAIOP Commands

### 8.2.3.2.13.1 help command

The **help** command displays usage information for the DPAIOP object.

SYNTAX:

**restool dpaiop help**

ARGUMENTS:

none

EXAMPLE:

```
$ restool dpaiop help
usage: restool dpaiop <command> [--help] [ARGS...]
Where <command> can be:
info - displays detailed information about a DPAIOP object.
create - creates a DPAIOP under the root DPRC
destroy - destroys a DPAIOP under the root DPRC

For command-specific help, use the --help option available for each command.
```

### 8.2.3.2.13.2 info command

The **info** command displays detailed information about a specific dpaiop object.

SYNTAX:

**restool dpaiop info** <dpaiop-object> [--verbose]

ARGUMENTS :

<dpaiop-object>

Specifies which dpaiop object to show detailed info for. The *dpaiop-object* argument is a string specifying the object name—e.g. “dpaiop.8”

**--verbose**

Shows extended/verbose information about the object

EXAMPLE:

```
$ restool dpaiop info dpaiop.5
```

dpmcp version: 1.0

dpmcp id: 5

plugged state: plugged

dpaio server layer version: 2.1.3

DPAIOP state: DPAIOP\_STATE\_RUNNING

### 8.2.3.2.13.3 create command

The **create** command creates a new DPAIOP. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpaio create** --aiop-id=<number> --aiop-container=<container-name>

ARGUMENTS :

**--aiop-container**=<container-name>

Specifies the AIOP container name, e.g. dprc.3, dprc.4, etc.

OPTIONS :

**--aiop-id**=<number>

Specifies the AIOP ID. Currently aiop container could only hold one dpaio. Valid number is 0. Default number is 0.

EXAMPLE:

Create a DPAIOP:

```
$ restool dpaio create --aiop-id=0 --aiop-container=dprc.3
dpaio.0 is created under dprc.3

$ restool dpaio create --aiop-container=dprc.3
dpaio.0 is created under dprc.3
```

### 8.2.3.2.13.4 create command

The **create** command creates a new DPAIOP. The name/id of the object created is displayed to stdout.

SYNTAX:

**restool dpaio create** --aiop-container=<container-name> [OPTIONS]

ARGUMENTS :

**--aiop-container**=<container-name>

Specifies the AIOP container name, e.g. dprc.3, dprc.4, etc.

OPTIONS :

**--container**=<container-name>

Specifies the parent container name. e.g. dprc.2, dprc.3 etc.  
If it is not specified, the new object will be created under the default dprc.

EXAMPLE:

Create a DPAIOP:

```
$ restool dpaiop create --aiop-container=dprc.3
dpaiop.0 is created under dprc.1
```

Create a DPAIOP as a child object of dprc.2:

```
$ restool dpaiop create --aiop-container=dprc.3 --container=dprc.2
dpaiop.0 is created under dprc.2
```

### 8.2.3.2.13.5 destroy command

The **destroy** command destroys a DPAIOP.

SYNTAX:

**restool dpaiop destroy** <dpaiop-object>

ARGUMENTS :

<dpaiop-object>

Specifies which DPAIOP to destroy.

EXAMPLE:

```
$ restool dpio destroy dpaiop.9
dpaiop.9 is destroyed
```

## 8.2.4 DPAA2 User Manual

DPAA2 is a hardware-level networking architecture found on some NXP SoCs. This section provides technical information on this architecture mainly for software developers.

[Click here](#) to access the DPAA2 User Manual PDF.

## 8.2.5 DPAA2 API Reference Manual

[Click here](#) to access the DPAA2 API Reference Manual PDF.

## 8.2.6 Backplane Support on Layerscape

### 8.2.6.1 Overview

This section describes how to enable backplane support for Layerscape devices with embedded support for this type of connection.

Ethernet operation over electrical backplanes, also referred to as “Backplane Ethernet,” combines the IEEE 802.3 Media Access Control (MAC) and MAC Control sublayers with a family of Physical Layers defined to support operation over a modular chassis backplane. Usually, there is no external PHY involved and the connection is made at the SoC’s PCS (Physical Coding Sublayer) level. Based on the link quality, a signal equalization is required. In cases where the link is realized based on passive direct attach cables, the link may need to be established with only the default (recommended) parameters for equalization. The standard states that a start-up algorithm should be in place in order to get the link up.

#### 8.2.6.1.1 10GBase-KR Support on Layerscape Platforms

Layerscape devices come with embedded support for backplane connections at different baud rates. 10G is present in custom boards of LS1088A, LS2088A, and LS1046A devices.

The enablement of backplane support is done in two parts. One refers to support from the device tree while the other is contained in the Linux kernel driver.

In the device tree, the following value is valid `backplane-mode`:

- `10gbase-kr`

In the Linux kernel driver, the implementation is different depending on each of the four different cases. However, the following changes are common for all:

- Advertise the link partner with the correct working mode.
- Put the lane in the correct mode (10GBase-KR).
- Use the recommended (if it is the case) parameters for pre- and post-tap coefficients in the lane initialization phase. This affects the starting point of the algorithm.
- Optionally, update the constraint relation between tap coefficients if this is needed.

### 8.2.6.1.2 Physical Layer Signaling System

The Backplane Ethernet extends the family of 10GBASE-R Physical Layer signaling systems to include the 10GBASE-KR. This embodiment specifies 10 Gb/s operation over two differential, controlled impedance pairs of traces (one pair for transmit, one pair for receive). This system employs the 10GBASE-R PCS, the serial PMA, and the 10GBASE-KR PMD sublayers.

The 10GBASE-KR PMD's control function implements the 10GBASE-KR start-up protocol. This protocol facilitates timing recovery and equalization while also providing a mechanism through which the receiver can tune the transmit equalizer to optimize performance over the backplane interconnect. The 10GBASE-KR PHY may optionally include 10GBASE-R Forward Error Correction (FEC).

Details about the aforementioned layers can be found in Clause 49, 51 and 74 of the [IEEE Std 802.3](#).

### 8.2.6.1.3 Auto-negotiation

Auto-negotiation allows the devices at both ends of a link segment to advertise abilities, acknowledge receipt, and discover the common modes of operation that both devices share. It also rejects the use of operational modes not shared by both devices. Auto-negotiation does not test link segment characteristics.

### 8.2.6.1.4 Link Training

Link Training occurs after auto-negotiation has determined the link to be a 10GBase-KR, but before auto-negotiation is done. It continuously exchanges messages (training frames) between the local and the remote device as part of the start-up phase. Link training also tunes the analog parameters of the remote and local SerDes transmitter to improve the link quality. Both LP (link partner/remote device) and LD (local device) perform link training in parallel. Link training stops when both sides decide that the link is passable. Then the link is considered up.

## 8.2.6.2 Enable Backplane Support on Layerscape

### 8.2.6.2.1 Setup

#### Hardware Setup

- Two custom boards (LS1088A, LS2088A, LS1046A) with no XFI retimers
- Passive direct attach cable (l <= 1M)

#### Software Setup

- Linux kernel with backplane support enabled

- Device tree for custom boards (LS1088A, LS2088A, LS1046A) with backplane PHY devices

#### NOTE

The custom boards (LS1088A, LS2088A, LS1046A) were used for this implementation/ demonstration. You may use any custom board that enables access to the Backplane Ethernet feature.

### 8.2.6.2.2 Enable Backplane Connection from MC

This step is required only for devices based on DPAA2 architecture.

Use `MAC_LINK_TYPE_BACKPLANE` for all ports that will be used for backplane connections. In order to do that in the MC data path configuration file, add an entry like below for all ports used:

```
board_info {
 ports {
 ...
 mac@1 {
 link_type = "MAC_LINK_TYPE_BACKPLANE";
 };
 ...
 };
};
```

Deploy this configuration file on the target board as per Data Path Configuration chapter from [DPAA2 User Manual](#).

NOTE: This is a very important step and omitting it can lead to an unreliable backplane connection. Random link-down or link-up events can be experienced. This is due to a concurrent access to MDIO bus between MC core (MC firmware) and GPP core (Linux kernel).

### 8.2.6.2.3 Enable Backplane Support in Linux Kernel

#### 8.2.6.2.3.1 Enable Backplane PHY Driver

Enable backplane support from:

```
Device Drivers->Network device support->PHY device support and infrastructure->FSL_BACKPLANE
```

#### 8.2.6.2.3.2 Add Backplane PHY Devices in Device Tree

##### 8.2.6.2.3.2.1 SerDes Device and Internal MDIO Buses

Normally the SerDes device and all internal MDIO buses should be listed in the SoC's common device tree source file:

```
<linux_kernel_repo>/arch/arm64/boot/dts/freescale/fsl-<device>xa.dtsi file.
```

To see if the SerDes module is listed, examine a block like the one below:

```
serdes1: serdes@1ea0000 {
 reg = <0x0 0x1ea0000 0 0x00002000>;
 compatible = "fsl,serdes-28g";
};
```

If the SerDes module is listed then it means that the `serdes1` (label for SerDes node) is registered and can be used. The only client of this node is the kernel backplane PHY driver which uses the node's unit address as a base address. The base address is mapped in the SOC's memory space to further access specific MDIO registers used to control the backplane connection.

In the DTS, there must be a `serdes1` node like the one represented above. If the node is not present, then it must be added. The device base address is listed in SoC's CCSR memory map. One thing that should be considered is endianness of SerDes module, which can be different than that of the GPP. In this case, the module's registers must be accessed using the endianness. Currently, the Linux kernel backplane PHY driver does not support access dependent on target endianness. One way to do this will be to use the endianness attribute in the device tree. This way the driver can be used on different targets without changes.

Next look after internal MDIO buses listed in the device tree. See the block below as an example:

```
pcs_mdio1: mdio@0x8c07000 {
 compatible = "fsl, fman-memac-mdio";
 reg = <0x0 0x8c07000 0x0 0x1000>;
 device_type = "mdio";
 little-endian;
};
```

The block above shows that `pcs_mdio1` is listed in the device tree. The unit address of this node (`0x8c7000`) is the WRIOP internal physical port 1 base address as it is mapped in the SoC memory space. The address `0x8c0000` is the WRIOP port block base address as it is listed in SoC reference manual. The address `0x7000` is the physical port offset in the WRIOP internal memory map. All `pcs_mdio` ports have an offset of `0x4000` between them, so the next port will be located at `0xb000` and so on. The attribute `fsl, fman-memac-mdio` means that the FSL MDIO driver will be used to access this MDIO bus. It is required to use a dedicated MDIO bus driver to access internal MDIO buses, because it uses proprietary MDIO control registers block and offset. See the [DPAA2 User Manual](#) for details about MDIO registers block.

The kernel MDIO driver used is:

```
<linux_kernel_repo>/drivers/net/Ethernet/freescale/xgmac_mdio.c
```

Internal MDIO buses should be listed for all PCS ports that support backplane KR connection in the device tree. This is because for every port used, the management registers are accessed through the MDIO bus. See DPAA2 architecture for details on how internal MDIO registers block is mapped for every physical port and MDIO registers subchapter of SerDes chapter from SoC reference manual.

If no internal MDIO bus is listed then add one internal MDIO bus for every PCS port target that will be used in a backplane connection.

### 8.2.6.2.3.2.2 Backplane PHY Devices

PCS ports are specific to each board. Backplane PHY devices should be added in board-specific device trees:

```
<linux_kernel_repo>/arch/arm64/boot/dts/freescale/fsl-<device>-<qds, rdb>.dts.
```

A backplane PHY device is registered on an internal MDIO bus. The block below is an example:

```
&pcs_mdio1 {
 pcs_phys1: ethernet-phy@0 {
 backplane-mode = "10gbase-kr";
 compatible = "ethernet-phy-ieee802.3-c45";
 reg = <0x0>;
 fsl, lane-handle = *lt;&serdes1>;
 fsl, lane-reg = <0x9c0 0x40>; /* lane H */
 };
};
```

Pcs-phys1 is listed on the MDIO bus and should be discovered when this bus is probed. The kernel backplane PHY driver should also register a PHY driver using PHY hardware ID (read via MDIO bus).

The `backplane-mode` attribute tells the kernel backplane PHY driver how to configure a specific SerDes lane. Currently, a SerDes lane can be configured as: 10gbase-kr

The `fsl, lane-handle` attribute is used to identify which SerDes lane the PCS port belongs to. In this case “serdes1” is used.

The `fsl, lane-reg` attribute is used to identify the SerDes lane used to send and receive data. 0x9c0 is the lane H offset in the SerDes1 internal memory map. See each platform's SoC Reference Manual for details and to find the other lane's offsets.

For LS2088A boards backplane PHY devices are added already for use with lane from H to E. Note that for SerDes 1 lane are numbered in the reversed order compared to WRIO physical ports and MACs.

If the backplane PHY device is not registered on internal MDIO buses for a specific board, then it can be added in the DTS.

#### 8.2.6.2.3.2.3 Connect with Backplane PHY Device Handle

Because the kernel PHY driver is instantiated by the kernel MAC driver, there should be a specified connection between the MAC and a specific PHY in the device tree. In the following example, the backplane PHY from SerDes lane H is used:

```
&dpmac1 {
 phy-handle = <&pcs_phy1>;
};
```

### 8.2.6.2.4 SerDes Setup

- Enable XFI protocol on SerDes lane - reset configuration word.
- Initialize the SerDes lane registers for the mode:

```
- Ethernet 10GBASE-KR
```

- Check the link capabilities with AN - software.
- Train the link - software.

### 8.2.6.2.5 Board Configuration

#### HW Adjustments

XFI retimers soldered on boards must be removed and PCS output signals should be routed directly to SFP+ cages pins. This is a very important operation and should be carried out carefully.

On SerDes1 module, XFI/10GBase-KR protocol will be activated on all eight lanes using 0x2a for protocol selection bits.

#### Connection Cables

Connect with passive direct attach cable.

Two custom boards (LS1088A, LS2088A, LS1046A) will be connected back to back with a passive direct attach copper cable, with a maximum length of 1m (for ex SFP-H10GB-CU1M).

## 8.2.6.3 Use Cases

### ping

In order to run a backplane *ping use case*, two boards must be connected back to back with a passive direct attach copper cable. Start MC with specified DPC file. Apply DPL using `fsl_mc apply dpl` command from U-boot and then boot Linux on both boards. After booting Linux, the interfaces must be configured properly for the two ports connected together using two IP addresses from the same IP class.

For example, use:

- On first board: `ifconfig ni0 1.1.1.1`
- On second board: `ifconfig ni0 1.1.1.2`

Once the interfaces are configured, traffic can be sent between the two the boards through the backplane link:

- On first board: `ping 1.1.1.2`
- On second board: `ping 1.1.1.1`

### netperf

the backplane *netperf use case* is similar to the ping usecase described above, and it is used for backplane performance benchmark. The board configuration must be done identically as described above. The difference is how traffic is sent between the two boards.

For example using UDP streams:

- On first board: `netperf -H 1.1.1.2 -l 60 -t UDP_STREAM -N &`
- On second board: `netperf -H 1.1.1.1 -l 60 -t UDP_STREAM -N &`

## 8.2.7 Soft Parser Support

### 8.2.7.1 Soft Parser Configuration Tool

#### 8.2.7.1.1 Introduction

This is a User Guide for SPC (Soft Parser Configuration) tool. The SPC tool allow users to extend the hard parser's capabilities to support custom protocols that are not supported by the hardware parser.

#### 8.2.7.1.2 Defining a custom protocol

The soft parser tool defines custom protocols using xml files, based on the NetPDL standard. It is important to note that even though the language used in the xml files is based on NetPDL, it doesn't follow its rules strictly; therefore, it is highly recommended to read this document.

XML rules: The xml document follows standard xml rules. The document is composed of several elements. Each element begins with a start tag and can contain attributes or child elements. If the element contains child elements, it must have a corresponding end-tag after them. An element without child elements, must end with a slash (/). Note that element and attribute names are always case sensitive.

In the custom protocol xml these names will not contain capital letters.

Comments always begin with "`<!--`" and end with "`-->`"

For example:

```
<element attribute1="value"> <!-- this is a comment -->
<child-element myAttribute="4"/>
</element>
<another-element attribute2="value2"/>
```



### 8.2.7.1.2.1 The <netpdl> element

The custom protocols document always begins with the <netpdl> root element. The end tag of the *netpdl* element should appear in the end of the document.

Attributes: No required attributes

Child elements: protocol

For example:

```
<netpdl>
...
</netpdl>
```

### 8.2.7.1.2.2 The <protocol> element

Each document can define one or more protocols. Every protocol should be defined separately within its own protocol element.

Attributes:

- Name – Required, possible value: string.

Defines a unique name for each protocol.

- Longname – Optional attribute, possible value: string.

Defines the name of the protocol for display purposes.

- Prevproto – Required, possible value: protocol name, the following previous protocols are supported:

The following table lists the protocols supported in the prevproto attribute:

**Table 45. Protocols supported in the prevproto attribute**

| Protocol | Layer |
|----------|-------|
| ethernet | 2     |
| llc_snap | 2     |
| vlan     | 2     |
| vxlان    | 2     |
| pppoe    | 2     |
| mpls     | 2     |
| arp      | 2     |
| ip       | 3     |
| ipv4     | 3     |
| ipv6     | 3     |
| gre      | 3     |
| minencap | 3     |
| otherl3* | 3     |
| tcp      | 4     |

*Table continues on the next page...*

**Table 45. Protocols supported in the prevproto attribute (continued)**

| Protocol   | Layer |
|------------|-------|
| udp        | 4     |
| ipsec_ah   | 4     |
| ipsec_esp  | 4     |
| sctp       | 4     |
| dccp       | 4     |
| otherl4*   | 4     |
| gtp        | 5     |
| esp        | 5     |
| finalshell | 5     |
| otherl5*   | 5     |

The prevproto attribute defines the previous protocol. The current custom protocol will be invoked only after the parser encounters the defined previous protocol. In the before section the soft parser will have access to all the fields defined in the previous protocol.

**NOTE**

\* The softparser xml has a somewhat different structure and behavior when otherl3 or otherl4 are defined as the previous protocol. See Section 2.2.1

**Child Elements:**

Format, execute-code

**Example:**

```
<protocol name="gtpu" longname="GTP-U" prevproto="#udp">
...
</protocol>
<protocol name="tcpExt" longname="tcp extension" prevproto="#tcp">
...
</protocol>
```

### 8.2.7.1.2.2.1 Use of “otherl3/otherl4/otherl5” as previous protocols

When otherl3 or otherl4 are defined as previous protocols (in the prevproto attribute of the protocol element) the custom protocol and previous protocol refer to the same position in the frame window. The otherl3 and otherl4 protocols have no defined size or defined fields, they are considered only as entry points for the softparser (or as termination points) and therefore they share the same starting offset with the custom protocol.

Since the otherl3/otherl4 only act as a link to the software parser, and hold no separate header which can be parsed, the before element cannot exist when these protocols are defined as the previous protocol.

### 8.2.7.1.2.3 The <format> element

The `format` element defines the format of the protocol header.

**Attribues:** None

**Child Elements:** Field

#### 8.2.7.1.2.4 The <fields> element

The `fields` element defines the fields of the protocol header.

**Attributes:** None

**Child Elements:** Field

#### 8.2.7.1.2.5 The <field> element

The `field` element defines a specific field in the custom protocol.

**Attributes:**

- **Type** – Required, possible values: "fixed" (for fields of byte-length size), "bit" (for fields of bit-length size).
- **Size** – Required, possible values: integer. The size of the field is in bytes.
- **Name** – Required, possible values: string. Unique name for the field.
- **longname** – Optional, possible values: string. Defines the name of the field for display purposes.
- **Mask** - Required only for bit fields, possible values: integer. Defines the specific bits in the current bytes which belong to this field.

The field elements appear one after the other and define the protocol's header frame. The first field begins in the first byte of the custom protocol's frame header, and its size is determined by the size attribute. The following fields follow the following rules:

- A fixed field or a field following a fixed field begins in the next byte which is the previous field's offset + the previous field's size.
- A bit field following a bit field begins in the next byte only if the last bit in the previous field's mask is 1.
- If two fields share the same offset (possible only when both fields are bitfields and the mask of the first field doesn't end with 1), they should have the same value in the size attribute.

**Example:**

```
<format>
<fields>
<field type="bit" name="flags" mask="0xE0" size="1"/>
<field type="bit" name="pt" mask="0x80" size="1"/>
<field type="bit" name="version" mask="0x07" size="1"/>
<field type="fixed" name="mtype" size="1"/>
<field type="fixed" name="length" size="2"/>
</fields>
</format>

<format>
<fields>
<field type="bit" name="version" mask="0xE0" size="1"/>
<field type="bit" name="pt" mask="0x10" size="1"/>
<field type="bit" name="flags" mask="0x07" size="1"/>
<field type="bit" name="flags1" mask="0x01" size="1"/>
<field type="bit" name="flags2" mask="0x10" size="1"/>
<field type="bit" name="flags3" mask="0x02" size="1"/>
<field type="fixed" name="mtype" size="1" longname="message type"/>
<field type="fixed" name="length" size="2" />
</fields>
</format>
```

The fields will be stored in the following bit offsets in the custom protocols header:

```
Version - 0-2
Pt - 3-3
```

```
Flags - 5-7
flags1 - 15-15
flags2 - 19-19
flags3 - 22-22
mtype - 24-31
length - 32-47
```

### 8.2.7.1.2.6 The <execute-code> element

This section contains all the code which should be executed for this custom protocol once the previous protocol has been reached. This element contains two child elements, `before` and `after`. At least one of the child elements must exist. If both child elements exist, the `before` element must appear before the `after` element.

**Attributes:** None

**Child elements:** `before` and `after`.

Example:

```
<execute-code>
<before>
...
</before>
<after headersize = "8">
</after>
</execute-code>
```

### 8.2.7.1.2.7 The <before> element

This section contains code which should be executed once the previous protocol has been encountered but before ensuring that the current frame belongs to the custom protocol. In other words, this code is usually used to confirm that the next frame belongs to the custom protocol and to perform any necessary preparations that are needed before processing the custom protocol header.

When the code in this section is analyzed, the frame window still points to the previous protocol's header and therefore the `$FW` variable still accesses the previous protocol in the `before` sections and the `$headerSize` variable returns the header size of the previous protocol. It is also possible to access specific fields from the previous protocol's header but not from the current protocol.

After the softparser reaches the end of the `before` section, the frame window moves to the custom protocol (as explained in the `after` section below). If no `after` element exists, the softparser jumps back to the hardparser at end of the `before` section.

The `before` element may only appear once in the `execute-code` element, and if an `after` element exists, it must appear after the `before` element.

**Attributes:** none

**Child Elements:** `if`, `switch`, `assign`, `action`

#### NOTE

When the previous protocol is `otherl3` or `otherl4`, the previous protocol and the custom protocol are treated as the same and begin in the same offset in the frame window. Therefore, the `before` section cannot exist when the previous protocol is `otherl3` or `otherl4`, and only an `after` element can be defined. See section 2.2.1 for more details.

### 8.2.7.1.2.8 The <after> element

This section contains the code which should be executed when a frame from the current custom protocol has been encountered. In contrast to the 'before' section, in the 'after' section it is possible to access fields from the current protocol, but not from the previous protocol. In the `after` section, the `$FW` variable accesses the current custom protocol and the `$headerSize` variable returns the header size of the current custom protocol.

After the end of the section, the frame window jumps to the end of the custom protocol's header and the program jumps back to the hardparser.

The `after` element may only appear once in the `execute-code` element, and if a `before` element exists, it must appear before the `after` element.

#### Attributes:

- **headerSize** – Optional, possible values: arithmetic expression, default value: calculated according to format element.

The user can define the header size for the custom protocol in this attribute. This information is needed to return to the parser exactly after the custom protocol header. If the header size isn't specified, the SPC assumes that the fields defined in the format element are the only fields in the custom protocol header and calculates the header size according to those fields. The `$headerSize` variable in the `after` section returns the value defined in this attribute (or the value calculated by default if the attribute is missing).

**Child Elements:** `if`, `switch`, `assign`, `action`

For example:

```
<protocol name="gtp" prevproto="#udp">
<format>
<fields>
<field type="bit" name="version" mask="0xE0" size="1"/>
</fields>
</format>
<execute-code>
<before>
<assign-variable name="$GPR1" value="udp.dport"/>
<!--ILLEGAL: <assign-variable name="$GPR1" value="version" -->
<assign-variable name="$shimr" value="$headerSize"/>
<!-- shimresult now holds udp's header size -->
</before>
<after headersize="4">
<!--ILLEGAL:<assign-variable name="$GPR1" value="udp.dport"> -->
<assign-variable name="$GPR1" value="version"/>
<assign-variable name="$shimr" value="$headerSize"/>
<!-- shimresult now equals 4-->
</after>
</execute-code>
</protocol>
```

### 8.2.7.1.2.9 Elements in the before and after sections

This section describes the elements in the `before` and `after` sections.

#### 8.2.7.1.2.9.1 The <assign-variable> element

The `assign-variable` element assigns an expression to a variable.

#### Attributes:

- **name** – Required, possible values: RA variables. The name of the variable which will be assigned a value.
- **value** – Required, possible value: arithmetic expression. The expression assigned to the variable.

**Child Elements:** None

Example:

```
<assign-variable name="$shimoffset_2" value="$shimoffset_1+12"/>
```

#### 8.2.7.1.2.9.1.1 The <if> element

The *if* element makes it possible to execute parts of the code only if certain conditions are met.

**Attributes:**

- Expr – Required, possible values: logical expression. Defines the condition which should be checked before executing the code.

**Child Elements:** if-true (required), if-false

Example

```
<if expr="$shimoffset_3==1">
<if-true>
...
</if-true>
<if-false>
</if-false>
</if>
```

##### 8.2.7.1.2.9.1.1.1 <if-true>

The *if-true* element defines code which should be executed if the expression defined in the 'if' element is true.

Attributes: none

Child elements: If, switch, assign, action (same child elements as in the before/after sections)

##### 8.2.7.1.2.9.1.1.2 <if-false>

The *if-false* element defines code which should be executed if the expression defined in the 'if' element is false.

Attributes: none

Child elements: If, switch, assign, action (same child elements as in the before/after sections)

#### 8.2.7.1.2.9.1.2 The <switch> element

The *switch* element defines an expression and a set of cases with values and code which should be executed if the value equals the expression. Each 'switch' element must have at least one 'case' child element.

**Note:** Only the code of the first case which matches the expression is executed, the rest of the values will be skipped (in c language terms - a break is automatically added after the code of each case).

**Attributes:**

- expr – Required, possible values: arithmetic expression.

Defines the value being checked.

**Child Elements:** Case and Default

Example:

```
<switch expr="$ShimOffset_3+1">
<case value="2">
<assign-variable name="$GPR1[1:1]" value="0"/>
</case>
<case value="3" maxvalue="4">
<assign-variable name="$GPR1[1:1]" value="1"/>
```

```
</case>
<default>
<assign-variable name="$GPR1[1:1]" value="2">
</default>
</switch>
```

#### 8.2.7.1.2.9.1.2.1 The <case> element

The `case` element matches a value or range of values against the switch expression.

##### **Attributes:**

- **value** – Required, possible values: Integer. If the value equals the switch expression and no earlier case has been matched, the code in the case element is executed.
- **maxvalue** – Optional, possible values: Integer. If the switch expression is equals or is larger than value and the expression equals or is smaller than maxvalue, and no earlier case has been matched, the code in the case element is executed.

**Child Elements:** If, switch, assign, action (same child elements as in the before/after sections).

#### 8.2.7.1.2.9.1.2.2 The <default> element

The `default` element contains code which should be executed if the expression in the switch element wasn't matched by any of the cases.

**Attributes:** None

**Child Elements:** If, switch, assign, action (same child elements as in the before/after sections).

#### 8.2.7.1.2.9.1.3 The <action> element

Jumps out of the custom protocol.

##### **Attributes:**

- **type** – Required, possible values: currently only 'exit' is supported for this attribute.
- **nextproto** – Optional, possible values protocol name. The following tables summarizes the list of available values for this attribute:

**Table 46. Possible values for the 'nextproto' attribute**

| Protocol | Application                                |
|----------|--------------------------------------------|
| ethernet | Jump to ethernet and continue hard parsing |
| llc_snap | Jump to llc_snap and continue hard parsing |
| vlan     | Jump to vlan and continue hard parsing     |
| vlan     | Jump to vxlan and continue hard parsing    |
| pppoe    | Jump to pppoe and continue hard parsing    |
| mpls     | Jump to mpls and continue hard parsing     |
| ipv4     | Jump to ipv4 and continue hard parsing     |
| ipv6     | Jump to ipv6 and continue hard parsing     |
| gre      | Jump to gre and continue hard parsing      |
| minencap | Jump to minencap and continue hard parsing |

*Table continues on the next page...*

**Table 46. Possible values for the 'nextproto' attribute (continued)**

| Protocol               | Application                                                                                                                                                                                                                       |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| otherl3                | Jump to otherl3 and continue hard parsing                                                                                                                                                                                         |
| tcp                    | Jump to tcp and continue hard parsing                                                                                                                                                                                             |
| udp                    | Jump to udp and continue hard parsing                                                                                                                                                                                             |
| ipsec_ah               | Jump to ipsec and continue hard parsing                                                                                                                                                                                           |
| ipsec_esp              | Jump to ipsec and continue hard parsing                                                                                                                                                                                           |
| sctp                   | Jump to sctp and continue hard parsing                                                                                                                                                                                            |
| dccp                   | Jump to dccp and continue hard parsing                                                                                                                                                                                            |
| otherl4                | Jump to otherl4 and continue hard parsing                                                                                                                                                                                         |
| after_ip               | Jump to the protocol which should follow the ip protocol. The next protocol is found according to the \$nxtHdr field (for details see the table below). The advance attribute cannot be set to 'no' when using this option.       |
| after_ethernet         | Jump to the protocol which should follow the ethernet protocol. The next protocol is found according to the \$nxtHdr field (for details see the table below). The advance attribute cannot be set to 'no' when using this option. |
| after_tcp              | Jump to the protocol which should follow the TCP protocol. The next protocol is found according to the \$nxtHdr field (for details see the table below). The advance attribute cannot be set to 'no' when using this option.      |
| after_udp              | Jump to the protocol which should follow the UDP protocol. The next protocol is found according to the \$nxtHdr field (for details see the table below). The advance attribute cannot be set to 'no' when using this option.      |
| return (default value) | Return to the hard parser. Continue parsing the frame header at the same position where soft parsing started. The advance attribute can not be set to 'yes' when using this option.                                               |
| none/ end_parse        | Finish parsing the frame header, don't return to the hard parser.                                                                                                                                                                 |

**Table 47. Next protocol values when nextproto is set to 'after\_ethernet'**

| \$nxtHdr value                           | Next Protocol |
|------------------------------------------|---------------|
| 0x05DC or less                           | llc_snap      |
| 0x0800                                   | ipv4          |
| 0x0806                                   | arp           |
| 0x86dd                                   | ipv6          |
| 0x8847, 0x8848                           | mpls          |
| 0x8100, 0x88A8, ConfigTPID1, ConfigTPID2 | Vlan          |
| 0x8864                                   | Pppoe         |
| Other value                              | otherl3       |



**Table 48. Next protocol value when nextproto is set to 'after\_ip'**

| \$nxtHdr value | Next Protocol |
|----------------|---------------|
| 4              | ipv4          |
| 6              | tcp           |
| 17             | udp           |
| 33             | dccp          |
| 41             | ipv6          |
| 50, 51         | ipsec         |
| 47             | gre           |
| 55             | minencap      |
| 132            | sctp          |
| Other value    | otherl4       |

**Table 49. Next protocol values when nextproto is set to 'after\_tcp' or 'after\_udp'**

| \$nxtHdr value | Next Protocol |
|----------------|---------------|
| 2123           | GTP(GTP-C)    |
| 2152           | GTP(GTP-U)    |
| 3386           | GTP(GTP')     |
| 4500           | ESP           |
| 4789           | VXLAN         |
| Other value    | Otherl5+      |

- **advance** – Optional, possible values: "yes", "no". Default value: "yes", unless 'end\_parse' or 'return' are set in the nextproto attribute, or in case the nextproto attribute isn't set, in those cases the default value is 'no'.

The attribute specifies whether the parser should move to the next frame header before jumping. This attribute has different meanings in the before and after sections. In the before section the parser will move the FW (frame window) past the previous protocol header until it reach the header of the custom protocol. In the after section the parser will move the FW past the current custom protocol header until it reaches the header of the next protocol. The FW is advanced according to the header size.

**Notes:**

- The frame window must advance when jumping to 'after\_ethernet' or 'after\_ip' and therefore the advance attribute cannot be set to 'no' in those cases.
- The frame window cannot advance when returning to the hard parser and therefore the advance attribute cannot be set to 'yes' when nextproto is set to 'return' or not set at all.

**Example:**

```
<action type="exit" advance = "yes" nextproto="#udp"/>
```

### 8.2.7.1.3 Expressions

Expressions are constructed of operands and operators. The simplest expression may contain only one operand. Most operators are dyadic, and separate two operands (such as +, -) and some operators are monadic and operate only on the operand following them (such as *not*).

#### 8.2.7.1.3.1 Operands

The following operands exist: Numbers, variables, fields, and expressions.

##### NOTE

All operands are limited to 64 bits (8 bytes).

##### 8.2.7.1.3.1.1 Numbers

Numbers can appear in a decimal (no prefix), binary (begin with 0b), or hexadecimal (begin with 0x) format.

Numbers are always limited to a 64-bit unsigned type. However, some operators are only executed on the 32 LSB of the number. Note that immediate primitive negative numbers are not supported, for examples the number -2 cannot appear in an expression. However, artificial negative value can be created using arithmetic expressions such as 1-3 (which returns 0xffffffff).

##### 8.2.7.1.3.1.2 Fields

Fields are defined in the protocol's `format` element. There are two ways to access fields, either by typing their name directly or by typing the name of protocol where the field is defined, then the dot character and then the name of the field. In the `before` section it is possible only to access fields from the previous protocol and in the `after` section, it is possible only to access the current custom protocol's fields.

##### NOTE

If the length of the field is longer than 8 bytes we cannot access it. This can be solved either by accessing the frame directly using the \$FW variable, or by splitting the field to several shorter fields.

Field example:

```
<protocol name="gtpu" prevproto="#ethernet">
<format>
<fields>
<field type="fixed" name="example" size="2"/>
</fields>
</format>
<execute-code>
<before>
<assign-variable name="$l2r" value="ethernet.type"/>
</before>
<after>
<assign-variable name="$shimOffset_2" value="example"/> </after>
</execute-code>
</protocol>
```

##### 8.2.7.1.3.1.3 Variables

All variables begin with the \$ prefix, and their name are case insensitive. The following variables exist: Frame window, header size, prevprotoOffset, parameter array, and result array variables.

###### 8.2.7.1.3.1.3.1 Result Array Variables

These variables return specific bytes in the result array.

#### Accessing the variables:

- `$variableName` – returns the entire variable
- `$variableName[byteOffset:bytesNumber]` – Returns the bytesNumber number of bytes in the variable starting from byteOffset. This is useful to access only specific bytes in the variable. In case bytesNumber equals zero, the entire variable is returned starting from byteOffset.

**Example:** The variable `$actiondescriptor` returns result array bytes 64-71 in the results array. Typing `$actiondescriptor[2:4]`, will return result array bytes 66-69, since 66 is in offset 2 of the variable (64 is offset 0) and the size requested is 4. The variable `$actiondescriptor[3:0]` will return result array bytes 67-71, since 67 is in offset 3 of the variable, and size requested is 0 so the entire variable starting with the specified offset (3) is returned.

**Other usage:** In addition to expressions, the result array variables can also be used in the left side of the assign-variable elements which modify the result arrays values.

The following results array variables exist:

**Table 50.**

| <u>Variable Name</u>                       | <u>Bytes referred to in the Result Array</u> |
|--------------------------------------------|----------------------------------------------|
| <code>gpr1</code>                          | 0-7                                          |
| <code>gpr2*</code>                         | 8-15                                         |
| <code>nxthdr</code>                        | 16-17                                        |
| <code>fafext</code>                        | 18-19                                        |
| <code>fafflags</code>                      | 20-31                                        |
| <code>shimoffset_1</code>                  | 32-32                                        |
| <code>shimoffset_2</code>                  | 33-33                                        |
| <code>ip_pidoffset</code>                  | 34-34                                        |
| <code>ethoffset</code>                     | 35-35                                        |
| <code>llc_snapoffset</code>                | 36-36                                        |
| <code>vlantcioffset_1</code>               | 37-37                                        |
| <code>vlantcioffset_n</code>               | 38-38                                        |
| <code>lastetypeoffset</code>               | 39-39                                        |
| <code>pppoeoffset</code>                   | 40-40                                        |
| <code>mplsoffset_1</code>                  | 41-41                                        |
| <code>mplsoffset_n</code>                  | 42-42                                        |
| <code>arpoffset</code>                     | 43-43                                        |
| <i>Table continues on the next page...</i> |                                              |

**Table 50. (continued)**

|                                            |         |
|--------------------------------------------|---------|
| ipoffset_1                                 | 43-43   |
| ipoffset_n                                 | 44-44   |
| minencapoffset                             | 44-44   |
| greoffset                                  | 45-45   |
| l4offset                                   | 46-46   |
| gtpoffset                                  | 47-47   |
| esppoffset                                 | 47-47   |
| ipsecoffset                                | 47-47   |
| routhdroffset1                             | 48-48   |
| routhdroffset2                             | 49-49   |
| nxthdroffset                               | 50-50   |
| fragoffset                                 | 51-51   |
| grossrunningsum                            | 52-53   |
| runningsum                                 | 54-55   |
| parseerrcode                               | 56-56   |
| softparsectx                               | 57-63   |
| ipv4sa                                     | 80-83   |
| ipv4da                                     | 84-87   |
| ipv6sa1                                    | 80-87   |
| ipv6sa2                                    | 88-95   |
| ipv6da1                                    | 96-103  |
| ipv6da2                                    | 104-111 |
| sperc                                      | 112-113 |
| iplength                                   | 114-115 |
| routtype                                   | 116-116 |
| fdlength                                   | 123-125 |
| <i>Table continues on the next page...</i> |         |

Table 50. (continued)

|              |         |
|--------------|---------|
| parseerrstat | 127-127 |
|--------------|---------|

\* Note: The \$GPR2 variable is used internally by the SPC Soft Parser Tool to calculate complex expression, including checksum operations. This variable shouldn't be used by the user. Use this variable only if necessary at your own risk.

#### 8.2.7.1.3.1.3.2 Parameter Array

This variable returns data from the parameter array. Since the parameter array is more than 8 bytes long, it is required to specify the specific bytes needed.

Accessing the variable: \$PA[byteOffset:byteNumber]. Returns the byteNumber number of bytes in the parameter array starting from byteOffset.

For example:

In order to access the fifth and sixth bytes (index at PA[4] and PA[5]) in the parameter array, we'll type \$PA[4:2]

#### 8.2.7.1.3.1.3.3 Header size variables

Returns the header size, or the default header size.

Accessing the variables: \$headerSize or \$defaultHeaderSize

- In the before section the \$headerSize of the previous protocol will be returned and accessing the \$defaultHeaderSize is not allowed.
- In the after section the \$defaultHeaderSize will return the number of bytes in the custom protocol's format fields. The \$headerSize will return the headerSize as defined by the user in the after element. If no headerSize has been defined by the user, the variable will return the same value as the \$defaultHeaderSize

#### 8.2.7.1.3.1.3.4 Frame Window

Returns data from the Frame Header. In the before section data is returned starting with the previous protocol's header. In the 'after' section data is returned starting with the custom protocol's header

Accessing the variable: \$variableName[bitOffset:bitNumber] – Returns the bitNumber number of bits in the parameter array starting from bitOffset.

Note: The FW uses similar syntax to the PA and RA variables but accesses specific **bits** instead of bytes.

Examples:

- In order to access the tenth and eleventh bits in the frame array (indexed at FW[9], FW[10]), we'll type \$FW[9:2].
- In order to access the entire third byte in the frame array we'll type \$FW[16:8].
- The conditions in the following example are always true since we access the same bits with the FW variable and through the fields.

```
<format>
<fields>
<field type="bit" name="first" size="1" mask = "0xE0"/>
<field type="bit" name="second" size="1" mask = "0x1"/>
<field type="bit" name="third" size="1" mask = "0xF"/>
<field type="fixed" name="fourth" size="2"/>
</fields>
</format>
...
<after>
<if expr = "first==$FW[0:3]" > ... </if>
```

```
<if expr = "second==$FW[7:1]" > ... </if>
<if expr = "third==$FW[8:4]" > ... </if>
<if expr = "fourth==$FW[16:16]" > ... </if>
</after>
```

#### 8.2.7.1.3.5 Variable prevprotoOffset

Returns the previous protocol's frame header offset. The variable has the same value in the before and after section, and always refers to the protocol defined in the prevproto attribute of the protocol element.

In the before section the FW's current location is equal to prevProtoOffset, in the after section the FW's current location is equal to prevProtoOffset+headerSize.

**Note:** This variable is a "shortcut" to the result array, and returns or modifies values taken directly from the RA. The following tables summarize the RA value returned for each previous protocol.

**Table 51.**

| Previous Protocol                         | Returned value from RA                                                                                                                                  |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ethernet                                  | \$Ethoffset                                                                                                                                             |
| Gre                                       | \$Greoffset                                                                                                                                             |
| ipv4, ipv6                                | \$Ipooffset_n                                                                                                                                           |
| llc_snap                                  | \$Llcsnapoffset                                                                                                                                         |
| Minencap                                  | \$Minencapoffset                                                                                                                                        |
| Mpls                                      | \$mplsoffset_n                                                                                                                                          |
| Pppoe                                     | \$Pppoeoffset                                                                                                                                           |
| tcp, udp, sctp, dccp, ipsec_ah, ipsec_esp | \$L4offset                                                                                                                                              |
| Vlan                                      | \$vlanoffset_n                                                                                                                                          |
| otherl3, otherl4                          | \$NxtHdrOffset – When the previous protocol is otherl3 or otherl4 the custom protocol and the previous protocol have the same offset. See section 2.2.1 |

#### 8.2.7.1.3.2 Operators

Many types of operators exist. Operators can receive several operands (usually one or two) or arithmetic or logical value and can return an arithmetic or logical value. An arithmetic value is a number, while a logical value is true or false. The following table describes all the operators and their properties. All dyadic operators (operators which receive two parameters) appear between two operands. All monadic operators appear before an operand.

**Table 52. Types of operators**

| Name                                | Parameters                       | Description                                                                       | Syntax |
|-------------------------------------|----------------------------------|-----------------------------------------------------------------------------------|--------|
| Greater than                        | Logical (Arithmetic, Arithmetic) | Checks if the value of the first expression is greater than the second expression | gt     |
| Table continues on the next page... |                                  |                                                                                   |        |

**Table 52. Types of operators (continued)**

| Name          | Parameters                                                                                               | Description                                                                                 | Syntax   |
|---------------|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|----------|
| Greater equal | Logical (Arithmetic, Arithmetic)                                                                         | Checks if the value of the first expression equals or is greater than the second expression | ge       |
| Less than     | Logical (Arithmetic, Arithmetic)                                                                         | Checks if the value of the first expression is less than the second expression              | Lt       |
| Less equal    | Logical (Arithmetic, Arithmetic)                                                                         | Checks if the value of the first expression equals or is less than the second expression    | le       |
| Equal         | Logical (Arithmetic, Arithmetic)                                                                         | Checks if the two expressions are equal                                                     | ==       |
| Don't equal   | Logical (Arithmetic, Arithmetic)                                                                         | Checks if the two expressions aren't equal                                                  | !=       |
| Logical and   | Logical (Logical, Logical)                                                                               | Checks if both expressions are true                                                         | and      |
| Logical or    | Logical (Logical, Logical)                                                                               | Checks if one of the expressions are true                                                   | or       |
| Logical not   | Logical (Logical)                                                                                        | Returns true if the expression is false and false otherwise                                 | Not      |
| Add           | 32bit Arithmetic (32bit Arithmetic, 32bit Arithmetic)                                                    | Return the sum of the expressions                                                           | +        |
| Subtract      | 32bit Arithmetic (32bit Arithmetic, 32bit Arithmetic)                                                    | Return the difference between two expressions (result of subtraction)                       | -        |
| Add carry     | 16bit Arithmetic (16bit Arithmetic, 16bit Arithmetic)                                                    | Return the sum of the two-expression summed with the carry after 32 bit.                    | Addc     |
| Bitwise or    | Arithmetic (Arithmetic, Arithmetic)                                                                      | Returns the result of a bitwise or operation on the two expressions                         | bitwor   |
| Bitwise xor   | Arithmetic (Arithmetic, Arithmetic)                                                                      | Returns the result of a bitwise xor operation on the two expressions                        | bitwxor  |
| Bitwise and   | Arithmetic (Arithmetic, Arithmetic)                                                                      | Returns the result of a bitwise and operation on the two expressions                        | bitwand  |
| Bitwise not   | Arithmetic (Arithmetic)                                                                                  | Returns the result of a bitwise not operation on the expression                             | bitwnot  |
| Shift left    | Arithmetic (Arithmetic, Integer – value up to 64)                                                        | Return the left expression shifted left by the right expression                             | shl      |
| Shift right   | Arithmetic (Arithmetic, Integer – value up to 64)                                                        | Return the left expression shifted right by the right expression                            | shr      |
| Concat        | Arithmetic (Arithmetic, Variable or Integer)                                                             | Special instruction explained below                                                         | concat   |
| Checksum      | Arithmetic (Arithmetic – value up to 0xffff, Arithmetic – value up to 256, Arithmetic – value up to 256) | Special instruction explained below                                                         | checksum |

### 8.2.7.1.3.2.1 The concat operator

The *concat* operator shifts the first argument left and inserts the second argument to its right. The concat operation can be executed on variables or integers. If the second argument is a variable, the first argument is shifted left according to the known size of the variable. The result array variables have constant sizes and the sizes of frame header's fields are set in the custom protocol document or the pdl document.

- If the user accesses only specific bits in the second argument, the first argument is shifted left only by the exact number of bits accessed.
- If the second argument is an integer, the first argument is shifted left by the smallest word size the integer fits in - 16, 32, 48 or 64.

#### NOTE

The second argument of a concat operation cannot be an expression since the compiler doesn't know at runtime the size of the expression and therefore can't shift the first argument properly. However, for expressions, the *concat* operation can simply be replaced by a shift operation (if the user know the number of bits to shift) and a bitwise or operation. It is still recommended to use concat instead of shift and bitwise left when performing the operation on variables or integers, to keep the final code shorter.

For example, the following if expression is true:

```
<assign-variable name="$shimr" value="2"/>
<assign-variable name="$GPR1[6:2]" value="3"/>
<if expr="1 concat $shimr concat $GPR1[6:2] concat 0x40000 ==
0x102000300040000">
```

### 8.2.7.1.3.2.2 The checksum operator

The *checksum* operator is a special operator with different behavior and syntax than the rest of the operators. It appears before three operands which have parentheses around them, and thus looks like a function - *checksum(expression, integer, integer)*. The first operand defines the initial checksum value, the second operand defines the frame window offset in which to start the checksum (relative to the current frame window location) and the third operand defines the length of the data, in bytes, on which the checksum operation should be calculated. Since it is only possible to access 256 bytes in the Frame Window the last two argument should be smaller or equal to 256. Using these values, the checksum executes the add carry (addc) operation on 2-bytes sized words in the frame window range defined. If the range selected contains an odd number of bytes to be check summed, the last byte is padded on the right with zeros to form a 16-bit word for checksum purposes. The total sum is added to the initial check sum value using another addc operation. Therefore, the first argument which defined the initial sum value must be smaller than 0xffff. The result of the final add operation is returned.

For example:

Suppose, we have the following frame, and the custom protocol starts in the 0xE offset (where 4500 appears).

FFFF FFFF FFFF 0CCB CC0D DDDD 0800 4500 002E 0000 4000 402F 2AA2 1000 0000 FFFE 0001 0308 0900 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 DA95 36D6 6f15 778c

The following if conditions will always be true:

```
<after>
<if expr="checksum(0x30a2,2,7+2) == 0xdaff">
...
<if expr="checksum(0,0,20) == 0xffff">
...
</after>
```

The first checksum operation above performs the following calculation:



0x30a2 + (0x002e add 0x0000 addc 0x4000 addc 0x402f addc 0x2a00)

The second checksum perform the following calculation:

0x0000 + (0x4500 addc 0x002e addc 0x0000 addc 0x4000 addc 0x402f addc 0x2aa2 addc 0x1000 addc 0x0000 addc 0xFFFFE addc 0x0001)

Normally any protocol should update the `$runningSum` variable with its calculated checksum. This action should be done on after block section of the execute-code element by using bitwise XOR operation.

Here is an example for the correct `$runningSum` update:

```
<after>
<if expr="checksum(0x30a2,2,7+2) == 0xdaaff">
...
<if expr="checksum(0,0,20) == 0xffff">
...
</after>
```

- where 46 in this example is the length of the current custom header

### 8.2.7.1.3.2.3 Expression priorities

Expressions containing multiple operators perform the operation according to the following rules, in the order they appear below:

1. First operations in parenthesis are performed.
2. Next operations which have a higher priority (see section 3.2.4) are performed.
3. Lastly, if there are several operations with the same priority, they are executed from left to right.

It is recommended to use parentheses when several operators appear in the same expression to make sure they are calculated correctly.

### 8.2.7.1.3.2.4 Specific operator priorities

If several operators appear in the same expressions without any parentheses separating them, they should be performed in the following order:

1. not, bitwise not, checksum
2. add, subtract, add carry
3. bitwise and, bitwise or, bitwise xor
4. shift right, shift left, concat
5. greater than, greater equal, less than, less equal, equal, not equal
6. and, or

### 8.2.7.1.3.2.5 Variables size

In most operations, the expression size is limited to 64 bits. However, there are a few exceptions: when shifting variables, the **shift** value must be equal or lower than 64 since there are only 64 bits in an expression.

The **add carry** operation can only be performed on 16bit variables and will always return a 16bit variable. The softparser will report an error if an add carry operation is performed on a constant larger than 16bit but won't be able to recognize a complex expression larger than 16bit, therefore it is the user's responsibility to perform the operation only on 16bit variables.

The **subtract** and **add** operators can only be performed on 32bit variables, and they will only return a 32-bit result. If two 32-bit expressions are added and their result is larger than 32 bits, only the carry will return, such that the returned value is a 32-bit variable. The softparser will report a warning if an add carry operation is performed on a constant larger than 32 bits but won't be able to recognize a complex expression larger than 32 bits.

There is an exception which allows performing add and subtract operations large values. Users can perform these operations with one 64-bit variable and one 32-bit variable and receive a 64-bit result, as long as the operation doesn't modify the 32 most significant bytes. In this case the 64-bit variable must appear on the left side of the operator. Working in this way in **not** recommended and should only be used if there is no other option or if performance is crucial.

For example:

The following if expressions are always true:

```
<if expr="0xffffffff+2 == 0x1">
<if expr="0x123456781+3 == 0x123456784">
The following if expression is false (and shouldn't appear in the xml):
<if expr="3+0x123456781 == 0x123456784">
```

### 8.2.7.1.3.3 Expression types

There are two main types of expressions: Logical expressions, which return `true` or `false` and arithmetic expressions, which return a numeric result.

#### 8.2.7.1.3.3.1 Logical expressions

Logical expression appears in the `expr` attribute of the `'if'` element.

These expressions always return a true or false value, and therefore they must use at least one logical operator which will separate arithmetic or logical operators.

Examples:

The following are logical expressions -

- $(4 == \$shimoffset\_1 \text{ or } 5 != \$shimoffset\_2)$
- $not(\$ShimOffset\_2 \text{ ge } \$ShimOffset\_1 \text{ or } \$ShimOffset\_1 \text{ lt } \$ShimOffset\_2)$

The following are **not** logical expressions -

- $(7 \text{ gt } 3 \text{ and } 2+7)$
- $(5 \text{ lt } 8 \text{ or } 7)$

#### 8.2.7.1.3.3.2 Arithmetic expressions

Arithmetic expressions always have a numeric result. They can hold a single operand (a number, variable or arithmetic expression), or more than one operands separated by arithmetic operators. Logical operators are not allowed in arithmetic expression.

Arithmetic expressions may appear in the following:

- The value attribute of the assign element.
- The headersize attribute of the after element.
- The expr attribute of the switch element

Examples:

The following are arithmetic expressions:

- $(\$FW[0:16] + 4)$
- $(\$shimOffset\_1 \text{ concat } 3)$
- $(3 + 7 + 8 + \$shimOffset\_2)$
- 4

The following are not arithmetic expression:

`4==$shimOffset_2`

## 8.2.7.1.4 FAF – frame attribute flags

FAF support was introduced in DPAA 2.0 and they provide information about parsed frame fields. These flags are populated by the Parser after frame parsing.

In SP for DPAA 2.0 was added the ability to access the FAF directly from FSL extension of NetPDL language.

For more detailed information about each FAF meaning and bit position inside *Parse Results* array, please refer to *DPAA 2.0 Parser Guide*.

### 8.2.7.1.4.1 Inspect FAF

FAF can be inspected from FSL NetPDL code by using if instruction with attribute *faf* and specify desired FAF name from the list of available FAF names presented below.

All frame attribute flags (HW FAFs and User defined FAFs) can be inspected by the Soft Parser.

```
<if faf="name">
<if-true>
.....
</if-true>
<if-false>
.....
</if-false>
</if>
```

### 8.2.7.1.4.2 Modify FAF

FAF can be modified by using new *set* / *reset* instructions introduced in FSL NetPDL for DPAA 2.0. Only user defined flags, can be set or reset by the Soft Parser.

To set a FAF flag use:

```
<set faf="name"/>
```

To reset a FAF flag use:

```
<reset faf="name"/>
```

#### 8.2.7.1.4.2.1 Available FAF attributes names

All available FAF names that can be used in FSL NetPDL as *faf* attributes and their meaning are listed in the following tables:

##### User defined FAFs:

Can be both set and inspected by the Soft Parser.

**Table 53. User defined FAFs attributes and their meaning**

|                                     |                     |
|-------------------------------------|---------------------|
| custom_0                            | User Defined Flag 0 |
| custom_1                            | User Defined Flag 1 |
| custom_2                            | User Defined Flag 2 |
| Table continues on the next page... |                     |

**Table 53. User defined FAFs attributes and their meaning (continued)**

|          |                     |
|----------|---------------------|
| custom_3 | User Defined Flag 3 |
| custom_4 | User Defined Flag 4 |
| custom_5 | User Defined Flag 5 |
| custom_6 | User Defined Flag 6 |
| custom_7 | User Defined Flag 7 |

**Hardware FAFs:**

Can only be inspected by the Soft Parser (as they are set by the HW Parser).

**Table 54. Hardware FAFs attributes and their meaning**

|                                            |                                                                              |
|--------------------------------------------|------------------------------------------------------------------------------|
| IPv6_route_hdr2_present                    | Routing header present in IPv6 header 2                                      |
| GTP_primed_detected                        | GTP Primed was detected                                                      |
| VLAN_prio_detected                         | VLAN with VID = 0 was detected                                               |
| PTP_detected                               | A PTP frame was detected                                                     |
| VxLAN_present                              | VXLAN was parsed                                                             |
| VxLAN_parsing_error                        | A VXLAN HXS parsing error was detected                                       |
| Ethernet_slow_protocol                     | Ethernet control protocol (MAC DA is 01:80:C2:00:00:00-01:80:C2:00:00:00:FF) |
| IKE_present                                | IKE was detected at UDP port 4500                                            |
| shim_soft_parsing_error                    | An SXS parsing error was found in the shim shell                             |
| parsing_error                              | A Parsing error was found, the error code is reported in the Parse Result    |
| Ethernet_MAC_present                       | Ethernet MAC was parsed                                                      |
| Ethernet_unicast                           | Ethernet MAC DA is Unicast                                                   |
| Ethernet_multicast                         | Ethernet MAC DA is Multicast                                                 |
| Ethernet_broadcast                         | Ethernet MAC DA is Broadcast                                                 |
| BPDU_frame                                 | MAC DA is 01:80:C2:00:00:00                                                  |
| FCoE_detected                              | FCoE frame detected. Ether type is 0x8906 detected                           |
| FIP_detected                               | FCoE initialization protocol detected. Ether type is 0x8914 detected         |
| Ethernet_parsing_error                     | An Ethernet HXS parsing error was found                                      |
| LLC_SNAP_present                           | LLC+SNAP was parsed                                                          |
| unknown_LL_C_OUI                           | (LLC is not AAAA03 or OUI is not zero or Ethernet Length is <= 8)            |
| LLC_SNAP_error                             | A LLC+SNAP HXS parsing error was found                                       |
| VLAN_1_present                             | At least one VLAN was parsed                                                 |
| VLAN_n_present                             | More than one VLAN was parsed                                                |
| VLAN_parsing_error                         | A VLAN HXS parsing error was found                                           |
| <i>Table continues on the next page...</i> |                                                                              |

**Table 54. Hardware FAFs attributes and their meaning (continued)**

|                                            |                                                                                                                                                      |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| PPPoE_PPP_present                          | PPPoE+PPP was parsed                                                                                                                                 |
| PPPoE_PPP_parsing_error                    | A PPPoE+PPP HXS parsing error was found                                                                                                              |
| MPLS_1_present                             | At least one MPLS was parsed                                                                                                                         |
| MPLS_n_present                             | More than one MPLS was parsed                                                                                                                        |
| MPLS_parsing_error                         | A MPLS HXS parsing error was found                                                                                                                   |
| ARP_present                                | ARP frame with Ethertype 0x0806                                                                                                                      |
| ARP_parsing_error                          | ARP HXS parsing error was found                                                                                                                      |
| L2_unknown_protocol                        | set when next HXS to be executed is the Other L3 shell                                                                                               |
| L2_soft_parsing_error                      | A L2 SXS parsing error was found                                                                                                                     |
| IPv4_1_present                             | IPv4 was parsed as first IP, IPv4 SA IPv4 DA IPv4 Protocol                                                                                           |
| IPv4_1_unicast                             | IPv4 was parsed as first IP, IPv4 DA is Unicast                                                                                                      |
| IPv4_1_multicast                           | IPv4 was parsed as first IP, IPv4 DA is Multicast                                                                                                    |
| IPv4_1_broadcast                           | IPv4 was parsed as first IP, IPv4 DA is Broadcast                                                                                                    |
| IPv4_n_present                             | IPv4 was parsed as last IP                                                                                                                           |
| IPv4_n_unicast                             | IPv4 was parsed as last IP, IPv4 DA is Unicast                                                                                                       |
| IPv4_n_multicast                           | IPv4 was parsed as last IP, IPv4 DA is Multicast                                                                                                     |
| IPv4_n_broadcast                           | IPv4 was parsed as last IP, IPv4 DA is Broadcast                                                                                                     |
| IPv6_1_present                             | IPv6 was parsed as first IP, IPv6 SA IPv6 DA IPv6 NextHeader are populated                                                                           |
| IPv6_1_unicast                             | IPv6 was parsed as first IP, IPv6 DA is Unicast                                                                                                      |
| IPv6_1_multicast                           | IPv6 was parsed as first IP, IPv6 DA is Multicast                                                                                                    |
| IPv6_n_present                             | IPv6 was parsed as last IP                                                                                                                           |
| IPv6_n_unicast                             | IPv6 was parsed as last IP, IPv6 DA is Unicast                                                                                                       |
| IPv6_n_multicast                           | IPv6 was parsed as last IP, IPv6 DA is Multicast                                                                                                     |
| IP_1_option_present                        | IP option present                                                                                                                                    |
| IP_1_unknown_protocol                      | not IP/GRE/MINENC/TCP/UDP/IPSec/SCTP/DCCP/ICMP/IGMP/ICMPv6UDP Lite                                                                                   |
| IP_1_packet_is_fragment                    | IPv4 “more fragments” flag is set or the “fragment offset” field is non-zero or IPv6 Fragment Extension Header present. IPv6FragOffset is populated. |
| ip_1_packet_is_initial_fragment            | IPv4 “more fragments” flag is set and the “fragment offset” field is 0 or IPv6 Fragment Extension Header present and “fragment offset” field is 0.   |
| IP_1_parsing_error                         | An IP 1 HXS parsing error was found                                                                                                                  |
| IP_n_option_present                        | IP option present                                                                                                                                    |
| IP_n_unknown_protocol                      | not IP/GRE/MINENC/TCP/UDP/IPSec/SCTP/DCCP/ICMP/IGMP/ICMPv6UDP Lite                                                                                   |
| <i>Table continues on the next page...</i> |                                                                                                                                                      |

**Table 54. Hardware FAFs attributes and their meaning (continued)**

|                                            |                                                                                                                                                    |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| IP_n_packet_is_fragment                    | IPv4 “more fragments” flag is set or the “fragment offset” field is non-zero or IPv6 Fragment Extension Header present.                            |
| IP_n_packet_is_initial_fragment            | IPv4 “more fragments” flag is set and the “fragment offset” field is 0 or IPv6 Fragment Extension Header present and “fragment offset” field is 0. |
| ICMP_detected                              | ICMP frame detected, IP Protocol is 1.                                                                                                             |
| IGMP_detected                              | IGMP frame detected, IP Protocol is 2 .                                                                                                            |
| ICMPv6_detected                            | ICMPv6 frame detected, IP Protocol is 3A.                                                                                                          |
| UDP_light_detected                         | UDP light detected, IP Protocol is 136                                                                                                             |
| IP_n_parsing_error                         | An IP n HXS parsing error was found                                                                                                                |
| Min_encap_present                          | Min. Encap was parsed, the parsed Original Destination Address replaces the IPv4 Destination Address                                               |
| Min_encap_s_flag_set                       | The S flag is set in Min. Encap, the parsed IP Src Address replaces the IPv4 Source Address                                                        |
| Min_encap_parsing_error                    | A Min. Encap HXS parsing error was found                                                                                                           |
| GRE_present                                | GRE was parsed                                                                                                                                     |
| GRE_R_bit_set                              | RFC1701 R bit set                                                                                                                                  |
| GRE_parsing_error                          | An GRE HXS parsing error was found                                                                                                                 |
| L3_unknown_protocol                        | set when next HXS to be executed is the Other L4 shell                                                                                             |
| L3_soft_parsing_error                      | A L3 SXS parsing error was found                                                                                                                   |
| UDP_present                                | UDP was parsed                                                                                                                                     |
| UDP_parsing_error                          | A UDP HXS parsing error was found                                                                                                                  |
| TCP_present                                | TCP was parsed                                                                                                                                     |
| TCP_options_present                        | offset value higher than 5                                                                                                                         |
| TCP_control_bits_6_11_Set                  | one or many of URG, ACK, PSH, RST, SYN, FIN bits are set                                                                                           |
| TCP_control_bits_3_5_Set                   | one or many of NS, CWR, ECE bits are set                                                                                                           |
| TCP_parsing_error                          | A TCP HXS parsing error was found                                                                                                                  |
| IPSec_present                              | IPSec was parsed                                                                                                                                   |
| IPSec_ESP_found                            | ESP found                                                                                                                                          |
| IPSec_AH_found                             | AH found                                                                                                                                           |
| IPSec_parsing_error                        | A IPSec HXS parsing error was found                                                                                                                |
| SCTP_present                               | SCTP was parsed                                                                                                                                    |
| SCTP_parsing_error                         | A SCTP HXS parsing error was found                                                                                                                 |
| DCCP_present                               | DCCP was parsed                                                                                                                                    |
| DCCP_parsing_error                         | A DCCP HXS parsing error was found                                                                                                                 |
| <i>Table continues on the next page...</i> |                                                                                                                                                    |

**Table 54. Hardware FAFs attributes and their meaning (continued)**

|                         |                                                         |
|-------------------------|---------------------------------------------------------|
| L4_unknown_protocol     | Set when next HXS to be executed is the Other L5+ shell |
| L4_soft_parsing_error   | A L4 SXS parsing error was found                        |
| GTP_present             | GTP was parsed.                                         |
| GTP_parsing_error       | A GTP HXS parsing error was found                       |
| ESP_present             | ESP was parsed                                          |
| ESP_parsing_error       | An ESP HXS parsing error was found                      |
| iSCSI_detected          | iSCSI detected. Port# 860                               |
| Capwap_control_detected | A Capwap-control frame was detected. Port# 5246         |
| Capwap_data_detected    | A Capwap-data frame was detected. Port# 5247            |
| L5_soft_parsing_error   | A L5SXS parsing error was found                         |
| IPv6_route_hdr1_present | Routing header present in IPv6 header 1                 |

## 8.2.7.1.5 Subroutines support

In SP for DPAA 2.0 was added support to create and call subroutines in FSL NetPDL language for code reusability purpose. Passing parameters is not allowed. Currently only a stack depth of one call is supported since this is supported by DPAA 2.0.

### 8.2.7.1.5.1 Defining a subroutine

A subroutine can be defined by using tag `<subroutine>` inside `<execute-code>` tag on the same level with `<before>` and `<after>` tags. The name of the subroutine must be specified by using attribute *name*.

```
<subroutine name="sub_name">
<!-- subroutine body -->
.....
</subroutine>
```

A subroutine body can contain all instructions supported the same like `<before>` and `<after>` sections but it cannot contain a call to another subroutine because DPAA 2.0 *gosub* instruction allows only one level of call stack.

Multiple subroutines can be defined the only constraint is to have different names.

### 8.2.7.1.5.2 Calling a subroutine

A subroutine can be called by using the tag `<gosub/>` in FSL NetPDL language and specify the name of the called subroutine by using attribute *name* inside this tag.

```
<gosub name="sub_name"/>
```

A subroutine can be called anywhere from inside sections `<before>` and `<after>`. The calls must substitute a set of several instructions for code reusability purpose.

### 8.2.7.1.5.3 Example of a subroutine usage

```
<execute-code>
 <before>

 <gosub name="sub_name"/>
 <after>
```

```
<gosub name=" sub_1"/>
<gosub name="sub_2"/>
.....
</before>
<after>
.....
<gosub name="sub_2"/>
.....
</after>
<subroutine name="sub_1">
<!-- subroutine 1 section -->
<assign-variable name="$gpr1" value="5"/>
<gosub name="sub_2"/> <!-- warning displayed and gosub is ignored -->
.....
</subroutine>

<subroutine name="sub_2">
<!-- subroutine 2 section -->
<assign-variable name="$gpr1" value="6"/>
.....
</subroutine>
</execute-code>
```

### 8.2.7.1.6 SP Hardware configuration file

The Soft Parser Configuration also requires Hardware related settings. All these hardware configurations must be specified in a separate XML file.

All hardware configurations are optional and in case they are not specified, the system uses default values. The entire hardware configuration XML file is optional and can miss entirely in which case the system uses a set of default values for all necessary hardware settings.

#### 8.2.7.1.6.1 The <spconfig> element

The SP hardware configuration file always begins with the <spconfig> root element.

The end tag of the spconfig element should appear in the end of the document.

**Attributes:** No required attributes

**Child Elements:** memorymap, device, parameters

For example:

```
<spconfig>
...
</spconfig >
```

#### 8.2.7.1.6.2 SoC configuration

The SP hardware configuration file defines the SoC attributes.

**Element:** soc

**Attributes:**

- name – optional, possible value: string. Specifies the SoC name used to run SP bytecode
- rev – optional, possible value: string. Specifies the SoC revision used



Example:

```
<!-- SP configuration file -->
<!-- optional: this configuration file is optional -->
<spconfig>
 <!-- SoC configuration -->
 <!-- optional -->
 <soc name="LS2088" rev="1.0" />
</spconfig>
```

### 8.2.7.1.6.3 Memory map configuration

The SP hardware configuration file can define parser memory map. This is optional, and it is used to define how protocols compiled bytecode is loaded in parser memory. This is useful for advanced users and provides full control over the parser bytecode memory.

#### 8.2.7.1.6.3.1 The *<memorymap>* element

The *memorymap* element is used to encapsulate the entire parser memory map definition for different bytecode sections.

#### 8.2.7.1.6.3.2 The *<bytecode>* element

The *bytecode* element is used to define all attributes for one bytecode section.

##### **Attributes:**

- **offset** – optional, possible value: numeric.  
Specifies the base address where this bytecode section must be loaded in parser memory.

#### 8.2.7.1.6.3.3 The *<load-on-parser>* element

The *load-on-parser* element is used to define on which parser this bytecode section must be loaded.

##### **Attributes:**

- **name** – optional, possible value: string.  
Specifies the parser where this bytecode section must be loaded  
Valid values: *wriop\_ingress*, *wriop\_egress*, *aiop*

#### 8.2.7.1.6.3.4 The *<load-protocol>* element

The *load-protocol* element is used to define which protocols from the ones defined in NetPDL protocol definition file must be included in this bytecode section.

##### **Attributes:**

- **name** – optional, possible value: string.  
Specifies the protocol name to be included in this bytecode section  
The protocol name must exist in NetPDL protocol definition file.

#### 8.2.7.1.6.3.5 Example for memory map definition

```
<!-- SP configuration file -->
<!-- optional: this configuration file is optional -->
<spconfig>

 <!-- optional -->
```

```
<!-- TODO: not implemented: 1 default bytecode section is used with all protocols -->
<memorymap>
 <!-- bytecode section -->
 <bytecode offset="0x40" >

 <!-- load this bytecode section on parsers -->
 <load-on-parser name="wriop_ingress" />
 <load-on- parser name="wriop_egress" />

 <!-- protocols to be included in this bytecode section -->
 <load-protocol name="afteth" />
 <load-protocol name="dap" />

 </bytecode>
</memorymap>

</spconfig>
```

#### 8.2.7.1.6.4 Device configuration

The SP hardware configuration file can define Parser device related settings. This is optional, and it is used to define all specific device parser settings (like what protocols should be enabled on initialization, by default on each parser).

##### 8.2.7.1.6.4.1 The *<device>* element

The *device* element is used to encapsulate the entire parser device definition for all available parsers.

##### 8.2.7.1.6.4.2 The *<parser>* element

The *parser* element is used to define all configurations for one parser.

###### **Attributes:**

- **name** – required, possible value: string.  
Specifies the parser for which this device configuration section is intended  
Valid values: wriop\_ingress, wriop\_egress, aiop\_ingress, aiop\_egress

##### 8.2.7.1.6.4.3 The *<enable-on-init>* element

The *enable-on-init* element is used to define which protocols are enabled by default on initialization for current parser.

###### **Attributes:**

- **protocol** – required, possible value: string.  
Specifies the protocol name to be enabled by default on initialization  
The protocol name must exist in NetPDL protocol definition file.

##### 8.2.7.1.6.4.4 Example to enable protocols

```
<!-- SP configuration file -->
<!-- optional: this configuration file is optional -->
<spconfig>

 <!-- optional: implicit all protocols are disabled on all parsers -->
 <device>
 <parser name="wriop_ingress">
 <enable-on-init protocol="afteth" />
```

```

 </parser>
 <parser name="wriop_egress">
 <enable-on-init protocol="afteth" />
 </parser>
</device>

</spconfig>

```

### 8.2.7.1.6.5 SP parameters configuration

The SP hardware configuration file can define parameters passed to SP. This is optional, and it is used to define all the necessary attributes of the parameters passed to SP.

#### 8.2.7.1.6.5.1 The *<parameters>* element

The *parameters* element is used to encapsulate the entire SP parameters definition.

#### 8.2.7.1.6.5.2 The *<parameter>* element

The *parameter* element is used to define all attributes for one parameter.

##### **Attributes:**

- **name** – required, possible value: string. Specifies the name of this parameter.
- **protocol** – required, possible value: string. Specifies the protocol name for which this parameter is intended. The protocol name must exist in NetPDL protocol definition file.
- **offset** – required, possible value: numeric/string. Specifies the offset in memory of this parameter. In case the keyword 'auto' is used, the offset is automatically calculated based on the previous parameter offset and size
- **size** – required, possible value: numeric. Specifies the size in bytes of this parameter.
- **value** – optional, possible value: numeric. Specifies the default value of this parameter. In case this attribute is missing, then the default value used for this parameter is zero.
- **type** – optional, possible value: string. Specifies the type of this parameter that define its runtime behavior.

##### **Valid options:**

- read-write – used to specify the parameter can be both read and written.
- read-only – used to specify the parameter is read only so cannot be written.

In case this attribute is missing, then the default value used for this parameter is read-write.

### 8.2.7.1.7 Tips and recommendations

This chapter lists the recommendations while using the Soft Parser Configuration tool.

#### 8.2.7.1.7.1 Updating important fields

The Soft Parser Configuration Tool allows users to define custom protocols, parse these protocols, and update any needed field. However, the tool does not update fields for the user (besides advancing the frame window– see the explanations on the before/after and action elements).

Therefore, when using the soft parser tool, some fields are left empty unless the user manually updates them. These fields might be needed in later stages to correctly interpret. A list of important fields which should be updated appears in the Parser document, under section 1.5.5 Soft HXS PR Updates. These fields include the \$nxtHdr, \$Runningsum, HXS offsets, Last E Type Offset, and \$nxtHdrOffset (see table 4.1.3.1). Notice that the HXS offset, \$nxtHdr, and \$nxtHdrOffset are also used internally by the softparser (see section 5.1.2), therefore these values should be modified carefully. The \$nxtHdr should be modified only if the custom protocol doesn't jump to 'after\_ip'/'after\_ethernet' or if the user wants to change the next protocol

when jumping to 'after\_ip'/'after\_ethernet'. The HXS and next header offsets should only be modified in the `after` section or in the `before` section if the parser exits in that section without advancing the frame header.

### 8.2.7.1.7.2 Refraining from modifying specific fields

Some fields in the RA are used internally by the soft parser and users should not modify these fields in certain conditions:

- `$GPR1` is used to store temporary values in complex operations, and therefore users should refrain from modifying it.
- `$nxtHdr` is used to calculate the next protocol when jumping to 'next\_ethernet' or 'next\_ip'. Therefore, it should not be modified when `nextproto` equals one of those values.
- `$prevProtoOffset` is used to advance the frame window between the `before` and `after` sections or when using the action element with the `advance` attribute in the `before` section. Therefore, it shouldn't be modified in the `before` section, unless softparser exits in that section without advancing the frame window. `$prevProtoOffset` can equal the following RA variables (which also shouldn't be modified in the same context): `$ethoffset`, `$greoffset`, `$ipoffset_n`, `$llc_snapoffset`, `minencapoffset`, `mplsoffset_n`, `pppoeoffset`, `l4offset`, `vlanoffset_n`, and `$nxtHdrOffset`.
- `$nxtHdrOffset` is used to advance the frame window between the `before` and `after` sections or when using the action element with the `advance` attribute in the `before` section. Therefore, it should not be modified in the `before` section, unless softparser exits in that section without advancing the frame window.

### 8.2.7.1.7.3 Setting the next protocol

The softparser can be used to add code for an existing protocol or to define an entirely new protocol. When it is used as an extension for an existing protocol and no new frame headers are being parsed, the `nextproto` attribute of the action element should be set to 'return'. In this case the `nextproto` attribute can also be left empty since 'return' is the default value. If 'return' is set the soft parser will execute the soft parser code and then the hardware parser will continue parsing at the same position in the frame header where it stopped earlier.

When the soft parser is used for a separate custom protocol with its own header, the hard parser should skip this custom protocol (since it won't recognize it and know how to parse it) and therefore the next protocol should be set to a specific protocol. If the next protocol is unknown the `nextproto` attribute in the action element can also be set to 'after\_ip' or 'after\_ethernet', in such cases the next protocol will be determined according to the value in the `$nxtHdr` field.

For example:

1. If we want to execute softparse code when we parse the ethernet protocol, our code will probably include an action like action below which will appear in the 'before' section:

```
<action type="exit" advance="no" nextproto="return">
```

2. If we want to add a custom protocol after Ethernet and then jump to ipv6 our code will probably include an action like action below which will appear in the 'after' section

```
<action type="exit" advance="yes" nextproto="ipv6">
```

3. If we want to add a custom protocol after Ethernet and we don't know where to jump next our code will probably include an action like the action below which will appear in the 'after' section

```
<action type="exit" advance="yes" nextproto="after_ethernet">
```

## 8.2.7.1.8 Limitations

This section describes limitations users should consider when working with the Soft Parser Configuration tool.

### 8.2.7.1.8.1 Complex expressions

The Soft Parser tool has limited abilities and cannot process any expression. Some expressions that contain many operations and parentheses might be too complicated for the Soft Parser. If you receive an error stating that an expression is too complex,

you can try simplifying it by splitting it to a few expressions, opening parenthesis, or storing temporary values in the result array variables. (\$GPR1 is recommended for storing temporary variables but refrain from storing in \$GPR2 which is used internally by the tool.) Notice that the checksum operation is especially prone to participate in expressions that are too complex.

### 8.2.7.1.8.2 Subroutines: not supported

Subroutines are not yet supported in the NetPDL language although the underlying infrastructure was implemented in SPC Tool for SPA (Soft Parser Assembler). Subroutine support at NetPDL level will be enabled in future versions of SPC Tool.

### 8.2.7.1.8.3 Memory map configuration: not supported

Parser memory map definition in SP Hardware configuration file is not supported by this SPC version. The system will use one default bytecode section that contains all protocols existent in NetPDL file and will be loaded on all three available parser memories: WRIOP Ingress, WRIOP Egress, AIOP.

However, the base address where bytecode is loaded in parser memory is supported and can be changed by defining the *offset* attribute on *bytecode* element in XML configuration file. Only the *offset* for the first *bytecode* section is considered while all other bytecode sections are ignored.

The other elements *<load-on-parser>* and *<load-protocol>* are not supported at all and will be ignored by this SPC version. A runtime warning message (*'unsupported element'*) is displayed if these tags are used.

### 8.2.7.1.8.4 Multiple custom protocols: not supported

Multiple custom protocols definition in a blob is not supported by this SPC Tool version so only one custom protocol can be defined and used. This limitation is due to the previous limitation related to memory map configuration is not supported because the memory map configuration is used to include different protocols in current bytecode section by using the tag *<load-protocol>*. A runtime warning message (*'unsupported element'*) is displayed if this tag is used.

### 8.2.7.1.8.5 Enabling protocols at initialization is mandatory

Because there is a limitation in MC related to Networking object API (for network objects DPNI, DPDMUX, DPSW) is not currently supported, this makes it a mandatory operation to enable desired protocols at initialization in the SPC tool by using the tag: *enable-on-init*.

If this step is not performed, no protocol is enabled and defined custom protocols are not usable because, by default, all protocols are disabled on all parsers.

This is an example for a minimal configuration file:

```
<spconfig>
 <device>
 <parser name="wriop_ingress">
 <enable-on-init protocol="protocol_name" />
 </parser>
 </device>
</spconfig>
```

### 8.2.7.1.9 Running the Soft Parser tool

The Soft Parser Tool should be executed using the *spc* executable file. The following command line options are relevant for the soft parser:

- *-s <custom\_protocol\_file>* - required. The file contains the xml with the description of all the custom protocols, as explained in this document.

- `-c <config_file>` - required. Specifies the SP hardware configuration file.
- `-d <pdl_file>` - optional. This file contains information regarding the protocols supported by the hard parser. If this option is missing, then the default pdl file will be used.
- `-i` - optional. Generate intermediate code.
- `-l <level>` - optional. Specify log level. The following choices are valid: none, err, warn, info, dbg1, dbg2, dbg3.

For more information type: `spc --help`

### 8.2.7.1.10 Output of the SPC tool

The output received after running SPC Tool is a **Soft Parser Blob** (\*.spb file). A soft parser blob is a binary file that contains entire configuration required to configure the Soft Parser (custom protocols bytecode and SP hardware configuration).

If the option `-i` is used, then additional files are generated: several levels of intermediate code (*parsed*, *ir*, *code*, *asm*) and *\_blob.h* file, which is the entire binary blob information dumped in human readable format as an array of bytes.

## 8.2.7.2 SPC on DPAA 2.x Based Platforms

### 8.2.7.2.1 Introduction

This document describes how to apply Soft Parser (SP) configuration on DPAA 2.x based platforms.

#### Solution overview

The architecture is based on using an offline tool to take in a text-based description of the protocol(s) to be parsed and produce a blob for Management Complex (MC) to load.

Loading of the blob is done at system boot by U-Boot. There is one blob per system and the soft parser sequence(s) can be used on any of the interfaces (physical ports or internal links).

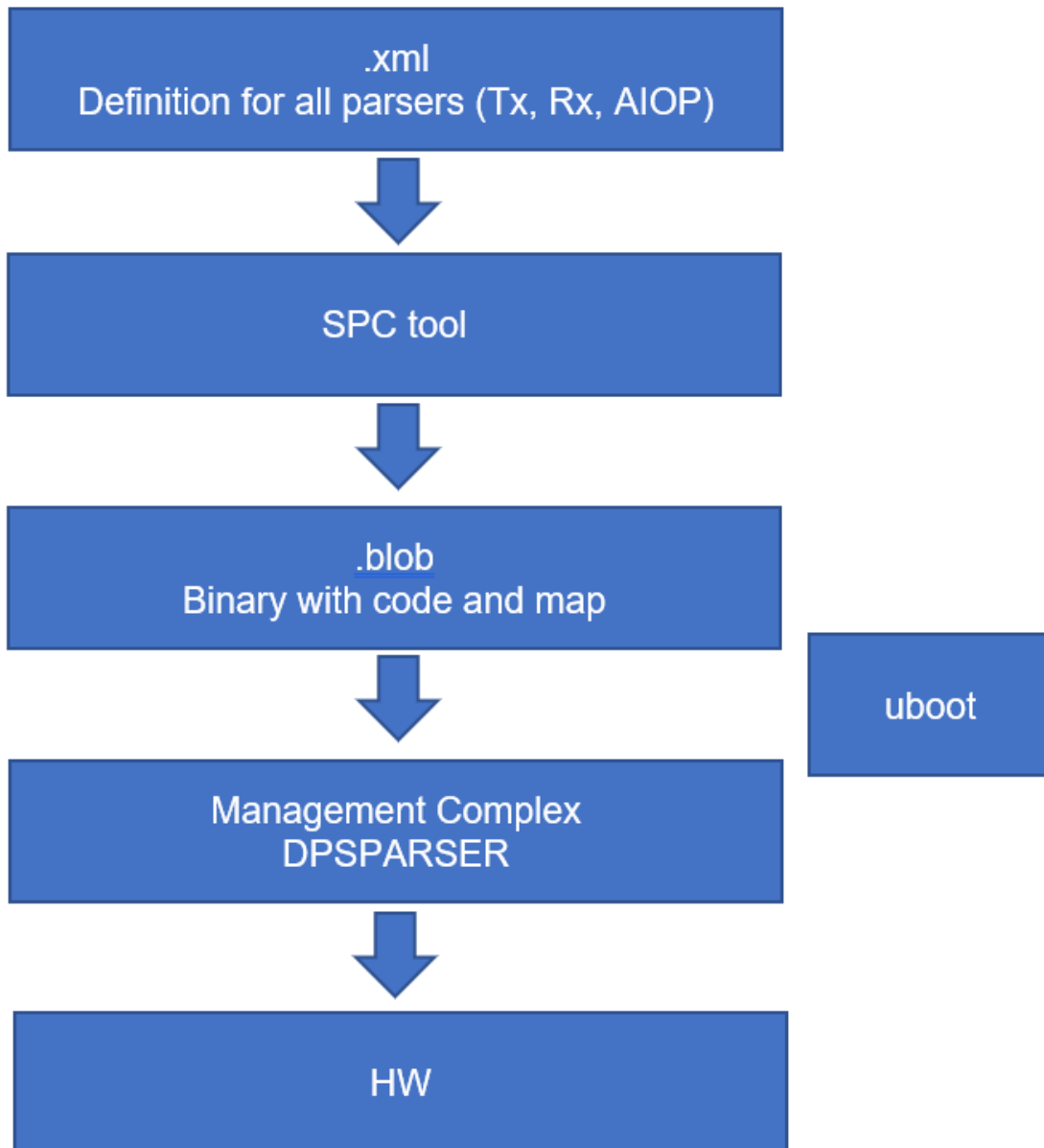


**Figure 56. High-level solution overview**

A soft parser blob is a binary file that encapsulates the entire configuration required to configure the Soft Parser HW module: custom protocols bytecode, SP protocols configuration, SP parameters, and soft parser hardware configuration. The soft parser blob file is generated by the SPC (Soft Parser Configuration) Tool. MC can be used to apply an SP Blob on hardware by using U-Boot command line.

### System Architecture

The high-level architecture for Soft Parser Programming is represented in the following picture with all modules involved and their interaction.

### Soft Parser Programming

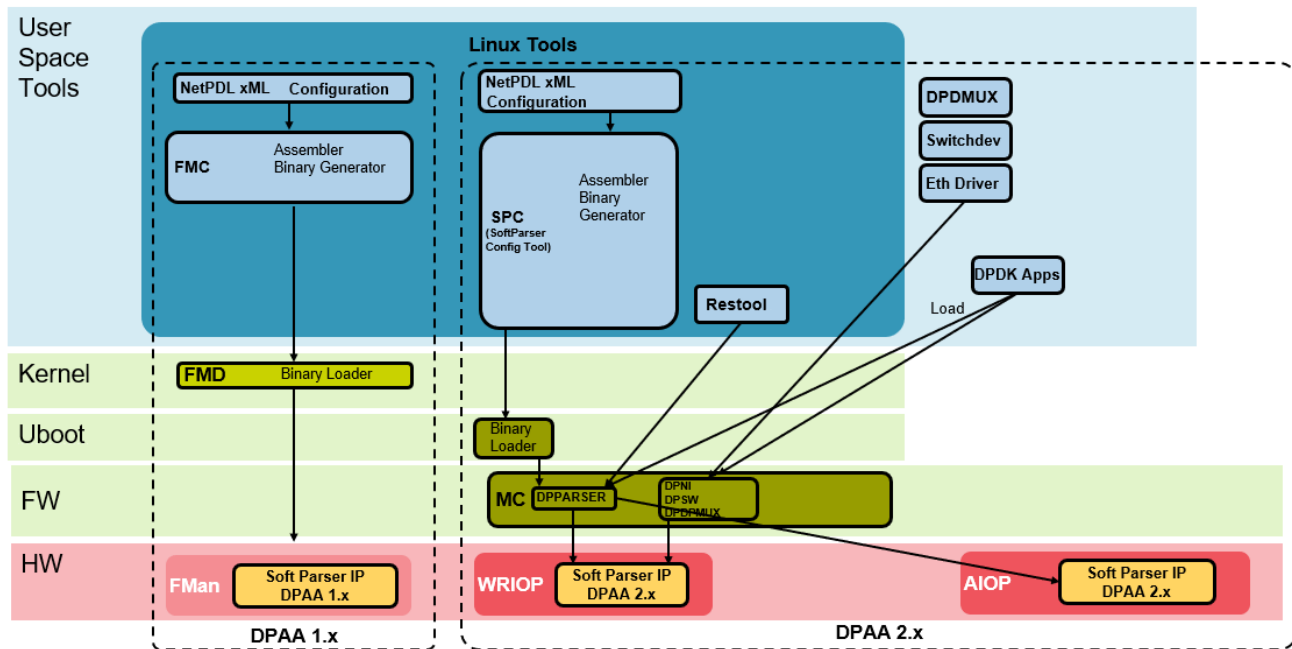


Figure 57. High-level system architecture

The architecture for SP programming on DPAA 2.x was designed to be used in a similar way as it is on DPAA 1.x platforms by taking as input an XML configuration file in NetPDL language.

This architecture is composed from the following modules:

- User space tools:
  - SPC – Soft Parser Configuration Tool
  - Restool
- User applications:
  - DPDK apps
- Binary loader:
  - U-Boot
- Firmware:
  - MC – Management Complex
- Hardware:
  - WRIOP – Soft Parser
  - AIOP – Soft Parser

## 8.2.7.2.2 Applying Soft Parser Blob at U-Boot command line

To load a blob and apply on HW, execute the following U-Boot command sequence:

- load MC:

```
tftp 0x80000000 /mc_spb_path.itb
```
- load DPC and DPL files:



```
tftp 0xa0000000 /dpc_path.dtb
tftp 0xa8000000 /dpl_path.dtb
```

- **load Soft Parser Blob file:**

```
tftp <blob_address> <soft_parser_blob_path.spb>
```

Example:

```
tftp 0xac000000 /soft_parser_blob.spb
```

- **start MC:**

```
fsl_mc start mc 0x80000000 0xa0000000
```

- **apply SPB:**

```
fsl_mc apply spb <blob_address>
```

Example:

```
fsl_mc apply spb 0xac000000
```

Execute the commands below to invoke MC to load, parse, verify, and apply configuration from a soft parser blob file.

If the blob was applied and the command succeeded, this will be confirmed at command line:  
fsl-mc: Applying soft parser blob... SUCCESS

If an error occurred and the command failed, the error will be displayed at command line:

fsl-mc: Applying soft parser blob... FAILED with error code = 1:

BLOB : Magic number does not match

or

fsl-mc: Applying soft parser blob... FAILED with error code = 29:

apply spb : Soft Parser BLOB is already applied

- **apply DPL:**

```
fsl_mc apply dpl 0xa8000000
```

- **boot Linux:**

```
setenv ethact e1000#0; tftp a0000000 /kernel_path.itb; bootm a0000000
```

At this point, Soft Parser is configured according to configuration existent in blob applied above and can be used.

After the Linux boot, the interfaces used to receive traffic must be configured from command line.

**Optionally** the MC console can be checked to verify that all soft parser actions performed:

- **verify blob actions performed:**

```
cat /dev/fsl_mc_console | grep BLOB
```

the log should contain similar line (otherwise the custom protocol is not usable):

```
[I, DPSPARSER] Soft Parser BLOB parsing : Completed
```

- **verify DPSPARSER actions performed:**

```
cat /dev/fsl_mc_console | grep DPSPARSER
```

the log should contain similar lines (otherwise the custom protocol is not enabled):

```
[I, DPSPARSER] Enable system WRIOP INGRESS SPs on PPID 0
```

```
[I, DPSPARSER] 'afteth' : HXS = 0x1 PC = 0x20 Parameters = 0
```

The interfaces used must be correctly configured by using *ifconfig* command.

An external traffic generator can be used to create test frames with custom protocols and then inject these frames in configured interfaces. These frames are then processed by the Soft Parser according to configuration applied.

### 8.2.7.2.3 Limitations

This section describes the limitations users should consider when working with Soft Parser Blob.

- The U-Boot command to load, parse, and apply a soft parser blob SPB file (*'apply spb'*) can be used only before applying the DPL file (*'apply dpl'*) command. Never try to use *'apply spb'* command after *'apply dpl'* command because this action results in an error and SPB configuration will not be applied.
- There is no support to load a soft parser blob (SPB) file from Linux. Currently this action can be performed only from U-Boot.
- Networking object API (for network objects DPNI, DPDMUX, DPSW) is not currently supported (as it is described in architecture document).
- For SPC Tool limitations see Soft Parser Configuration Tool User Guide.

# Chapter 9

## Linux user space

### 9.1 Libraries

#### 9.1.1 OpenSSL

##### 9.1.1.1 Overview

The Secure Socket Layer (SSL) protocol is the most widely deployed application protocol to protect data during transmission by encrypting the data using popular cipher algorithms such as AES, DES and 3DES.

Apart from encryption it also provides message authentication services using popular hash/digest algorithms such as SHA1 and MD5. SSL is widely used in application web servers (HTTP) and other applications such as SMTP POP3, IMAP, Proxy servers etc., where protection of data in transit is essential.

There are various version of SSL protocol such as SSLv3, TLSv1.0, TLSv1.1, TLSv1.2, TLSv1.3 and DTLS (Datagram TLS). Of all the SSL protocol versions, TLSv1.0 and SSLv3 are in commonly in use, with other versions seeing more adoption as well.

This document introduces NXP SSL acceleration solution on QorIQ platforms using OpenSSL.

##### OpenSSL Software architecture

The OpenSSL library has several sub-components such as:

1. SSL protocol library
2. Crypto library (Symmetric and Asymmetric cipher support, digest support etc.)
3. Certificate Management

The following figure presents the general interconnect architecture for OpenSSL and the interfaces with hardware acceleration drivers:

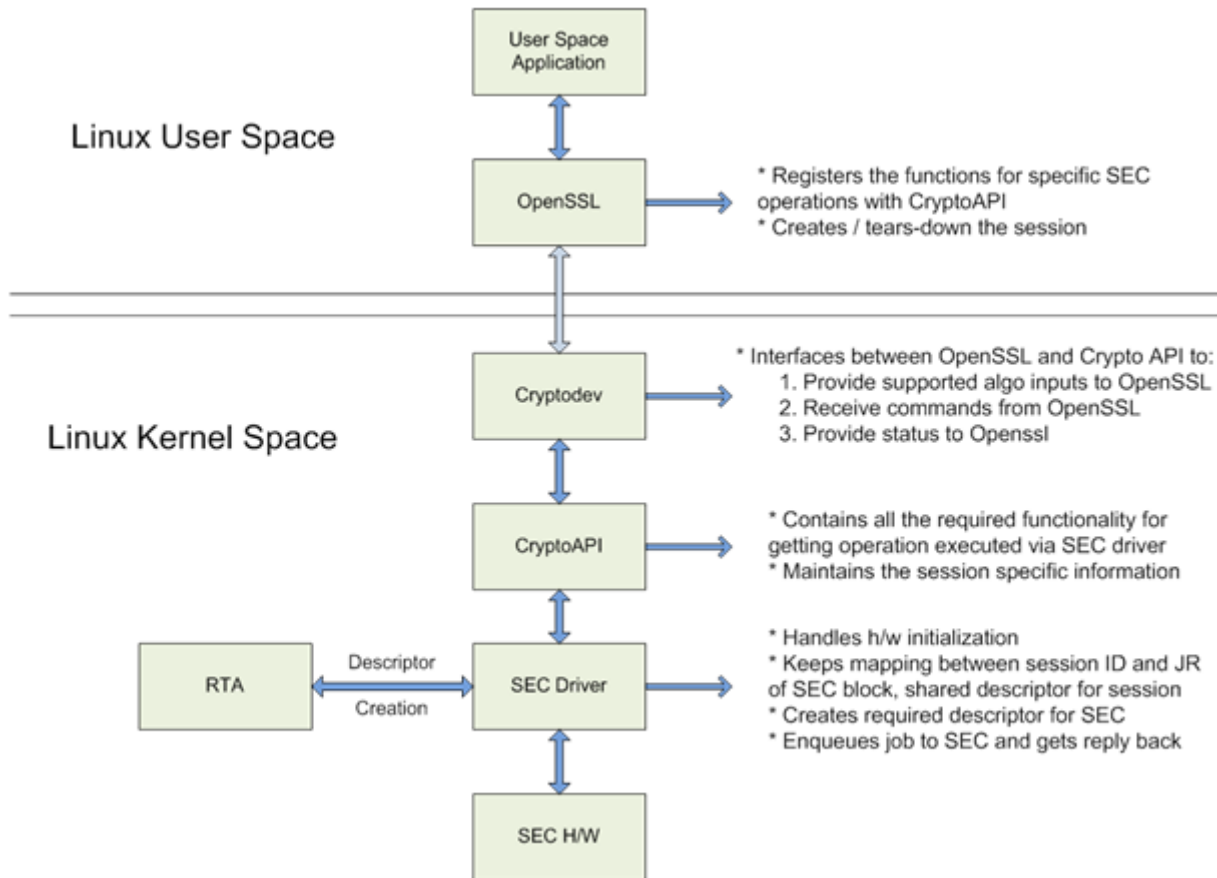


Figure 58. OpenSSL interface with Linux kernel

### OpenSSL's ENGINE Interface

OpenSSL Crypto library provides Symmetric and Asymmetric (PKI) cipher support that is used in a variety of applications such as OpenSSH, OpenVPN, PGP, IKE, XML-SEC etc. The OpenSSL Crypto library provides software support for:

1. Cipher algorithms
2. Digest algorithms
3. Random number generation
4. Public Key Infrastructure

Apart from the software support, OpenSSL can offload these functions to hardware accelerators via the ENGINE interface. The ENGINE interface provides callback hooks that integrate hardware accelerators with the crypto library. The callback hooks provide the glue logic to interface with the hardware accelerators. Generic offloading of cipher and digests algorithms through Linux kernel is possible with cryptodev engine.

### NXP solution for OpenSSL hardware offloading

The following layers can be observed in NXP's solution for OpenSSL hardware offloading:

- OpenSSL (user space) - implements the SSL protocol
- cryptodev-engine (user space) - implements the OpenSSL ENGINE interface; talks to cryptodev-linux (/dev/crypto) via ioctl, offloading cryptographic operations in kernel
- cryptodev-linux (kernel space) - Linux module that translates ioctl requests from cryptodev-engine into calls to Linux Crypto API

- Linux Crypto API (kernel space) - Linux kernel crypto abstraction layer
- CAAM driver (kernel space) - Linux device driver for the CAAM (Cryptographic Acceleration and Assurance Module) crypto engine

The following can be offloaded in hardware in current SDK (not a complete list):

- SSL: TLS v1.0 with one-shot cipher modes (a single ioctl for both encryption and authentication):
  - AES128-SHA
  - AES256-SHA
- Crypto algorithms:
  - AES-CBC
  - 3DES
- Digest algorithms:
  - MD5
  - SHA1
  - SHA256
- Public key algorithms:
  - RSA

### 9.1.1.2 Manual Build of OpenSSL with Cryptodev Engine Support

This chapter is optional since the root filesystem can be configured to include both OpenSSL and cryptodev automatically.

```
$ cd flexbuild
$ source setup.env

Build cryptodev-linux:
$ flex-builder -c cryptodev-linux -a arm64 # automatically setup cross-toolchain and fetch
cryptodev-linux repository to build

Build OpenSSL:
$ flex-builder -c openssl -a arm64

Merge OpenSSL and cryptodev-linux components into target rootfs:
$ flex-builder -i merge-component -a arm64

Generate bootpartition tarball:
$ flex-builder -i mkbootpartition -a arm64
```

Follow flexbuild documentation to finalize the building of the root filesystem and kernel on the host system.

#### Manual build

Manual build or rebuild may be necessary for a number of reasons. This section describes how to build and install OpenSSL natively on the target system. Cross-build procedure requires appropriate toolchains and is not described.

Both OpenSSL and cryptodev must be installed since they depend on each-other:

```
$ git clone https://source.codeaurora.org/external/qoriq/qoriq-components/openssl
$ git clone https://source.codeaurora.org/external/qoriq/qoriq-components/cryptodev-linux
```

Linux user space

Libraries

Build cryptodev and optionally run self tests:

```
$ cd cryptodev-linux
$ make
$ sudo make install
```

```
$ sudo modprobe cryptodev
$ make check
```

Build openssl with cryptodev support:

```
$ cd openssl
$./Configure -DHAVE_CRYPTODEV --prefix=/usr/local/ --openssldir=/usr/local/openssl linux-aarch64
shared
$ make
$ sudo make install
```

After installation verify that the binary is linking with the correct share library from /usr/local/lib:

```
$ ldd /usr/local/bin/openssl
```

If the binary is linking with the original library from /usr/lib then it may be necessary to adjust the linker paths. Put /usr/local/lib in a line before /usr/lib inside /etc/ld.so.conf and then update the linker cache:

File: /etc/ld.so.conf

```
...
/usr/local/lib
...
/usr/lib
...
```

```
$ sudo ldconfig
$ ldd /usr/local/bin/openssl
```

### 9.1.1.3 Hardware Offloading with OpenSSL

#### Overview

OpenSSL can delegate execution of crypto operations to a variety of hardware devices through the engine interface. On top of this interface is implemented the engine cryptodev which is used to offload crypto operations to hardware devices under the control of the operating system kernel. Cryptodev engine was originally developed for OpenBSD and later the same API was ported to GNU/Linux operating system by several drivers like OCF and cryptodev-linux.

Cryptodev-linux is a Linux kernel driver that exposes the internal crypto API to user-space via the device file /dev/crypto. User-space applications use ioctl system calls to ask the Linux kernel to perform crypto operations on their behalf. The Linux kernel supports a multitude of crypto algorithms with software implementations running on CPU. Drivers for hardware accelerators are installed with higher priority and override software implementations with no further configuration.

From the point of view of any application, the fastest implementation of an algorithm is used transparently. This behavior is transferred also to cryptodev interface which is oblivious to the fact that an algorithm may run on CPU or on a hardware accelerator. For this reason, it is the job of the operator of the application to ensure that hardware kernel drivers are available before running the application.

This translates simply to running modprobe if the NXP SEC driver is not built-in the kernel. In our case:

```
modprobe caamalg
modprobe caamhash
modprobe caam_pkc
```

NXP platforms have several SEC frontends exposed with different device drivers: JRI (job ring), QI (queue interface), DPSECI. Refer to your [platform and SEC guide](#) for available Linux kernel drivers. Usually, at least the JR driver should be available for any platform

For QI frontend the symmetric cyphers has a kernel module called caamalg\_qi. This driver installs algorithms with lower priority than caamalg so they will be shadowed by the latter. To use the QI frontend load this driver instead of caamalg:

```
modprobe caamalg_qi
```

DPSECI frontend has yet another driver caamalg\_qi2 which currently is always built-in the kernel.

### Verify setup

There are a few simple steps to confirm if crypto hardware drivers are available. Running these steps contribute to a smooth experience and easier debugging if things go wrong.

Linux kernel can check and report ciphers availability with the help of tcrypt module. After probing tcrypt, crypto algorithms will be listed in /proc/crypto. Tcrypt module is not always available in default kernels but it is a simple way to run tests and list all available crypto algorithms:

```
$ modprobe tcrypt
$ grep aes /proc/crypto
<...>
```

Load CAAM device drivers if they are not built-in and check their interrupt count:

```
modprobe caamalg (or caamalg_qi)
modprobe caamhash
modprobe caam_pkc
<...>
grep tls /proc/crypto
name : tls10(hmac(sha1),cbc(aes))
driver : tls10-hmac-sha1-cbc-aes-caam-qi
grep rsa /proc/crypto
<...>
```

Hardware operations can be monitored with the interrupt counters for CAAM JR and QI (DPAA and DPAA2) interfaces:

```
cat /proc/interrupts | grep jr
88: 0 0 0 0 26 0 0 0 OpenPIC 88 Level ffe301000.jr
89: 0 0 0 0 0 1117204 0 0 OpenPIC 89 Level ffe302000.jr
90: 0 0 0 0 0 0 24 0 OpenPIC 90 Level ffe303000.jr
91: 0 0 0 0 0 0 0 24 OpenPIC 91 Level ffe304000.jr
```

```
cat /proc/interrupts | grep -i qman
108: 0 0 0 0 0 0 0 7508 OpenPIC 108 Level QMan portal 7
110: 0 0 0 0 0 0 7524 0 OpenPIC 110 Level QMan portal 6
112: 0 0 0 0 0 7542 0 0 OpenPIC 112 Level QMan portal 5
114: 0 0 0 0 7565 0 0 0 OpenPIC 114 Level QMan portal 4
116: 0 0 0 0 7576 0 0 0 OpenPIC 116 Level QMan portal 3
118: 0 0 7524 0 0 0 0 0 OpenPIC 118 Level QMan portal 2
120: 0 7535 0 0 0 0 0 0 OpenPIC 120 Level QMan portal 1
```

```
122: 7521 0 0 0 0 0 0 0 0 OpenPIC 122 Level QMan portal 0
470: 0 0 0 0 0 0 0 0 0 OpenPIC 2006 Edge qman-err
```

```
cat /proc/interrupts | grep DPIO
<...>
```

The interrupt counters may also be incremented during networking operations unrelated to crypto. Further analysis is required to understand the source of their modification.

Load cryptodev driver and check if OpenSSL communicates with it. If cryptodev driver is not loaded, openssl will report only dynamic engine support and all operations will be done in software by OpenSSL itself.

```
openssl engine
(dynamic) Dynamic engine loading support

modprobe cryptodev
ls /dev/crypto
<...>
openssl engine
(cryptodev) BSD cryptodev engine
(dynamic) Dynamic engine loading support
```

## Offloading Symmetric and Public Key Algorithms

With cryptodev and SEC drivers loaded there is no other configuration necessary for OpenSSL to run crypto operations through hardware accelerator. OpenSSL will automatically use cryptodev engine if available. Some applications that link with OpenSSL like OpenSSH will automatically use the available accelerator. Others, like nginx web server may need explicit activation in their configuration file.

```
modprobe cryptodev
openssl speed -evp AES128-SHA -elapsed
<...>
openssl speed rsa1024
<...>
```

## TLS 1.0 Offloading in Nginx Server

Nginx does not use any openssl engines by default. If an engine is to be used, including cryptodev, it must be explicitly listed in nginx configuration file. Here is a fragment of nginx configuration file that activates cryptodev and allows hardware offloading of TLS1.0 record layer protocol:

/etc/nginx/nginx.conf:

```
ssl_engine cryptodev;
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000; #for 4 Core CPU; For 2 Core CPU worker_cpu_affinity 01
10;
...
HTTPS server
#
server {
 listen 443;
 server_name localhost;

 ssl on;
 ssl_certificate server.crt;
 ssl_certificate_key server.key;
 ssl_session_timeout 5m;
```



```

ssl_protocols TLSv1;
ssl_ciphers AES128-SHA:AES256-SHA;
ssl_prefer_server_ciphers on;

location / {
 root /var/www/localhost/html;
 index index.html index.htm;
}
...

```

Worker processes and affinity should be set according to the number of CPU cores available on the platform. Refer to nginx documentation for more details.

### TLS1.0 Record Layer Testing

We will use only OpenSSL functionality to verify the offloading of TLS record layer.

First create the RSA public and private keys to be used by the server:

```
$ openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

Start https webserver:

```
modprobe cryptodev
$ openssl s_server -key key.pem -cert cert.pem -accept 44330 -www -cipher AES128-SHA -tls1
```

And connect to it with the client from another console:

```
$ openssl s_client -connect localhost:44330
(or)
$ echo "GET /" | openssl s_client -connect localhost:44330 -quiet
```

Just like with other tests, hardware offloading can be verified by listing the interrupt counters of the SEC driver.

## 9.1.1.4 TLS Ciphersuites and TLS Protocol Versions

Please refer to the official RFC documents for an up-to-date list of supported algorithms:

[Transport Layer Security \(TLS\) Parameters](#)

**Table 55. OpenSSL CipherSuite Compatibility**

| CipherSuite                         | TLS Protocol Version |
|-------------------------------------|----------------------|
| SSL_RSA_WITH_NULL_MD5               | SSL3.0               |
| SSL_RSA_WITH_NULL_SHA               | SSL3.0               |
| SSL_RSA_EXPORT_WITH_RC4_40_MD5      | SSL3.0               |
| SSL_RSA_WITH_RC4_128_MD5            | SSL3.0               |
| SSL_RSA_WITH_RC4_128_SHA            | SSL3.0               |
| SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5  | SSL3.0               |
| SSL_RSA_WITH_IDEA_CBC_SHA           | SSL3.0               |
| Table continues on the next page... |                      |

**Table 55. OpenSSL CipherSuite Compatibility (continued)**

| CipherSuite                            | TLS Protocol Version |
|----------------------------------------|----------------------|
| SSL_RSA_EXPORT_WITH_DES40_CBC_SHA      | SSL3.0               |
| SSL_RSA_WITH_DES_CBC_SHA               | SSL3.0               |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA          | SSL3.0               |
| SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA   | SSL3.0               |
| SSL_DH_DSS_WITH_DES_CBC_SHA            | SSL3.0               |
| SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA       | SSL3.0               |
| SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA   | SSL3.0               |
| SSL_DH_RSA_WITH_DES_CBC_SHA            | SSL3.0               |
| SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA       | SSL3.0               |
| SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA  | SSL3.0               |
| SSL_DHE_DSS_WITH_DES_CBC_SHA           | SSL3.0               |
| SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA      | SSL3.0               |
| SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA  | SSL3.0               |
| SSL_DHE_RSA_WITH_DES_CBC_SHA           | SSL3.0               |
| SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA      | SSL3.0               |
| SSL_DH_anon_EXPORT_WITH_RC4_40_MD5     | SSL3.0               |
| SSL_DH_anon_WITH_RC4_128_MD5           | SSL3.0               |
| SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA  | SSL3.0               |
| SSL_DH_anon_WITH_DES_CBC_SHA           | SSL3.0               |
| SSL_DH_anon_WITH_3DES_EDE_CBC_SHA      | SSL3.0               |
| SSL_FORTEZZA_KEA_WITH_NULL_SHA         | SSL3.0               |
| SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA | SSL3.0               |
| SSL_FORTEZZA_KEA_WITH_RC4_128_SHA      | SSL3.0               |
| TLS_RSA_WITH_NULL_MD5                  | TLS1.0               |
| TLS_RSA_WITH_NULL_SHA                  | TLS1.0               |
| TLS_RSA_EXPORT_WITH_RC4_40_MD5         | TLS1.0               |
| TLS_RSA_WITH_RC4_128_MD5               | TLS1.0               |
| TLS_RSA_WITH_RC4_128_SHA               | TLS1.0               |
| TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5     | TLS1.0               |
| TLS_RSA_WITH_IDEA_CBC_SHA              | TLS1.0               |
| TLS_RSA_EXPORT_WITH_DES40_CBC_SHA      | TLS1.0               |
| Table continues on the next page...    |                      |

**Table 55. OpenSSL CipherSuite Compatibility (continued)**

| CipherSuite                           | TLS Protocol Version |
|---------------------------------------|----------------------|
| TLS_RSA_WITH_DES_CBC_SHA              | TLS1.0               |
| TLS_RSA_WITH_3DES_EDE_CBC_SHA         | TLS1.0               |
| TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA  | TLS1.0               |
| TLS_DH_DSS_WITH_DES_CBC_SHA           | TLS1.0               |
| TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA      | TLS1.0               |
| TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA  | TLS1.0               |
| TLS_DH_RSA_WITH_DES_CBC_SHA           | TLS1.0               |
| TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA      | TLS1.0               |
| TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA | TLS1.0               |
| TLS_DHE_DSS_WITH_DES_CBC_SHA          | TLS1.0               |
| TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA     | TLS1.0               |
| TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA | TLS1.0               |
| TLS_DHE_RSA_WITH_DES_CBC_SHA          | TLS1.0               |
| TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA     | TLS1.0               |
| TLS_DH_anon_EXPORT_WITH_RC4_40_MD5    | TLS1.0               |
| TLS_DH_anon_WITH_RC4_128_MD5          | TLS1.0               |
| TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA | TLS1.0               |
| TLS_DH_anon_WITH_DES_CBC_SHA          | TLS1.0               |
| TLS_DH_anon_WITH_3DES_EDE_CBC_SHA     | TLS1.0               |
| TLS_RSA_WITH_AES_128_CBC_SHA          | TLS1.0               |
| TLS_RSA_WITH_AES_256_CBC_SHA          | TLS1.0               |
| TLS_DH_DSS_WITH_AES_128_CBC_SHA       | TLS1.0               |
| TLS_DH_DSS_WITH_AES_256_CBC_SHA       | TLS1.0               |
| TLS_DH_RSA_WITH_AES_128_CBC_SHA       | TLS1.0               |
| TLS_DH_RSA_WITH_AES_256_CBC_SHA       | TLS1.0               |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA      | TLS1.0               |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA      | TLS1.0               |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA      | TLS1.0               |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA      | TLS1.0               |
| TLS_DH_anon_WITH_AES_128_CBC_SHA      | TLS1.0               |
| TLS_DH_anon_WITH_AES_256_CBC_SHA      | TLS1.0               |
| Table continues on the next page...   |                      |

**Table 55. OpenSSL CipherSuite Compatibility (continued)**

| CipherSuite                                | TLS Protocol Version |
|--------------------------------------------|----------------------|
| TLS_RSA_WITH_CAMELLIA_128_CBC_SHA          | TLS1.0               |
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA          | TLS1.0               |
| TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA       | TLS1.0               |
| TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA       | TLS1.0               |
| TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA       | TLS1.0               |
| TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA       | TLS1.0               |
| TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA      | TLS1.0               |
| TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA      | TLS1.0               |
| TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA      | TLS1.0               |
| TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA      | TLS1.0               |
| TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA      | TLS1.0               |
| TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA      | TLS1.0               |
| TLS_RSA_WITH_SEED_CBC_SHA                  | TLS1.0               |
| TLS_DH_DSS_WITH_SEED_CBC_SHA               | TLS1.0               |
| TLS_DH_RSA_WITH_SEED_CBC_SHA               | TLS1.0               |
| TLS_DHE_DSS_WITH_SEED_CBC_SHA              | TLS1.0               |
| TLS_DHE_RSA_WITH_SEED_CBC_SHA              | TLS1.0               |
| TLS_DH_anon_WITH_SEED_CBC_SHA              | TLS1.0               |
| TLS_ECDH_RSA_WITH_NULL_SHA                 | TLS1.0               |
| TLS_ECDH_RSA_WITH_RC4_128_SHA              | TLS1.0               |
| TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA         | TLS1.0               |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA          | TLS1.0               |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA          | TLS1.0               |
| TLS_ECDH_ECDSA_WITH_NULL_SHA               | TLS1.0               |
| TLS_ECDH_ECDSA_WITH_RC4_128_SHA            | TLS1.0               |
| TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA       | TLS1.0               |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA        | TLS1.0               |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA        | TLS1.0               |
| TLS_ECDHE_RSA_WITH_NULL_SHA                | TLS1.0               |
| TLS_ECDHE_RSA_WITH_RC4_128_SHA             | TLS1.0               |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA        | TLS1.0               |
| <i>Table continues on the next page...</i> |                      |

**Table 55. OpenSSL CipherSuite Compatibility (continued)**

| CipherSuite                           | TLS Protocol Version |
|---------------------------------------|----------------------|
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA    | TLS1.0               |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA    | TLS1.0               |
| TLS_ECDHE_ECDSA_WITH_NULL_SHA         | TLS1.0               |
| TLS_ECDHE_ECDSA_WITH_RC4_128_SHA      | TLS1.0               |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA | TLS1.0               |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA  | TLS1.0               |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA  | TLS1.0               |
| TLS_ECDH_anon_WITH_NULL_SHA           | TLS1.0               |
| TLS_ECDH_anon_WITH_RC4_128_SHA        | TLS1.0               |
| TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA   | TLS1.0               |
| TLS_ECDH_anon_WITH_AES_128_CBC_SHA    | TLS1.0               |
| TLS_ECDH_anon_WITH_AES_256_CBC_SHA    | TLS1.0               |
| TLS_RSA_WITH_NULL_SHA256              | TLS1.2               |
| TLS_RSA_WITH_AES_128_CBC_SHA256       | TLS1.2               |
| TLS_RSA_WITH_AES_256_CBC_SHA256       | TLS1.2               |
| TLS_RSA_WITH_AES_128_GCM_SHA256       | TLS1.2               |
| TLS_RSA_WITH_AES_256_GCM_SHA384       | TLS1.2               |
| TLS_DH_RSA_WITH_AES_128_CBC_SHA256    | TLS1.2               |
| TLS_DH_RSA_WITH_AES_256_CBC_SHA256    | TLS1.2               |
| TLS_DH_RSA_WITH_AES_128_GCM_SHA256    | TLS1.2               |
| TLS_DH_RSA_WITH_AES_256_GCM_SHA384    | TLS1.2               |
| TLS_DH_DSS_WITH_AES_128_CBC_SHA256    | TLS1.2               |
| TLS_DH_DSS_WITH_AES_256_CBC_SHA256    | TLS1.2               |
| TLS_DH_DSS_WITH_AES_128_GCM_SHA256    | TLS1.2               |
| TLS_DH_DSS_WITH_AES_256_GCM_SHA384    | TLS1.2               |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256   | TLS1.2               |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256   | TLS1.2               |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256   | TLS1.2               |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384   | TLS1.2               |
| TLS_DHE_DSS_WITH_AES_128_CBC_SHA256   | TLS1.2               |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA256   | TLS1.2               |

*Table continues on the next page...*

**Table 55. OpenSSL CipherSuite Compatibility (continued)**

| CipherSuite                             | TLS Protocol Version |
|-----------------------------------------|----------------------|
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256     | TLS1.2               |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384     | TLS1.2               |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256    | TLS1.2               |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384    | TLS1.2               |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256    | TLS1.2               |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384    | TLS1.2               |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256  | TLS1.2               |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384  | TLS1.2               |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256  | TLS1.2               |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384  | TLS1.2               |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256   | TLS1.2               |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384   | TLS1.2               |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   | TLS1.2               |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   | TLS1.2               |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | TLS1.2               |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 | TLS1.2               |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | TLS1.2               |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | TLS1.2               |
| TLS_DH_anon_WITH_AES_128_CBC_SHA256     | TLS1.2               |
| TLS_DH_anon_WITH_AES_256_CBC_SHA256     | TLS1.2               |
| TLS_DH_anon_WITH_AES_128_GCM_SHA256     | TLS1.2               |
| TLS_DH_anon_WITH_AES_256_GCM_SHA384     | TLS1.2               |

## 9.1.2 Runtime Assembler Library Reference

Use the Runtime Assembler Library to write SEC descriptors.

### 9.1.2.1 Runtime Assembler Library Reference

Use the Runtime Assembler Library to write SEC descriptors. This reference describes the structure, concept, functionality, and high level API.

[Click here](#) to access the Writing descriptors for NXP CAAM using RTA library PDF.

## 9.2 Data Plane Development Kit (DPDK)

### 9.2.1 Introduction

DPDK is an user space packet processing framework.

This guide contains instructions for installing and configuring the user space Data Plane Development Kit (DPDK) v17.11 software. Besides highlighting the applicable platforms, this guide describes steps for compiling and executing sample DPDK applications in a Linux application (*linuxapp*) environment over Layerscape boards.

OVS-DPDK is a popular software switching package which uses DPDK as the underlying platform. The guide also detail methods to execute *ovs-dpdk* in conjunction with DPDK over Layerscape boards.

### 9.2.1.1 Platform-specific Details

DPDK supports LX2160 family of SoCs. This section explains LX2160 block diagram and mapping of Ethernet port names appearing on the chassis front panel of LX2160ARDB with the physical port names used in the DPL file. This Ethernet port mapping information is required for configuring Ethernet ports for DPDK.

## Hardware specifications

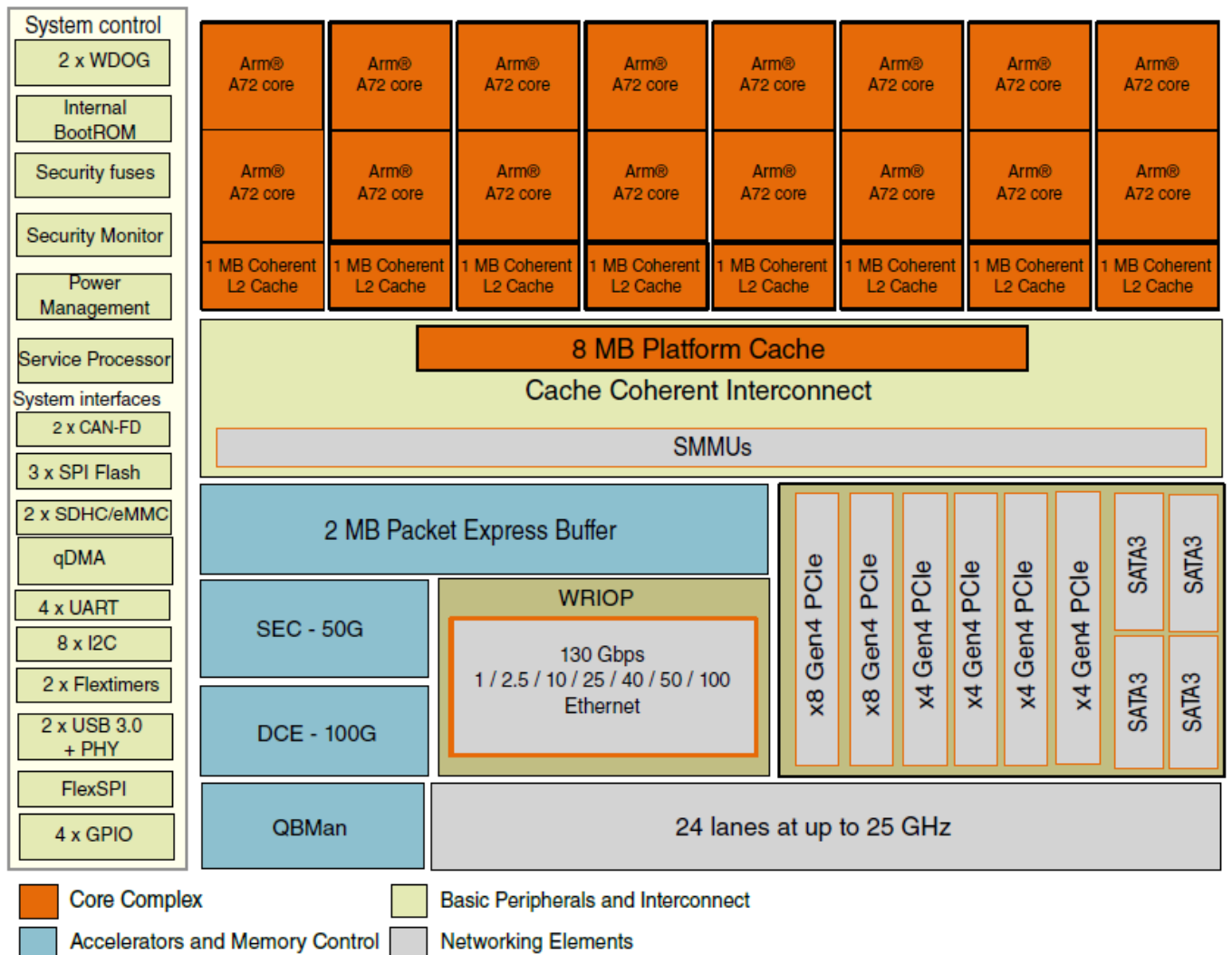


**Figure 1-1. LX2160A block diagram**



Figure 59. LX2160ARDB Ethernet ports

Table 56. LX2160ARDB Ethernet port mapping

| Port name on chassis | Physical port name used in DPL file | Description       |
|----------------------|-------------------------------------|-------------------|
| 40G MAC2             | dpmac.2                             | 40G               |
| 10G MAC3             | dpmac.3                             | 10G - copper port |
| 10G MAC4             | dpmac.4                             | 10G - copper port |
| 25G MAC5             | dpmac.5                             | 25G               |
| 25G MAC6             | dpmac.6                             | 25G               |
| 1G MAC               | dpmac.17                            | 1G – copper port  |
| 1G MAC               | dpmac.18                            | 1G – copper port  |

Table 57. Minimum frame size for port line rate performance

| Port Line Rate [Gbps] | Minimum Frame Size for Port Line Rate Performance [bytes] |                 |                 |                     |                 |                 |                     |                 |                 |
|-----------------------|-----------------------------------------------------------|-----------------|-----------------|---------------------|-----------------|-----------------|---------------------|-----------------|-----------------|
|                       | Platform Clock Frequency [MHz]                            |                 |                 |                     |                 |                 |                     |                 |                 |
|                       | 800                                                       |                 |                 | 700 <sup>[11]</sup> |                 |                 | 650 <sup>[11]</sup> |                 |                 |
|                       | RX                                                        | TX Multi-Queues | TX Single Queue | RX                  | TX Multi-Queues | TX Single Queue | RX                  | TX Multi-Queues | TX Single Queue |
| 25                    | 64                                                        | 64              | 128             | 64                  | 64              | 128             | 64                  | 128             | 128             |
| 40                    | 64                                                        | 128             | 256             | 64                  | 128             | 256             | 64                  | 256             | 256             |
| 50                    | 64                                                        | 128             | 256             | 128                 | 256             | 256             | 128                 | 256             | 256             |
| 100                   | 128                                                       | 256             | 512             | N/A                 | N/A             | N/A             | N/A                 | N/A             | N/A             |

1 The results in this table were originally generated using tests running in a unit-level environment. The 800 MHz results were subsequently validated in the sub-system level test bench. However, the 700 and 650 MHz results have not been validated in the sub-system level simulation.

[11] The results in this table were originally generated using tests running in a unit-level environment. The 800 MHz results were subsequently validated in the sub-system level test bench. However, the 700 and 650 MHz results have not been validated in the sub-system level simulation.



## 9.2.1.2 References

**Table 58. DPDK Application References**

| Sample Applications                                          | DPDK Web Manual Link               | Description                                                                              |
|--------------------------------------------------------------|------------------------------------|------------------------------------------------------------------------------------------|
| Layer-2 Forwarding ( <code>l2fwd</code> )                    | <a href="#">l2fwd usage</a>        | Layer 2 Forwarding sample application setup and usage guide.                             |
| Layer-2 Forwarding with Crypto ( <code>l2fwd-crypto</code> ) | <a href="#">l2fwd-crypto</a>       | Layer 2 Forwarding with Crypto sample application setup and usage guide.                 |
| Layer-3 Forwarding ( <code>l3fwd</code> )                    | <a href="#">l3fwd usage</a>        | Layer 3 Forwarding sample application setup and usage guide.                             |
| IPSec Gateway ( <code>ipsec-secgw</code> )                   | <a href="#">ipsec-secgw usage</a>  | IPSec Security Gateway sample application setup and usage guide.                         |
| PMD Test Application ( <code>testpmd</code> )                | <a href="#">testpmd usage</a>      | Guide for test application which can be used to test all PMD supported features.         |
| DPDK Web Guide                                               | <a href="#">DPDK Documentation</a> | Link to DPDK Web Manual containing information about all supported PMD and Applications. |

**Table 59. Release References**

| Component | Base Release Versions |
|-----------|-----------------------|
| DPDK      | 17.11                 |
| OVS       | 2.9                   |

## 9.2.2 DPDK Overview

Key goal of the DPDK is to provide a simple, complete framework for fast packet processing in data plane applications. Using the APIs provided as part of the framework, applications can leverage the capabilities of underlying network infrastructure.

The framework creates a set of libraries for target environments, layered through an Environment Abstraction Layer (EAL) which hides all the device glue logic beneath a set of consistent APIs. These environments are created through the use of configuration files. Once the EAL library is created, the user may link with the library to create their own applications. Various other libraries, outside of EAL, including the Hash, Longest Prefix Match (LPM) and rings libraries are also available for performing specific operations. Sample applications are also provided to help understand various features and uses of DPDK framework.

DPDK implements a run-to-completion model for packet processing where all resources must be allocated prior to calling data plane applications, running as execution units on logical processing cores. In addition, a pipeline model may also be used by passing packets or messages between cores via rings. This allows work to be performed in stages, resulting in more efficient use of code on cores.

More information on general working of DPDK can be found through [DPDK website](#).

### 9.2.2.1 DPDK Platform Support

This section describes the NXP Data Path Acceleration Architecture, see the diagram below:

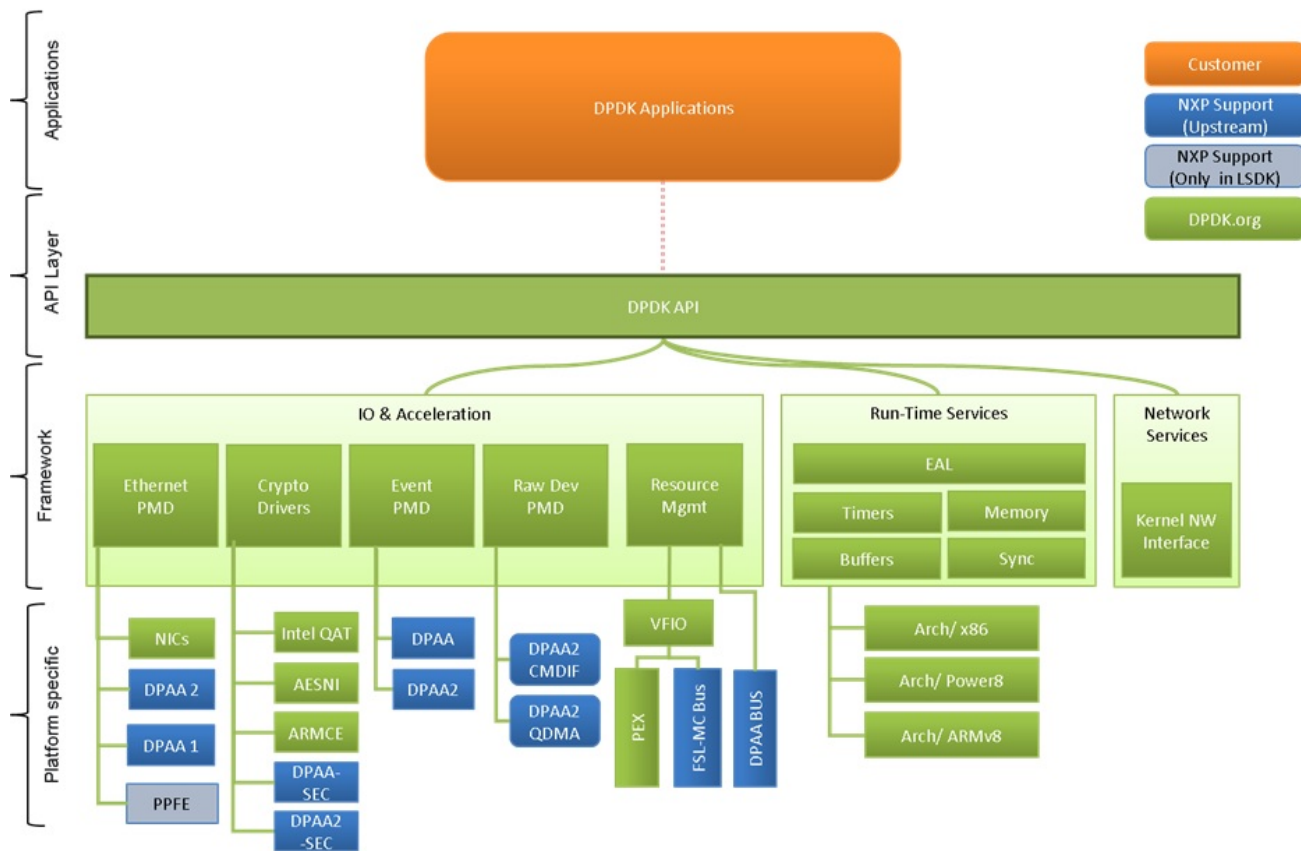


**Figure 60. DPK Architecture with NXP Components**

The NXP Data Path Acceleration Architecture comprises a set of hardware components which are integrated via a hardware queue manager and use a common hardware buffer manager. Software accesses the DPAA via hardware components called "Software Portals". These directly provide queue and buffer manager operations such as enqueues, dequeues, buffer allocations, and buffer releases and indirectly provide access to all of the other DPAA hardware components via the queue manager.

NXP DPAA architecture based *PMD (Poll Mode Drivers)* has been added to DPDK infrastructure to support seamless working on NXP platform. With the addition of these drivers, DPDK framework on NXP platforms permits Linux user space applications to be build using standard DPDK APIs in a portable fashion. The drivers directly access the DPAA queue and buffer manager software portals in a high performance manner and the internal details remains hidden from higher level DPDK framework. Besides drivers for network interfaces, drivers (PMDs) for interfacing with Crypto (CAAM) block have also been included in the DPDK source code.

#### NOTE

Since this guide contains support for PPFE, DPAA2 and DPAA platforms, the following markers are used throughout the guide:

- DPAA2 – This marker marks the steps/text applicable only for DPAA2 platforms, for example, LS2088
- DPAA – This marker marks the steps/text applicable only for DPAA platforms, for example, LS1043
- PPFE - This marker marks the steps/text applicable only for PPFE platforms, for example, LS1012

All other steps which don't have any marker are applicable for both the platforms.

---

**NOTE**

See [DPDK Performance Reproducibility Guide](#) on page 534 to tune the system for best DPDK performance on NXP platforms.

---

---

**NOTE**

**Multi-thread environment**

DPDK was originally designed for Intel architectures, however efforts are underway to make it multiple architecture friendly. There are still some restrictions which should be taken care when used on NXP platforms.

1. Multiple pthreads

DPDK usually pins one pthread per core to avoid the overhead of task switching. This allows for significant performance gains, but lacks flexibility and is not always efficient. DPDK is comprised of several libraries - some of the functions in these libraries can be safely called from multiple threads simultaneously, while others cannot.

The run-time environment of the DPDK is typically a single thread per logical core. It is best to avoid sharing data structures between threads and/or processes where possible. Where this is not possible, the execution blocks must access the data in a thread-safe manner. Mechanisms such as atomic variables or locking can be used to allow execution blocks to operate serially. However, this can effect the performance of the application.

2. Fast-path APIs

Applications operating in the data plane are performance sensitive but certain functions within those libraries may not be safe to call from multiple threads simultaneously.

The Hash, LPM, Mempool libraries and RX/TX in the PMD are examples of such multi-thread unsafe functions. The RX/TX of the *PMD* are the most critical aspects of a DPDK application and it is recommended that no locking be used with these paths as it will impact performance. However, these functions can be safely used from multiple threads when each thread is performing I/O on a different NIC queue. If multiple threads are to use the same hardware queue on the same NIC port, then locking or some other form of mutual exclusion is necessary. In the NXP implementation, each thread has to use a software portal (DPIO) instance to access the underlying DPAA hardware. Thus, it is recommended that only one thread per logical core should be created for RX/TX and other I/O access to DPAA hardware.

---

## 9.2.2.2 DPAA: Supported DPDK Features

Following is the list of DPDK NIC features which DPAA driver support:

- Allmulticast mode
- Basic stats
- Extended stats
- Flow control
- Firmware Version information
- Jumbo frame
- L3 checksum offload
- L4 checksum offload
- Link status
- MTU update
- Promiscuous mode

- Queue start/stop
- Speed Capabilities
- Scattered RX
- Unicast MAC filter
- RSS Hash
- Packet type parsing
- ARMv8

### 9.2.2.3 DPAA2: Supported DPDK Features

Following is the list of DPDK NIC features which DPAA2 driver support:

- Allmulticast mode
- Basic stats
- Firmware Version information
- Flow control
- Jumbo frame
- L3 checksum offload
- L4 checksum offload
- Link Status
- Link Status Events
- MTU update
- Packet type parsing
- Promiscuous mode
- Queue start/stop
- RSS hash
- Unicast MAC filter
- VLAN offload
- VLAN filter
- Speed capabilities
- ARMv8
- Linux VFIO
- Extended stats

### 9.2.2.4 PPFE supported DPDK features

Following is the list of DPDK NIC features which PPFE driver support:

- ALLmulticast mode
- Basic Stats
- MTU update
- Promiscuous mode

- Packet type parsing
- ARMv8

## 9.2.3 Build DPDK

This section includes three sub-sections which detail:

1. Building DPDK binaries (libraries and sample applications) using the Flexbuild build system.
2. Building DPDK binaries as standalone package, through DPDK's own build system.
3. Building Pktgen application which can be used as a software packet generator using DPDK as underlying layer.

### 9.2.3.1 Build DPDK using Flexbuild

DPDK is one of the application packages of the Flexbuild system. This section details method to build DPDK as a standalone package within the Flexbuild environment. It is assumed that the Flexbuild environment has already been configured before executing the commands below.

See [LX2160A BSP user guide](#) on page 32 for complete details of using the Flexbuild build system.

Once the Flexbuild environment has been setup, following commands can be used to build DPDK applications and libraries. Generated files (libraries and binaries) would be available in the `<Flexbuild>/build/apps/components_arm64` folder. Once the *rootfs* (root filesystem) is generated, the *components\_arm64* folder would be merged in it.

```
flex-builder -c openssl -a arm64 # to resolve the dependency on OpenSSL package
flex-builder -c linux -a arm64 # to resolve the dependency of KNI module
```

```
flex-builder -c dpdk -a arm64 # build dpdk application
```

```
flex-builder -c pktgen-dpdk -a arm64 # to generate dpdk pktgen application
```

#### NOTE

DPDK is dependent on OpenSSL package for software crypto and OpenSSL PMD. It is necessary to build OpenSSL before DPDK in Flexbuild environment to suffice this dependency. If building DPDK on target platform, it is possible that OpenSSL libraries are already available in library path. In this case, building OpenSSL library would not be required.

See [How to build LSDK with Flexbuild](#) on page 48 for packing these binaries into the target *rootfs* using the Flexbuild build system.

#### Layout of DPDK Binaries

Single image of DPDK binary supports both the DPAA and DPAA2, and PPFE platforms. Once the DPDK package has been installed, binaries would be available in `/usr/local/bin` folder in the *rootfs*. Flexbuild system generates a single *rootfs* for all NXP platforms it supports.

```
/usr/local/bin # Contains the sample applications listed in Table 58. DPDK
Application References on page 477
/usr/local/include/dpdk # All DPDK header necessary for external application development
/usr/local/lib # Various static DPDK libraries for external application development
```

DPDK binaries have been placed in the `/usr/local/bin` folder to take advantage of the binary search path set in the `PATH` variable. In case the `PATH` variable doesn't contain the `/usr/local/bin` by default, it can be added to it to enable BASH command completion.

At various places in this document, above binaries would be referred for representing execution as well as other information. It is assumed that execution is being done either using the `PATH` variable set, as explained above, or with absolute path to the binaries.

Besides the above folders, another set of files are also available in *rootfs* to support DPDK application execution. These files are available in the `/usr/local/dpdk` folder in the *rootfs*.

Table below depicts various DPDK artifacts which are available in the Flexbuild generated *rootfs*:

| S/No | File/Image name related to <code>/usr/local/</code>                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                              |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | <code>./lib/lib*.a</code>                                                                                                                                                                                                          | DPAA, DPAA2, and PPFE<br>Static Libraries for compiling external applications.                                                                                                                                                                                                           |
| 2    | <code>./include/dpdk/*.h</code>                                                                                                                                                                                                    | DPAA, DPAA2, and PPFE<br>Headers for compiling external applications.                                                                                                                                                                                                                    |
| 2    | <code>./bin/l2fwd</code><br><code>./bin/l3fwd</code><br><code>./bin/l2fwd-crypto</code><br><code>./bin/ipsec-secgw</code><br><code>./bin/testpmd</code>                                                                            | DPAA, DPAA2, and PPFE<br>DPDK Example applications and PMD test application.                                                                                                                                                                                                             |
| 3    | <code>./dpaa/usdpaa_config_ls&lt;PLAT&gt;.xml</code><br><code>./dpaa/usdpaa_policy_hash_ipv4_1queue.xml</code><br><code>./dpaa/usdpaa_policy_hash_ipv4_2queue.xml</code><br><code>./dpaa/usdpaa_policy_hash_ipv4_4queue.xml</code> | DPAA Only.<br>FMC Configurations and Policy files.<br><br><PLAT> is platform name for DPAA platform, for example <code>ls1043</code> or <code>ls1046</code> .<br>Each Policy file for defining the number of queues per port as mentioned in its name.                                   |
| 4    | <code>./dpaa2/dynamic_dpl.sh</code><br><code>./dpaa2/destroy_dynamic_dpl.sh</code>                                                                                                                                                 | DPAA2 Only.<br>Dynamic DPL container creation and teardown script.                                                                                                                                                                                                                       |
| 5    | <code>./share/dpdk/usertools/dpdk-setup.sh</code><br><code>./share/dpdk/usertools/dpdk-devbind.py</code>                                                                                                                           | DPAA, DPAA2, and PPFE<br>DPDK NIC binding utility.<br>This is only applicable for executing DPDK applications in VM.                                                                                                                                                                     |
| 6    | <code>./disable_services.sh</code>                                                                                                                                                                                                 | When executing a Ubuntu OS over Layerscape board, performance on core 0 can become non-deterministic because of OS services and threads.<br><br>This script disables the extra services on a Ubuntu OS so that all cores of a board can be used without major impact to the performance. |

*Table continues on the next page...*

Table continued from the previous page...

|    |                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                       |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7. | ./examples/ipsec_secgw/ep0.cfg<br>./examples/ipsec_secgw/ep1.cfg<br>./ipsec/ep0_64X64.cfg<br>./ipsec/ep1_64X64.cfg<br>./ipsec/ep0_64X64_proto.cfg<br>./ipsec/ep0_64X64_sha256.cfg<br>./ipsec/ep1_64X64_proto.cfg<br>./ipsec/ep1_64X64_sha256.cfg | Configuration files for ipsec-gw example application.<br><br>The ep0 and ep1 files are standard configurations for 2 tunnels for encryption and decryption, each. The ep0_64X64 and ep1_64X64 are for 64 tunnels for encryption and decryption, each. |
| 8. | /usr/bin/pktgen                                                                                                                                                                                                                                  | Packetgeneration application                                                                                                                                                                                                                          |
| 9. | ./debug_dump.sh                                                                                                                                                                                                                                  | Dumping the debug data for further analysis.                                                                                                                                                                                                          |

### 9.2.3.2 Standalone build of DPDK Libraries and Applications

This section details steps required to build DPDK binaries (libraries and example applications) in a standalone environment. This environment can either be on a host enabled for cross building for Layerscape boards or directly on the Layerscape target board.

#### NOTE

This section primarily focuses on standalone building of DPDK on a host machine using cross compilation for Layerscape boards as target. Though, necessary notes have been added to enable compilation directly on target boards. Refer [How to build LSDK with Flexbuild](#) on page 48 for creating an environment suitable for building DPDK on Layerscape boards.

For steps detailing building DPDK using Flexbuild system, refer [How to build LSDK with Flexbuild](#) on page 48 and [Build DPDK using Flexbuild](#) on page 481.

#### Obtain the DPDK source code

The DPDK source code contains all the necessary libraries for build example applications as well as test applications. The source code also includes various configuration and scripts for supporting build and execution. Obtain the DPDK source code using the link below:

- ```
git clone https://source.codeaurora.org/external/qoriq/qoriq-components/dpdk -b github.qoriq-os/integration
```

Once the above repository has been cloned, DPDK source code is available for compilation. This source is common for both, DPAA, DPAA2, and PPFE platforms.

Prerequisites before compiling DPDK

Before compiling DPDK as a standalone build, following dependencies need to be resolved independently:

- Platform compliant and compiled Linux Kernel source code so that KNI modules can be built.
 - This is optional and if KNI module support is not required, this can be ignored.
 - For details of compiling platform compliant Linux Kernel, refer [How to build LSDK with Flexbuild](#) on page 48.
 - For disabling KNI module, see notes below.
- OpenSSL libraries required for building software crypto driver (OpenSSL PMD).

- OpenSSL package needs to be separately compiled and libraries installed at a known path before DPDK build can be done.
- This is optional and if software crypto driver support is not required, this dependency can be ignored.

NOTE

Refer to [How to build LSDK with Flexbuild](#) on page 48 for more information on how to build OpenSSL as part of Flexbuild system. If using Flexbuild and referring to this link for building OpenSSL package, commands specified below can be skipped.

Following steps are for building OpenSSL as a standalone package, outside the Flexbuild system. This is not a preferred way and should be used only if Flexbuild system is not available. Follow the steps given below to build OpenSSL package.

```
git clone git://git.openssl.org/openssl.git # Clone the OpenSSL source code
cd openssl # Change into cloned directory
git checkout OpenSSL_1_1_0 # Checkout the specific branch supported by DPDK
```

Export the Cross Compilation tool chain for building OpenSSL for target. The following step for exporting cross compilation toolchain is required only when compiling on Host. On a target board, it is assumed default build toolchain would be used.

```
export CROSS_COMPILE=<path to uncompressed toolchain archive>/bin/aarch64-linux-gnu-
```

Configure the OpenSSL build system with following command. The `--prefix` argument specifies a path where OpenSSL libraries would be deployed after build completes. This is also a path which would be provided to DPDK build system for accessing the compiled OpenSSL libraries.

```
./Configure linux-aarch64 --prefix=<OpenSSL library path> shared
```

```
make depend
make
make install
export OPENSSL_PATH=<OpenSSL library path>
```

NOTE

When building DPDK on target board, it is possible that OpenSSL libraries required by DPDK are already available as part of the *rootfs*, in which case external compilation of OpenSSL package would not be required.

- For disabling OpenSSL PMD support, see notes below.

Compiling DPDK

Follow the below steps to compile DPDK once the above prerequisites are resolved. These steps are common for DPAA and DPAA2 targets and are needed only when cross compiling on a host for Layerscape boards as target. In case of direct compilation on target boards, it is assumed that prerequisites would be satisfied using the root filesystem. In case root filesystem doesn't contain necessary prerequisites, below steps would be required once prerequisites have been built/obtained independently.

1. Setup the environment for compilation

- a. Setup Linux Kernel path. This is optional and required only for KNI and ixgb_uio module compilation. Skip it, if ixgb_uio or KNI module or KNI example application is not required.

```
export RTE_KERNELDIR=<Path to compiled Linux kernel to compile KNI kernel module>
```

- b. Setup cross compilation toolchain.

This step is required only on the host environment where default toolchain is not for target boards. When compiling on a target board, this step can be skipped.

```
export CROSS=<path to cross-compile toolchain>
```

- c. Setup OpenSSL path for software crypto drivers (OpenSSL PMD). This is optional and can be skipped in case software crypto driver (OpenSSL PMD) support is not required.

```
export OPENSSL_PATH=<path to installed OpenSSL>
```

2. Use DPDK build system for compiling DPDK.

NOTE

DPDK binaries generated using below steps are compatible for DPAA, DPAA2, and PPFE platforms.. This is also valid when DPDK is build through Flexbuild build system. Refer [How to build LSDK with Flexbuild](#) on page 48 for steps to build DPDK using Flexbuild build system.

- a. Execute the following command:

```
make T=arm64-dpaa-linuxapp-gcc install DESTDIR=<location to install DPDK>
```

Where `DESTDIR=<location to install DPDK>` is an optional parameter to deploy all the DPDK binaries (libraries and example applications) to a standard Linux package specific layout within a directory represented by this parameter. Alternatively, a directory named `arm64-dpaa-linuxapp-gcc` is also created and binaries and libraries are also available in it.

- b. **Disabling KNI and other kernel module compilation:** In case DPDK kernel modules is not required (`RTE_KERNELDIR` variable is not set), use the following command. `DESTDIR` can be added, as explained above, if required.

```
make T=arm64-dpaa-linuxapp-gcc CONFIG_RTE_KNI_KMOD=n CONFIG_RTE_EAL_IGB_UIO=n  
CONFIG_RTE_EAL_IGB_UIO=n install
```

- c. **Enabling software crypto driver support:** Software crypto driver (OpenSSL PMD) is disabled by default. If it is required set `OPENSSL_PATH` variable, use the following command. `DESTDIR` can be added, as explained above, if required.

```
make T=arm64-dpaa-linuxapp-gcc CONFIG_RTE_LIBRTE_PMD_OPENSSL=y EXTRA_CFLAGS="-I$  
{OPENSSL_PATH}/include/" EXTRA_LDFLAGS="-L${OPENSSL_PATH}/lib/" install
```

- d. In case KNI is not required and software crypto support is required, use the following command. `DESTDIR` can be added, as explained above, if required..

```
make T=arm64-dpaa-linuxapp-gcc CONFIG_RTE_LIBRTE_PMD_OPENSSL=y EXTRA_CFLAGS="-I$  
{OPENSSL_PATH}/include/" EXTRA_LDFLAGS="-L${OPENSSL_PATH}/lib/" CONFIG_RTE_KNI_KMOD=n  
CONFIG_RTE_EAL_IGB_UIO=n install
```

NOTE

For more information about the DPDK build system, refer [DPDK Documentation](#).

NOTE

DPDK `arm64-dpaa-linuxapp-gcc` folder contains `.config` file for storing the build configuration. Another way of disabling or enabling support features, like KNI and software crypto drivers, is to edit this file before executing the `make` command. If this method is adopted, parameters to command line for disabling the feature are not required.

NOTE

If KNI or software crypto driver support is disabled using the `make` command line parameters, it would *not* modify the configuration file for DPDK in the `<target>` folder. Every subsequent compilation of DPDK or example application would need to include the same command line arguments to avoid failure because of missing features which were not compiled. Or, edit the `.config` folder in the `arm64-dpaa-linuxapp-gcc` build folder.

Compiling DPDK Example applications

Once the DPDK source code has been compiled, the DPDK example applications can be built independently as required.

1. Before the example applications can be built, the path to DPDK SDK needs to be set which includes the DPDK source code. This would be used by build system to look for compiled libraries and headers.

```
export RTE_SDK=<path to DPDK source code, where compilation was done>
```

2. Target should be set to same value as done for compilation of DPDK.

```
export RTE_TARGET=arm64-dpaa-linuxapp-gcc
```

3. Once the above variables are set, example applications can be compiled using the following commands:

```
make -C examples/l3fwd          # for the L3 forwarding application
```

```
make -C examples/l2fwd          # for the L2 forwarding application
```

```
make -C examples/ip_fragmentation # for the IP fragmentation application
```

```
make -C examples/ip_reassembly   # for the IP reassembly application
```

```
make -C examples/ipsec-secgw      # for the IPSec gateway application
```

```
make -C examples/ipsec-secgw CONFIG_RTE_LIBRTE_PMD_OPENSSL=y EXTRA_CFLAGS="-I${OPENSSL_PATH}/  
include/" EXTRA_LDFLAGS="-L${OPENSSL_PATH}/lib/" # for IPSec application with openssl  
PMD
```

```
make -C examples/l2fwd-crypto      # for the L2 forwarding with crypto support application
```

```
make -C examples/l2fwd-crypto CONFIG_RTE_LIBRTE_PMD_OPENSSL=y EXTRA_CFLAGS="-I${OPENSSL_PATH}/  
include/" EXTRA_LDFLAGS="-L${OPENSSL_PATH}/lib/" # for L2 forwarding crypto operations  
with openssl PMD
```

Above are sample commands for a limited set of DPDK example applications. Other applications too be compiled using similar command pattern.

NOTE

- All the example applications currently supported by DPDK are available as part of the DPDK source code in the `./examples/` folder. Other examples can also be compiled using the pattern stated above.
- It should be noted though that verification of all example applications has not been done for DPAA, DPAA2 and PPFE platforms
- `testpmd` is not supported on PPFE platform. Only `l2fwd`, `l3fwd` and `ipsec-secgw` examples are supported by PPFE.

4. Once the example application are compiled, the binaries would be available in the following folder within the DPDK source code folder:

```
examples/l3fwd/build/app/*           # for L3 forwarding application binary and map file
```

Similar to the pattern above, binaries for the example applications compiled would be available in their respective `build/app` folder.

Besides the above example application, DPDK also provides a `testpmd` binary which can be used for comprehensive verification of DPDK driver (PMD) features for available and compatible devices. This binary is compiled by default during DPDK source compilation explained in [Compiling DPDK](#) section.

NOTE

Only a small set of DPDK example applications are currently deployed to root filesystem (`/usr/local/bin`) when compiling DPDK through Flexbuild build system. These are: `l2fwd`, `l3fwd`, `l2fwd-crypto`, `ipsec-gw` and `testpmd`.

9.2.3.3 DPDK based Packet Generator

Pktgen is a packet generator powered by DPDK. It requires DPDK environment for compilation and DPDK compliant infrastructure for execution. DPAA and DPAA2 DPDK PMD (Poll Mode Drivers) can be used by Pktgen for building a packet generator using the DPAA infrastructure.

Prerequisites for compiling Pktgen

For compiling Pktgen, **libpcap** library is required. If **Pktgen** is being built as a cross compiled target, the `libpcap` too should be compiled against the same compiled. If using the Flexbuild system, `libpcap` can be obtained as an external package from Ubuntu repository. Refer [How to build LSDK with Flexbuild](#) on page 48 for more information.

NOTE

For `libpcap` library compilation and deployment, refer [Tcpdump and libpcap project](#) pages. `Libpcap` current and past releases can be obtained from [this](#) link. Documentation for `libpcap` is included in its source code. Also note that `libpcap` should be compiled for target board if working in a cross compilation environment.

Obtaining the Pktgen source code

Fetch the **Pktgen** source code using the following clone command:

```
git clone http://dpdk.org/git/apps/pktgen-dpdk
git checkout pktgen-3.4.3
```

Compiling Pktgen

Compilation steps below assume that compiled DPDK binaries (libraries and headers) are available in build directory generated by DPDK. Refer [DPDK Build Steps](#) for compiling DPDK and creating the build (`arm64-dpaa-linuxapp-gcc`) directory. Further, it is expected that `libpcap` libraries and headers are also present in this build folder.

Export the path to DPDK build environment and build folder defined by the compilation target:

```
export RTE_SDK=<path to compiled DPDK source code containing build folder>
```

```
export RTE_TARGET=<arm64-dpaa-linuxapp-gcc or arm64-dpaa-linuxapp-gcc> # Select the build folder
based on required DPAA or DPAA2 target>
```

Build the source code:

```
make
```

Before executing the Pktgen application

For executing the Pktgen application, *Pktgen.lua* file and *pktgen* binary are needed on the execution environment. If build was done using a cross compiled environment, transfer these binaries to the target environment from the build host. If the compilation was done on the target board, skip this step.

```
cd <Pktgen compiled source code>
cp Pktgen.lua <target board>
cp app/app/arm64-dpaa-linuxapp-gcc/pktgen <target board>
```

9.2.3.4 Build OVS-DPDK using Flexbuild

OVS is a popular multilayer virtual switch for enabling massive network automation through programmatic extensions. OVS-DPDK is one of the application packages of the Flexbuild system which used DPDK as underlying framework. This section details method to build OVS-DPDK as a standalone package within the Flexbuild environment. It is assumed that the Flexbuild environment has already been configured before executing the commands below. Refer to [LX2160A BSP user guide](#) on page 32 for complete details of using the Flexbuild build system.

NOTE
In the Flexbuild configurations, OVS-DPDK needs to be configured to 'y' for enabling packaging of OVS-DPDK in Flexbuild generated root filesystem, if not already enabled. For more information, refer [How to build LSDK with Flexbuild](#).

Once the Flexbuild environment has been setup, following commands can be used to build OVS-DPDK package. Generated files (libraries and binaries) would be available in the <Flexbuild>/build/apps/components_arm64 folder. Once the rootfs (root filesystem) is generated, the *components_arm64* folder would be merged in it.

```
$ flex-builder -c ovs-dpdk
```

NOTE
OVS-DPDK is dependent on DPDK package as it is used as its underlying framework. Flexbuild is designed to compile DPDK before OVS-DPDK if not already built.

Layout of OVS-DPDK Binaries

A OVS-DPDK binary image supports both the DPAA and DPAA2 platforms. Once the OVS-DPDK package has been installed, binaries would be available in /usr/local/ folder in the rootfs. Flexbuild system generates a single rootfs for all NXP platforms it supports.

NOTE
OVS-DPDK binaries are deployed into the root filesystem as per the default layout of installation target for OVS-DPDK build system.

Table below depicts various OVS-DPDK artifacts which are available in the Flexbuild generated rootfs:

S/No	File/Image name related to /usr/local/	Description
Table continues on the next page...		

Table continued from the previous page...

1	<pre>./bin/ovs-dpctl ./bin/ovs-vsctl ./bin/ovsdb-client ./sbin/ovsdb-server ./sbin/ovs-vswitchd</pre> And various other binaries installed by OVS package as default.	For both, DPAA and DPAA2, platforms. Various OVS binaries.
2	<pre>./share/man/man7/ovs-*</pre>	Various OVS man-pages.

9.2.3.5 Virtual machine (VM or guest) images

This section describes steps for deploying a Virtual Machine and executing DPDK applications in it. Additionally, OVS-DPDK package is used for deploying a software switch on the host machine through which virtual machines communicate with other virtual machine or external network.

NOTE

For obtaining necessary artifacts (kernel image, rootfs) for booting up a virtual machine on Layerscape board, refer [Configuring and Building](#) on page 554 KVM/Qemu.

9.2.4 Executing DPDK Applications on Host

This section describes how to execute DPDK and related applications in both Host and VM environments.

NOTE

IP_ADDR_BRD, IP_ADDR_IMAGE_SERVER, and TFTP_BASE_DIR are not U-Boot or Linux environment variables. They are used in this document to represent:

1. **IP_ADDR_BRD**: IP address of target board in test setup.
2. **IP_ADDR_IMAGE_SERVER**: IP address of the machine where all the software images are kept. These images are transferred to the board using either tftp or scp.
3. **TFTP_BASE_DIR**: TFTP base directory of TFTP server running on the machine where images are kept.

9.2.4.1 Prerequisites for running DPDK Applications

This section describes the procedures once the target platform is booted up and logged into the Linux shell. This section is applicable to DPAA, DPAA2 and PPFE platforms and is organized as follows:

- Generic Setup contain common steps to be executed before executing any of DPDK sample application or external DPDK applications. One of these sections would be relevant depending on the platform DPAA, DPAA2 or PPFE being used.
- Application specific sections contain steps on how to execute the DPDK example and related applications.

For more details refer the following topics:

- [Test Environment Setup](#) on page 492
- [Generic Setup - DPAA](#) on page 493
- [Generic Setup - DPAA2](#) on page 495
- [Generic Setup - PPFE](#) on page 496

9.2.4.2 Booting up the Target board

Follow the instructions mentioned in [LSDK Quick Start](#) on page 32 to get the target board up and working.

NOTE

While bringing up a DPAA2 platform specific board (LS2088A or LS1088A), use the following boot arguments to obtain best performance. This can be done by appending the following string to the `bootargs` environment variable in `uboot`.

```
default_hugepagesz=1024m hugepagesz=1024m hugepages=8 isolcpus=1-7
```

Above setting insures that at least 8GB of hugepages are available with the application. `isolcpus` insures that Linux Kernel doesn't use these CPUs for scheduling its tasks - that prevents context switching of any application running on these cores. If the installed memory is lesser, lower number of hugepages can be used.

NOTE

DPAA1 platorms (LS1043ARDB, LS1046A) may have lower memory e.g (2GB) and lower number of cores. In that case, following string can be appended to `bootargs`:

```
default_hugepagesz=2MB hugepagesz=2MB hugepages=448 isolcpus=1-3  
bportals=s0 qportals=s0
```

Above setting insures that at least 448 hugepages are available with the application. `isolcpus` insures that Linux Kernel doesn't use these CPUs for scheduling its tasks - that prevents context switching of any application running on these cores. If the installed memory is lower, you may use lower number of hugepages. The `bportals` and `qportals` ensures that only 1 portal is available for kernel use (since only one core is for kernel), rest are available for user space.

NOTE

PPFE platform (LS1012ARDB) have lower memory e.g (1GB) and single core. In that case, following string can be appended to `bootargs`:

```
default_hugepagesz=2MB hugepagesz=2MB hugepages=256
```

NOTE

For UEFI, to update the boot arguments please refer to UEFI section in the user manual.

Update `grub.cfg` file for hugepage and `isolcpus` related changes.

On DPAA2 platforms: "rootwait=20 default_hugepagesz=1024m hugepagesz=1024m hugepages=8 isolcpus=1-7"

On DPAA1 platforms: "rootwait=20 default_hugepagesz=2MB hugepagesz=2MB hugepages=448 isolcpus=1-3 bportals=s0 qportals=s0"

NOTE

For the DPAA platform, DPDK specific Device Tree file (for example, `fsl-ls1046a-rdb-usdpaa.dtb` for LS1046A and `fsl-ls1043a-rdb-usdpaa.dtb` for LS1043A) should be used for booting up the board. This Device tree file is configured to provide userspace applications with network interfaces.

Also note that once the above mentioned Device Tree configuration is used, all FMAN ports would be available in the userspace only. Changes to the Device Tree file would be required to assign some of the FMAN ports to Linux Kernel.

One can use the following method to replace default `fsl-ls104xa-rdb-sdk.dtb` with `fsl-ls104xa-rdb-usdpaa.dtb` to support DPDK on LS104XRDB platforms.

Example 1: After entering Ubuntu on the board, run following instructions for LS1046ARDB:

```
cd /boot
mv fsl-ls1046a-rdb-sdk.dtb fsl-ls1046a-rdb-ori.dtb
ln -s fsl-ls1046a-rdb-usdpaa.dtb fsl-ls1046a-rdb-sdk.dtb
```

then reboot the board

As an alternative, the following method can be used.

Example 2: On the host computer, run the following instructions for LS1046ARDB:

```
cd flexbuild
source setup.env
sed -i 's/fsl-ls1046a-rdb-sdk.dtb/fsl-ls1046a-rdb-usdpaa.dtb/g'
configs/board/ls1046ardb/manifest
flex-builder -i mkdistroscr -a arm64
```

The new auto boot script will be in `build/firmware/u-boot/ls1046ardb/ls1046ardb_boot.scr`.

Then replace the old non-DPAA boot script in SD cards' boot partition with the one you just generated.

NOTE

Optionally follow the below instructions to assign one of the FMAN ports on LS104x (DPAA) RDB boards to Linux.

With standard flexbuild generated dtb all interfaces will be assigned to either Linux or Userspace. When using `fsl-ls1043a-rdb-sdk.dtb` or `fsl-ls1046a-rdb-sdk.dtb` all network interfaces will be assigned to Linux. When using `fsl-ls1046a-rdb-usdpaa.dtb` or `fsl-ls1046a-rdb-usdpaa.dtb` all network interfaces will be assigned to user space. The example below shows the changes that are required to assign one network interface to Linux and configure FMAN to support DPDK applications.

Example: Modify `fsl-ls1046a-rdb-usdpaa.dts` file to assign FMAN ports to Linux by removing the following ethernet node that corresponds to FM0-MAC3 (RGMII-1).

```
ethernet@2 {  
    compatible = "fsl,dpa-ethernet-init";  
    fsl,bman-buffer-pools = <0x7 &0x8 &0x9>;  
    fsl,qman-frame-queues-rx = <0x54 1 0x55 1>;  
    fsl,qman-frame-queues-tx = <0x74 1 0x75 1>;  
};
```

Then modify the file `usdpaa_config_ls1046.xml` (located in `/usr/local/dpdk/dpaa`) by removing the corresponding port entry. For example the below entry needs to be removed for FM0-MAC3(RGMII-1):

```
<port type="MAC" number="3"  
policy="hash_ipsec_src_dst_spi_policy_mac3"/>
```

9.2.4.3 Prerequisites for running DPDK Applications

This section describes the procedures once the target platform is booted up and logged into the Linux shell. This section is applicable to DPAA, DPAA2 and PPFE platforms and is organized as follows:

- [Generic Setup - DPAA](#) on page 493 , [Generic Setup - DPAA2](#) on page 495 and [Generic Setup - PPFE](#) contain common steps to be executed before executing any of DPDK sample application or external DPDK applications. One of these sections would be relevant depending on the platform, DPAA, DPAA2 or PPFE, being used.
- Application specific sections contain steps on how to execute the DPDK example and related applications.

9.2.4.3.1 Test Environment Setup

Test Environment Setup

Various sample application execution steps are detailed in the following sections. Figure below describes the setup containing the DUT (Device Under Test) and the Packet Generator (Spirent, Ixia or any other software/hardware packet generator). This is applicable for the commands provided in following section.

The setup includes a one-to-one link between DUT and Packet generator unit. DPDK application running on the DUT is expected to forward the traffic from one port to another. The setup below and commands described in following sections can be scaled for more number of ports.

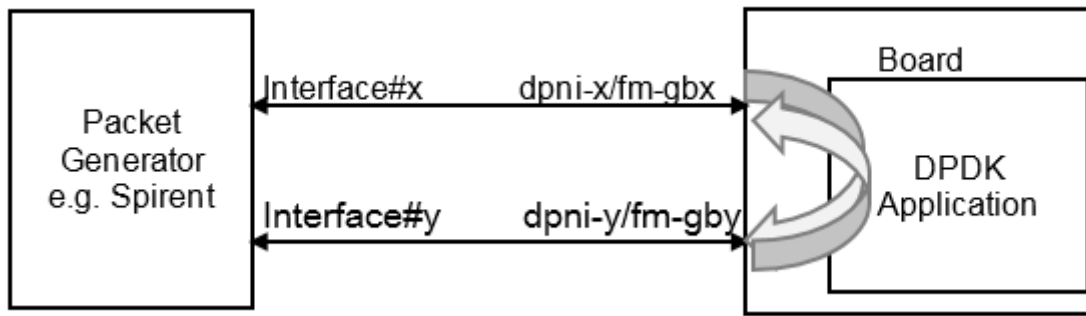


Figure 61. Test Setup

9.2.4.3.2 Generic Setup - DPAA

This section details steps required to setup necessary environment for execution of DPDK applications on DPAA platform. This section is applicable for sample as well as any external DPDK applications. For further details about the applicable configuration file for DPAA platform, refer to [Build OVS-DPDK using Flexbuild](#) on page 488. For DPAA2 platform specific setup, refer to [Generic Setup - DPAA2](#) on page 495.

DPAA Hardware Configuration files

NOTE

FMAN based configuration for queues is not a default configuration for DPAA driver. Default configuration is non-FMC mode where number of queues per interface are automatically configured based on the application request using DPDK APIs. But, *FMAN* based configuration is recommended configuration for custom use cases of flows and fine-tuning of flows. Following section details configuration using *FMC* using the shipped configuration files.

DPAA platforms supports hardware acceleration of packet queues. These queues need to be configured in the *FMAN* (*Frame Manager*) prior to being used. This can be done by choosing the appropriate policy configuration file packaged along with Flexbuild rootfs or DPDK source code.

Either of 1, 2, or 4 queue based policy files can be selected before application is executed. For example, 1 queue policy file would define single queue per physical interface of DPAA. Similarly, 2 and 4 queue are for defining 2 or 4 queues for each defined interface, respectively.

NOTE

Once selected, it is not possible to change the configuration for number of max queues, without rebooting the board. Configuration file should be selected based on requirement and this constraint.

Following are the available platform specific configuration files:

- `usdpaa_config_ls1043.xml` for LS1043A board
- `usdpaa_config_ls1046.xml` for LS1046A board

Following are the available policy files:

- `usdpaa_policy_hash_ipv4_1queue.xml` for 1 queue per port
- `usdpaa_policy_hash_ipv4_2queue.xml` for 2 queues per port
- `usdpaa_policy_hash_ipv4_4queue.xml` for 4 queues per port

NOTE

It is important to execute the applications using the same queue configuration as per the policy file used. This is because once the queue configuration is done, DPAA hardware would distribute packets across configured number of queues. Not consuming packets from any queue would lead to queue buildup eventually stopping the I/O.

Setting up the DPAA Environment

Configure number of queues using environment variable:

```
export DPAA_NUM_RX_QUEUES=<Number of queues>
```

Based on the number of queues defined in the above parameter, select the policy configuration file and execute the `fmc` binary:

```
fmc -c <Configuration file> -p <Policy File> -a
```

For example, in case of LS1043 platform, using 1 queue, following would be the command to execute:

```
export DPAA_NUM_RX_QUEUES=1  
fmc -c ./usdpaa_config_ls1043.xml -p ./usdpaa_policy_hash_ipv4_1queue.xml -a
```

NOTE

It is important that value of `DPAA_NUM_RX_QUEUES` matches to the policy file being used. In case of mismatch, DPDK application may show unexpected behavior.

NOTE

LSDK 18.03 (or dpdk release 18.02) onwards DPAA platforms enables the push mode by default. That is, first 4 queues of an interface would be configured in Push mode, thereafter, all queues would use the default pull configuration. Push mode queues support higher performance configuration than standard pull mode queues, but are limited in numbers. To toggle the number of push mode queues, use the following environment variable:

```
#export DPAA_PUSH_QUEUES_NUMBER=0 <default value is 4>
```

Do note that configuring larger number of push mode queues than available (achievable), would lead to I/O failure.

Setup hugepages for DPDK application to use for packet and general buffers. This step can be ignored if hugepages are already mounted. Use command `mount | grep hugetlbfs` to check if hugepages are already setup.

```
mkdir /mnt/hugepages
```

```
mount -t hugetlbfs none /mnt/hugepages
```

Hereafter, DPDK sample applications are ready to be executed on the DPAA platform.

Cleanup of the DPAA Environment

To remove the configuration done using the `fmc` tool, use the `-x` parameter. It is a good practice to cleanup the configuration before setting up a new configuration. Even in cases where change of configuration is required, for example, increasing the number of queues supported, following command can be used for cleaning up the previous configuration.

```
fmc -x
```

9.2.4.3.3 Generic Setup - DPAA2

This section details steps required to setup necessary environment for execution of DPDK applications over DPAA2 platform. This section is applicable for sample as well as any external DPDK applications. For further details about the applicable configuration file for DPAA2 platform, refer to [Build OVS-DPDK using Flexbuild](#) on page 488. For DPAA platform specific setup, refer to [Generic Setup - DPAA](#) on page 493.

These steps must be performed before running any of the DPDK application on host.

Setting up the DPAA2 Environment

For executing DPDK application on DPAA2 platform, a resource container needs to be created which contains all necessary interfaces to the DPAA2 hardware blocks. Necessary configuration scripts are provided with DPDK package for creating and destroying containers.

1. Configure the DPAA2 resource container with `dynamic_dpl.sh` script. This script is available under `/usr/local/dpdk/dpaa2/dpdk-extras` folder in the rootfs.

```
cd /usr/local/dpdk/dpaa2/      # Or, any other folder if custom installation of DPDK is done
./dynamic_dpl.sh <DPMAC1.id> <DPMAC2.id> ... <DPMACn.id>
```

In the above command, `<DPMAC1.id>` refers to the DPAA2 MAC resource, for example, `dpmac.1` or `dpmac.2`. Modify the above command as per the number of physical MAC ports required by the application (constrained by availability and connectivity on the DUT).

Output of `dynamic_dpl.sh` command shows the name of the container created. This name is passed to DPDK applications using the `DPRC` environment variable. Following block shows sample output of the `dynamic_dpl.sh` command:

```
##### Container dprc.2 is created #####
Container dprc.2 have following resources :=>

* 16 DPBP
* 5 DPCON
* 4 DPSECI
* 3 DPNI
* 10 DPIO
* 10 DPCI

##### Configured Interfaces #####

Interface Name      Endpoint      Mac Address
=====
dpni.1              dpmac.1      -Dynamic-
dpni.2              dpmac.2      -Dynamic-
```

The MAC addresses are auto-assigned by the DPDK applications after fetching information from the firmware. These would be same as the one programmed by u-boot. For creating flows, see the application output or note the MAC addresses during board bootup. Testpmd application can also be used to find the MAC address assigned.

NOTE

It is possible to modify the number of interfaces (DPBP, DPCON, DPNI, etc) in a container. This can be done by defining environment variable `COMPONENT_COUNT=<number>` before executing the script. For example, to set number of DPBP to 4, use `export DPBP_COUNT=4`.

Though the flexibility has been provided to modify the interfaces in the container, note that resources need to be balanced and changing any count will require corresponding changes to other interfaces. Incorrect changes can render the DPDK application unable to execute.

2. Setup the environment variable using the container name reported by `dynamic_dpl.sh` command:

```
export DPRC=dprc.2
```

Once the above setup is complete, DPDK application can be executed on the DPAA2 platform.

Teardown of DPAA2 Environment

It might be required to change the configuration of the resource contain to modify the components included in it. As the number of resources in the system are limited, number of containers which can be created as also limited. It is possible to remove an existing container and create another.

Execute the following command to teardown a container:

```
cd /usr/local/dpdk/dpaa2                # Or, any other folder if custom installation of DPDK is
done
./destroy_dynamic_dpl.sh <Container Name> # for example, "dprc.2"
```

9.2.4.3.4 Generic Setup - PPFE

This section provides steps required to setup necessary environment for execution of DPDK applications over PPFE platform.

These steps must be performed before running any of the DPDK application on host.

Setting up the PPFE Environment

For executing DPDK application on PPFE platform, a kernel module `pfe.ko` must be loaded in user space mode which will do the necessary initialization to run the DPDK applications. By default, `pfe.ko` will be loaded automatically during kernel bootup. User must ensure the value of `/sys/module/pfe/parameters/us` is 1 to check `pfe.ko` module is loaded in user space mode. If `/sys/module/pfe/parameters/us` is not 1, then user shall unload the module and then load again with module argument as `us=1`.

```
rmmod pfe.ko
insmod pfe.ko us=1
```

Additionally, user must run the below commands to fulfill DPDK applications huge pages requirements.

```
mkdir /mnt/hugepages
mount -t hugetlbfs none /mnt/hugepages
```

9.2.4.3.5 DPAA2: Multiple parallel DPDK Applications

This section describes steps for executing multiple parallel DPDK application on DPAA2 platform.

For executing multiple DPDK applications, each application instance should run with its own resource container (*DPRC*). This constraint is because of the way DPDK framework is designed to use a given container for exclusive use, irrespective of resources within, and bind it using VFIO layer. This design prevents parallel access to single resource container from multiple parallel instances of a single DPDK application, multiple parallel execution of different DPDK applications.

Creating Multiple DPRC instances

Using the resource container script documented in [this section](#), create multiple resource container instances on host. Following command creates a resource container with 2 network interfaces (and all other resources necessary to run a DPDK application).

First DPRC: (assuming name as `dprc.2` through rest of the document)

```
cd /usr/local/dpdk/dpaa2                # Or, any other folder if custom installation of DPDK is
done
./dynamic_dpl.sh <DPMAC1.id> <DPMAC2.id> # For example, execute ./dynamic_dpl.sh dpmac.1 dpmac.2
```

Second DPRC: (assuming name as `dprc.3` through rest of the document)

```
cd /usr/local/dpdk/dpaa2          # Or, any other folder if custom installation of DPDK is
done
./dynamic_dpl.sh <DPMAC3.id> <DPMAC4.id> # For example, execute ./dynamic_dpl.sh dpmac.3 dpmac.4
```

Executing multiple DPDK Applications

Once the resource containers are created, on two separate terminals, execute the following commands to run **l2fwd** application, bridging traffic between both interfaces available in the container:

```
export DPRC=dprc.2
cd /usr/local/bin
./l2fwd -c 0x3 -n 1 --file-prefix=p1 --socket-mem=1024 -- -p 0x3 -q 1
```

Some of the arguments, which are deviations from general **l2fwd** command, are explained below:

--file-prefix: Each DPDK Application attempts to allocate some hugepages for DMA'd area. This allocation is done in the hugepages through the use of *hugepage* mount, by creating and mapping a file. This arguments instructs the EAL to append a string to the file name. This way, multiple instances, having different such arguments, wouldn't attempt to open same hugepage mapping file.

--socket-mem: Passed to EAL, this instructs the EAL to allocate only specified amount of memory from the hugepages. By default, if this is not provided, a DPDK application would acquire all possible hugepages (all free pages) available on the Linux system.

For the second instance, command like following can be executed:

```
export DPRC=dprc.3
cd /usr/local/bin
./l2fwd -c 0xc -n 1 --file-prefix=p2 --socket-mem=1024 -- -p 0x3 -q 1
```

Note the difference of values for `-c` and `--file-prefix` between the first and second command.

9.2.4.4 DPDK example applications

DPDK example application binaries are available in the `/usr/local/bin` folder in the Flexbuild generated rootfs.

NOTE

Command snippets below assume that commands are executed while being present in `/usr/local/bin` or appropriate `PATH` variable has been set. Also, a DPDK binary can be executed on both, DPAA and DPAA2, platform without any modifications.

NOTE

Only a selected few DPDK example applications have been deployed in the root filesystem by default. For non-deployed example application, compilation needs to be done using DPDK source code. Refer [Standalone build of DPDK Libraries and Applications](#) on page 483 for more details.

NOTE

For PPFE platform, since LS1012ARDB has only 1 core, so `-c` with 0x1 is only acceptable core mask for all DPDK applications. Additionally, user must provide the `--vdev` argument with value `eth_pfe` to enable ethernet device for DPDK applications.

L2fwd – Layer 2 Forwarding Application

Sample application to show forwarding between multiple ports based on the Layer 2 information (switching).

```
l2fwd -c 0x2 -n 1 -- -p 0x1 -q 1 -T 0
```

In the above command: `-c` refers to the core mask for cores to be assigned to DPDK; `-p` is the port mask for ports to be used by application; `-q` defines the number of queues to serve on each port. Other command line parameters may also be provided - for a complete list, refer [L2 Forwarding Sample Application \(in Real and Virtualized Environments\)](#).

NOTE

- `isolcpus` provided as boot argument to u-boot assures that isolated cores are not scheduled by Linux kernel. Using Core 0 for DPDK application can lead to non-deterministic behavior, including drop in performance. It is recommended that DPDK application core mask values avoid using Core 0.
- L2fwd application periodically prints the I/O stats. To avoid CPU core to be interrupted because of these scheduled prints, `-t 0` option can be appended at the end of command line.
- Command to run `l2fwd` on LS1012ARDB:

```
./l2fwd -c 0x1 -n 1 --vdev 'eth_pfe0' --vdev='eth_pfe1' -- -p 0x3 -q 3
```

NOTE

For best performance on LS1046ARDB, use the following command. This includes an option `-b 7` which sets optimal I/O burst size:

```
l2fwd -c 0x2 -n 1 -- -p 0x1 -q 1 -T 0 -b 7
```

L2fwd-qdma - Layer 2 Forwarding using QDMA

Sample application to show forwarding between multiple ports based on the Layer 2 information (switching) using QDMA.

```
l2fwd-qdma -c 0x2 -n 1 -- -p 0x1 -q 1 -m 1 -T 0
```

In the above command: `-c` refers to the core mask for cores to be assigned to DPDK; `-p` is the port mask for ports to be used by application; `-q` defines the number of queues to serve on each port. `-m` mode specifies HW (`-m` is 0) or Virtual (`-m` is 1) mode for QDMA queues. Apart from `-m` parameter other parameters are similar to DPDK `l2fwd` application -refer [L2 Forwarding Sample Application \(in Real and Virtualized Environments\)](#).

L3fwd – Layer 3 Forwarding Application

Sample application to show forwarding between multiple ports based on the Layer 3 information (routing).

```
l3fwd -c 0x6 -n 1 -- -p 0x3 --config="(0,0,1),(1,0,2)"
```

In the above command: `-c` refers to the core mask for cores to be assigned to DPDK; `-p` is the port mask for ports to be used by application; `--config` is (*Port, Queue, Core*) configuration used by application for attaching cores to queues on each port. Other command line parameters may also be provided - for a complete list, refer [L3 Forwarding Sample Application](#).

Other variations of the above command described below change the configuration of ports, queue and cores services them.

1. 4 core - 2 Port, 2 queues per port:

```
l3fwd -c 0xF -n 1 -- -p 0x3 -P --config="(0,0,0),(0,1,1),(1,0,2),(1,1,3)"
```

2. 4 core - 2 Port with destination MAC address:

```
l3fwd -c 0xF -n 1 -- -p 0x3 -P --config="(0,0,0),(0,1,1),(1,0,2),(1,1,3)" --eth-dest=0,11:11:11:11:11:11 --eth-dest=1,00:00:00:11:11:11
```

3. 8 core - 2 Port with 4 queues per port:

```
l3fwd -c 0xFF -n 1 -- -p 0x3 -P --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),(1,0,4),(1,1,5),(1,2,6),(1,3,7)"
```

NOTE

Although, above command snippets use the Core 0 for DPDK application, for best performance Core 0 use is not recommended as Linux OS schedules its tasks on it. It is also recommended that `isolcpus` be used in Linux boot argument to prevent Linux from scheduling tasks on other Cores.

NOTE

Example command to run l3fwd on LS1012ARDB:

```
./l3fwd -c 0x1 --vdev='eth_pfe0' --vdev='eth_pfe1' -n 1 -- -p 0x3 --config="(0,0,0),(1,0,0)" -P
```

NOTE

For best performance on LS1046ARDB, use the following command. This includes an option `-b 7` which sets optimal I/O burst size:

```
l3fwd -c 0xF -n 1 -- -p 0x3 -P -b 7 --config="(0,0,0),(0,1,1),(1,0,2),(1,1,3)"
```

This is valid for any configuration of cores, queues and ports (i.e., `--config` option).

NOTE

For LX2 Platform, while running on all available cores, the core mask parameters passed to `l3fwd` needs to be adjusted for 16 available cores. Following is an example of using all 16 cores on LX2, 2 Ports, 8 queues per port:

```
l3fwd -c 0xffff -n 1 -- -p 0x3 -P --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),(0,4,4),(0,5,5),(0,6,6),(0,7,7),(1,0,8),(1,1,9),(1,2,10),(1,3,11),(1,4,12),(1,5,13),(1,6,14),(1,7,15)"
```

L2fwd-Crypto – Layer 2 Forwarding using DPAA/DPAA2 CAAM Hardware

This variation of Layer 2 forwarding application uses DPAA/DPAA2 CAAM block for encryption of packets.

- Layer 2 forwarding with Cipher only support:

```
l2fwd-crypto -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_ONLY --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10
```

- Layer 2 forwarding with Cypher-Hash support:

```
l2fwd-crypto -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_HASH --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --auth_algo sha1-hmac --auth_op GENERATE --auth_key_random_size 64
```

- Layer 2 forwarding with Hash only support:

```
l2fwd-crypto -c 0x2 -n 1 -- -p 0x1 -q 1 --chain HASH_ONLY --auth_algo sha1-hmac --auth_op
GENERATE --auth_key_random_size 64
```

L2fwd-Crypto – Layer 2 Forwarding using OpenSSL Software Instructions

This variation of Layer 2 forwarding application uses OpenSSL library for performing software crypto operations. Internally, the OpenSSL library would use the ARMCE instructions specific for ARM CPUs. For DPDK, this application uses the OpenSSL PMD as its underlying driver.

NOTE

This command requires support of OpenSSL package while building the DPDK applications. Refer [this section](#) of this document, for details about toggling compilation of software crypto support, which includes the OpenSSL driver.

- Cipher_only
 - For DPAA Platform
 - 1 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_ONLY --
cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:
0e:0f:10 --cryptodev_mask 0x10
```

- 2 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl0" --vdev "crypto_openssl1" -c 0x6 -n 1 -- -p 0x3 -q 1 --
chain CIPHER_ONLY --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key
01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --cryptodev_mask 0x30
```

- 4 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl0" --vdev "crypto_openssl1" --vdev "crypto_openssl2" --
vdev "crypto_openssl3" -c 0xf -n 1 -- -p 0xf -q 1 --chain CIPHER_ONLY --cipher_algo aes-cbc
--cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --
cryptodev_mask 0xf0
```

- For DPAA2 Platform
 - 1 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_ONLY --
cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:
0e:0f:10 --cryptodev_mask 0x100
```

- 2 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl0" --vdev "crypto_openssl1" -c 0x6 -n 1 -- -p 0x3 -q 1 --
chain CIPHER_ONLY --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key
01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --cryptodev_mask 0x300
```


- 4 core: Depending on the platform being executed on, append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl0" --vdev "crypto_openssl1" --vdev "crypto_openssl2" --
vdev "crypto_openssl3" -c 0xf -n 1 -- -p 0xf -q 1 --chain CIPHER_ONLY --cipher_algo aes-cbc
--cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --
cryptodev_mask 0xF00
```

- Cipher_hash

- For DPAA Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_HASH --
cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:
0e:0f:10 --auth_algo sha1-hmac --auth_op GENERATE --cryptodev_mask 0x10 --
auth_key_random_size 64
```

- For DPAA2 Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain CIPHER_HASH --
cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key 01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:
0e:0f:10 --auth_algo sha1-hmac --auth_op GENERATE --cryptodev_mask 0x100 --
auth_key_random_size 64
```

In the above ccommands, for scaling to multiple cores or ports, toggle the -c and -p arguments as described above.

- Hash_cipher

- For DPAA Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain HASH_CIPHER --
auth_algo sha1-hmac --auth_op GENERATE --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key
01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --cryptodev_mask 0x10 --auth_key_random_size
64
```

- For DPAA2 Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain HASH_CIPHER --
auth_algo sha1-hmac --auth_op GENERATE --cipher_algo aes-cbc --cipher_op ENCRYPT --cipher_key
01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10 --cryptodev_mask 0x100 --auth_key_random_size
64
```

In the above commands, for scaling to multiple cores or ports, toggle the -c and -p arguments.

- Hash_only

- For DPAA Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain HASH_ONLY --auth_algo
sha1-hmac --auth_op GENERATE --cryptodev_mask 0x10 --auth_key_random_size 64
```

- For DPAA2 Platform:

- 1 core: Append the above blacklisting parameter to the end of this command:

```
l2fwd-crypto --vdev "crypto_openssl" -c 0x2 -n 1 -- -p 0x1 -q 1 --chain HASH_ONLY --auth_algo
sha1-hmac --auth_op GENERATE --cryptodev_mask 0x100 --auth_key_random_size 64
```

- For scaling to multiple cores or ports, toggle the `-c` and `-p` arguments as described above.

For more information on L2fwd-crypto application, refer to [L2 Forwarding with Crypto Sample Application](#).

NOTE

Example command to run l2fwd-crypto with openssl on LS1012ARDB (cipher only):

```
./l2fwd-crypto -c 0x1 --vdev='eth_pfe0' --vdev='crypto_openssl' -n 1 -- -p 0x1 -q 1 --chain
CIPHER_ONLY --cipher_algo aes-cbc --cipher_key 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f --
cipher_op ENCRYPT
```

IPSec-secgw – IPSec Gateway using DPAA/DPAA2 CAAM Hardware

For IPSec application, two DUTs need to be configured as endpoint 0 (ep0) and endpoint 1 (ep1). Assuming that endpoint have **4 ports** each:

- Connect Port 1 and Port 3 of the ep0 and ep1 to each other (back-to-back).
- Connect Port 0 and Port 2 of the ep0 and ep1 to packet generator (for example, Spirent).

The Stream generated by packet generator needs to have IP addresses in following pattern:

```
EP0:
port 0: 32 flows with destination IP: 192.168.1.XXX, 192.168.2.XXX, ..... ,192.168.31.XXX,
192.168.32.XXX
port 2: 32 flows with destination IP: 192.168.33.XXX, 192.168.34.XXX, ..... ,192.168.63.XXX,
192.168.64.XXX
EP1:
port 0: 32 flows with destination IP: 192.168.101.XXX, 192.168.102.XXX, ..... ,192.168.131.XXX,
192.168.132.XXX
port 2: 32 flows with destination IP: 192.168.133.XXX, 192.168.134.XXX, ..... ,192.168.163.XXX,
192.168.164.XXX
```

Above represents default configurations for the endpoints in `ep0_64X64.cfg` and `ep1_64X64.cfg`. Custom port mappings, SA/SP and the routes can be configured in the corresponding configuration file named as `ep0.cfg` and `ep1.cfg` for respective endpoint. These files are available in Flexbuild generated rootfs; for further details, refer [this table](#).

For more information, refer to [IPsec Security Gateway Sample Application](#).

Endpoint 0 (ep0) configuration:

```
ipsec-secgw -c 0xf -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" -f ep0_64X64.cfg
```

Endpoint 1 (ep1) configuration:

```
ipsec-secgw -c 0xf -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" -f ep1_64X64.cfg
```

Running IPSec Gateway application with protocol offload

The DPAA/DPAA2 CAAM hardware also support IPSec protocol offload. The command and configurations are exactly same except the cfg files. For protocol offload, the cfg files are `ep0_64X64_proto.cfg` and `ep1_64X64_proto.cfg`. Performance with protocol offload would be much better than the standard case. *In case of platforms which have 8 cores, the command for 8 core will also be exactly same as non-offload case, except the name of the cfg files.*

Endpoint 0 (ep0) configuration:

```
ipsec-secgw -c 0xf -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" -f
ep0_64X64_proto.cfg
```

Endpoint 1 (ep1) configuration:

```
ipsec-secgw -c 0xf -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" -f
ep1_64X64_proto.cfg
```

Running IPsec Gateway Application with 8 Cores

For running IPsec application with multiple queues using 64X64 tunnels and with 8 cores, following command and configuration needs to be done:

Endpoint 0 (ep0) configuration: Sample configuration for this is available in ep0_64X64.cfg file available in /usr/local/dpdk/dpaa2/ folder in root filesystem.

```
ipsec-secgw -c 0xFF -- -p 0xf -P -u 0xa --config="(0,0,0),(0,1,1),(1,0,2),(1,1,3),(2,0,4),(2,1,5),
(3,0,6),(3,1,7)" -f ep0_64X64.cfg
```

Endpoint 1 (ep1) configuration: Sample configuration for this is available in ep1_64X64.cfg file available in /usr/local/dpdk/dpaa2/ folder in root filesystem.

```
ipsec-secgw -c 0xFF -- -p 0xf -P -u 0xa --config="(0,0,0),(0,1,1),(1,0,2),(1,1,3),(2,0,4),(2,1,5),
(3,0,6),(3,1,7)" -f ep1_64X64.cfg
```

Running IPsec Gateway Application with 16 Cores on LX2 Platform

For running IPsec application with multiple queues using 64X64 tunnels and with 16 cores, following command and configuration needs to be done:

Endpoint 0 (ep0) configuration: Sample configuration for this is available in ep0_64X64_sha256_proto.cfg file available in /usr/local/dpdk/dpaa2/ folder in root filesystem.

```
./ipsec-secgw -c 0xFFFF -- -p 0xf -P -u 0xa --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),(1,0,4),
(1,1,5),(1,2,6),(1,3,7),(2,0,8),(2,1,9),(2,2,10),(2,3,11),(3,0,12),(3,1,13),(3,2,14),(3,3,15)" -f
ep0_64X64_sha256_proto.cfg
```

Endpoint 1 (ep1) configuration: Sample configuration for this is available in ep1_64X64_sha256_proto.cfg file available in /usr/local/dpdk/dpaa2/ folder in root filesystem.

```
./ipsec-secgw -c 0xFFFF -- -p 0xf -P -u 0xa --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),(1,0,4),
(1,1,5),(1,2,6),(1,3,7),(2,0,8),(2,1,9),(2,2,10),(2,3,11),(3,0,12),(3,1,13),(3,2,14),(3,3,15)" -f
ep1_64X64_sha256_proto.cfg
```

IPSec-secgw – IPsec Gateway using OpenSSL PMD

The command, flow stream and port configuration is similar to the [IPSec-secgw – IPsec Gateway using DPAA/DPAA2 CAAM Hardware](#) on page 502 command, flow stream and port configuration, except that it uses OpenSSL PMD for crypto operations. Internally, the OpenSSL PMD uses the ARMCE instructions for the ARM CPUs for performing crypto operations.

- For DPAA Platform:
 - Endpoint 0 configuration

```
ipsec-secgw -c 0xf --vdev "crypto_openssl" -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),
(3,0,3)" --cryptodev_mask 0x10 -f ep0_64X64.cfg
```

Endpoint 1 configuration

```
ipsec-secgw -c 0xf --vdev "crypto_openssl" -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" --cryptodev_mask 0x10 -f ep1_64X64.cfg
```

- For DPAA2 Platform:

- Endpoint 0 configuration

```
ipsec-secgw -c 0xf --vdev "crypto_openssl" -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" --cryptodev_mask 0x100 -f ep0_64X64.cfg
```

Endpoint 1 configuration

```
ipsec-secgw -c 0xf --vdev "crypto_openssl" -- -p 0xf -P -u 0xa --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3)" --cryptodev_mask 0x100 -f ep1_64X64.cfg
```

NOTE

Example command to run ipsec-secgw with openssl on LS1012ARDB:

```
./ipsec-secgw -c 0x1 --vdev='eth_pfe0' --vdev='eth_pfe1' --  
vdev='crypto_openssl' -- -p 0x3 -P -u 0x2 --config="(0,0,0),(1,0,0)"  
-f ep0_64X64.cfg
```

KNI - Using Kernel Network Interface Module

The Kernel NIC Interface (KNI) is a DPDK control plane solution that allows userspace applications to exchange packets with the kernel networking stack. For details please refer: http://dpdk.org/doc/guides/sample_app_ug/kernel_nic_interface.html

Loading the KNI kernel module without any parameter. By default only one kernel thread is created for all KNI devices for packet receiving in kernel side:

```
#insmod rte_kni.ko
```

Affine the kni task to a single core e.g core number #1

```
#taskset -pc 1 `pgrep -fl kni_single | awk '{print $1}'`
```

Run the kni application

```
kni [EAL options] -- -P -p PORTMASK --config="(port,lcore_rx,lcore_tx[,lcore_kthread,...])  
[,port,lcore_rx,lcore_tx[,lcore_kthread,...]]"  
#./kni -c 0xf -n 1 -- -p 0x3 -P --config="(0,0,1),(1,0,1)"  
where config is : (PORT, kni lcore Rx core, kni lcore tx core )
```

On another console check the interfaces with:

```
#ifconfig -a
```

Enable the given interface and assign IP address (if any)

Pktgen – DPDK based Software Packet Generator

Pktgen is a software packet generator based on DPDK. Refer [DPDK based Packet Generator](#) on page 487 for steps required for building *Pktgen*.

All the commands below assume that *Pktgen* application is either executed from current folder or appropriate path environment variable has been set.

1. 3 Port, 1 Core each

```
pktgen -l 0-3 -n 1 --proc-type auto --file-prefix pg --log-level 8 -- -T -P -m "[1].0, [2].1, [3].2"
```

2. 1 Port, 2 Core

```
pktgen -l 0-3 -n 1 --proc-type auto --file-prefix pg --log-level 8 -- -T -P -m "[1:2].0"
```

3. To start or stop traffic on a specific port:

```
start 0 # start <port number>
stop 0 # stop <port number>
```

4. To start or stop traffic on all ports:

```
str
stp
```

9.2.4.5 Command interface (CMDIF) demo application

DPDK based Command interface (CMDIF) demo application demonstrates the communication between GPP and AIOP using DPDK API's and Command Interface library. Command Interface library is provided as a lib module within `examples/cmdif/` (`examples/cmdif/lib/librte_cmdif.a`).

This application requires a corresponding process running on AIOP core/s, which will read and respond to CMDIF application. CMDIF application is only supported on DPAA2 which will have AIOP.

NOTE

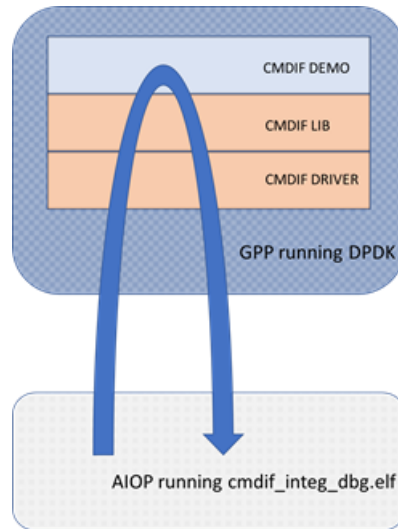
Include the library `librte_cmdif.a`, when you are writing an application over DPAA2 CMDIF based raw device.

The application verifies the following:

1. CMDIF client (where GPP is the client and AIOP is the server)
2. CMDIF server (where GPP is the server and AIOP is the client)

CMDIF Client (GPP is client)

In the CMDIF client, the GPP is the client and the AIOP is the server. Requests are initiated by the GPP and are sent to the AIOP core. The AIOP responds back with the response.

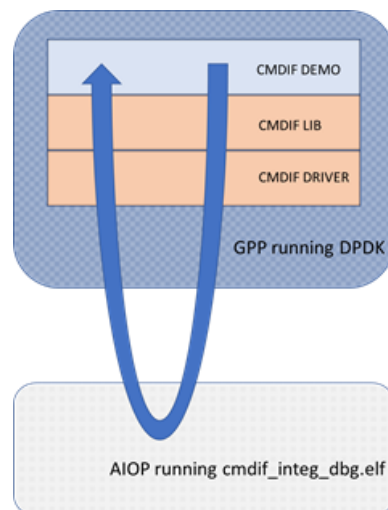


The CMDIF client (demo) is responsible for the following:

- Opens a CI communication channel using a single DPCI device, defined in container used by application
- Sends multiple messages from GPP to AIOP using synchronous commands
- Sends and receive response for multiple messages from GPP to AIOP using asynchronous commands
- Application Validates the response received from the AIOP Server application and prints the result on console
- Closes the opened CI communication channels

CMDIF Server (GPP is server)

In the CMDIF server, the GPP is the server and the AIOP is the client. Requests are initiated by the AIOP and are sent to the GPP core. The GPP responds back to the AIOP with success or error.



The CMDIF server (demo) is responsible for the following:

- Registers the server module
- Opens the Sever session
- Initiates the client open on the AIOP client
- Receives requests/commands from the AIOP
- Closes the server session

- Unregisters the module

Running demo application

The demo application showcases only a single thread or core use-case, thus supporting the coremask with single core.

NOTE

`dynamic_dpl.sh` is not required to run along with `cmdif_demo`.

Executing demo application example also requires the following:

- Running `dynamic_AIOP_dpl.sh` (*instead of `dynamic_dpl.sh`*)
- Loading the `cmdif_integ_dbg.elf` (provided in AIOPSL - <https://github.com/qorik-open-source/aiopsl/tree/integration/demos/images>) using the `aiop_tool` which needs to run in background.

For example:

```
./dynamic_AIOP_dpl.sh
export DPRC = > dprc container created for GPP<
aiop_tool load -g dprc.3 -f cmdif_integ_dbg.elf &
cmdif_demo -c 0x2
```

Description about the command:

- `dynamic_AIOP_dpl.sh` – creates three containers
 - First one for the AIOP
 - Second one for the `aiop_tool` which loads the AIOP FW
 - Third one for DPDK's use (**Use this container name as `$DPRC` export variable**)
- The `-c` option enables cores 2

Expected output

The application should print below logs on console in case of CMDIF client:

- PASSED open commands
- PASSED synchronous send commands
- PASSED asynchronous send/receive commands
- PASSED: close commands

Also, verify that application prints below logs in console in case of CMDIF server:

- PASSED cmdif session open
- PASSED sync command
- PASSED Async commands
- PASSED Isolation context test
- PASSED cmdif session close

9.2.5 OVS-DPDK and DPDK in VM with VIRTIO Interfaces

DPDK example and DPDK-based applications can also run inside the virtual machine. This section describes steps to run these applications inside the virtual machine on both DPAA and DPAA2 platforms.

The virtual machine runs inside the host Linux system and is launched by an application called QEMU.

NOTE

While using the virtual machine, the console logs for the guest Linux do not appear on the host Linux console (i.e UART). The guest logs are exposed through `telnet`, and they can be accessed by doing `telnet` on the host board's IP Address (`IP_ADDR_BRD`) and `GUEST_CONSOLE_TELNET_PORT`. Each Virtual machine that is run on a single host is allocated a different `GUEST_CONSOLE_TELNET_PORT`, and this port number is specified by user running virtual machine through the QEMU command line.

Following is the layout of the sub-sections of this chapter:

- [Generic steps](#) on page 508 describing steps required for QEMU setup for both, DPAA and DPAA2 platforms.
- [Configuring OVS](#) on page 508 describing steps necessary to launch OVS-DPDK on the host machine for switching traffic between VMs and external network.
- Various sections for launching a virtual machine and executing a DPDK application:
 - [Launch Virtual Machine](#) on page 511 for launching a virtual machine.
 - [Accessing virtual machine console](#) on page 513 for accessing a virtual machine console from a network connected machine over `telnet`.
 - [Launching two virtual machines](#) on page 513 for launching more than one virtual machine.
 - [Running DPDK applications in VM](#) on page 514 for running DPDK applications in the virtual machine.
- [Multi Queue VIRTIO support](#) on page 515 describes steps for DPDK application using multiple queues.

9.2.5.1 Generic steps

Refer to [Configuring and Building](#) on page 554 KVM/Qemu for detailed information about deploying virtual machines using KVM/QEMU using Layerscape boards.

The reference above serves as base for deploying Virtual Machines and DPDK application in them. All following sections assume that kernel image and virtual machine rootfs is available with DPDK sample application images in it.

NOTE

Give IP Address to the board so that virtual machine console can be accessed using `telnet`.

```
ifconfig eth<x> <IP_ADDR_BRD>
```

9.2.5.2 Configuring OVS

OVS-DPDK application binary and configuration files are available in the `/usr/local` folder in the Flexbuild generated rootfs.

It is assumed that before executing command snippets in this section, necessary steps mentioned in [Generic steps](#) on page 508 have already been executed.

NOTE

Command snippets below assume that commands are executed while being present in `/usr/local/` folder. Or, appropriate `PATH` variable has been set. As the OVS commands are spread across multiple folder, each command snippet also shows the location of these binaries relative to above folder.

Command snippets below assume that commands are executed while being present in this folder or appropriate `PATH` variable has been set.

OVS is used as a back-end for VHOST USER ports. The physical ports on the target platform and the vhost user ports (virtio devices) are added to ovs-vswitch and the flows in OVS are programmed so as to establish traffic switching between physical ports and vhost devices as follows:

- Incoming traffic Physical port1 => output to vhost-user port 1

- Incoming traffic on vhost-user port1 => output on physical port 1
- Incoming traffic on physical port 2 => output on vhost-user port 2
- Incoming traffic on vhost-user port 2 => output on physical port 2

The following steps must be followed to setup OVS as vhost switching back-end:

1. Reset the OVS environment.

```
pkill -9 ovs
```

```
rm /usr/local/etc/openvswitch/conf.db
```

```
rm -rf /usr/local/var/run/openvswitch/vhost-user-1
```

```
rm -rf /usr/local/var/run/openvswitch/vhost-user-2
```

2. Specify the initial Open vSwitch (OVS) database to use:

```
mkdir -p /usr/local/etc/openvswitch # If the folder doesn't already exist
```

```
mkdir -p /var/log/openvswitch # to ensure that OVS logging can be done
```

```
mkdir -p /usr/local/var/run/openvswitch
```

```
cd /usr/bin/ovs-dpdk/  
./ovsdb-tool create /usr/local/etc/openvswitch/conf.db ./vswitch.ovsschema
```

```
./ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock --  
remote=db:Open_vSwitch,Open_vSwitch,manager_options --pidfile=/tmp/ovsdb-server.pid --detach --  
log-file=/var/log/openvswitch/ovs-vswitchd.log
```

```
export DB_SOCKET=/usr/local/var/run/openvswitch/db.sock
```

3. Configure the OVS to support DPDK ports:

```
./ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
```

4. Configure OVS to work with 1G Hugepages

```
export SOCK_MEM=1024
```

```
./ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-socket-mem="$SOCK_MEM"
```

5. Define Cores for OVS Operations

```
export OVS_SERVICE_MASK=0x1  
export OVS_CORE_MASK=0x6
```

```
./ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-lcore-mask=$OVS_SERVICE_MASK
```

```
./ovs-vsctl --no-wait set Open_vSwitch . other_config:pmd-cpu-mask=$OVS_CORE_MASK
```

NOTE

OVS_CORE_MASK should be chosen such as to not include Core 0. OVS_SERVICE_MASK should be any core which is not already assigned to OVS_CORE_MASK. This way, OVS services threads (defined by OVS_SERVICE_MASK) will not compete for CPU scheduling with OVS I/O threads (OVS_CORE_MASK). OVS_SERVICE_MASK can be set to Core 0 as defined in example above

6. Start the ovs-vswitchd daemon:

```
./ovs-vswitchd unix:$DB_SOCKET --pidfile --detach -c $OVS_CORE_MASK
```

NOTE

--detach option makes the daemon run in background. If this option is given same shell can be used to run further commands, otherwise ssh to the target board and run further commands. Each time you reboot or there is an OVS termination, you need to rebuild the OVS environment and repeat steps 1-6 of this section

7. Create an OVS bridge.

```
./ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
```

8. Create DPDK port

For creating DPDK ports with OVS, platform specific port information needs to be provided to OVS.

- ```
./ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk options:dpdk-devargs=dpni.1
```

```
./ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk options:dpdk-devargs=dpni.2
```

Above commands attach the DPAA2 ports dpni.1 and dpni.2 with OVS. In case different ports are required, above command should be modified accordingly.

### 9. Create vhost-user port

```
./ovs-vsctl add-port br0 vhost-user1 -- set Interface vhost-user1 type=dpdkvhostuser
```

```
./ovs-vsctl add-port br0 vhost-user2 -- set Interface vhost-user2 type=dpdkvhostuser
```

### 10. Commands to Configure Multi Queues

```
./ovs-vsctl set Interface dpdk0 options:n_rxq=4
./ovs-vsctl set Interface dpdk1 options:n_rxq=4
./ovs-vsctl set Interface dpdk0 options:n_txq=4
./ovs-vsctl set Interface dpdk1 options:n_txq=4
```

#### NOTE

The above commands are required only in case of multi-queue use case(Four queues been used in above reference commands). For single queue mode no commands needed as OVS by default configures single queue.

### 11. Delete OVS flow

```
./ovs-ofctl del-flows br0
```

### 12. Set OVS flow rules for external-to-external path:

#### NOTE

The commands below configure a hard-coded bi-directional data path between Port 1 and Port 2. Use this step only for OVS external-to-external testing. For OVS Host-to-VM configuration, skip and continue with next step.

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=1,actions=output:2
```

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=2,actions=output:1
```

### 13. Set OVS flow rules between Host to VM:

#### NOTE

The steps below configure OVS such that Port 1 <=> Port 3 and Port 2 <=> Port 4 are connected to each other. If a different configuration is required, the commands below should be altered as well as VM configurations.

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=1,actions=output:3
```

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=3,actions=output:1
```

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=2,actions=output:4
```

```
./ovs-ofctl add-flow br0 -O OpenFlow13 table=0,in_port=4,actions=output:2
```

#### NOTE

OVS Switch (ovs-vswitchd) must be run before launching the virtual machine using QEMU, otherwise the virtual machine launch will fail.

14. Run the following command to enable emc-cache lookups in OVS. This helps in enhancing the lookup speed to ensure better performance.

```
./ovs-vsctl --no-wait set Open_vSwitch . other_config:emc-insert-inv-prob=1
```

### 15. Verify the Flows inserted:

```
./ovs-ofctl dump-flows br0
```

## 9.2.5.3 Launch Virtual Machine

This section describes necessary environment setup and commands for launching a Virtual Machine (VM).

It is assumed that before executing command snippets in this section, necessary steps mentioned in [Generic steps](#) on page 508 and [Configuring OVS](#) on page 508 have already been executed.

### Setup the environment

For accessing the VM, *telnet* is used. This environment variable defines the *telnet* port to be used.

```
export GUEST_CONSOLE_TELNET_PORT=4446 # Telnet port to be used for accessing the virtual machine
```

```
export ROOTFS_IMG=<VM_ROOTFS_IMG>
```

Define other environment variables which are used by the QEMU command to configure the virtual machine environment:

```
export VM_MEM=2048M
export VM_CORES=2
export NUM_QUEUES=1
```

#### NOTE

- VM\_CORES are the number of cores to reserve for the virtual machine operation.

Export the following paths:

```
export VHOST1_PATH=/usr/local/var/run/openvswitch/vhost-user1
export VHOST2_PATH=/usr/local/var/run/openvswitch/vhost-user2
```

### Launch QEMU and virtual machine

Launch the QEMU emulator using the following command.

```
qemu-system-aarch64 -nographic -object memory-backend-file,id=mem,size=$VM_MEM,mem-path=/mnt/
hugepages,share=on -cpu host -machine type=virt -kernel /boot/Image -enable-kvm -serial tcp::
$GUEST_CONSOLE_TELNET_PORT,server,telnet -append 'root=/dev/vda rw console=ttyAMA0,115200 rootwait
earlyprintk' -m $VM_MEM -numa node,memdev=mem -chardev socket,id=char1,path=$VHOST1_PATH -netdev
type=vhost-user,id=hostnet1,chardev=char1,vhostforce,queues=$NUM_QUEUES -device virtio-net-
pci,disable-modern=false,addr=0x3,netdev=hostnet1,id=net1,mrg_rxbuf=off -chardev
socket,id=char2,path=$VHOST2_PATH -netdev type=vhost-
user,id=hostnet2,chardev=char2,vhostforce,queues=$NUM_QUEUES -device virtio-net-pci,disable-
modern=false,addr=0x4,netdev=hostnet2,id=net2,mrg_rxbuf=off -smp $VM_CORES -S -drive if=none,file=
$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo
```

#### NOTE

For best performance, Core 0 in the VM should not be used for DPDK I/O threads.

Also, to avoid system services from using GPUs scheduled for DPDK I/O threads, it is recommended that `isolcpus` be used for isolating cores from Linux Kernel scheduling in VM. The exact configuration is dependent on number of CPU assigned by QEMU to VM using the `VM_CORES` environment variable.

Append `isolcpus=1-$VM_CORES` to the `'root=/dev/vda rw console=ttyAMA0,115200 rootwait earlyprintk'` string in the `qemu-system-aarch64` command given above.

#### NOTE

Extra care should be taken for value assigned to `mem-path` variable. It should point to a valid mounted hugepage filesystem. In case the value assigned to `mem-path` is not a valid hugepage filesystem, Qemu would create a mmap'd file for its work which might negatively impact performance.

Following logs will appear on the host UART console:

```
QEMU 2.5.0 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: disconnected:telnet::4446,server
```

#### NOTE

Complete QEMU logs are visible only when `telnet` is used for logging into the guest machine, as described in [Accessing virtual machine console](#) on page 513.

The `-s` option mentioned in the `qemu` command stops the virtual machine bootup after initial setup. Run the `info cpus` command on QEMU CLI interface to see the QEMU threads.

```
(qemu) info cpus
* CPU #0: thread_id=2559
 CPU #1: (halted) thread_id=2560
```

SSH on the board (telnet to IP address `IP_ADDR_BRD`) from other console and affine the threads to the cores using the `taskset` command:

```
taskset -p 0x4 <tid1>
taskset -p 0x8 <tid1>
```

#### NOTE

It is recommended to affine the VCPUs to the cores on which OVS threads are not running. For better performance VCPU threads should be given one physical CPU each if possible.

Run the `c` command from the QEMU CLI to continue the VM boot-up:

```
(qemu) c
```

## 9.2.5.4 Accessing virtual machine console

Telnet to the `IP_ADDR_BRD` at port `GUEST_CONSOLE_PORT` from any machine, which can reach `IP_ADDR_BRD` over network:

```
telnet 192.168.1.141 4446
Trying 192.168.1.141...
Connected to 192.168.1.141.
Escape character is '^'.
[0.000000] Booting Linux on physical CPU 0x0
[0.000000] Initializing cgroup subsys cpuset
[0.000000] Initializing cgroup subsys cpu
[0.000000] Initializing cgroup subsys cpuacct
[0.000000] Linux version 4.4.65 (root@dash1) (gcc version 5.4.0 20160609 (Ubuntu/Linaro
5.4.0-6ubuntu1~16.04.4)) #1 SMP PREEMPT Fri Jun 23 07:34:43 IST 2017
```

Only a partial terminal output has been shown above.

## 9.2.5.5 Launching two virtual machines

This section describes steps for launching 2 virtual machine simultaenously for multiple VM use case.

#### NOTE

- Memory assigned to each virtual machine should not exceed the total number of huge pages assigned on system. In following example, 2048Mb to each virtual machine has been specified and verified to be working correctly.
- Console telnet port of both virtual machine must be different. In the below example, VM1 has port 4446 and VM2 has port 4447 configured for telnet. Modify the command accordingly if different values are required.

Launch VM1:

```
qemu-system-aarch64 -nographic -object memory-backend-file,id=mem,size=$VM_MEM,mem-path=/mnt/
hugepages,share=on -cpu host -machine type=virt -kernel /boot/Image -enable-kvm -serial tcp::
4446,server,telnet -append 'root=/dev/vda rw console=ttyAMA0,115200 rootwait earlyprintk' -m
$VM_MEM -numa node,memdev=mem -chardev socket,id=char1,path=$VHOST1_PATH -netdev type=vhost-
```

```
user,id=hostnet1,chardev=char1,vhostforce,queues=$NUM_QUEUES -device virtio-net-pci,disable-
modern=false,addr=0x3,netdev=hostnet1,id=net1,mrg_rxbuf=off -chardev socket,id=char2,path=
$VHOST2_PATH -netdev type=vhost-user,id=hostnet2,chardev=char2,vhostforce,queues=$NUM_QUEUES -
device virtio-net-pci,disable-modern=false,addr=0x4,netdev=hostnet2,id=net2,mrg_rxbuf=off -smp
$VM_CORES -S -drive if=none,file=$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo
```

**Launch VM2:**

```
qemu-system-aarch64 -nographic -object memory-backend-file,id=mem,size=$VM_MEM,mem-path=/mnt/
hugepages,share=on -cpu host -machine type=virt -kernel /boot/Image -enable-kvm -serial tcp::
4447,server,telnet -append 'root=/dev/vda rw console=ttyAMA0,115200 rootwait earlyprintk' -m
$VM_MEM -numa node,memdev=mem -chardev socket,id=char1,path=$VHOST1_PATH -netdev type=vhost-
user,id=hostnet1,chardev=char1,vhostforce,queues=$NUM_QUEUES -device virtio-net-pci,disable-
modern=false,addr=0x3,netdev=hostnet1,id=net1,mrg_rxbuf=off -chardev socket,id=char2,path=
$VHOST2_PATH -netdev type=vhost-user,id=hostnet2,chardev=char2,vhostforce,queues=$NUM_QUEUES -
device virtio-net-pci,disable-modern=false,addr=0x4,netdev=hostnet2,id=net2,mrg_rxbuf=off -smp
$VM_CORES -S -drive if=none,file=$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo
```

## 9.2.5.6 Running DPDK applications in VM

All the DPDK applications mentioned in this section have been tested in following configuration:

- Two Physical network interfaces.
- Two virtio-net devices in the virtual machine.

Following figure illustrates the test setup:

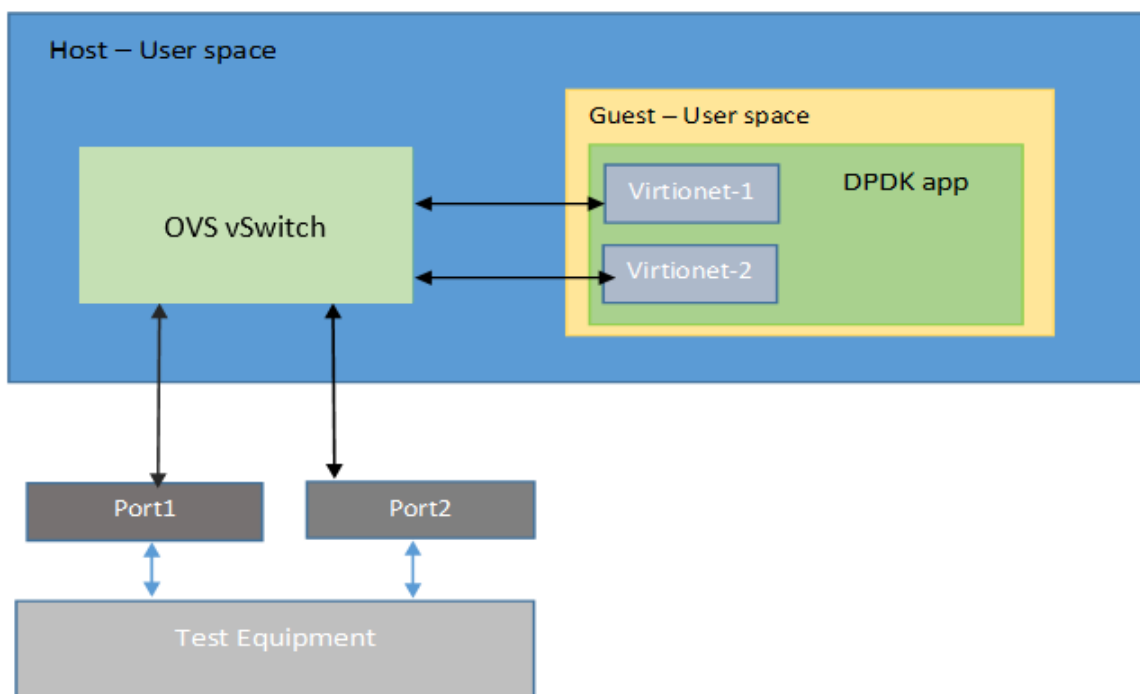


**Figure 62. DPDK virtionet test setup**

### Generic Setup

DPDK example application binaries are available in the `/usr/local/bin` folder in the Flexbuild generated roots.

- Setup Hugepages

```
mkdir /mnt/hugepages
mount -t hugetlbfs none /mnt/hugepages
echo 512 > /proc/sys/vm/nr_hugepages ; for dpaa1 change change size as 256
```

**NOTE**

For the below commands, it is assumed that they are executed from `/usr/local` folder. Modify the commands for different path or `PATH` variable configuration.

- Setup the devices using DPDK Scripts

```
/usr/share/usertools/dpdk-devbind.py --status
/usr/share/usertools/dpdk-devbind.py -b uio_pci_generic 0000:00:03.0
/usr/share/usertools/dpdk-devbind.py -b uio_pci_generic 0000:00:04.0
```

## Run DPDK Applications

**NOTE**

Using Core 0 for DPDK application can lead to non-deterministic behavior, including drop in performance. It is recommended that DPDK application core mask values avoid using Core 0.

Executing `l2fwd` application:

```
bin/l2fwd -c 0x2 -n 1 -- -p 0x1 -q 1 -T 0
```

Executing `l3fwd` application:

```
bin/l3fwd -c 0x2 -n 1 -- -p 0x1 --config="(0,0,1)" -P --parse-ptype
```

## Testpmd

- For TX only:

```
./bin/testpmd -c 0x3 -n 1 -- -i --nb-cores=1 --portmask=0x1 --nb-ports=1 --forward-mode=txonly
--disable-hw-vlan --port-topology=chained
```

- For RX only:

```
./bin/testpmd -c 0x3 -n 1 -- -i --nb-cores=1 --portmask=0x1 --nb-ports=1 --forward-mode=rxonly
--disable-hw-vlan --port-topology=chained
```

## 9.2.5.7 Multi Queue VIRTIO support

To scale the performance against the number of VM cores, the VIRTIO devices need to be configured with multiple queues. This section explains the steps required for setup multi queue VIRTIO devices.

See **Generic Setup** of DPAA platform including configuration necessary for defining multiple queues before DPDK application is executed. No special setup is required for DPAA2 before DPDK application start. Further, refer [Configuring OVS](#) on page 508 for setting OVS-DPDK on the host. Steps defined below build upon the configurations and steps provided in these sections for multiqueue support.

QEMU commands for multiqueue vhost devices are different and are shown later in the section.

### Additional steps for setup of OVS

Besides the steps mentioned in [Configuring OVS](#) on page 508, following changes are required to modify the number of supported queues in the virtual machine.

Run following commands after adding DPDK and vhost-user ports to the bridge:

```
./ovs-vsctl set Interface dpdk0 options:n_rxq=2
./ovs-vsctl set Interface dpdk1 options:n_rxq=2
./ovs-vsctl set Interface dpdk0 options:n_txq=2
./ovs-vsctl set Interface dpdk1 options:n_txq=2
./ovs-vsctl set Interface vhost-user1 options:n_rxq=2
./ovs-vsctl set Interface vhost-user2 options:n_rxq=2
./ovs-vsctl set Interface vhost-user1 options:n_txq=2
./ovs-vsctl set Interface vhost-user2 options:n_txq=2
```

### Launch VM with multiqueue VHOST devices

Similar to the steps mentioned in [Launch Virtual Machine](#) on page 511, following steps are required to start the virtual machine. Changes are highlighted with **bold**:

#### NOTE

Command snippets shown below are valid for DPAA2 platform. Replace **dpaa2** with **dpaa** for equivalent command on DPAA platform.

```
export GUEST_CONSOLE_TELNET_PORT=4446
export VM_MEM=2048M # For DPAA1 use VM_MEM=650M
export VM_CORES=2
```

```
export NUM_QUEUES=2
```

```
export ROOTFS_IMG=<VM_ROOTFS_IMG>
```

```
export VHOST1_PATH=/usr/local/var/run/openvswitch/vhost-user1
export VHOST2_PATH=/usr/local/var/run/openvswitch/vhost-user2
```

```
qemu-system-aarch64 -nographic -object memory-backend-file,id=mem,size=$VM_MEM,mem-path=/mnt/
hugepages,share=on -cpu host -machine type=virt -kernel /boot/Image -enable-kvm -serial tcp::
$GUEST_CONSOLE_TELNET_PORT,server,telnet -append 'root=/dev/vda rw console=ttyAMA0,115200 rootwait
earlyprintk' -m $VM_MEM -numa node,memdev=mem -chardev socket,id=char1,path=$VHOST1_PATH -netdev
type=vhost-user,id=hostnet1,chardev=char1,vhostforce,queues=$NUM_QUEUES -device virtio-net-
pci,disable-modern=false,addr=0x3,netdev=hostnet1,mq=on,id=net1,mrg_rxbuf=off,vectors=6 -chardev
socket,id=char2,path=$VHOST2_PATH -netdev type=vhost-
user,id=hostnet2,chardev=char2,vhostforce,queues=$NUM_QUEUES -device virtio-net-pci,disable-
modern=false,addr=0x4,netdev=hostnet2,mq=on,id=net2,mrg_rxbuf=off,vectors=6 -smp $VM_CORES -S -
drive if=none,file=$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo
```

### DPDK applications in VM

Connect to VM terminal as explained in [Accessing virtual machine console](#) on page 513. Once logged-in as Guest, DPDK applications using multiple queues can be run in VM.

#### NOTE

If the number of queues defined for DPDK application in VM is *not* equal to number of queues (NUM\_QUEUES) defined in QEMU command, the application may fail to start.

#### NOTE

For the below commands, it is assumed that they are executed from /usr/local folder. Modify the commands for different path or PATH variable configuration.

Besides the above steps, all steps are same as described in [single queue VM usecase](#).



Setup the devices using DPDK scripts:

```
/usr/share/usertools/dpdk-devbind.py --status
```

```
/usr/share/usertools/dpdk-devbind.py -b uio_pci_generic 0000:00:03.0
```

```
/usr/share/usertools/dpdk-devbind.py -b uio_pci_generic 0000:00:04.0
```

Execute `l3fwd` application:

```
./examples/l3fwd -c 0x3 -n 1 -- -p 0x3 --config="(0,0,0),(0,1,0),(1,0,1),(1,1,1)" -P --parse-ptype
```

Execute `testpmd` application:

```
./bin/testpmd -c 3 -n 1 -- -i --nb-cores=1 --nb-ports=1 --total-num-mbufs=1025 --forward-mode=txonly --disable-hw-vlan --rxq=2 --txq=2 --port-topology=chained
```

## 9.2.6 Enabling DPAA2 direct assignment for DPDK

The DPAA2 architecture supports the assignment of direct dpaa2 resource access from the QEMU guest VM (Kernel or userspace app). See **Direct assigned devices**.

This section describes necessary environment setup and commands for launching a Virtual Machine (VM) with VFIO device passthrough or direct device assignment support.

### NOTE

ARM-V8 currently support VM to work in NO-IOMMU mode only. Which means that all HW access will use physical address mode only. The default sdk code is build with virtual addressing mode only. You will need to rebuild the the dpdk for `arm64-dpaa2-linuxapp-gcc`, using `CONFIG_RTE_LIBRTE_DPAA2_USE_PHYS_IOVA=y` to use the dynamic IOVA mode.

To rebuild the DPDK binaries in dynamic IOVA mode: Change `CONFIG_RTE_LIBRTE_DPAA2_USE_PHYS_IOVA=y` in `config/defconfig_arm64-dpaa2-linuxapp-gcc` or, `config/defconfig_arm64-dpaa2-linuxapp-gcc`. (whichever config you are using to build the DPDK).

Then follow the instructions in [Standalone build of DPDK Libraries and Applications](#) on page 483 section to build DPDK applications.

You can transfer the applications manually to the virtual machine using the host-vm connections as suggested to configure in next section.

### 9.2.6.1 Launch Virtual Machine

1. Make sure that Kernel is enabled for direct assignment mode. See, [Host kernel: Enabling DPAA2 direct assignment](#) on page 556.
2. The default QEMU present in filesystem may not support the direct assignment feature. See, [Building QEMU](#) on page 558

Execute the following commands to build the QEMU 2.9 with VFIO passthrough support locally:

```
git clone https://source.codeaurora.org/external/qoriq/qoriq-components/qemu
cd qemu
git checkout qemu-2.9
git submodule update --init dtc
```

Make sure your m/c has the required packages to build QEMU. Refer the example below:

```
#update your ubuntu m/c with required packages.
apt-get install pkg-config
apt-get install libglib2.0-dev
apt-get install libpixman-1-dev
apt-get install libaio-dev
apt-get install libusb-1.0-0-dev
```

Now, build the QEMU

```
./configure --prefix=/root/qemu-2.9 --target-list=aarch64-softmmu --enable-fdt --enable-kvm --
with-system-pixman
make
make install
```

The new QEMU will be installed in /root/qemu-2.9 folder.

### 3. Create DPAA2 resources for the VM guest kernel and VM guest userspace (dpdk) on the board.

The dynamic scripts to support the dpaa2 resource creation are available in (/usr/local/dpdk/dpaa2) for ubuntu rootfs. It is also part of the DPDK source code in 'nxp' folder.

```
export DPDK_SCRIPTS=/usr/local/dpdk/dpaa2
```

Create a dpni based interface for file transfer and communication between host and VM guess kernel.

```
ls-addni -n
Output:---Created interface: eth0 (object:dpni.1, endpoint:)
```

Next create the VM guest kernel container. See *How to use DPAA2 direct assignment* chapters for further information.

A sample vm\_linux conf file is provided in scripts to create the vm guest kernel container. In this, the number of resources are good for 2 core VM. The previously created dpni object is also passed to connect it with guest kernel container.

```
source $DPDK_SCRIPTS/dynamic_dpl.sh -c $DPDK_SCRIPTS/vm_linux.conf dpni.1
```

Next step is to create the container for VM guest userspace for DPDK

```
source $DPDK_SCRIPTS/dynamic_dpl.sh -c $DPDK_SCRIPTS/vm_dpdk.conf <dpmac.1> <dpmac.2>
```

#### NOTE

Make sure to enter the created parent DPRC into vm\_dpdk.conf

For the rest of the chapter, it is assumed that VM guest kernel container is dprc.2 and VM guest userspace child container is dprc.3 (nested).

Create an ethernet connect between host and VM for communication/transfer. This was already created and passed during vm-linux container.

```
#assign IP to host interface created to communicate with VM (dprc.2, eth0)
ifconfig eth0 192.168.2.2
```

### 4. Create hugepages mount

```
echo hugetlbfs /mnt/hugetlbfs hugetlbfs defaults,mode=0777 0 0 >> /etc/fstab
mkdir /mnt/hugetlbfs
mount /mnt/hugetlbfs
```

### 5. Launch qemu (Version: 2.9.0) using following command:

For generating a root filesystem image, refer [Creating a guest Linux root filesystem](#). Assign the `ROOTFS_IMG` in below command with the absolute path to the generated image.

```
export ROOTFS_IMG=/ubuntu_bionic_arm64_rootfs.ext4.img # Telnet port to be used for accessing
this instance of virtual machine export GUEST_CONSOLE_TELNET_PORT=4446
export KERNEL_IMG=/root/Image-4.14 export KERNEL_IMG=/root/Image-4.14
```

**Define other environment variables which are used by the QEMU command to configure the virtual machine environment:**

```
export VM_MEM=4096M(2048M for LS1088ARDB)
```

1. Add the device command below (for the GUEST KERNEL DPRC to be assigned) to the QEMU command line:

```
-device vfio-fsl-mc,host=dprc.2
```

Also, make sure to specify the appropriate number of cores for the guest VM. It should match the number of dpio objects created in the child container. In our case, 1 core.

```
-smp $VM_CORES
```

2. Start QEMU with `-S` option (the vcpu threads are not yet started). We need this in order for the Ethernet drivers in the guest to correctly bind the objects to the cores.

```
single core VM launch /root/qemu-2.9/bin/qemu-system-aarch64 -smp $VM_CORES -m $VM_MEM -mem-
path /mnt/hugetlbfs -cpu host -machine type=virt,gic-version=3 -kernel $KERNEL_IMG -enable-kvm -
display none -serial tcp::$GUEST_CONSOLE_TELNET_PORT,server,telnet -drive if=none,file=
$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo -append 'root=/dev/vda rw
console=ttyAMA0 rootwait earlyprintk' -monitor stdio -device vfio-fsl-mc,host=dprc.2 -S
```

# Two core VM launch (check the isolcpus for core #1 in bootargs).

#### NOTE

- For best performance, Core 0 in the VM should not be used for DPDK I/O threads.
- To avoid system services from using GPUs scheduled for DPDK I/O threads, it is recommended that isolcpus be used for isolating cores from Linux Kernel scheduling in VM. The exact configuration is dependent on number of CPU assigned by QEMU to VM using the `VM_CORES` environment variable.

Append `isolcpus=1-$VM_CORES` to the `'root=/dev/vda rw console=ttyAMA0,115200 rootwait earlyprintk'` string in the `qemu-system-aarch64` command given above.

```
/root/qemu-2.9/bin/qemu-system-aarch64 -smp $VM_CORES -m $VM_MEM -mem-path /mnt/hugetlbfs
-cpu host -machine type=virt,gic-version=3 -kernel $KERNEL_IMG -enable-kvm -display none
-serial tcp::$GUEST_CONSOLE_TELNET_PORT,server,telnet -drive
if=none,file=$ROOTFS_IMG,id=foo,format=raw -device virtio-blk-device,drive=foo -append
'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk isolcpus=1' -monitor stdio
-device vfio-fsl-mc,host=dprc.2 -S
```

#### NOTE

Make sure to specify the appropriate number of cores for the guest VM. It should match the number of dpio objects created in. Also, make sure that the `/mnt/hugetlbfs` folder exists and is mounted when starting QEMU.

Following logs will appear on the host UART console:

QEMU 2.9.0 monitor - type 'help' for more information

```
(qemu) qemu-system-aarch64: -serial tcp::4446,server,telnet: QEMU waiting for connection on: disconnected:telnet::4446,server
```

3. Launch VM using : telnet <Board ip addr> <GUEST\_CONSOLE\_TELNET\_PORT> For example, telnet localhost 4446

Make sure to assign each vcpu thread to one physical CPU only.

Get the VM thread IDs entering QEMU shell.

```
(qemu) info cpus
* CPU #0: thread_id=7211
CPU #1: (halted) thread_id=7212
```

Assign one vcpu thread to one core only.

```
$ taskset -p 0x1 7211
pid 7211's current affinity mask: ff
pid 7211's new affinity mask: 1
$ taskset -p 0x2 7212
pid 7212's current affinity mask: ff
pid 7212's new affinity mask: 2
```

start the vcpu threads:

```
(qemu) c
```

## 9.2.6.2 Accessing the virtual machine console

```
root@ls2088ardb:~# telnet localhost 4446
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[0.000000] Booting Linux on physical CPU 0x0
[0.000000] Linux version 4.14.16-00004-ga5a4b5d (b10814@bf-netperf1.idc) (gcc version 7.2.1
20171011 (Linaro GCC 7.2-2017.11)) #2 SMP PREEMPT Tue Apr 3 12:24:09 IST 2018
[0.000000] Boot CPU: AArch64 Processor [410fd082]
[0.000000] Machine model: linux,dummy-virt
[0.000000] efi: Getting EFI parameters from FDT:
[0.000000] efi: UEFI not found.
[0.000000] cma: Reserved 16 MiB at 0x00000000ff000000
[0.000000] NUMA: No NUMA configuration found
----\
Ubuntu 16.04.3 LTS localhost ttyAMA0
localhost login: root
Password:
Last login: Wed May 2 20:08:32 UTC 2018 on ttyAMA0
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.14.16-00004-ga5a4b5d aarch64)
```

Only a partial terminal output has been shown above.

Use "root" as login & password

Execute the following commands:

```
echo 1000 > /proc/sys/vm/nr_hugepages
child DPRC container for VM guest userspace
export DPRC=dprc.3
echo 1 > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode
```

```
echo vfio-fsl-mc > /sys/bus/fsl-mc/devices/$DPRC/driver_override
echo $DPRC > /sys/bus/fsl-mc/drivers/vfio-fsl-mc/bind
```

Set the Start CPU core as per taskset (w.r.t host) to run DPDK app. This should be the first physical core which is assigned to VM. E.g if you are running two core VM and Core #2 and #3 are given to VM.

```
export HOST_START_CPU=2
```

### Setup Hugepages

```
echo hugetlbfs /mnt/hugetlbfs hugetlbfs defaults,mode=0777 0 0 >> /etc/fstab
mkdir /mnt/hugetlbfs
mount /mnt/hugetlbfs
```

configure the host connection for SCP, ssh and file transfer

```
ifconfig eth1 192.168.2.1
```

## 9.2.6.3 Running DPDK applications with direct device assignments

All the DPAA2 based dpdk application will work in VM similar to the host.

If the dpdk example applications are not present, you can bring them via scp/tftp using eth1 interface.

### NOTE

Using Core 0 for DPDK application can lead to non-deterministic behavior, including drop in performance. It is recommended that DPDK application core mask values avoid using Core 0.

Refer some example test commands below:

```
#one core VM (core #0 for dpdk)
./l3fwd -c 0x1 -n 1 --log-level=bus.fslmc,8 -- -p 0x1 -P --config="(0,0,0)"
./l2fwd-crypto -c 0x1 -n 1 --log-level=bus.fslmc,8 -- -p 0x1 -q 1 --chain HASH_ONLY
--auth_algo sha2-256-hmac --auth_op GENERATE --auth_key_random_size 64
```

#two core VM (core #1 for DPDK)

```
./l3fwd -c 0x2 -n 1 -- -p 0x1 -P --config="(0,0,1)"
./l3fwd -c 0x2 -n 1 -- -p 0x3 -P --config="(0,0,1),(1,0,1)"
./testpmd -c 0x3 -n 1 -- -i --portmask=0x3 --nb-cores=1 --forward-mode=txonly
```

## 9.2.7 DPDK on Docker

### 9.2.7.1 Docker Overview

Docker provides an environment for a given image, over which any user space application can be executed. An image must contain/expose all the tools which are required to run any application.

For more information on Docker, see <https://docs.docker.com/engine/userguide/>.

### 9.2.7.2 DPAA1-Platform

### 9.2.7.2.1 Running the Docker Container

To execute Docker, make sure you have completed the following prerequisites:

1. The Docker daemon must be running. If not, follow the instructions given at the link below to execute the daemon.

<https://docs.docker.com/engine/docker-overview/>

2. The Docker tool must be installed, which will be working as the client to run the Docker container.

Download the required image, which should be run as an environment. Use the command below to get generic prebuilt images:

```
docker pull <distribution>:<tag>

e.g. docker pull Ubuntu:latest /*for standard latest Ubuntu version*/
```

All downloaded images can be verified using the command below:

```
docker images
```

Once images are downloaded, the Docker container can be started using the steps below:

```
/*Below commands will execute a docker container named as docker0*/

/*Running docker container*/
docker run
--privileged /*It provides privilege to docker container to
 access host completely*/
--interactive /*Docker container will be running state */

/*Exporting host environment variable to docker container*/
--env LD_LIBRARY_PATH=/usr/local/lib

/*User defined name to docker container */
--name=docker0 --hostname=docker0

/*container will be detached once it is launched and host prompt will
be available to use*/
--detach

/*Exporting host partitions to docker container*/
--volume=/usr:/usr
--volume=/sys:/sys
--volume=/dev:/dev ubuntu: latest

/*Attaching docker console*/
docker exec -it docker0 bash
```

### 9.2.7.2.2 Running the DPDK Application

Once docker is launched and connected, then execute the DPDK application by running the respective command. The command below is a sample to run DPDK L3FWD:

- export DPAA\_DYNAMIC\_DIST=1
- l3fwd -c 0x0C -n 1 -p 0x30 --config="(4,0,2),(5,0,3)" -P

## 9.2.7.3 DPAA2-Platform

### 9.2.7.3.1 Traffic Multiplexer/De-Multiplexer

On the DPAA2 architecture, the MC provides various methods by which incoming traffic can be split of over the multiple DPNI's. The sections below provide more information.

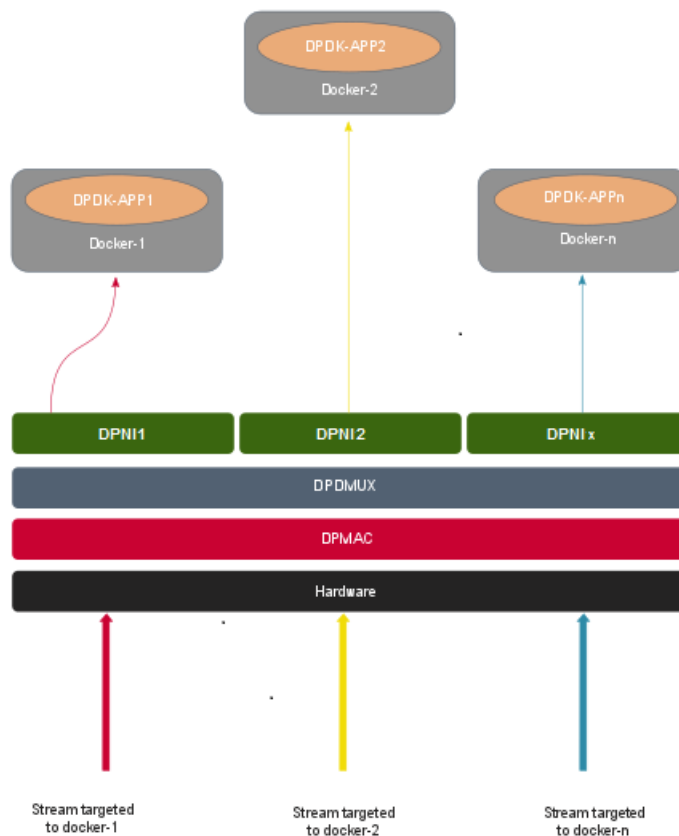
#### Using DPDMUX

MC provides an object (DPDMUX) which splits incoming traffic over the multiple DPNI's based on following parameters:

1. MAC based classification
2. VLAN based classification
3. MAC + VLAN base classification
4. User defined key based classification.

DPDMUX has its own filter table which consists of default filtering rules. Default filtering rules are a combination of MAC address configured on DPNI and port information as a destination. Once the DPDMUX object is connected to a given DPNI, then the entry for a particular DPNI will be added to the filtering table. All incoming default traffic will be distributed based on the destination MAC address in the packet. The user may add more entries to the filtering table as per his/her requirement.

The diagram below shows a sample use case for DPDMUX and associated links for a single DPMAC object. It can be extended up-to a maximum number of DPMACs, each having its own DPDMUX object.



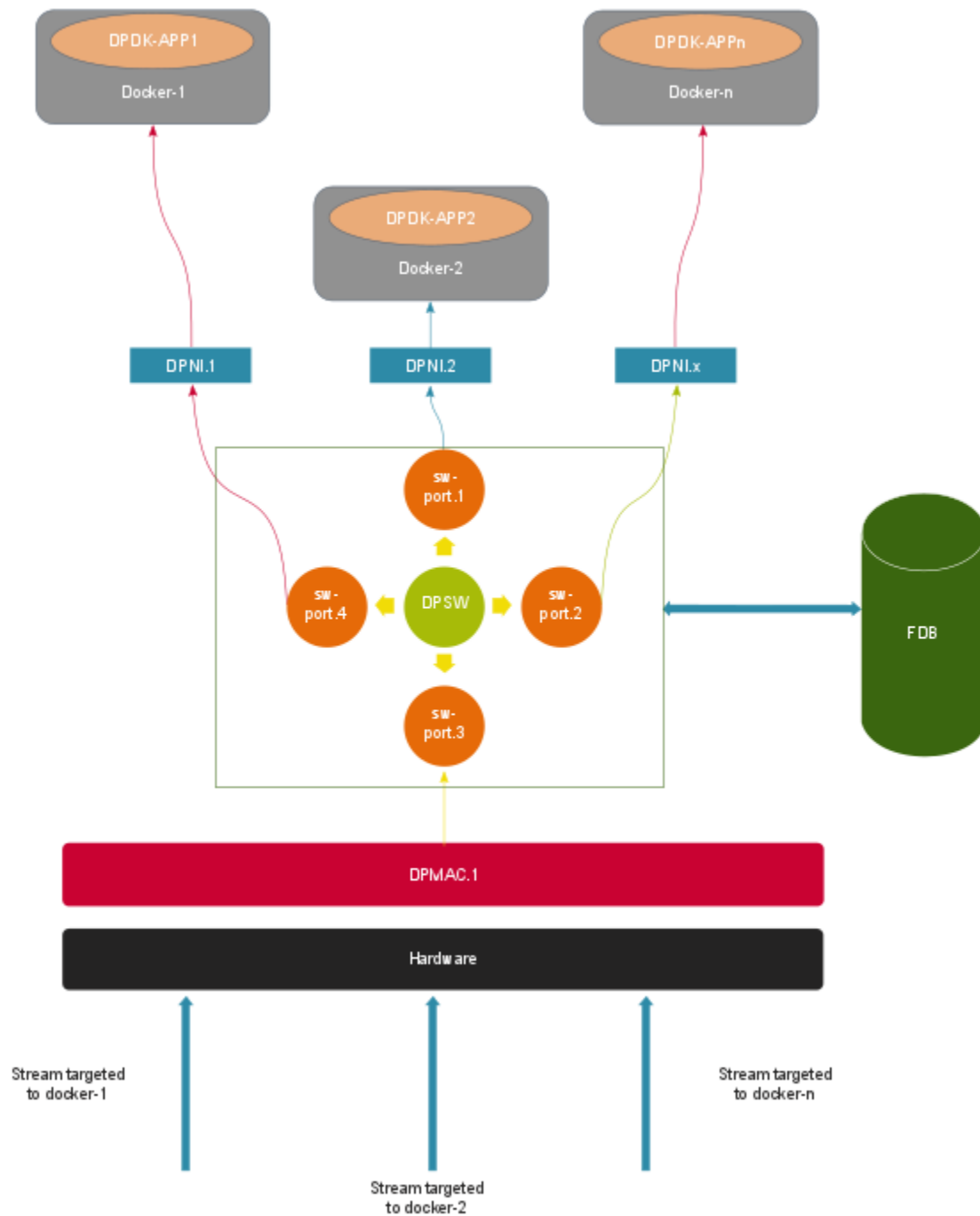
## Using DPSW

MC also provides another object(DPSW) which internally implements DPAA2 H/W Switch. This Switch instance can also be used for traffic forwarding to multiple hosts. On LS2088 there is only once instance of DPSW that can be created and required ports will be connected to the same DPSW instance.

DPSW has its own filter table which populates dynamically with source MAC address and port which packet is received on. Default incoming traffic will be flooded to all ports except ingress port and filtering rules will be learnt into filtering table. After learning, same packet will be forwarded to the destined port only.

Below diagram shows a sample use case for DPSW and associated links for single DPMAC object. It can be extended up-to maximum number of DPMACs with same DPSW instance.





## 9.2.7.3.2 Docker's Resource Setup

### 9.2.7.3.2.1 Application Container Configuration

For each Docker instance, a [DPRC](#) needs to be created containing DPAA2 hardware blocks necessary for the Docker container.

A helper script `dynamic_dpl.sh`, part of the LSDK rootfs, can be used for creating such DPRC. For example, following command snippet creates a DPRC containing 8 DPNI objects (logical network interfaces) which are not backed by any

physical link (DPMAC) and have MAC addresses starting from 00:00:00:00:05:00. For more details about creating DPRC, refer [Creating DPRCs](#) on page 328

```
/usr/local/dpdk/dpaa2/dynamic_dpl.sh dpni dpni dpni dpni dpni dpni dpni dpni -b 00:00:00:00:05:00
```

The output of the above command would be similar to:

```
Container dprc.2 is created

Container dprc.2 have following resources :=>

* 16 DPBP
* 8 DPCON
* 8 DPSECI
* 8 DPNI
* 10 DPIO
* 2 DPCI

Configured Interfaces

Interface Name Endpoint Mac Address
=====
dpni.1 UNCONNECTED 00:00:00:00:05:01
dpni.2 UNCONNECTED 00:00:00:00:05:02
dpni.3 UNCONNECTED 00:00:00:00:05:03
dpni.4 UNCONNECTED 00:00:00:00:05:04
dpni.5 UNCONNECTED 00:00:00:00:05:05
dpni.6 UNCONNECTED 00:00:00:00:05:06
dpni.7 UNCONNECTED 00:00:00:00:05:07
dpni.8 UNCONNECTED 00:00:00:00:05:08
```

Each such DPRC would be assigned to a Docker container. Thus, multiple such DPRC would have to be created as per the use-case and Docker instances required for it.

#### NOTE

Resources available on a DPAA2 system are limited and assigning them to DPRC can result error if requested resources are not available. For the above script output, if the script doesn't return any error and all the DPNI's have different MAC addresses, result can be considered success. In case of error or failure to assign MAC addresses, resource assignment to the DPRCs need to be restructured.

### 9.2.73.2.2 Kernel Container Configuration

The kernel has its defined root container (dprc.1). The user will be adding required traffic multiplexer/de-multiplexer objects into the kernel's container and connecting the application container's network interfaces for traffic distribution.

#### Configuration using DPDMUX

```
/*Create DPDMUX objects with total number of required links i.e. downlinks and uplinks both. Here
dpdmux.0 object is created*/

restool dpdmux create --num-ifs=3 --method DPDMUX_METHOD_MAC --max-dmat-entries=8 --max-mc-groups=8
--manip=DPDMUX_MANIP_NONE

/*Connecting downlinks and uplinks with above created DPDMUX */
restool dprc connect dprc.1 --endpoint1=dpmac.x --endpoint2=dpdmux.0.0
restool dprc connect dprc.1 --endpoint1=dpni.y --endpoint2=dpdmux.0.1
```

```
restool dprc connect dprc.1 --endpoint1=dpni.z --endpoint2=dpdmux.0.2
```

```
/*Where x, y and z are object indices created in resource containers*/
```

### Configuration using DPSW

```
/*Create DPSW object with total number of required links i.e. downlinks and uplinks both. Here dpsw.
0 object is created */
restool dpsw create --num-ifs=3
```

```
/*Connecting downlinks and uplinks with above created DPSW */
restool dprc connect dprc.1 --endpoint1=dpmac.x --endpoint2=dpsw.0.0
restool dprc connect dprc.1 --endpoint1=dpni.y --endpoint2=dpsw.0.1
restool dprc connect dprc.1 --endpoint1=dpni.z --endpoint2=dpsw.0.2
```

```
/*Where x, y and z are object indices which are created in resource containers*/
```

### 9.2.7.3.3 Running the Docker Container

To execute Docker, make sure you have completed the following prerequisites:

1. The Docker daemon must be running. If not, follow the instructions given at the link below to execute the daemon.

<https://docs.docker.com/engine/docker-overview/>

2. The Docker tool must be installed, which will be working as the client to run the Docker container.

Download the required image, which should be run as an environment. Use the command below to get generic prebuilt images:

```
docker pull <distribution>:<tag>
```

```
e.g. docker pull Ubuntu:16.04 for standard Ubuntu version 16.04
```

All downloaded images can be verified using the command below:

```
docker images
```

Once images are downloaded, the Docker container can be started using the steps below:

```
/*Below commands will execute a docker container named as docker0*/

/*Exporting application resource container to docker tool*/
export DPRC="dprc.<index>"
export VFIO_NO=`readlink /sys/bus/fsl-mc/devices/$DPRC/iommu_group | xargs basename`

/*Running docker container*/
docker run
--privileged /*It provides privilege to docker container to
access host completely*/
--interactive /*Docker container will be running state */

/*Exporting host environment variable to docker container*/
--env DPRC=$DPRC --env LD_LIBRARY_PATH=/usr/local/lib

/*Exporting host devices to docker container*/
--device=/dev/vfio/vfio:/dev/vfio/vfio
--device=/dev/vfio/$VFIO_NO:/dev/vfio/$VFIO_NO
```

```

/*User defined name to docker container */
--name=docker0 --hostname=docker0

/*container will be detached once it is launched and host prompt will
be available to use*/
--detach

/*Exporting host partitions to docker container*/
--volume=/usr:/usr
--volume=/sys:/sys
--volume=/dev:/dev ubuntu:16.04

/*Attaching docker console*/
docker exec -it docker0 bash

```

### 9.2.73.4 Running the DPDK Application

Once docker is launched and connected, then execute the DPDK application by running the respective command. The command below is a sample to run DPDK L3FWD:

```

./l3fwd -c 0xFF -n 4 -- -p 0xFF -P --config="(0,0,0),(1,0,1),(2,0,2),(3,0,3),(4,0,4),(5,0,5),
(6,0,6),(7,0,7)" -P

```

### 9.2.73.5 Example Configuration: Using DPDMUX

On Host:

```

export MAX_QOS=16
export NXP_CHRT_PERF_MODE=1
export DPNI_NORMAL_BUF=1

```

Create container for docker0:

```

./dynamic_dpl.sh dpni dpni dpni dpni dpni dpni dpni dpni -b 00:00:00:00:05:00

```

```

Container dprc.2 is created
Container dprc.2 have following resources :=>

```

```

* 16 DPBP
* 8 DPCON
* 8 DPSECI
* 8 DPNI
* 10 DPPIO
* 2 DPCI

```

```

Configured Interfaces

```

| Interface Name | Endpoint    | Mac Address       |
|----------------|-------------|-------------------|
| dpni.1         | UNCONNECTED | 00:00:00:00:05:01 |
| dpni.2         | UNCONNECTED | 00:00:00:00:05:02 |
| dpni.3         | UNCONNECTED | 00:00:00:00:05:03 |
| dpni.4         | UNCONNECTED | 00:00:00:00:05:04 |
| dpni.5         | UNCONNECTED | 00:00:00:00:05:05 |
| dpni.6         | UNCONNECTED | 00:00:00:00:05:06 |
| dpni.7         | UNCONNECTED | 00:00:00:00:05:07 |
| dpni.8         | UNCONNECTED | 00:00:00:00:05:08 |

## Create container for docker1:

```
./dynamic_dpl.sh dpni dpni dpni dpni dpni dpni dpni dpni -b 00:00:00:00:05:08

Container dprc.3 is created
Container dprc.3 have following resources :=>
* 16 DPBP
* 8 DPCON
* 8 DPSECI
* 8 DPNI
* 10 DPPIO
* 2 DPCI
Configured Interfaces
Interface Name Endpoint Mac Address
=====
dpni.9 UNCONNECTED 00:00:00:00:05:09
dpni.10 UNCONNECTED 00:00:00:00:05:0a
dpni.11 UNCONNECTED 00:00:00:00:05:0b
dpni.12 UNCONNECTED 00:00:00:00:05:0c
dpni.13 UNCONNECTED 00:00:00:00:05:0d
dpni.14 UNCONNECTED 00:00:00:00:05:0e
dpni.15 UNCONNECTED 00:00:00:00:05:0f
dpni.16 UNCONNECTED 00:00:00:00:05:10
```

## Create DPDMUX objects with downlinks and uplinks

```
restool dpdmux create --num-ifs=2 --method DPDMUX_METHOD_MAC --max-dmat-entries=8 --max-mc-groups=8
--manip=DPDMUX_MANIP_NONE
restool dpdmux create --num-ifs=2 --method DPDMUX_METHOD_MAC --max-dmat-entries=8 --max-mc-groups=8
--manip=DPDMUX_MANIP_NONE
restool dpdmux create --num-ifs=2 --method DPDMUX_METHOD_MAC --max-dmat-entries=8 --max-mc-groups=8
--manip=DPDMUX_MANIP_NONE
restool dpdmux create --num-ifs=2 --method DPDMUX_METHOD_MAC --max-dmat-entries=8 --max-mc-groups=8
--manip=DPDMUX_MANIP_NONE
```

## Create uplink connections

```
restool dprc connect dprc.1 --endpoint1=dpdmux.0.0 --endpoint2=dpmac.1
restool dprc connect dprc.1 --endpoint1=dpdmux.1.0 --endpoint2=dpmac.2
restool dprc connect dprc.1 --endpoint1=dpdmux.2.0 --endpoint2=dpmac.3
restool dprc connect dprc.1 --endpoint1=dpdmux.3.0 --endpoint2=dpmac.4
```

## Create downlink connections for docker0

```
restool dprc connect dprc.1 --endpoint1=dpni.1 --endpoint2=dpdmux.0.1
restool dprc connect dprc.1 --endpoint1=dpni.2 --endpoint2=dpdmux.1.1
restool dprc connect dprc.1 --endpoint1=dpni.3 --endpoint2=dpdmux.2.1
restool dprc connect dprc.1 --endpoint1=dpni.4 --endpoint2=dpdmux.3.1
```

## Create downlink connections for docker1

```
restool dprc connect dprc.1 --endpoint1=dpni.5 --endpoint2=dpdmux.0.2
restool dprc connect dprc.1 --endpoint1=dpni.6 --endpoint2=dpdmux.1.2
```

```
restool dprc connect dprc.1 --endpoint1=dpni.7 --endpoint2=dpdmux.2.2
restool dprc connect dprc.1 --endpoint1=dpni.8 --endpoint2=dpdmux.3.2
```

**NOTE:** The above commands are for 1G test. In case 10G port is to be used append the above commands to create uplink and downlink connections to append "--committed-rate=10000 --max-rate=10000"

### Running DPDK L2fwd on docker0

```
export DPRC="dprc.2"
export VFIO_NO=`readlink /sys/bus/fsl-mc/devices/$DPRC/iommu_group | xargs basename`

docker run --privileged --interactive --env DPRC=$DPRC --env LD_LIBRARY_PATH=/usr/local/lib --
device=/dev/vfio/vfio:/dev/vfio/vfio --device=/dev/vfio/$VFIO_NO:/dev/vfio/$VFIO_NO --name=docker0
--hostname=docker0 --detach --volume=/usr:/usr --volume=/sys:/sys --volume=/dev:/dev ubuntu:18.04

docker exec -it docker0 bash

cd /usr/local/bin

./l2fwd -c 0xF0 -n 1 --file-prefix=docker0 --socket-mem=2048 -- -p 0x0F -q 1
```

### Running DPDK L2fwd on docker1

```
export DPRC="dprc.3"
export VFIO_NO=`readlink /sys/bus/fsl-mc/devices/$DPRC/iommu_group | xargs basename`

docker run --privileged --interactive --env DPRC=$DPRC --env LD_LIBRARY_PATH=/usr/local/lib --
device=/dev/vfio/vfio:/dev/vfio/vfio --device=/dev/vfio/$VFIO_NO:/dev/vfio/$VFIO_NO --name=docker1
--hostname=docker1 --detach --volume=/usr:/usr --volume=/sys:/sys --volume=/dev:/dev ubuntu:18.04

docker exec -it docker1 bash

cd /usr/local/bin

./l2fwd -c 0xF0 -n 1 --file-prefix=docker1 --socket-mem=2048 -- -p 0x0F -q 1
```

#### NOTE

The above set of commands are for reference on LS2088A. On LS1088 DPDMUX object supports upto 4 downlinks (dpni's). These can be assigned to a docker instance as per requirement. For example, one usecase would assign two dpni's in each of the two docker container instances however other usecase would be to assign one dpni to each of four docker instances.

## 9.2.73.6 Example Configuration: Using DPSW

### On Host

#### Common Container settings:

```
export MAX_QOS=16
export NXP_CHRT_PERF_MODE=1
export DPNI_NORMAL_BUF=1
```

### Create container for docker0:

```
./dynamic_dpl.sh dpni -b 00:00:00:00:05:00

Container dprc.2 is created

Container dprc.2 have following resources :=>
* 16 DPBP
* 8 DPCON
* 8 DPSECI
* 1 DPNI
* 10 DPPIO
* 2 DPCI

Configured Interfaces

Interface Name Endpoint Mac Address
=====
dpni.1 UNCONNECTED 00:00:00:00:05:01
```

### Create container for docker1:

```
./dynamic_dpl.sh dpni -b 00:00:00:00:05:01

Container dprc.3 is created

Container dprc.3 have following resources :=>
* 16 DPBP
* 8 DPCON
* 8 DPSECI
* 1 DPNI
* 10 DPPIO
* 2 DPCI

Configured Interfaces

Interface Name Endpoint Mac Address
=====
dpni.2 UNCONNECTED 00:00:00:00:05:02
```

### Create DPSW objects

```
restool dpsw create --num-ifs=3
restool dprc connect dprc.1 --endpoint1=dpsw.0.0 --endpoint2=dpmac.1
```

### Create downlink connections for docker0

```
restool dprc connect dprc.1 --endpoint1=dpni.1 --endpoint2=dpsw.0.1
```

### Create downlink connections for docker1

```
restool dprc connect dprc.1 --endpoint1=dpni.2 --endpoint2=dpsw.0.2
```

**Running DPDK L2fwd on docker0**

```
export DPRC="dprc.2"
export VFIO_NO=`readlink /sys/bus/fsl-mc/devices/$DPRC/iommu_group | xargs basename`

docker run --privileged --interactive --env DPRC=$DPRC --env LD_LIBRARY_PATH=/usr/local/lib --
device=/dev/vfio/vfio:/dev/vfio/vfio --device=/dev/vfio/$VFIO_NO:/dev/vfio/$VFIO_NO --name=docker0
--hostname=docker0 --detach --volume=/usr:/usr --volume=/sys:/sys --volume=/dev:/dev ubuntu:18.04

docker exec -it docker0 bash

cd /usr/local/bin
./l2fwd -c 0x04 -n 1 --file-prefix=docker0 --socket-mem=2048 -- -p 0x01 -q 1
```

**Running DPDK L2fwd on docker1**

```
export DPRC="dprc.3"
export VFIO_NO=`readlink /sys/bus/fsl-mc/devices/$DPRC/iommu_group | xargs basename`

docker run --privileged --interactive --env DPRC=$DPRC --env LD_LIBRARY_PATH=/usr/local/lib --
device=/dev/vfio/vfio:/dev/vfio/vfio --device=/dev/vfio/$VFIO_NO:/dev/vfio/$VFIO_NO --name=docker1
--hostname=docker1 --detach --volume=/usr:/usr --volume=/sys:/sys --volume=/dev:/dev ubuntu:18.04

docker exec -it docker1 bash

cd /usr/local/bin
./l2fwd -c 0x08 -n 1 --file-prefix=docker1 --socket-mem=2048 -- -p 0x01 -q 1
```

**NOTE**

The above commands are for LS2088A only as LS1088A doesn't support DPSW object.

## 9.2.8 Known Limitations and Future Work

**Generic Limitations:**

1. The hardware internally uses the hardware access portals for each thread doing packet I/O, which limits the number of I/O threads thereby impacting the performance. The number of available portals is different for DPAA and DPAA2.
2. Not all functionalities supported by DPDK framework have been implemented by PPFE, DPAA and DPAA2 drivers (PMDs). For list of supported features, refer PPFE: Supported DPDK Features, [DPAA: Supported DPDK Features](#) on page 479 and [DPAA2: Supported DPDK Features](#) on page 480.
3. Using Core 0 for I/O related work is known to impact performance - whether on host or in VM. Disabling services can result in normal performance but the results are non-deterministic. Affining Core 0 to I/O should be avoided as much as possible.
4. It has been observed that PCI NIC card events can lead to performance drop on certain platforms. The behavior is non-deterministic across platforms. For peak performance numbers, PCI NIC cards should be disabled.
5. DPDK docker support is currently only available for DPAA2 and DPAA platforms.
6. I/O from secondary process is not supported in multi-process mode across any supported platform.

**PPFE Specific Limitations:**

1. While using PPFE in user space, if the kernel mode pfe module is loaded before using the user space mode, the HIF rings do not get cleaned sometimes and user need to restart the application again till the rings are cleaned (DPDK-1373).
2. SG is not supported. It will be supported in next release.



3. Link update for PPFE interfaces is not supported. It will be supported in next release.
4. Multiple buffer pools are not supported.
5. Some of the features are listed in DPDK-1375 which are currently not supported.

#### DPAA Specific Limitations:

1. Ports assigned to user space cannot be assigned dynamically to kernel space or vice-versa.
2. When running the default config (no DPAA\_FMC\_MODE), the system take upto 5 second in clean up the resources.

## 9.2.9 Troubleshooting

Following are some common steps and suggestions outlined for best performance from DPDK Applications:

1. To obtain best performance, please ensure that the boot-up time command line arguments are similar to below:

For DPAA2:

```
console=ttyS1,115200 root=/dev/mmcblk0p3 earlycon=uart8250,mmio,0x21c0600
default_hugepagesz=1024m hugepagesz=1024m hugepages=8 isolcpus=1-7
```

For DPAA:

```
console=ttyS0,115200 root=/dev/mmcblk0p3 earlycon=uart8250,mmio,0x21c0500 default_hugepagesz=2m
hugepagesz=2m hugepages=448 isolcpus=1-3 bportals=s0 qportals=s0
```

`isolcpus` in the above ensures that only Linux Kernel schedules its threads on Core 0 only. Core 1-x would be used for DPDK application threads.

Hugepage count defined by `hugepages` should also be modified to maximum possible so as to allow DPDK applications to have larger buffers.

#### NOTE

The value of `hugepages` is dependent on the size of RAM available on the board. Value should be selected based on specific use-case as any memory allocated for hugepage is not usable for Linux Kernel OS operations.

2. If there is issue with reception of transmission of packets, verify the following points:
  - a. Ensure that no error has been reported by DPDK application at startup. Generally the output is descriptive enough for cause of problem.
  - b. Check the mapping of ports against the physical ports:
    - In case of DPAA platform, ensure that the mapping of physical interfaces with DPDK ports is correct. Refer LS1043ARDB Port Layout or LS1046ARDB Port Layout.
    - In case of DPAA2 platform, ensure that correct `dpni.X` has been used in the `dynamic_dpl.sh` script while creating the `dprc` containers. A common pitfall is to use an incorrect `dpni` as against the physical port being used for IO.
  - c. Ensure that traffic generator to board connectivity is proper. You may run `testpmd` in `tx_only` mode to validate if the packets are going out on specific interfaces. For information about `testpmd` application and its supported arguments, refer [the web documentation](#).
  - d. Ensure that the traffic generator stream settings are correct and enough streams are being generated for proper distribution between DPDK application cores.
  - e. Ensure that the MAC address of stream generated by traffic generator matches that of the `dpni` port, or the interface is in promiscuous mode.
3. If the performance is not as expected:

- a. Ensure that the stream configuration of the traffic generator is appropriate and that it can generate multiple streams. In case the streams have all same IP destination and/or source, the distribution of traffic across multiple cores wouldn't happen.

#### NOTE

For obtaining best performance, it is important to configure the number of streams from packet generator adequately. If the number of streams generated by packet generator are not adequate, it would lead to improper distribution across the queues defined (especially in case of multiple queue setup) and eventually lack of performance.

- b. Using standard process tools in Linux, for example `ps`, `top`, verify that all the DPDK application threads have been started (as per application configuration on command line) and busy looping.
4. DPAA2 - the default mode for packet rx is prefetch mode, where it is expected that the rx API is receiving equal number of buffers in each call. However, if you want different number of buffers in each call, please do "export DPAA2\_NO\_PREFETCH\_RX=1"
5. For DPAA2 Platform certain tuning parameters are available. User can enable them according to the requirements.
  - To offload the RX error packet drop (parsing error) handling in hardware. Set,

```
#export DPAA2_PARSE_ERR_DROP=1
```
  - To disable the TX congestion control - i.e. infinite size of TX queues, set:

```
#export DPAA2_TX_CGR_OFF=1
```
  - To configure the TQ queue congestion control - taildrop size in byte (default is 64K bytes), set:

```
#export DPAA2_TX_TAILDROP_SIZE=<size>
```
6. System tuning parameters can be checked with "debug\_dump.sh" script located in "/usr/local/dpdk" directory. You can share the output with support team for further analysis.
7. DPAA2 status can be checked from restool commands. (e.g. `restool dpni info dpni.1`)
8. DPAA2 - Application hangs during initialization when number of buffers are very large (more than 1 Million) Problem occurs due to maximum limit of number of buffers in QBMAN. It is configurable in `dpc.0x2A_0x41.dts`. To configure number of buffers, following node needs to be added if node is not present otherwise add only entry marked.controllers.

```
{
 qbman {

 total_bman_buffers=<0x1B7790>;
 };
};
```

## 9.2.10 DPDK Performance Reproducibility Guide

This chapter describes various cases and points which are important for obtaining best performance from DPDK software on the NXP platforms. This is a suggestive list of best practices and optimal configurations which can help extract maximum performance of the NXP DPAA hardware.

#### NOTE

The practices mentioned in this chapter are based on tests in controlled environment. These are not intended for production or deployment without adequate analysis of the impact on use-cases.

This document is divided into two broad sections: Steps required before booting up the Linux Kernel and steps required before DPDK application execution.

## Before booting up Linux

### 1. Choosing Optimal Board Support Packages (BSP)

- Choosing a compatible board support package is critical for functionality as well as performance of DPDK application.

For DPAA and DPAA2 platforms, select the top frequency RCW/PBL binaries stably supported by boards. For example, for LS2088ARDB DPAA2, Rev 1.1 boards with frequency of **2100x800x2133** is known to perform best. Other frequency, though stable, would result in slower performance. Below table describes an indicative set of known BSP files for DPDK supported SoC.

### 2. Disabling hardware prefetching through u-boot

- For LS2088A DPAA2 platform, it is possible to disable hardware prefetching through u-boot. This can enhance performance in multicore scenario.
- For disabling hardware prefetching, following command should be used on u-boot prompt:

```
setenv hwconfig 'fsl_ddr:bank_intlv=auto;core_prefetch:disable=0xFE'
```

#### NOTE

Please change the `disable=` parameter based on the platform being used. For example, for LS1046/LS1043, having 4 cores, use `disable=0xE`, and for LX2 having 16 cores, use `disable=0xFFFE`.

After executing the above command, board bank needs to be reset for the setting to take place. In the above command, field `disable=0xFE` defines the mask for disabling prefetching on specific cores. For example, for disabling prefetching on 3rd and 4th core, use `disable=0x0C`.

Also, it should be noted that disabling prefetching on Core 0 is not supported.

#### NOTE

This setting doesn't have impact on single core case. Maximum performance gain is observed when all 8 cores of LS2088 board are being used (of which 7 cores have prefetching disabled as Core 0 doesn't support this feature).

### 3. Linux Boot Arguments

- For DPAA platform, if the on-board memory is limited (e.g. LS1043 RDB), following configuration should be appended to default boot arguments:

```
default_hugepagesz=2m hugepagesz=2m hugepages=448 isolcpus=1-3 bportals=s0 qportals=s0
```

Through the above boot arguments, 896 Mb of hugepages have been assigned for all DPDK applications (448 pages of 2M size each).

`isolcpus` isolates the CPUs 1, 2, 3 from Linux Kernel process schedulers' scheduling algorithm. All System Service would be scheduled on Core 0 and that should be avoided in application configuration for I/O threads.

- For DPAA2 platform, following configuration should be appended to default boot arguments:

```
default_hugepagesz=1024m hugepagesz=1024m hugepages=8 isolcpus=1-7
```

It is recommended to use 1G huge page size for DPAA2 platform.

#### NOTE

Change the value of `isolcpus` parameter based on the platform being used. For example, for LX2 platform use `isolcpus=1-15`.

- In case UEFI based booting is used, the boot arguments are changed from `grub.cfg`. Please refer to UEFI section on how to update the arguments.

**NOTE**

It should be noted that CPU isolation configuration cannot be changed in a running Linux Kernel. Whereas, huge page configuration can be changed from Linux prompt by writing to `/proc/sys/vm/nr_hugepages` file. Thus, CPU isolation should be carefully decided before booting up Linux Kernel.

**NOTE**

`nousb` can be appended to boot arguments to disable USB in Linux Kernel. This prevents any interrupts from USB devices to be serviced by CPU cores. This is especially important when Core 0 is being used for DPDK I/O performance. But, this option should only be used if there is no dependency of USB devices for system execution, for example, a USB mass storage which contains either the root filesystem or extra filesystem containing data necessary for execution.

4. For Best performance, use the data cores as isolated cpus and operate them in tickless mode on kernel version 4.4 above. For this:

- a. Compile the Kernel with `CONFIG_NO_HZ_FULL=y`
- b. Add bootargs with `'isolcpus=1-7 rcu_nocbs=1-7 nohz_full=1-7'` for 8 core platform and `'isolcpus=1-3 rcu_nocbs=1-3 nohz_full=1-3'` for 4 core platform

**NOTE**

The `CONFIG_NO_HZ_FULL` linux kernel build option is used to configure a tickless kernel. The idea is to configure certain processor cores to operate in tickless mode and these cores do not receive any periodic interrupts. These cores will run dedicated tasks (and no other tasks will be scheduled on such cores obviating the need to send a scheduling tick). A `CONFIG_HZ` based timer interrupt will invalidate L1 cache on the core and this can degrade dataplane performance by a few % points (to be quantified, but estimated to be 1-3%). Running tickless typically means getting 1 timer interrupt/sec instead of 1000/sec

## 5. Setup of the Performance Validation Environment

- It is important that the environment for performance verification uses a balanced core loading approach. Each core should be loaded with equal number of Rx/Tx queues, irrespective of their count. Images below describe some of the I/O scenario using an example setup containing a target board and a packet generator. In all the cases shown, it is assumed that each port has a single queue being serviced by a CPU core. Also, even though below images show 8 ports, it is a generic representation. DPAA boards may not have 8 equal ports (1G/10G) - this representation is assuming traffic is always distributed across equal capacity ports.

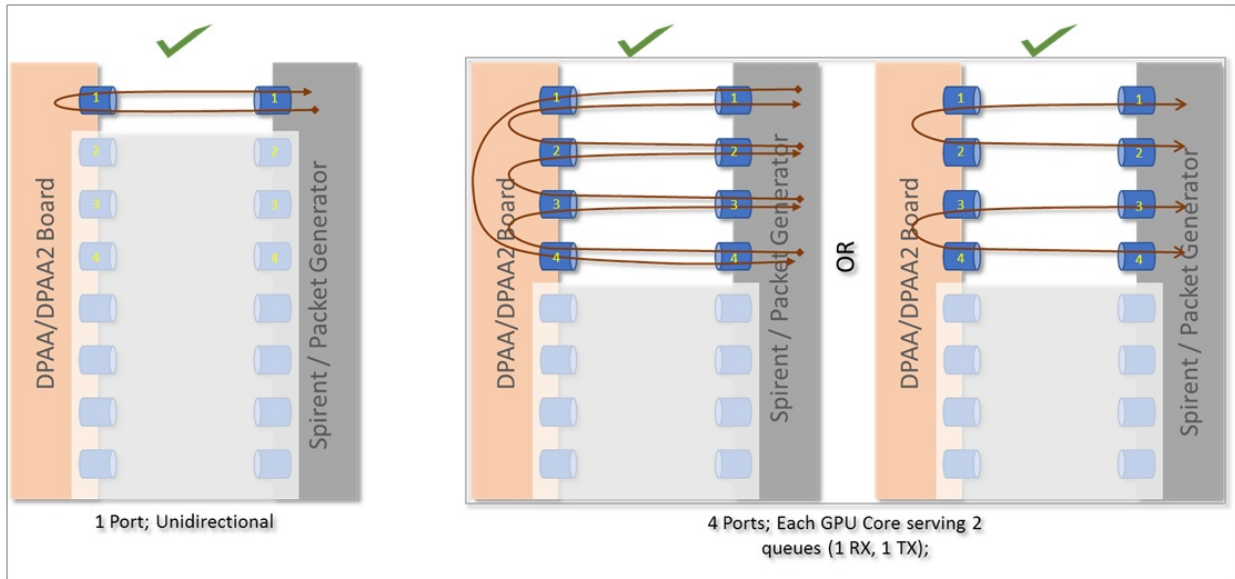


Image above describes 2 cases: One for single port and another for 4 ports. It can be noted that all the cores are equally loaded (equal number of cores, irrespective numbers of ports being serviced). Further, the 4 port case shows that there is more than one way to move stream of packets. (Note the direction of arrows in each case.)

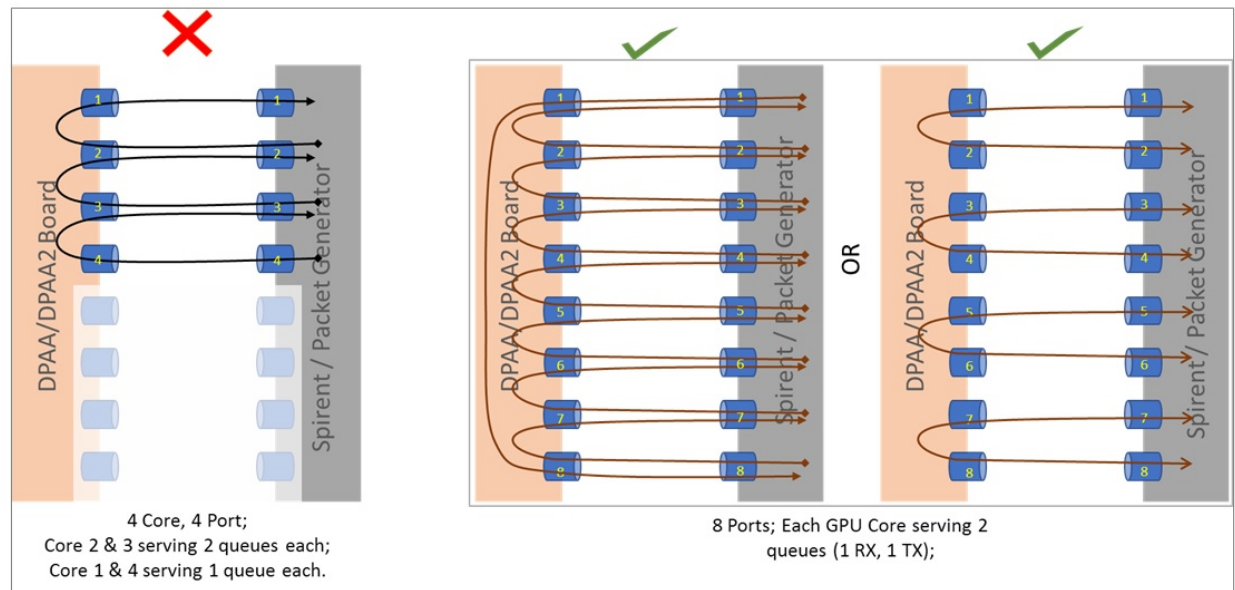


Image above describes a case with 4 ports where the CPU cores are not equally loaded. This is not a recommended combination as this would mean some streams being served (packet per second) slower than others. 8 port combination shown in the image above extends the mapping of 4 ports shown in image before. Once again, it should be noted that there are multiple ways to create a balanced set of streams. A performance setup should choose one baseline and all performance reports should be based on that baseline.

#### NOTE

For Performance measurement, performing I/O across non-equal capacity ports (1G=>10G, vice-verse) is not a valid case. This would lead to build up of queues on higher capacity links eventually stopping traffic when hardware is unable to obtain buffers for storing new incoming packets - eventually stopping traffic.

## 6. Uninstalling PCI Ethernet (e1000) NIC Cards

- It has been observed that when PCI Ethernet card (for example, on DPAA/DPAA2 RDB boards Intel e1000) are installed, they have a tendency to poll frequently the CPU cores (Core 0, in case of isolation). This has adverse impact on the application performance if DPDK I/O threads are scheduled on same cores which services these interrupts.
- For best performance, such PCI Ethernet cards should be uninstalled from the hardware. If un-installation is not possible, see the comments mentioned in section below to disable the interface by unlinking it from the Linux Kernel.

**NOTE**

`nopci` can be appended to boot arguments to completely disable PCI devices from being detected by Linux Kernel. This prevents PCI interrupts from being serviced by CPU. But, this option should not be used if there is dependency on any PCI device for system execution.

**Before and during DPDK Application start****1. Setting real-time priority for DPDK Application**

- In full fledged distributions, like Ubuntu, the root filesystem contains various system services by default. These services are targeted towards a generic environment. Many of these services require periodic CPU cycles. DPDK I/O threads execute as a run-to-completion process, infinitely looping over CPUs they are affined to. Services which require periodic CPU cycles can interrupt the DPDK I/O threads causing loss of packets and/or latency. Ideally, such services should be disabled or a rootfs without such services should be used for optimal performance. But, in case this cannot be done, real-time priority of application can also achieve desired results.
- Execute the script `/usr/local/dpdk/enable_performance_mode.sh`. Care should be taken to run the DPDK application from same shell as the one on which script was executed. This is because the script sets some environment variables which are used by DPDK application to define real-time priorities for its threads. This script is also designed to set to "performance" mode the CPU scaling governor. This prevents the CPU from putting itself into lower power state when not busy. This causes loss of traffic in initial I/O streams when the CPU is expected to spin upto its maximum frequency.

**NOTE**

This script sets the real-time priorities for any DPDK application which is run after the script has been executed. This also applies to application configured to run on Core 0. Thus, it is important to consider the implication. If the application is run on Core 0 and it is busy in I/O, it can lead to CPU stall causing complete lock-up. DPDK sample applications like `l2fwd`, `l3fwd`, `ipsec-secgw` are designed to relinquish the CPU when no I/O is being done. That way, using sample application, all core performance can be calculated. Similar care should be taken while developing custom DPDK applications. **This script is NOT designed to be used with Virtualization cases (Qemu, VM) and OVS.**

**NOTE**

Though this script doesn't necessarily require core isolation and tickless kernel, it is still recommended that I/O cores be isolated and tickless kernel be used to get the best performance environment. Also, this script assumes that it is a Ubuntu environment with power governor support and that no other process is running in priority higher than DPDK application.

**NOTE**

An opposite script, `/usr/local/dpdk/disable_performance_mode.sh`, is also available. This puts the processor back in the "on-demand" scaling governor configuration and also removed the environment variables. It is important to run this script once performance verification of a DPDK sample application has been completed. This would avoid issues with inadvertently executing DPDK application on Core 0 and causing a lock-up.

**2. Using High Performance (PEB) Buffer (Only for DPAA2)**

- In DPAA2 platform, while creating the resource container using the `dynamic_dpl.sh` script, it is possible to toggle between high performance PEB buffers and normal buffers (DDR). By default, the high performance buffers are enabled for LS2088A; for LS1088A, default configuration is normal buffers.

#### NOTE

For LS2088A, it is recommended to use high performance buffers which is enabled by default. Though, there is caveat to this as described below.

PEB buffers are limited resources. Overusage of buffers, either through large number of queues or deep taildrop settings, can cause the PEB buffers to overflow causing a interruption of I/O. The hardware might also enter a state from which it will not recover until board is restarted.

Exact limitations of number of queues is based on various parameters and cannot be stated objectively without defining the use-case. As a thumb-rule, refrain from using PEB buffers if configuration requires more than 1 queue per CPU core to be used, assuming all ports and CPU cores are being employed.

For toggling between normal and high performance buffers, use the following environment variable **before** executing the `dynamic_dpl.sh` script:

```
export DPNI_NORMAL_BUF=1
disables high performance buffers; enables normal buffers
```

### 3. Disabling PCI Ethernet (e1000) NICs

- As mentioned in the above section, it is preferable if no PCI Ethernet hardware (like e1000 on DPAA/DPAA2 boards) is installed. But, if it is not possible to uninstall a hardware device, following command can be used to unlink the Ethernet card from PCI driver in the Linux Kernel thereby preventing the CPU cores from being interrupted with periodic interrupts. This is specially important when all core performance is to be recorded.

```
echo 1 > /sys/bus/pci/devices/<PCI device BDF address>/remove
```

In the above command, replace `<PCI device BDF address>` with appropriate BDF format bus address of the PCI device, for example `0000:01:00.0`, after properly bypassing the `:` character in the name to avoid failure reported by Linux Bash prompt. For example, `echo 1 > /sys/bus/pci/devices/0000\:01\:00.0/remove`.

This command would unlink the PCI device with BDF address `0000:01:00.0` from its PCI driver's control, thereby disabling it from Linux Kernel.

#### NOTE

Once the device is unlinked from the PCI driver, it would not be usable through the Linux Kernel interface until bound to same or another PCI driver. It is out of scope for this document to record steps necessary for linking a PCI device to a PCI driver to bring it under Linux Kernel control.

### 4. DPDK Optimal Example Application Configuration

- Avoiding Core 0
  - As mentioned above, distributions like Ubuntu have large number of system services. Though some of these services can be disabled, there would always be cases of interrupts or un-interruptible services which would require Core 0 cycles. Isolating the cores through Linux Kernel can be done using Linux boot arguments. This would allow isolated cores to be used exclusively for DPDK I/O threads.
  - Once a configuration of isolated cores is set, similar configuration should be done in DPDK application using the `-c` or `--coremask` command line option.
  - If 4 core (in LS1043A or LS1046A) or 8 core (LS1088A or LS2088A) performance is required, system services should be disabled. Though, it should be noted that performance number using Core 0 show un-deterministic behavior of latency and packet losses. For example, LS2088A has been observed to perform fairly stable on 8 core configuration with services disabled, but same cannot be stated for LS1088A boards.
- Avoiding Core 0 in case of Virtual Machine



- Core 0 impact on the DPDK I/O performance is valid for host as well as for Virtual Machine (VM). While configuring DPDK application in VM, Core 0 should be avoided. The Qemu configuration should be such as to avoid using the Host's Core 0 for any VM logical core which is running DPDK I/O threads.
- For a VM environment, OVS or similar switching stack maybe used on the host. Qemu configuration should be such as to avoid mapping the logical cores (VCPU) assigned to VM with any of the CPU cores which run the switching stack threads. `taskset` command is recommended for affining the Qemu threads (serving VM VCPUs) to a particular core. Refer [Launch QEMU and virtual machine](#) for more details.
- Using Multi-queue configuration to spread load across multiple CPUs
  - DPDK applications can utilize RSS based spreading of incoming frames across multiple queues servicing a particular port. This is especially helpful in obtaining better performance by utilizing 1:N mapping of ports to CPU cores. That is, more than 1 CPU core serves a single port.

This requires adequate configuration of Port-Queue-Core combination through DPDK application command line. For example, `l3fwd` application can be configured to use 8 ports on a LS2088A board for serving 2 ports using the following command:

```
l3fwd -c 0xFF -n 1 -- -p 0x3 --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),(1,0,4),(1,1,5),
(1,2,6),(1,3,7)"
```

In the above command, the `--config` argument takes multiple tuples of (*port, queue, core*). Note that Port number 0 is being served by Core 0, 1, 2 and 3 using separate queue numbers.

Using similar configuration described for `l2fwd` application above, optimal utilization of Cores can be achieved. The command line options vary with DPDK application and DPDK online web manual should be referred for specific example applications.

Though the above command snippet utilizes Core 0, necessary care should be taken as described in text above.

- As mentioned above, DPDK uses RSS (Receive Side Scaling) to spread the incoming frames across multiple queues. Multi-queue setup needs to be supported by varying flows from the Packet Generator. The flows created should be such as to have varying Layer-2 or Layer-3 field values.
  - As flow distribution is based on hash over Layer-2 and Layer-3 fields, it is possible that lower number of flows would distribute unevenly across queues. Number of flows created should be large enough to spread equally across all the configured queues.
- Consideration for CPU clusters
  - SoC have multiple clusters housing one or more CPUs. Each cluster shares a L2 cache. In general, this allows threads sharing data over CPUs from same cluster to perform better than threads sharing data across CPUs from different clusters.
  - For best performance, it is recommended that DPDK application configuration for selecting CPU cores should be such to either use all CPUs from same cluster or spread queues equally across clusters. When this is combined with Core 0 issue, it implies that using Cluster having Core 0 might perform slightly worse than using cluster which doesn't use Core 0.
- Using limited number of I/O buffers
  - DPDK allows an application to change the number of maximum in-flight buffers. This is especially useful when there is memory constraint and DPDK application has limited resources.
  - Each buffer, for processing, has to be fetched into the system caches (L2/L1). Larger the number of buffers in-flight simultaneously, more would be the flushing of buffer addresses. To avoid excessive pressure on the L2 caches (eviction, hit, miss cycle), lower number of buffers should be used. Exact numbers would depend on the use-case and resources available.



For example, in case of `l3fwd` application, `--socket-mem=1025` like EAL argument can be provided to the application as shown in command snippet below. Note that the argument has been provided before the `--` - these are passed to DPDK framework rather than the application itself.

```
./l3fwd -c 0xFF -n 1 --socket-mem=1025 -- -p 0x1 --config="(0,0,0),(0,1,1),(0,2,2),(0,3,3),
(0,4,4),(0,5,5),(0,6,6),(0,7,7)"
```

- Degradation of OVS performance with increase in flows
  - It has been observed that OVS doesn't perform well when the number of flows are large. This is because of OVS's inherent design to use a flow matching table of size 8000. If larger than 8000 flows are used, the overall performance degrades because of hash collisions. If more than 8000 flows are required, use the following command **after** OVS bridge has been created:

```
ovs-vsctl set bridge br0 other-config:flow-eviction-threshold=65535
```

This command would set the size of OVS internal flow table to 65535.

- Use `-n 1` as argument passed to DPDK EAL
  - `-n` argument for DPDK application is for defining number of DDR channels for the system - which is typically valid for NUMA architectures. This parameter is used for mempool memory alignments. For NXP SoCs, this should be set to "1". NXP SoCs supported by DPDK are non-NUMA.

# Chapter 10

## Virtualization

### 10.1 KVM/QEMU

#### 10.1.1 KVM/QEMU Overview

This document is a guide and tutorial to building and using KVM (Kernel-based Virtual Machine) on NXP QorIQ SoCs.

Virtualization provides an environment that enables running multiple operating systems on a single computer system. Virtualization uses hardware and software technologies together to enable this by providing an abstraction layer between system hardware and the OS. The isolated environment in which OSes run is known as a *virtual machine* (or VM). The abstraction layer that manages all this is referred to as a *hypervisor or virtual machine manager*. The hypervisor layer operates at a privilege level higher than that of the operating systems, thus enabling it to enforce system security, ensure that virtual machines cannot interfere with each other, and transparently provide other services such as I/O sharing to the VM.

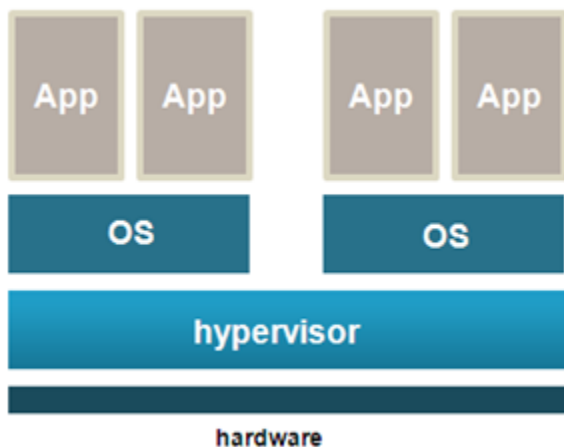


Figure 63.

KVM is a Linux kernel driver that together with QEMU, an open source machine emulator, provides an open source virtualization platform based on Linux. KVM and QEMU together act as a virtual machine manager that can boot and run operating systems in virtual machines. See Figure below.

In this document the term *host* kernel refers to the underlying instance of Linux with the KVM driver that acts as the hypervisor. The term *guest* refers to the operating system, such as Linux, that runs in a virtual machine. A virtual machine will be referred to as a "VM".

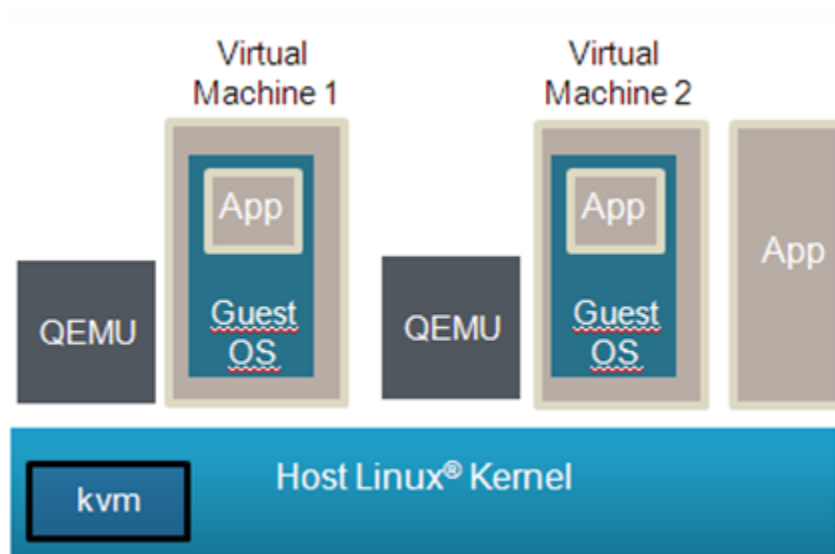


Figure 64.

NXP QorIQ SoCs based on ARM v7 and ARM v8 CPUs are supported.

### 10.1.1.1 Using QEMU and KVM

#### 10.1.1.1.1 Overview of Using QEMU

QEMU is used to start virtual machines. The QEMU application is named **qemu-system-arm** (for 32 bit platforms) or **qemu-system-aarch64** (for 64 bit platforms).

In addition to the QEMU executable itself, the following is a list of the minimum components that must be available on the target system to launch a virtual machine using QEMU:

- The host Linux kernel on the target must be built with virtualization support for KVM enabled .
- A guest OS kernel image (e.g. zImage or Image for Linux)
- A guest root filesystem (If needed by the guest OS. For example, a Linux guest requires a rootfs.)
- Recommended: A working network interface (to interface to the guest's console and the QEMU monitor)

The QEMU Emulator User Documentation [1] (see [References](#) on page 551) contains complete documentation for all QEMU command line arguments. The Table below summarizes some of the flags and arguments for basic operation.

Table 60.

| Argument          | Descriptions                                                                                                            |
|-------------------|-------------------------------------------------------------------------------------------------------------------------|
| -enable-kvm       | Specifies that the Linux KVM should be used for the virtual machine's CPUs                                              |
| -nographic        | Disables graphical output-console will be on emulated serial port.                                                      |
| -M <i>machine</i> | Specifies the type of virtual machine. One value is supported: <ul style="list-style-type: none"> <li>• virt</li> </ul> |

*Table continues on the next page...*

**Table 60. (continued)**

| Argument                            | Descriptions                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -smp <i>cpu_count</i>               | <p>Specifies the number of CPUs for the virtual machine.</p> <p>The number of virtual CPUs allowed is the same as the value of the CONFIG_NR_CPUS config option in the host Linux kernel. To see this value issue the following command from Linux on the target board:</p> <pre>zcat /proc/config.gz   grep NR_CPUS</pre>                                                                                     |
| -kernel <i>file</i>                 | Specifies the guest OS image. The supported image types are in <i>Image</i> format (the generic Linux kernel binary image file) and <i>zImage</i> (a compressed version of the Linux kernel image)                                                                                                                                                                                                             |
| -initrd <i>file</i>                 | Specifies a root filesystem image                                                                                                                                                                                                                                                                                                                                                                              |
| -append <i>cmdline</i>              | Use <i>cmdline</i> as the guest OS kernel command line (passed in the bootargs property of the /chosen node in the guest device tree)                                                                                                                                                                                                                                                                          |
| -serial <i>dev</i>                  | <p>Redirects the virtual serial port to the host device <i>dev</i>. QEMU supports many possible host devices. Please refer to the QEMU User Documentation [1] (see <a href="#">References</a> on page 551) for complete details.</p> <p>Note: if using a tcp device with the <b>server</b> option QEMU will wait for a connection to the device before continuing unless the <b>nowait</b> option is used.</p> |
| -m <i>megs</i>                      | <p>Specifies the size of the VM's RAM in megabytes. This option is ignored if using direct mapped memory.</p> <p>See <a href="#">Virtual Machine Memory</a> on page 545 for further details on options for allocating memory.</p>                                                                                                                                                                              |
| -mem-path <i>path</i>               | <p>Specifies the path to a file from which to allocate memory for the virtual machine. This option should be used to allocate memory from hugetlbfs.</p> <p>See <a href="#">Virtual Machine Memory</a> on page 545 for further details on options for allocating memory.</p>                                                                                                                                   |
| -monitor <i>dev</i>                 | <p>Redirects the QEMU monitor to the host device <i>dev</i>. QEMU supports many possible host devices. Please refer to the QEMU User Documentation [1] (see <a href="#">References</a> on page 551) for complete details.</p> <p>Note: if using a tcp device with the <b>server</b> option QEMU will wait for a connection to the device before continuing unless the <b>nowait</b> option is used.</p>        |
| -S                                  | Do not start CPU at startup (you must type 'c' in the monitor). This can be useful if debugging.                                                                                                                                                                                                                                                                                                               |
| -gdb <i>dev</i>                     | Wait for gdb connection on device <i>dev</i>                                                                                                                                                                                                                                                                                                                                                                   |
| -drive [ <i>args</i> ]              | <p>Used to create a virtual disk in a virtual machine.</p> <p>See <a href="#">Virtual block devices</a> on page 546 for additional information.</p>                                                                                                                                                                                                                                                            |
| Table continues on the next page... |                                                                                                                                                                                                                                                                                                                                                                                                                |

Table 60. (continued)

| Argument                                              | Descriptions                                                                                                                                                                                                                                          |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -netdev [args]<br>-device virtio-net-device<br>[args] | The -netdev and -device virtio-net-device arguments specify the network backend and front end for creating virtual network devices in virtual machines.<br><br>See <a href="#">Virtual network interfaces</a> on page 546 for additional information. |
| -cpu model                                            | Select CPU model. Only one model is supported: <ul style="list-style-type: none"> <li>• host</li> </ul>                                                                                                                                               |

Below is an example command line a user would run from the host Linux to start virt virtual machine booting a Linux guest:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/Image -
enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio
```

### 10.1.1.1.2 Virtual Machine Memory

QEMU allocates and loads images into a VM's memory prior to starting the VM. The amount of memory needed for a virtual machine will be dependent on the workload to be run in the VM. There are two ways to allocate memory:

#### 1. Allocation via hugetlbfs

Hugetlbfs is a Linux mechanism that allows applications to allocate memory backed large physically contiguous regions of memory. QEMU can take advantage of hugetlbfs for allocation of memory for virtual machines, which can provide a significant performance improvement over malloc allocated memory. Hugetlbfs allocated memory provides the flexibility of memory that can be allocated and freed with performance comparable to direct mapped memory.

The **-mem-path** argument to QEMU specifies the path to the hugetlbfs mount point where the huge pages should be allocated from.

The **-m** argument to QEMU specifies the amount of memory to allocate to the virtual machine. There are no constraints on the size passed to this argument other than that the amount of memory must fit within the constraints of the system and be enough for the workload in the VM.

See the how-to article [Quick-start Steps to Run KVM Using Hugetlbfs](#) on page 560 for an example of how to use hugetlbfs.

#### 2. Allocation via malloc

The default for QEMU is to allocate guest memory by the standard malloc facility available to user space applications in Linux. The amount of memory is specified with the -m command line argument. Malloc'ed memory has the flexibility of being allocated and freed by QEMU as needed. However, malloc'ed memory is backed by 4KB physical pages that are not contiguous and emulation is required by KVM to present a contiguous guest physical memory region to the VM. This approach is discouraged since the emulation can result in a substantial performance penalty for certain workloads.

The guest device tree generated by QEMU will contain a memory node that specifies the total amount of memory.

#### NOTE

A virtual machine's memory is part of the address space of the QEMU process. This means that the amount of memory allocated to a VM is limited by the standard limits that exist for Linux processes. A 32-bit host kernel has a 2GiB virtual address space used for stack, text, and other data, and this limits the amount of memory that can be allocated to a VM.

### 10.1.1.1.3 Virtual network interfaces

QEMU provides a number of options for creating virtual network interfaces in virtual machines. Virtual network interfaces are specified using the QEMU command line and guest software sees them as memory mapped devices.

There are two aspects of virtual network interfaces with QEMU:

1. The network “front-end”, which is the network card as seen by the guest. This is specified with the **-device** QEMU argument. The argument to specify a virtio network front end would look like: **-device virtio-net-pci**
2. The network “backend”, which connects the network card to some network. Network backend options include user mode networking, a host TAP interface, sockets, or virtual distributed Ethernet. The network backend is specified using the **-netdev** command line argument of QEMU. Note: It is possible to connect two virtual machines using virtual network interfaces. Normally QEMU userspace process emulates I/O accesses from the guest. However, there is an in-kernel implementation: *vhost-net* which puts the data plane emulation code into the kernel.

For example, to use a virtio NIC card with a TAP interface back-end the QEMU command line argument would look like:

```
-netdev tap,id=tap0,script=/root/qemu-ifup -device virtio-net-pci,netdev=tap0
```

The script “/root/qemu-ifup” is a script that QEMU invokes and passes the TAP interface name as an argument. For example, the script could add the TAP interface to an Ethernet bridge.

See the QEMU Users Manual [1] (see [References](#) on page 551) for detailed information about command line options and the types of network interfaces and backends. For best performance, the virtio front-end is recommended.

For additional information about QEMU networking see the references in [For More Information](#) on page 552.

For a detailed example, see the how-to article [How to Use Virtual Network Interfaces Using Virtio](#) on page 562 .

### 10.1.1.1.4 Virtual block devices

There are a number of approaches to provide a virtual disk to a KVM/QEMU virtual machine. A guest disk image can be a single raw file on the host filesystem, a file in a virtual disk format such as qcow2 and vdi, or a block device on the host Linux system. The virtual disk is assigned on the QEMU command line. In the example below, the file **my\_guest\_disk** is a disk image and is assigned to the VM when QEMU is launched: `-drive file=my_guest_disk,cache=none,if=virtio`

Refer to the QEMU Users manual [1] (see [References](#) on page 551) for details on the types of virtual disk images that may be created and the related arguments to QEMU.

QEMU allows for various storing caching attributes to be set for the guest. The cache option is specified with `cache=` property. The following options are supported:

- `writethrough`: The host page cache is used, but the data is written to the physical device. This mode ensures data integrity.
- `writeback`: This is the default mode (when the cache property is missing). The host page cache is used, the normal page cache management will handle the write to the storage device.
- `none`: The host page cache is bypassed, the guests writes go directly to the storage device. The storage device may have a write cache.
- `directsync`: The host page cache is bypassed and the data is written to the physical device.
- `unsafe`: The flush commands to ensure the data integrity are ignored.

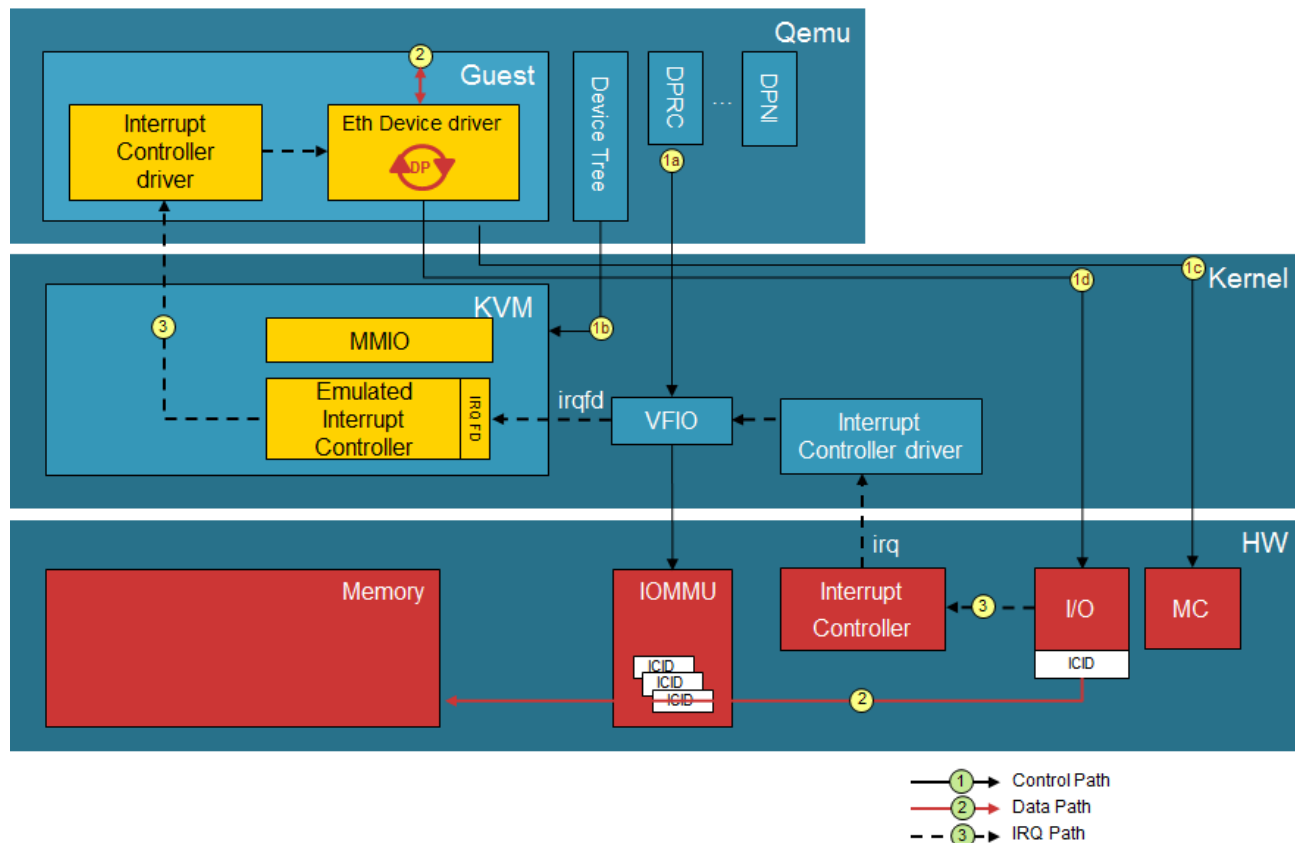
For a detailed example, see How to use **Virtual Disks Using Virtio**.

### 10.1.1.1.5 Direct assigned devices

#### VFIO - Virtual Function I/O

The VFIO is a Linux userspace driver infrastructure, an IOMMU/device agnostic framework for exposing direct device access from userspace. For the highest possible I/O performance, virtual machines make use of direct device access,





The root container always belongs to the Linux host, while any child container can be assigned to user-space applications such as DPDK or virtual machines - QEMU. In the context of *direct device assignment*, this means that any DPAA2 object that needs to be made available to a guest VM should be placed in a child container and, furthermore, the child container should be bound to the VFIO FSL MC driver. One can find more on how to use this feature in the [How to use DPAA2 direct assignment without scripts](#) on page 566 chapter.

### 10.1.1.1.6 VMs and the Linux Scheduler

Each virtual machine appears to the host Linux as a process with each virtual CPU in the VM implemented as a thread. A VM appears as an instance of QEMU when looking at Linux processes as can be seen in the example below:

```
$ ps -ef
 o
 o
root 1333 1 0 Oct01 ttyS0 00:00:00 -sh
root 1336 2 0 08:24 ? 00:00:00 [kworker/u4:2]
root 1372 1333 18 08:27 ttyS0 00:00:17 qemu-system-arm -enable-kvm -m
root 1361 1304 0 08:28 ? 00:00:00 sshd: root@pts/0
root 1363 1361 0 08:28 pts/0 00:00:00 -sh
 o
 o
```

CPUs appear as threads. To see thread IDs use the `info cpus` command in the QEMU monitor. Example of a VM with 8 virtual CPUs:

```
(qemu) info cpus
* CPU #0: thread_id=1984
```



```

CPU #1: (halted) thread_id=1985
CPU #2: (halted) thread_id=1986
CPU #3: (halted) thread_id=1987
CPU #4: (halted) thread_id=1988
CPU #5: (halted) thread_id=1989
CPU #6: (halted) thread_id=1990
CPU #7: (halted) thread_id=1991

```

To see the QEMU threads using the ps command:

```

root@ls_machine:~# ps -eL | grep qemu
1981 1981 ttyS1 00:00:00 qemu-system-aar
1981 1982 ttyS1 00:00:00 qemu-system-aar
1981 1983 ttyS1 00:00:00 qemu-system-aar
1981 1984 ttyS1 00:00:00 qemu-system-aar
1981 1985 ttyS1 00:00:00 qemu-system-aar
1981 1986 ttyS1 00:00:00 qemu-system-aar
1981 1987 ttyS1 00:00:00 qemu-system-aar
1981 1988 ttyS1 00:00:00 qemu-system-aar
1981 1989 ttyS1 00:00:00 qemu-system-aar
1981 1990 ttyS1 00:00:00 qemu-system-aar
1981 1991 ttyS1 00:00:00 qemu-system-aar

```

Being a Linux thread means that standard Linux mechanisms can be used to control aspects of how the threads are scheduled relative to other threads/processes. These mechanisms include:

- process priority
- CPU affinity
- isolcpus
- cgroups

### 10.1.1.2 Virtual Machine Overview

A guest OS running in a KVM/QEMU virtual machine "sees" a hardware environment similar to running on a physical board. The guest sees CPUs, memory, and a number of I/O devices. Some aspects of this environment are virtualized (emulated in software by KVM/QEMU) but this virtualization is mostly transparent to the guest, and changes to the guest are typically not required to run in a virtual machine.

The number of virtual machines that can be run simultaneously is only limited by the amount of available resources (like any other application on Linux).

KVM/QEMU implements a generic virt machine which is described completely by the device tree. The virtual machine contains the following resources:

- one or more ARMv7/ARMv8 virtual CPUs
- memory
- virtual console based on an emulated PL011
- virtio over PCI (used for virtual devices such as block and network devices)
- ARM Virtual Generic Interrupt Controller
- ARM virtual timer and counter

### 10.1.1.3 Introduction to KVM and QEMU

QEMU (pronounced KYOO-em-yoo) is a software-based machine emulator that emulates a variety of CPUs and hardware systems. KVM is a Linux kernel device driver that provides virtual CPU services to QEMU. The two software components work together as a virtual machine manager.

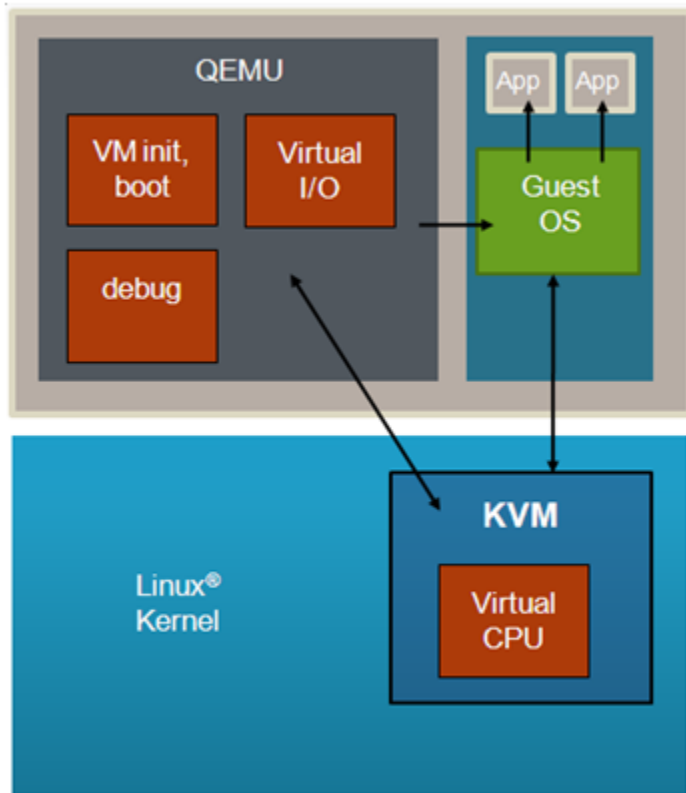


Figure 65.

QEMU is a Linux user-space application that runs on the host Linux instance and is used to start and manage a virtual machine. QEMU provides the following:

- A command line interface that provides extensive customization and configuration of a virtual machine when it is started-- e.g. type of VM, which images to load, and how virtual devices are configured
- Loading of all images needed by the guest-- e.g kernel images, root filesystem, guest device tree
- Setting the initial state of the VM and booting the guest
- Virtual I/O services, such as virtual network interfaces and virtual disks
- Debug services-which provide the capability to debug a guest OS using GDB (similar to a virtual JTAG)

KVM is a device driver in the Linux kernel whose key role in the VM architecture is to provide virtual CPU services. These services involve two aspects:

1. First, KVM provides an API set that QEMU uses to set and get the state of virtual CPUs and run them. For example, QEMU sets the initial values of the CPU's registers before starting the VM.
2. Second, after KVM starts a guest OS, certain operations (such as privileged instructions) performed by the OS cause an exception (or exit) into the host Linux kernel that must be handled and processed by KVM. This handling of traps is referred to as "emulation". These traps are transparent to the guest.

The KVM API is documented in the Linux kernel-- Documentation/virtual/kvm/api.txt.

KVM/QEMU supports virtual I/O which allows sharing of physical I/O devices by multiple VMs. Virtual network and block I/O are supported. See [For More Information](#) on page 552 for references that provide additional information on virtio.

### 10.1.1.4 Device Tree Overview

A device tree is a data structure that describes hardware resources such as CPUs, memory, and I/O devices. An device tree aware OS is passed a device tree which it reads to determine what hardware resources are available.

The host Linux kernel is booted first by a bootloader, for example u-boot (an open source bootloader). U-boot passes the kernel a **hardware** device tree that lists and describes all system hardware resources available to the host kernel (CPUs/cores, memory, interrupt controller and I/O).

Similarly, when a guest OS is booted in a KVM/QEMU virtual machine, QEMU passes it a **guest** device tree that describes all the hardware resources in the VM. See Figure below.

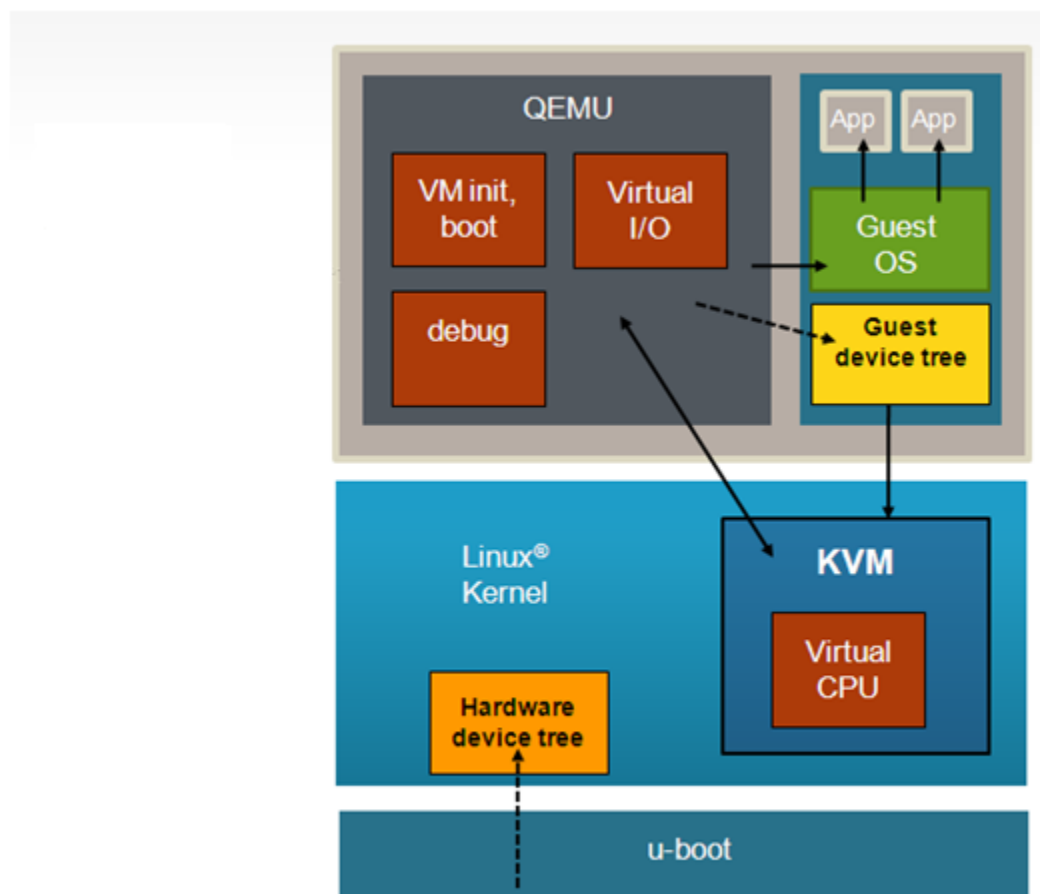


Figure 66.

The guest device tree is generated by QEMU and is used to define the resources a virtual machine will see. The guest device tree defines CPUs, memory, and I/O devices. QEMU places the guest device tree in the virtual machine's memory prior to starting the virtual machine.

### 10.1.1.5 References

[1] QEMU Emulator User Documentation: <http://qemu.weilnetz.de/qemu-doc.html>

[2] The Linux usage model for device tree data: <https://www.kernel.org/doc/Documentation/devicetree/usage-model.txt>

[3] Specification for virtio devices: <http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.pdf>

## 10.1.1.6 For More Information

### KVM

- KVM website: <http://www.linux-kvm.org>
- ARM VM specification: <http://lwn.net/Articles/589122/>
- Supporting KVM on ARM architecture: <http://lwn.net/Articles/557132/>

### QEMU

- QEMU website: <http://www.qemu.org/>

### Device Trees

- devicetree.org website: <http://devicetree.org>
- DTC, the device tree compiler is available at: <https://git.kernel.org/pub/scm/utils/dtc/dtc.git>. DTC also includes a library called libfdt which can be used by software to parse device trees.

Virtio-- a framework for doing virtual I/O using KVM/QEMU

- <http://www.ibm.com/developerworks/linux/library/l-virtio/>
- <http://ozlabs.org/~rusty/virtio-spec/virtio-paper.pdf>
- <http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.pdf>

### Virtual Networking with QEMU

- <http://wiki.qemu.org/Documentation/Networking>
- <http://www.linux-kvm.org/page/Networking>

## 10.1.1.7 Virtual machine reference

### 10.1.1.7.1 VM Overview

In general the architecture of KVM/QEMU is such that few changes should be needed to guest software to run in a VM-- i.e. a full virtualization approach is used, which means that virtual CPUs and virtual I/O devices behave like the physical hardware they are emulating.

However, there are some differences between virtual machines and native hardware that should be considered when targeting an OS to a KVM virtual machine. These differences can be divided into 2 general categories that will be discussed in further detail in this section:

1. Initial state and boot
2. CPUs

### 10.1.1.7.2 Memory Map of Virtual I/O Devices

The virt virtual machine contains a small subset of the devices found on a SoC. The available devices will be represented in the device tree passed to the guest at boot (e.g. virtual interrupt controller, virtual PCIE controller).

### 10.1.1.7.3 Virtual machine state at initialization

#### 10.1.1.7.3.1 Initial State and Boot

When booting the Host, kernel is entered into the HYP mode for ARMv7 respectively EL2 privilege level for ARMv8. After the boot the kernel uses a stub to install KVM and switches back to SVC, respectively EL1. The virtual machine has no virtualization extensions available, so the guest kernel will be entered in SVC mode (ARMv7) respectively EL1 (ARMv8).

In case of a real hardware the boot program will provide some services before giving control to the OS. The necessary steps needed to be done by the bootloader are described in the kernel documentation: *Documentation/arm/Bootimg/* (ARMv7), *Documentation/arm64/booting.txt* (ARMv8). In case of virtualization, KVM/QEMU makes the necessary actions to put hardware into the initial state (as seen by the guest) and also will take the role of the bootloader and makes the necessary settings.

It is recommended that a guest OS be minimally device tree aware. The libfdt library (available with the DTC tool) provides a full range of APIs to parse and manipulate device trees and will make the process of adding device tree awareness to an OS straightforward.

### 10.1.1.73.2 Initial State of Virtual CPUs

In a VM with multiple virtual CPUs, CPU #0 is the boot CPU and all other vcpus in the partition are considered secondary. The boot method for the secondary CPUs is PSCI.

The virtual CPU entry conditions comply with the entry conditions specified by *linux/Documentation/arm/Bootimg* (ARMv7) or *Documentation/arm64/booting.txt* (ARMv8)

### 10.1.1.74 Virtual CPUs

#### 10.1.1.74.1 Virtual CPU Specification

Software running in a virtual machine sees a virtual CPU that emulates an ARMv7/ARMv8 core without virtualization extensions.

The virtual CPU type will match that of the host hardware platform.

#### 10.1.1.74.2 Time in the Virtual CPU

ARM architecture has an optional extension, the generic timers, which provides:

- a counter (*physical counter*) that measures passing of time in real time
- a timer (*physical timer*) for each CPU. The timer is programmed to raise an interrupt to the CPU after a certain amount of time has passed.

The generic timers include virtualization support by introducing:

- a new counter, the *virtual counter*
- a new timer, the *virtual timer*.

This allows the virtual machine to have direct access to reading (virtual) counters and programming (virtual) timers without trapping.

KVM uses the physical timers in the host, the virtual machine access to the physical timers being disabled.

The virtual machine accesses the virtual timer and can, in this way, directly access the timer hardware without trapping to the hypervisor. However, the virtual timers do not raise virtual interrupts, but hardware interrupts which trap to the hypervisor. KVM injects a corresponding virtual interrupt into the VM when it detects that the virtual timer expired.

### 10.1.1.75 VGIC

The ARM Generic Interrupt Controller (GIC) provides hardware support for virtualization. The guest is able to mask, acknowledge and EOI interrupts without trapping to the hypervisor. However, there is a central part of the GIC called distributor which is responsible for interrupt prioritization and distribution to each CPU which does not provide virtualization extensions and for this part KVM provides an in-kernel emulation. Also, all the physical interrupts cannot be directly received by the guest. Instead, the KVM will program a virtual interrupt which will be raised in the guest. But, with the virtualization support in the GIC controller, when the guest is ACK-ing and EOI-ing the virtual interrupt, there is no need to trap into KVM.

QEMU/KVM provides 2 flavours of an emulated GIC:

- a GICv2 emulation which is the default option. Example command line: `-machine type=virt`
- a GICv3 emulation selected by the `gic-version` property. Example command line: `-machine type=virt,gic-version=3`. The GICv3 emulated interrupt controller is available only for platforms that have a physical GICv3 interrupt controller.

## 10.1.2 Configuring and Building

### 10.1.2.1 Overview

Linux with KVM enabled and QEMU can be built as part of the standard build process used to build the NXP LSDK.

The build instructions in the sections that follow assume a successful build/installation of the host. Please refer to the LSDK documentation for the host installation steps.

By default, the QEMU package installed on the target board will be the one retrieved from the Ubuntu 16.04 sources. In order to use features such as DPAA2 Direct Assignment or PCIE Direct Assignment, please refer to the [Building QEMU](#) on page 558 chapter in order to compile and build the necessary QEMU 2.9.0 version.

### 10.1.2.2 Quick Start - Recommended Configuration Options

The steps below show all the recommended configuration options to enable in order to build a kernel with virtual I/O enabled with the same kernel image serving as both host and guest. The sections that follow explain these options in further detail.

*Note:* The configuration options to run virtual machines are enabled by default in the LSDK. However they are listed here for reference.

1. From the main menuconfig window enable virtualization:

```
[*] Virtualization
```

2. In the virtualization menu enable the following options:

```
[*] Kernel-based Virtual Machine (KVM) support
```

3. Enable network bridging

```
Networking support --->
 Networking options --->
 <*> 802.1d Ethernet Bridging
```

4. Enable virtio PCI

```
Device Drivers --->
 Virtio drivers --->
 <*> PCI driver for virtio devices
```

5. Enable virtio for block devices

```
Device Drivers --->
 [*] Block devices --->
 <*> Virtio block driver
```

6. Enable virtio for network devices

```
Device Drivers --->
 [*] Network device support
```

```
[*] Network core driver support
 <*> Universal TUN/TAP device driver support
 <*> Virtio network driver
```

#### 7. Enable vhost for virtio network devices

```
[*] Virtualization
 <*> Host kernel accelerator for virtio net
```

#### 8. Enable Huge TLB file support

```
File Systems --->
 Pseudo filesystems --->
 [*] Huge TLB file system support
```

#### 9. Enable guest serial support

```
Device Drivers --->
 Character devices --->
 Serial drivers --->
 <*> ARM AMBA PL011 serial port support
 [*] Support for console on AMBA serial port
```

#### 10. Enable VFIO support

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework
```

#### 11. Enable VFIO support for QorIQ DPAA2 fsl-mc (Management Complex) devices

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework (VFIO [=y]) --->
 [*] VFIO No-IOMMU support ----
 <*> VFIO support for QorIQ DPAA2 fsl-mc bus devices
```

#### 12. Enable support for PCI VFIO

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework (VFIO [=y]) --->
 [*] VFIO No-IOMMU support ----
 <*> VFIO support for PCI devices
```

### 10.1.2.3 Host Kernel: Enabling KVM

This section describes the core, basic options needed to enable KVM in the host kernel. KVM is enabled in the host kernel under the virtualization menu of the main kernel menuconfig window.

```
[*] Virtualization
```

Core KVM support is enabled as follows:

```
[*] Kernel-based Virtual Machine (KVM) support
```

### 10.1.2.4 Host Kernel: Enabling Virtual Networking

[Virtual network interfaces](#) on page 546 describes how virtual networking can be used to give each VMs a virtual network interface which share physical network interfaces in Linux.

One common approach to configuring virtual networking is for QEMU to use a tun/tap interface bridged to a physical network interface. To do this Ethernet bridging and the kernel's tun/tap features must be enabled in the host kernel:

```
Networking support --->
 Networking options --->
 <*> 802.1d Ethernet Bridging
Device Drivers --->
 [*] Network device support
 [*] Network core driver support
 <*> Universal TUN/TAP device driver support
```

In order to enable vhost-net, the following config option should be enabled:

```
[*] Virtualization
 <*> Host kernel accelerator for virtio net
```

### 10.1.2.5 Host kernel: Enabling DPAA2 direct assignment

[Direct assigned devices](#) on page 546 chapter describes the mechanism used to passthrough fsl-mc bus devices to guest VMs using the VFIO framework. This section lists the Kconfig options that should be enabled in the Linux host kernel in order to support *DPAA2 Direct Assignment*.

Enable VFIO framework support

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework
```

Enable VFIO support for QorIQ DPAA2 fsl-mc (Management Complex) devices

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework (VFIO [=y]) --->
 [*] VFIO No-IOMMU support ----
 <*> VFIO support for QorIQ DPAA2 fsl-mc bus devices
```

#### NOTE

"VFIO No-IOMMU support" option is needed (only) for VFIO support in guest (e.g. DPDK in guest userspace)

### 10.1.2.6 Host kernel: Enabling PCIE direct assignment

[Direct assigned devices](#) on page 546 chapter describes the mechanism used to passthrough PCI devices using the VFIO framework.

This section lists the required Kconfig options in the host Linux kernel in order to use the aforementioned feature.

Enable VFIO framework support

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework
```



Enable support for PCI VFIO

```
Device Drivers --->
 <*> VFIO Non-Privileged userspace driver framework (VFIO [=y]) --->
 <*> VFIO support for PCI devices
```

### 10.1.2.7 Guest kernel: Enabling console

QEMU emulates an AMBA/PL011 console.

Below the kernel configuration options are shown to enable console:

```
Device Drivers --->
 Character devices --->
 Serial drivers --->
 <*> ARM AMBA PL011 serial port support
 [*] Support for console on AMBA serial port
```

### 10.1.2.8 Guest Kernel: Enabling Network and Block Virtual I/O

Virtio is a framework for doing paravirtualized I/O using QEMU/KVM. In order to support communication between guest and hypervisor virtio uses a PCI transport protocol.

Below the kernel configuration options are shown to enable virtio-pci:

```
Device Drivers --->
 Virtio drivers --->
 <*> PCI driver for virtio devices
```

Below the kernel configuration options are shown to enable virtio drivers in the Linux kernel to support networking I/O and block (disk) I/O.

```
Device Drivers --->
 [*] Network device support
 [*] Network core driver support
 <*> Virtio network driver
Device Drivers --->
 [*] Block devices --->
 <*> Virtio block driver
```

### 10.1.2.9 Building kernel with KVM support using flexbuild

#### NOTE

The steps presented here assume an understanding of using the `flex-builder` script to build LSDK. For details refer to LSDK building instructions.

The kernel can be built using the `flex-builder` utility (For more information please refer to the LSDK building instructions):

```
flex-builder -c linux -a arm64 // build linux kernel for arm64 platforms
flex-builder -c linux -a arm32 // build linux kernel for arm32 v8 platforms
flex-builder -c linux -a arm32 -m ls1021atwr // build for LS1021a arm32 v7 platform
```

If the kernel configuration needs to be changed, the custom option should be invoked and the necessary changes performed:

```
flex-builder -c linux:custom -a arm64 // generate custom kernel .config for arm64 in interactive
memu
flex-builder -c linux -a arm64 // build kernel based on the customized .config above
```

The same kernel image will be used by both guest and host.

## 10.1.2.10 Building QEMU

`flex-builder` script is used to generate the host root file system (For more details please see the LSDK building instructions). The generated host root filesystem already contains the QEMU installed. In the case the user wants to use a different QEMU version, this new version should be manually compiled and installed on the target (*Note:* The provided steps are targeting 64 bit platforms).

In order to use the DPAA2 Direct Assignment feature, a user should use the following commands in order to compile and install the proper QEMU version on the target.

1. Clone QEMU:

```
$ git clone https://source.codeaurora.org/external/qoriq/qoriq-components/qemu
```

2. Get the right branch:

```
$ git checkout qemu-2.9
```

3. Install the fdt library. The fdt library is a dependency, QEMU 2.9 uses a newer library than the one from the Ubuntu 16.04 userland, so compile it locally.

```
$ git submodule update --init dtc
```

4. Install the dependencies. These are the minimum dependencies required:

```
$ apt-get install pkg-config
$ apt-get install libglib2.0-dev
$ apt-get install libpixman-1-dev
$ apt-get install libaio-dev
$ apt-get install libusb-1.0-0-dev
```

If the last two dependencies are not present, the config step will not complain but will not build the required support.

5. Configure and compile QEMU

```
$./configure --prefix=<folder_where_the_qemu_will_be_installed> --target-list=aarch64-softmmu
--enable-fdt --enable-kvm --with-system-pixman
$ make
```

6. Install QEMU

```
$ make install
```

7. Include the folder containing the qemu executable in the system path.

```
$ export PATH=<folder_where_the_qemu_will_be_installed>/bin:$PATH
```

8. Make sure that the minimum required libraries are linked:

```
$ ldd qemu-system-aarch64
linux-vdso.so.1 => (0x0000ffffbad94000)
```

```

libz.so.1 => /lib/aarch64-linux-gnu/libz.so.1 (0x0000ffffbad35000)
libaio.so.1 => /lib/aarch64-linux-gnu/libaio.so.1 (0x0000ffffbad23000)
libpixmap-1.so.0 => /usr/lib/aarch64-linux-gnu/libpixmap-1.so.0 (0x0000ffffbacbe000)
libutil.so.1 => /lib/aarch64-linux-gnu/libutil.so.1 (0x0000ffffbacab000)
libnuma.so.1 => /usr/lib/aarch64-linux-gnu/libnuma.so.1 (0x0000ffffbac8d000)
libusb-1.0.so.0 => /lib/aarch64-linux-gnu/libusb-1.0.so.0 (0x0000ffffbac67000)
libglib-2.0.so.0 => /lib/aarch64-linux-gnu/libglib-2.0.so.0 (0x0000ffffbab60000)
libstdc++.so.6 => /usr/lib/aarch64-linux-gnu/libstdc++.so.6 (0x0000ffffba9d1000)
libm.so.6 => /lib/aarch64-linux-gnu/libm.so.6 (0x0000ffffba924000)
libgcc_s.so.1 => /lib/aarch64-linux-gnu/libgcc_s.so.1 (0x0000ffffba903000)
libpthread.so.0 => /lib/aarch64-linux-gnu/libpthread.so.0 (0x0000ffffba8d7000)
libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffffba790000)
/lib/ld-linux-aarch64.so.1 (0x0000ffffbad69000)
libudev.so.1 => /lib/aarch64-linux-gnu/libudev.so.1 (0x0000ffffba75f000)
libpcrc.so.3 => /lib/aarch64-linux-gnu/libpcrc.so.3 (0x0000ffffba6ee000)

```

9. Make sure that the qemu version is the expected one.

```

$ qemu-system-aarch64 --version
QEMU emulator version 2.9.0 ...

```

### 10.1.2.11 Creating a host Linux root filesystem

Creating a Linux root filesystem is out of the scope of this document. Please reference the NXP LSDK documentation on how to create root filesystems with `flex-builder` installer script. This section describes the software components needed on the host root filesystem to use KVM/QEMU.

The host root filesystem is the filesystem booted by the host kernel. The host rootfs is distinct from a guest root filesystem which may be needed by certain guest such as Linux.

A host root filesystem capable of running Linux as a guest needs the following components:

- Guest Linux kernel image (e.g. Image, zImage)
- QEMU executable (`qemu-system-aarch64` or `qemu-system-arm`)
- Guest root filesystem

Example host root filesystem layout with the required components to boot a Linux guest:

```

/root/zImage # guest Linux kernel
/root/ubuntu_bionic_arm64_rootfs.ext4.img # guest virtual disk image
/usr/bin/qemu-system-arm # QEMU for ARMv7 platforms
/usr/bin/qemu-system-aarch64 # QEMU for ARMv8 platforms

```

### 10.1.2.12 Creating a guest Linux root filesystem

In order to run a virtual machine, a guest Linux root filesystem is needed. There are various possibilities to host a guest root filesystem: a ramdisk, a virtual disk image, a block device on the host Linux system.

Also there are multiple virtual disk formats. `qemu-img` command can be used to generate, alter and convert between various virtual disk image formats.

A raw virtual disk can be created with the `flex-builder` script (the command is actually a wrapper over `qemu-img`):

```
flex-builder -i mkguestrfs -a <arch> -B 2G
```

The command will generate a 2GB raw disk image:

```
build/images/ubuntu_bionic_arm64_rootfs.ext4.img
```

## 10.1.3 KVM/QEMU How-to's

### 10.1.3.1 Quick-start Steps to Build and Deploy KVM

The following steps show how to build and deploy the necessary components in order to run virtual machines:

1. Build and install the LSDK on the board. (for details see [#unique\\_57](#))
2. Build the guest virtual disk (for details see [Creating a guest Linux root filesystem](#) on page 559)
3. Transfer the guest virtual image and the guest image on the host. The guest image (Image or zImage) is already in the /boot partition on the host system.

### 10.1.3.2 Quick-start Steps to Run KVM Using Hugetlbfs

This example assumes that the host Linux kernel is booted, has a working network interface, and the following images are present in the host root filesystem:

- Guest kernel image (*/boot/zImage* or */boot/Image*)
- Guest virtual disk image (*/root/ubuntu\_bionic\_arm64\_rootfs.ext4.img*)
- QEMU (*/usr/bin/qemu-system-arm* or */usr/bin/qemu-system-aarch64*)

Mount the HugeTLB filesystem on the host:

```
echo 512 > /proc/sys/vm/nr_hugepages
mkdir /mnt/hugetlbfs #any mount point can be used
mount -t hugetlbfs none /mnt/hugetlbfs/
```

This example will use 512 2M pages (2M is the default huge page size)

Start QEMU specifying the 2MB huge page pool as the file from which to allocate memory. In this example 512MB of memory is allocated to the VM:

64 bit ARMv8:

```
qemu-system-aarch64 -smp 8 -m 1024 -mem-path /mnt/hugetlbfs -cpu host -machine type=virt,gic-version=3 -kernel /boot/Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio
```

#### NOTE

On the GICv3 capable platforms the following emulated GIC controllers can be used:

- an emulated GICv3 interrupt controller can be used: `-machine type=virt,gic-version=3`
- an emulated GICv2 interrupt controller can be used: `-machine type=virt`

The ITS emulation is supported only with a GICv3 emulated interrupt controller.

On the GICv2 capable platforms only an emulated GICv2 interrupt controller can be used: `-machine type=virt`

32bit ARMv7:

```
qemu-system-arm -smp 2 -m 512 -mem-path /mnt/hugetlbfs -cpu host -machine type=virt -kernel /boot/zImage -enable-kvm -display none -serial tcp::4446,server,telnet -drive if=none,file=ubuntu_bionic_arm32_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio
```

**NOTE**

Make sure that the /mnt/hugetlbfs folder exists and is mounted when starting QEMU.

Explanation of the command line options:

- `-smp 2`: specifies the number of virtual CPUs.
- `-m 512`: the amount of memory for the VM
- `-mem-path /mnt/hugetlbfs`: allocates from hugetlbfs based memory
- `-cpu host`: the type of the CPU. In this case it is the same as the host CPU
- `-machine type=virt,gic-version=3`: the type of the virtual machine: virt machine + an GICv3 emulated interrupt controller
- `-machine type=virt`: the type of the virtual machine: virt machine + an GICv2 emulated interrupt controller
- `-kernel /boot/Image`: name of guest Linux kernel
- `-enable-kvm`: specifies that KVM should be used
- `-serial tcp::4446,server,telnet`: provide an emulated serial port (telnet server) on port 4446 on the host Linux system. Default behavior will be for QEMU to wait until the user connects to this port before booting the VM.
- `-drive if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-device,drive=foo`: creates a virtio based virtual disk (for details see [Virtual block devices](#) on page 546)
- `-append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk'`: guest Linux boot args
- `-display none`: do not display video output
- `-monitor stdio`: start QEMU monitor

At this point QEMU is waiting for a telnet connection to the virtual machine's console (port 4446 of the target board) prior to starting the virtual machine.

Connect to QEMU via telnet to start the virtual machine booting. In this example the target board has IP address 192.168.4.100.

```
-bash-3.2$ telnet 192.168.4.100 4446
Trying 192.168.4.100...
Connected to 192.168.4.100 (192.168.4.100).
Escape character is '^]'.
[0.000000] Booting Linux on physical CPU 0x0
[0.000000] Initializing cgroup subsys cpuset
[0.000000] Initializing cgroup subsys cpu
[0.000000] Initializing cgroup subsys cpuacct
.....
[OK] Reached target Multi-User System.
[OK] Reached target Graphical Interface.
 Starting Update UTMP about System Runlevel Changes...
[OK] Started Update UTMP about System Runlevel Changes.

Ubuntu 16.04.2 LTS localhost ttyAMA0

localhost login: root
Password:
Last login: Mon Jun 5 22:07:09 UTC 2017 on ttyAMA0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.65-00001-g6fed54f aarch64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
root@localhost:~#
```

### 10.1.3.3 How to Use Virtual Network Interfaces Using Virtio

As discussed in [Virtual network interfaces](#) on page 546, there are two aspects of virtual network interfaces-- 1) the "front end" (the device as seen by the guest OS) and 2) the "backend" (the means by the virtual device is connected to the network).

This example uses a "virtio" model NIC card and a tap network backend. The virtual network interface is bridged via a TAP interface to the physical network. The guest OS is Linux.

When starting QEMU we will add the following arguments to create the virtual network interface:

```
-netdev tap,id=tap0,script=/home/root/qemu-ifup,downscript=no,ifname="tap0" -device virtio-net-pci,netdev=tap0
```

Perform the following steps:

1. Enable virtio networking in the host and guest Linux kernels.
2. On the host Linux create a bridge to the physical network interface to be used by the virtual network interface in the virtual machine using the **brctl** command. In this example the physical interface being used is eth2:

```
brctl addbr br0
ifconfig br0 192.168.3.30 netmask 255.255.248.0
ifconfig eth2 0.0.0.0
brctl addif br0 eth2
```

3. Create a qemu-ifup script on the host Linux system. For the TAP backend type, when QEMU creates the virtual network interface it invokes a user-created script that allows customization of how the TAP interface is to be handled. The name of the TAP interface created by QEMU is passed as an argument. In this example we will bridge the the TAP interface to the bridge created in step #2. See the example qemu-ifup script below:

```
#!/bin/sh
TAP interface will be passed in $1
bridge=br0
guest_device=$1
ifconfig $guest_device 0.0.0.0 up
brctl addif $bridge $guest_device
```

4. When starting QEMU specify that the network device type is "virtio" and specify the path to the script created in step #3:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-device,drive=foo -netdev tap,id=tap0,script=qemu-ifup,downscript=no,ifname="tap0" -device virtio-net-pci,netdev=tap0 -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio
```

5. In the guest OS the virtual network interface will appear and can be brought up and assigned an IP address in the normal way. In the example below (the commands are run from the guest command shell) the virtio interface is eth0.

```
root@localhost:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: enp0s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
```

```

 link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1
 link/sit 0.0.0.0 brd 0.0.0.0
6: docker0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
 link/ether 02:42:a5:57:0b:85 brd ff:ff:ff:ff:ff:ff
 inet 172.17.0.1/16 scope global docker0
 valid_lft forever preferred_lft forever

root@localhost:~# ethtool -i enp0s1
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:00:01.0
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no

$ ifconfig enp0s1 192.168.3.31 netmask 255.255.248.0

```

### 10.1.3.4 How to use vhost-net with virtio

vhost-net is a character device that can be used to reduce the number of system calls involved in virtio networking. vhost-net moves network packets between the guest and the host system using the Linux kernel, bypassing QEMU.

In order to use vhost-net perform the following steps:

1. Enable virtio networking and vhost-net in the host and guest Linux kernels.
2. On the host Linux create a bridge to the physical network interface to be used by the virtual network interface in the virtual machine using the **brctl** command. In this example the physical interface being used is eth2:

```

brctl addbr br0
ifconfig br0 192.168.3.30 netmask 255.255.248.0
ifconfig eth2 0.0.0.0
brctl addif br0 eth2

```

3. Create a qemu-ifup script on the host Linux system. For the TAP backend type, when QEMU creates the virtual network interface it invokes a user-created script that allows customization of how the TAP interface is to be handled. The name of the TAP interface created by QEMU is passed as an argument. In this example we will bridge the the TAP interface to the bridge created in step #2. See the example qemu-ifup script below:

```

#!/bin/sh
TAP interface will be passed in $1
bridge=br0
guest_device=$1
ifconfig $guest_device 0.0.0.0 up
brctl addif $bridge $guest_device

```

4. When starting QEMU specify that the network device type is "virtio" and that vhost-net (**vhost=on** parameter) is used:

```

qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/
Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -netdev tap,id=tap0,script=qemu-ifup,downscript=no,ifname="tap0",vhost=on -

```

```
device virtio-net-pci,netdev=tap0 -append 'root=/dev/vda rw console=ttyAMA0 rootwait
earlyprintk' -monitor stdio
```

5. In the guest the virtual interface will come up as described in [How to Use Virtual Network Interfaces Using Virtio](#) on page 562. In the Host kernel the vhost thread can be seen consuming CPU:

```
2928 root 20 0 3258364 458340 19956 S 109.3 3.1 1:59.36 qemu-system-
aar
2944 root 20 0 0 0 0 R 99.7 0.0 1:43.52
vhost-2928
3020 root 20 0 225660 1224 1068 S 88.7 0.0 0:05.75 iperf
```

### 10.1.3.5 How to Use Virtual Disks Using Virtio

As discussed in [Virtual block devices](#) on page 546, there are a number of formats available for virtual disk images.

The example below uses a raw file. The steps below go through the process of creating a virtual disk image, assigning it to a VM, partitioning the disk, creating a filesystem on it, and mounting it.

1. On the host Linux, create a binary image to represent the guest disk. For example to create a 16MB disk:

```
$ dd if=/dev/zero of=my_guest_disk bs=4K count=4K
```

2. Start QEMU, specifying the name of the virtual disk file for the -drive argument:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/
Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -drive if=none,file=my_guest_disk,cache=none,id=user,format=raw -device virtio-
blk-pci,drive=user -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor
stdio
```

3. After the guest has booted the virtual disk is visible as a block device in /dev with the name vda, vdb, etc. In this example we have actually two virtual disks: one for the guest rootfs (vda) and one for my\_guest\_disk.

```
$ ls -l /dev/vdb
brw-rw---- 1 root disk 254, 0 Jan 1 1970 /dev/vdb
```

A virtual block device can be treated like any other hard disk. It can be partitioned, formatted, and mounted.

4. Configure a partition on the disk with fdisk:

```
root@localhost:~# fdisk /dev/vdb

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xc9820d64.

Command (m for help):
```

Display the partition table:

```
Command (m for help): p
Disk /dev/vdb: 16 MiB, 16777216 bytes, 32768 sectors
```



```
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc9820d64
```

```
Command (m for help):
```

#### Create a new partition:

```
Command (m for help): n
Partition type
 p primary (0 primary, 0 extended, 4 free)
 e extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-32767, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-32767, default 32767):
```

```
Created a new partition 1 of type 'Linux' and of size 15 MiB.
```

```
Command (m for help):
```

#### Display the new partition:

```
Command (m for help): p
Disk /dev/vdb: 16 MiB, 16777216 bytes, 32768 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc9820d64
```

| Device    | Boot | Start | End   | Sectors | Size | Id | Type  |
|-----------|------|-------|-------|---------|------|----|-------|
| /dev/vdb1 |      | 2048  | 32767 | 30720   | 15M  | 83 | Linux |

#### Write the partition table to disk and exit:

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

#### 5. Create a filesystem on the new partition:

```
root@localhost:~# mkfs.ext4 /dev/vdb1
mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 15360 1k blocks and 3840 inodes
Filesystem UUID: 8f0c49e4-2737-498e-a984-c5f05ba59b99
Superblock backups stored on blocks:
 8193

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

## 6. Mount the filesystem:

```
root@localhost:~# mount /dev/vdb1 /boot/
root@localhost:~# echo "A virtual disk" > /boot/test.txt
root@localhost:~# cat /boot/test.txt
A virtual disk
```

### 10.1.3.6 How to use virtual disks using virtio-blk-dataplane

Virtio-blk-dataplane was developed for high performance disk I/O, especially for high IOPS devices. The QEMU performs the disk I/O in a dedicated thread that is optimized for I/O performance.

In this example an sdcard is used, a block device on the Linux host.

#### 1. Start QEMU:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/
Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -object iothread,id=iothread0 -drive if=none,file=/dev/
mmcblk0,cache=none,id=drive0,format=raw,aio=native -device virtio-blk-
pci,drive=drive0,scsi=off,iothread=iothread0 -append 'root=/dev/vda rw console=ttyAMA0 rootwait
earlyprintk' -monitor stdio
```

#### 2. After the guest boots, the virtual disk is visible as a block device with the name vda, vdb, etc.

```
root@localhost:~# fdisk /dev/vdb

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): p
Disk /dev/vdb: 16 MiB, 16777216 bytes, 32768 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc9820d64

Device Boot Start End Sectors Size Id Type
/dev/vdb1 2048 32767 30720 15M 83 Linux
```

In this case the disk has 1 partition. The partition can be mounted and used.

### 10.1.3.7 How to use DPAA2 direct assignment without scripts

As presented in the introductory chapter [Direct assigned devices](#) on page 546, the DPAA2 architecture has the concept of MC containers which are arranged in a *tree structure*. While the root container always belongs to the host Linux, the child containers can be *directly assigned* to a user-space application such as DPDK or, as in our case, to a QEMU guest VM.

In the pursuit of creating a guest VM with one DPAA2 network interface directly assigned, we first need to create the child container and all the necessary MC objects.

In order to determine the number of DPAA2 objects needed to create a network interface [DPAA2 objects dependencies](#) on page 334. For our example the following rule applies:

- the DPIO number should be equal to the number of cores for the guest VM to be deployed ( for better performance)

- the DPCON number is equal to the number of cores multiplied by the number of interfaces
- one DPBP and DPMCP object for each network interface

The following section describes the steps to be followed in order to create a *single core VM with one DPAA2 network interface assigned*. The objects are created using the restool userspace program. For more details about the restool usage see [DPRCs and restool](#) on page 331 chapter.

## 1. Create and populate the child container

- Create the necessary MC objects
  - create the child container (this container will be assigned to the guest)

```
$ restool dprc create dprc.1
dprc.2 is created under dprc.1
```

- create the necessary objects in the child container

```
$ restool dpio create --container=dprc.2
dpio.11 is created under dprc.2
$ restool dpcon create --num-priorities=2 --container=dprc.2
dpcon.3 is created under dprc.2
$ restool dpmcp create --container=dprc.2
dpmcp.25 is created under dprc.2
$ restool dpbp create --container=dprc.2
dpbp.4 is created under dprc.2
$ restool dpni create --container=dprc.2
dpni.3 is created under dprc.2
```

- Change the plugged state of the newly created objects to *plugged*.

```
$ restool dprc assign dprc.2 --object=dpio.11 --plugged=1
$ restool dprc assign dprc.2 --object=dpcon.3 --plugged=1
$ restool dprc assign dprc.2 --object=dpmcp.25 --plugged=1
$ restool dprc assign dprc.2 --object=dpbp.4 --plugged=1
$ restool dprc assign dprc.2 --object=dpni.3 --plugged=1
```

- Check if objects were created properly by listing the contents of the child container:

```
$ restool dprc show dprc.2
dprc.2 contains 4 objects:
object label plugged-state
dpni.3 label plugged
dpbp.4 label plugged
dpmcp.25 label plugged
dpio.11 label plugged
dpcon.3 label plugged
```

- Connect the dpni object to the required dpmac in your scenario:

```
$ restool dprc connect dprc.1 --endpoint1=dpni.3 --endpoint2=dpmac.3
```

## 2. Bind the newly created DPRC device to the vfio-fsl-mc driver

```
$ echo vfio-fsl-mc > /sys/bus/fsl-mc/devices/dprc.2/driver_override
$ echo dprc.2 > /sys/bus/fsl-mc/drivers/vfio-fsl-mc/bind
```

3. Add the device command below (for the DPRC to be assigned) to the QEMU command line:

```
-device vfio-fsl-mc,host=dprc.2
```

Also, make sure to specify the appropriate number of cores for the guest VM. It should match the number of dpio objects created in the child container. In our case, 1 core.

```
-smp 1
```

4. Make sure to assign each vcpu thread to one physical CPU only

- Start QEMU with -S option (the vcpu threads are not yet started). We need this in order for the Ethernet drivers in the guest to correctly bind the objects to the cores.

```
qemu-system-aarch64 -smp 1 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/
Image -enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor
stdio -device vfio-fsl-mc,host=dprc.2 -S
```

Get the VM thread IDs entering QEMU shell

```
(qemu) info cpus
* CPU #0: thread_id=4952
```

- Assign one vcpu thread to one core only.

```
$ taskset -p 0x1 4952
pid 4952's current affinity mask: ff
pid 4952's new affinity mask: 1
```

- Start the vcpu threads.

```
(qemu) c
```

### 10.1.3.8 How to use DPAA2 direct assignment with scripts

While in the previous chapter, [How to use DPAA2 direct assignment without scripts](#) on page 566, we saw how to use the DPAA2 Direct Assignment feature manually, by creating each individual DPAA2 object needed in the child DPRC, in this chapter we will present a second method to create the desired configuration for a child container that will be *assigned* to the guest VM.

In order to describe the DPAA2 object configuration for a guest VM, thus a child DPRC, we employ the *DPL - Data Path Layout* syntax. The *restool* package has a new helper script, *ls-append-dpl*, that can parse DPL files which describe a child DPRC configuration and create that scenario using the *restool* tool.

One can check if the aforementioned script is available:

```
$ which ls-append-dpl

$ ls-append-dpl --help
Usage: /usr/local/bin/ls-append-dpl [options] <dpl-file>

Options:
-h, --help
 Print this help and exit
root@localhost:~#
```

The next section will describe how to use the *ls-append-dpl* script in order to create the child container that will be assigned to the guest VM. The next section will cover only the DPRC creation process, step #1 from the previous chapter, while the remaining steps are still the same.

### Single core guest with one network interface

Applying the [rule](#) presented before, we already know that in order to assign a network interface to a *single core guest* the child container should contain: 1 DPNI, 1 DPBP, 1 DPMCP, 1 DPIO and 1 DPCON.

- Create the DPL file

The file *vm\_1\_core.dts* is a text file that uses the DPL syntax and describes the required configuration for a child container that will be used for a *single core, one network interface* guest.

It has the exact same syntax as a DPL file used to describe the static host configuration. In the *vm\_1\_core.dts* file we can see that a *dprc* object is described:

```
dprc@2 {
 compatible = "fsl,dprc";
 parent = "dprc.1";
}
```

The *parent* property is mandatory and it should describe the parent container for the new one.

In this simple configuration, the single *dpni* created is connected to the *dpmac@1* in the *connections* section as follows:

```
connection@1{
 endpoint1 = "dpni@1";
 endpoint2 = "dpmac@1";
};
```

If you want to connect the *dpni@1* with any other object just change the value of *endpoint2*. For example, for a connection to be established with *dpmac@2* change the fragment to:

```
endpoint2 = "dpmac@2";
```

- Deploy the DPL configuration

```
$ ls-append-dpl vm_1_core.dts
Created the following objects:
dpmcp.50
dpni.1
dpio.8
dpcon.1
dprc.2
dpbp.1
```

### Multi core guest with one network interface

In order to transition from *1 core* guest to a *multi core* one, only the number of *dpio* and *dpcon* objects described in the DPL file need to be changed. Thus, in the case of a guest VM with 8 cores and one DPAA2 network interface, the DPL files should list and describe: 8 *dpio*, 8 *dpcon*, 1 *dpmcp*, 1 *dpbp*, 1 *dpni*.

The *vm\_8\_core.dts* describes the configuration required for a 8 core guest VM with one DPAA2 interface. You can use it in a similar fashion:

```
$ ls-append-dpl vm_8_core.dts
```

## ANNEX 1 - vm\_1\_core.dts

```
/dts-v1/;
/ {
 dpl-version = <10>;
 /*****
 * Containers
 *****/
 containers {

 dprc@2 {
 compatible = "fsl,dprc";
 parent = "dprc.1";
 options = "DPRC_CFG_OPT_SPAWN_ALLOWED", "DPRC_CFG_OPT_ALLOC_ALLOWED",
"DPRC_CFG_OPT_OBJ_CREATE_ALLOWED", "DPRC_CFG_OPT_IRQ_CFG_ALLOWED";

 objects {

 /* ----- DPBPs -----*/
 obj_set@dpbp {
 type = "dpbp";
 ids = <1>;
 };

 /* ----- DPCONS -----*/
 obj_set@dpcon {
 type = "dpcon";
 ids = <1>;
 };

 /* ----- DPIOs -----*/
 obj_set@dpio {
 type = "dpio";
 ids = <1>;
 };

 /* ----- DPMCPs -----*/
 obj_set@dpmcp {
 type = "dpmcp";
 ids = <1>;
 };

 /* ----- DPNIs -----*/
 obj_set@dpni {
 type = "dpni";
 ids = <1>;
 };
 };
 };
 };

 /*****
 * Objects
 *****/
 objects {

 dpbp@1 {
 compatible = "fsl,dpbp";
 };
 };
}
```

```

dpcon@1 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
};

dpio@1 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpmcp@1 {
 compatible = "fsl,dpmcp";
};

dpni@1 {
 compatible = "fsl,dpni";
 type = "DPNI_TYPE_NIC";
 options = "DPNI_OPT_NO_FS";
 num_queues = <8>;
 num_tcs = <1>;
 mac_filter_entries = <16>;
 vlan_filter_entries = <0>;
 fs_entries = <0>;
 qos_entries = <0>;
};

/*****
 * Connections
 *****/
connections {

 connection@1{
 endpoint1 = "dpni@1";
 endpoint2 = "dpmac@1";
 };
};

```

## ANNEX 2 - vm\_8\_core.dts

```

/dts-v1/;
/ {
 dpl-version = <10>;
 /*****
 * Containers
 *****/
 containers {

 dprc@2 {
 compatible = "fsl,dprc";
 parent = "dprc.1";
 options = "DPRC_CFG_OPT_SPAWN_ALLOWED", "DPRC_CFG_OPT_ALLOC_ALLOWED",
"DPRC_CFG_OPT_OBJ_CREATE_ALLOWED", "DPRC_CFG_OPT_IRQ_CFG_ALLOWED";

 objects {

 /* ----- DPBPs ----- */

```

```

 obj_set@dpbp {
 type = "dpbp";
 ids = <1>;
 };

 /* ----- DPCONS ----- */
 obj_set@dpcon {
 type = "dpcon";
 ids = <1 2 3 4 5 6 7 8>;
 };

 /* ----- DPIOs ----- */
 obj_set@dpio {
 type = "dpio";
 ids = <1 2 3 4 5 6 7 8>;
 };

 /* ----- DPMCPs ----- */
 obj_set@dpmcp {
 type = "dpmcp";
 ids = <1>;
 };

 /* ----- DPNIs ----- */
 obj_set@dpni {
 type = "dpni";
 ids = <1>;
 };
};

};

/*****
 * Objects
 *****/
objects {

 dpbp@1 {
 compatible = "fsl,dpbp";
 };

 dpcon@1 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
 };

 dpcon@2 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
 };

 dpcon@3 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
 };

 dpcon@4 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
 };
};

```



```
dpcon@5 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
};

dpcon@6 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
};

dpcon@7 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
};

dpcon@8 {
 compatible = "fsl,dpcon";
 num_priorities = <0x2>;
};

dpio@1 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@2 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@3 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@4 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@5 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@6 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
};

dpio@7 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
```

```

 num_priorities = <0x8>;
 };

 dpio@8 {
 compatible = "fsl,dpio";
 channel_mode = "DPIO_LOCAL_CHANNEL";
 num_priorities = <0x8>;
 };

 dpmcp@1 {
 compatible = "fsl,dpmcp";
 };

 dpni@1 {
 compatible = "fsl,dpni";
 type = "DPNI_TYPE_NIC";
 options = "DPNI_OPT_NO_FS";
 num_queues = <8>;
 num_tcs = <1>;
 mac_filter_entries = <16>;
 vlan_filter_entries = <0>;
 fs_entries = <0>;
 qos_entries = <0>;
 };
};

/*****
 * Connections
 *****/
connections {

 connection@1{
 endpoint1 = "dpni@1";
 endpoint2 = "dpmac@1";
 };
};
};

```

### 10.1.3.9 How to use PCIe direct assignment

Select the PCIe device that will be assigned to Virtual Machine. For example, it is e1000e PCI network device (0000.01.00.0).

#### 1. Bind the PCI device to the VFIO driver:

- Assume e1000e device with identity 0000.01.00.0

```

echo vfio-pci > /sys/bus/pci/devices/0000\:01\:00.0/driver_override
echo 0000:01:00.0 > /sys/bus/pci/drivers/e1000e/unbind
echo 0000:01:00.0 > /sys/bus/pci/drivers/vfio-pci/bind

```

#### 2. All device in the iommu-group must be assigned to same virtual machine.

- The command below will list all devices in the same iommu-group:

```

ls -l /sys/bus/pci/devices/0000:06:0d.0/iommu_group/devices

```

- All devices must be bound to VFIO using step (1) above.

3. Add the device command below to the QEMU command line for all devices in the `iommu-group`:

```
-device vfio-pci,host=0000:01:00.0
```

4. Device will be available in Virtual Machine.

This feature is enabled for LS1088 and LS2088 devices only.

### 10.1.3.10 Passthrough of USB Devices

USB devices can be assigned to virtual machines. When the device is assigned to the virtual machine it becomes the private resource of the VM and it cannot be used by the host Linux. The virtual machine sees a XHCI USB controller on its PCI bus. The XHCI controller supports USB 3.0 devices.

There are 2 approaches for passing through a USB device:

1. by specifying the USB vendor ID and product ID of the device
2. by specifying the USB bus and port number

In the examples below, the **-device nec-usb-xhci** argument specifies that a PCI-based XHCI USB controller should be added to the PCI bus. The **-device usb-host** identifies the specific USB device being passed through.

To assign the device by vendor and product ID, first identify the device using the `lsusb` command. For example:

```
root@localhost:~# lsusb
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 13fe:3600 Kingston Technology Company Inc. flash drive (4GB, EMTEC)
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

To assign the Kingston USB disk, specify the following **-device** arguments to QEMU:

```
-device nec-usb-xhci,id=xhci
-device usb-host,bus=xhci.0,vendorid=0x13fe,productid=0x3600
```

To assign the device by USB bus and host number, use the `lsusb` command:

```
root@localhost:~# lsusb -t
/: Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/lp, 5000M
/: Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/lp, 480M
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/lp, 5000M
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/lp, 480M
 |__ Port 1: Dev 2, If 0, Class=Mass Storage, Driver=usb-storage, 480M
```

In this example, the storage device can be seen on bus 1, port 1. The info `usbhost` in the QEMU monitor can also be used to display the host USB bus and port numbers for all USB devices.

To assign the Kingston USB disk by bus and port number, specify the following **-device** arguments to QEMU:

```
-device nec-usb-xhci,id=xhci
-device usb-host,bus=xhci.0,hostbus=1,hostport=1
```

### 10.1.3.11 Debugging: How to Examine Initial Virtual Machine State with QEMU

It can be helpful when debugging to examine the state of the virtual machine prior to executing the first instruction of the guest OS.

To do this, start QEMU with the -S option.

Example:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/Image
-enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio -S
```

The console was started with the "-serial tcp::4446,server,telnet" option so QEMU waits for a connection prior to starting initialization. Use telnet to connect to port 4446 of the target.

At this point QEMU initializes the VM, but does not execute the entry point to the guest OS. The monitor prompt can now be used to examine initial state:

```
QEMU 2.5.0 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: disconnected:telnet::4446,server

(qemu)
```

To see where boot images are loaded and placed by QEMU use the **info roms** command:

```
(qemu) info roms
addr=0000000000000000 size=0x000038 mem=ram name="smpboot"
addr=0000000040000000 size=0x000028 mem=ram name="bootloader"
addr=0000000040080000 size=0xf0aa00 mem=ram name="/boot/Image"
addr=0000000048000000 size=0x010000 mem=ram name="dtb"
/rom@etc/acpi/tables size=0x200000 name="etc/acpi/tables"
/rom@etc/table-loader size=0x000880 name="etc/table-loader"
/rom@etc/acpi/rsdp size=0x000024 name="etc/acpi/rsdp"
(qemu)
```

A trivial bootloader is loaded at the start of guest memory at 0x40000000

The kernel image (zImage) is loaded at 0x40080000.

To examine the initial state of registers use the **info registers** command:

```
(qemu) info registers
PC=0000000040000000 SP=0000000000000000
X00=0000000000000000 X01=0000000000000000 X02=0000000000000000 X03=0000000000000000
X04=0000000000000000 X05=0000000000000000 X06=0000000000000000 X07=0000000000000000
X08=0000000000000000 X09=0000000000000000 X10=0000000000000000 X11=0000000000000000
X12=0000000000000000 X13=0000000000000000 X14=0000000000000000 X15=0000000000000000
X16=0000000000000000 X17=0000000000000000 X18=0000000000000000 X19=0000000000000000
X20=0000000000000000 X21=0000000000000000 X22=0000000000000000 X23=0000000000000000
X24=0000000000000000 X25=0000000000000000 X26=0000000000000000 X27=0000000000000000
X28=0000000000000000 X29=0000000000000000 X30=0000000000000000
PSTATE=400003c5 -Z-- EL1h
q00=0000000000000000:0000000000000000 q01=0000000000000000:0000000000000000
q02=0000000000000000:0000000000000000 q03=0000000000000000:0000000000000000
q04=0000000000000000:0000000000000000 q05=0000000000000000:0000000000000000
q06=0000000000000000:0000000000000000 q07=0000000000000000:0000000000000000
q08=0000000000000000:0000000000000000 q09=0000000000000000:0000000000000000
q10=0000000000000000:0000000000000000 q11=0000000000000000:0000000000000000
q12=0000000000000000:0000000000000000 q13=0000000000000000:0000000000000000
q14=0000000000000000:0000000000000000 q15=0000000000000000:0000000000000000
q16=0000000000000000:0000000000000000 q17=0000000000000000:0000000000000000
q18=0000000000000000:0000000000000000 q19=0000000000000000:0000000000000000
```

```
q20=0000000000000000:0000000000000000 q21=0000000000000000:0000000000000000
q22=0000000000000000:0000000000000000 q23=0000000000000000:0000000000000000
q24=0000000000000000:0000000000000000 q25=0000000000000000:0000000000000000
q26=0000000000000000:0000000000000000 q27=0000000000000000:0000000000000000
q28=0000000000000000:0000000000000000 q29=0000000000000000:0000000000000000
q30=0000000000000000:0000000000000000 q31=0000000000000000:0000000000000000
FPCR: 00000000 FPSR: 00000000
(qemu)
```

The program counter is set to 0x40000000 which is the effective address of the entry point of the kernel.

### 10.1.3.12 Debugging: How to Profile Virtualization Overhead with KVM

Running software in a virtual machine can cause additional overhead that affects performance. The virtualization overhead is directly related to the number of times the hypervisor (KVM) is invoked to handle exception conditions that may occur in the virtual machine. These exception handling events are referred to as 'exits', because guest context is exited.

Examples of exits include things such the guest executing a privileged instruction, access a privileged CPU register, accessing a virtual I/O device, or a hardware interrupt such as a decremented interrupt.

The type and number of exits that occur is workload dependent.

KVM implements a mechanism in which different events are logged. These events are actually tracepoint events, and perf nicely integrates with them. You have to compile the host kernel with the following options:

```
Kernel hacking --->
 [*] Tracers --->
 [*] Trace process context switches and events
```

#### Counting Events

A count of a subset of KVM events that occur can be seen under debugfs. To see this first mount debugfs:

```
mount -t debugfs none /sys/kernel/debug
```

The statistics can be seen using perf tool:

```
perf stat -e "kvm:*" -p 1395
^C
Performance counter stats for process id '1395':

 5678 kvm:kvm_entry
 5678 kvm:kvm_exit
 3121 kvm:kvm_guest_fault
 2278 kvm:kvm_irq_line
 0 kvm:kvm_mmio_emulate
 0 kvm:kvm_emulate_cp15_imp
 2438 kvm:kvm_wfi
 0 kvm:kvm_unmap_hva
 2 kvm:kvm_unmap_hva_range
 0 kvm:kvm_set_spte_hva
 0 kvm:kvm_hvc
 3119 kvm:kvm_userspace_exit
 0 kvm:kvm_set_irq
 0 kvm:kvm_ack_irq
 4068 kvm:kvm_mmio
```

```
0 kvm:kvm_fpu
0 kvm:kvm_age_page

59.316709040 seconds time elapsed
```

## Tracing events

Detailed traced can be generated using ftrace:

```
[enable ftrace in kernel: events and system calls]
$echo 1 > /sys/kernel/debug/tracing/events/kvm/enable
$cat /sys/kernel/debug/tracing/trace_pipe

qemu-system-arm-1366 [000] ... 716.115891: kvm_guest_fault: ipa 0x9000000, hsr 0x93430046,
hxfar 0xa084c030, pc 0x80266a9c
qemu-system-arm-1366 [000] ... 716.115892: kvm_mmio: mmio write len 2 gpa 0x9000030 val 0xf01
qemu-system-arm-1366 [000] ... 716.115895: kvm_userspace_exit: reason KVM_EXIT_MMIO (6)
qemu-system-arm-1366 [000] d... 716.115907: kvm_entry: PC: 0x80266aa0
qemu-system-arm-1366 [000] d... 716.116234: kvm_exit: PC: 0x800cf508
qemu-system-arm-1366 [000] d... 716.118274: kvm_entry: PC: 0x800cf508
qemu-system-arm-1366 [000] d... 716.118704: kvm_exit: PC: 0x0000981c
qemu-system-arm-1366 [000] d... 716.120737: kvm_entry: PC: 0x0000981c
qemu-system-arm-1366 [000] d... 716.121159: kvm_exit: PC: 0x800bb104
qemu-system-arm-1366 [000] d... 716.123197: kvm_entry: PC: 0x800bb104
qemu-system-arm-1366 [000] d... 716.123620: kvm_exit: PC: 0x8009cae0
qemu-system-arm-1366 [000] d... 716.125696: kvm_entry: PC: 0x8009cae0
qemu-system-arm-1366 [000] d... 716.126091: kvm_exit: PC: 0x800c90f4
qemu-system-arm-1366 [000] d... 716.128130: kvm_entry: PC: 0x800c90f4
qemu-system-arm-1366 [000] d... 716.128561: kvm_exit: PC: 0x801f37f4
qemu-system-arm-1366 [000] d... 716.130594: kvm_entry: PC: 0x801f37f4
qemu-system-arm-1366 [000] d... 716.130623: kvm_exit: PC: 0x8020576c
qemu-system-arm-1366 [000] d... 716.130635: kvm_entry: PC: 0x8020576c
qemu-system-arm-1366 [000] d... 716.131018: kvm_exit: PC: 0x43014750
qemu-system-arm-1366 [000] d... 716.133053: kvm_entry: PC: 0x43014750
qemu-system-arm-1366 [000] d... 716.133478: kvm_exit: PC: 0x80205778
qemu-system-arm-1366 [000] d... 716.135555: kvm_entry: PC: 0x80205778
```

## 10.1.3.13 Debugging virtual machines

### 10.1.3.13.1 QEMU Monitor

When starting QEMU, a monitor shell is available that can be used to control and see the state of VM. By default this monitor is started in the Linux shell where QEMU is invoked.

See example below of the output when starting QEMU. The user can interact with the monitor at the (qemu) prompt.

```
QEMU 2.5.0 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: disconnected:telnet::4446,server
```

The monitor can also be exposed over a network port by using the `-monitor dev` command line option. See [Overview of Using QEMU](#) on page 543 and the QEMU user's manual [1] (see [References](#) on page 551).

Refer to the QEMU user's manual [1] for a complete listing of the monitor commands available. Below is a list of some useful commands supported in the NXP SDK implementation of QEMU:

- **help** - lists all the available commands with usage information
- **info cpus** - displays the state and thread ID of all virtual CPUs
- **info registers** - displays the contents of the default vcpu's registers

- **cpu cpu\_number** - sets the default vcpu number
- **system\_reset** - resets the VM
- **x/fmt addr** -- virtual memory dump starting at 'addr'
- **xp/fmt addr** -- physical memory dump starting at 'addr'

### 10.1.3.13.2 QEMU GDB Stub

QEMU supports debugging of a VM using gdb. QEMU contains a gdb stub that can be attached to from a host system and allows standard source level debugging capabilities to examine the state of the VM and do run control.

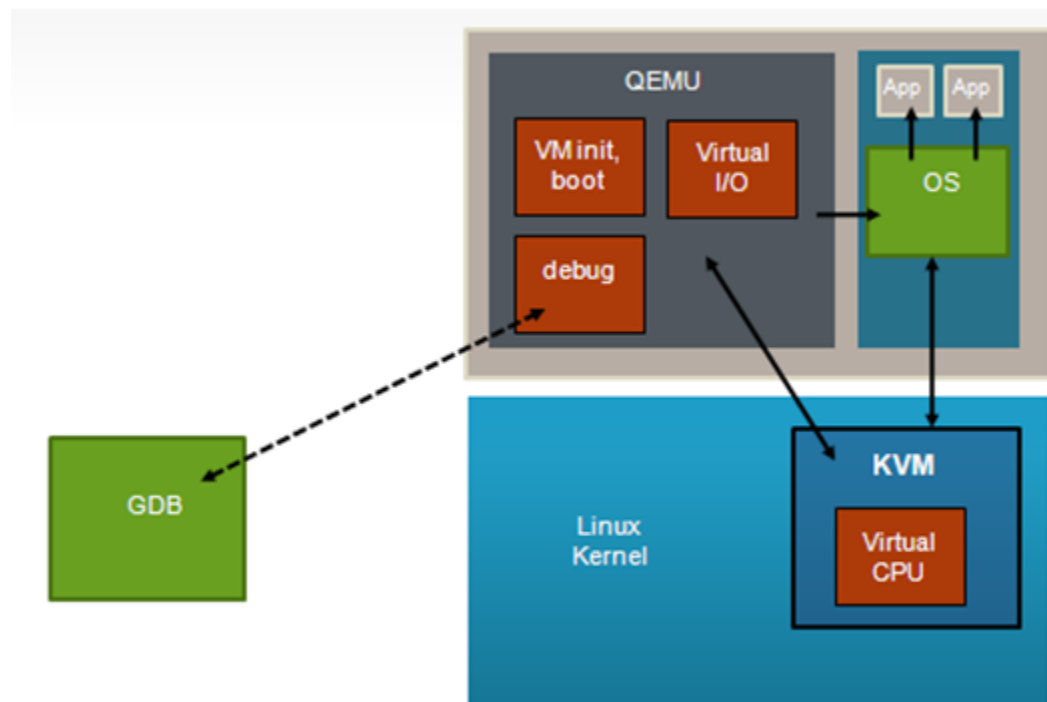


Figure 67.

To use the gdb stub, start QEMU with the `-gdb dev` option where `dev` specifies the type of connection to be used. See the QEMU user's manual [1] (see [References](#) on page 551) for details.

One useful option when debugging is the `-S` argument to QEMU which causes QEMU to wait to start the first instruction of the guest until told to start using the monitor (**continue** command).

In the example below the `tcp` device type is used. A gdb stub will be active on port 4445 of the host Linux kernel when starting QEMU:

```
qemu-system-aarch64 -smp 8 -m 1024 -cpu host -machine type=virt,gic-version=3 -kernel /boot/Image
-enable-kvm -display none -serial tcp::4446,server,telnet -drive
if=none,file=ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -device virtio-blk-
device,drive=foo -append 'root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk' -monitor stdio -
gdb tcp::4444
```

After the guest has been started normally, gdb can be used to connect to the VM (in this example the host kernel has an ip address of 192.168.3.30):

```
(gdb) target remote 192.168.4.100:4444
Remote debugging using 192.168.4.100:4444
0xffff000008096258 in ?? ()
```

Debugging with gdb can then proceed normally:

```
(gdb) p/x $pc
$4 = 0xffff000008096258
```

## 10.2 Linux Containers (LXC) for NXP QorIQ User's Guide

### 10.2.1 Introduction to Linux Containers

#### 10.2.1.1 NXP LXC Release Notes

This document describes current limitations in the release of LXC for NXP SoCs.

Copyright (C) 2017 NXP Semiconductors, Inc.

NXP LXC Release Notes  
04/27/2016

#### Overview

-----

This document describes new features, current limitations, and working demos in Linux Containers (LXC) for QorIQ Layerscape SDK.

#### Fixes

-----

- o Seccomp support on ARMv8 platforms.
- o Unprivileged containers support on ARMv8 platforms.

#### SDK Demo List

-----

- o Basic container usage flow and management commands
- o Container networking setups
  - o Shared networking
  - o Private NICs
  - o Ethernet bridge
  - o MACVLAN
  - o VLAN
- o Adjusting container capabilities
- o Tuning container resource usage
- o Running application containers
- o Isolating USDPAA applications in LXC containers. This has been tested using the USDPAA reflector app in a Multiple Instance Scenario on a DPAA board. After partitioning the board resources in order to support multiple reflector instances, these have been further isolated in container environments.



- o Running an unprivileged container linked to a host bridge.
- o Running containers with Seccomp protection.

## 10.2.1.2 Overview

This document is a guide and tutorial to using Linux Containers on NXP ARMv7 and ARMv8-based SoCs.

Linux Containers is a lightweight virtualization technology that allows the creation of environments in Linux called "**containers**" in which Linux applications can be run in isolation from the rest of the system and with fine grained control over resources allocated to the container (e.g. CPU, memory, network).

There are 2 implementations of containers in the LSDK:

- **LXC.** LXC is a user space package that provides a set of commands to create and manage containers and uses existing Linux kernel features to accomplish the desired isolation and control.
- **Libvirt.** The libvirt package is a virtualization toolkit that provides a set of management tools for managing virtual machines and Linux containers. The libvirt driver for containers is called "lxc", but the libvirt "lxc" driver is distinct from the user space LXC package.

Applications in a container run in a "sandbox" and can be restricted in what they can do and what visibility they have. In a container:

- An application "sees" only other processes that are in the container.
- An application has access only to network resources granted to the container.
- If configured as such, an application "sees" only a container-specific root filesystem. In addition to limiting access to data in the system's host rootfs, by limiting the `/dev` entries that exist in the containers rootfs this limits the devices that the container can access.
- The file POSIX capabilities available to programs are controlled and configured by the system administrator.
- The container's processes run in what is known as a "control group" which the system administrator can use to monitor and control the container's resources.

Why are containers useful? Below are a few examples of container use cases:

- **Application partitioning** -- control CPU utilization between high priority and low priority applications, control what resources applications can access.
- **Virtual private server** -- boot multiple instances of user space, each which effectively looks like a private instance of a server. This approach is commonly used in website infrastructure.
- **Software upgrade** -- run Linux user space in a container, when it becomes necessary to upgrade applications in the system, create and test upgraded software in a new container. The old container can be stopped and the new container can be started as desired.
- **Terminal servers** -- user accesses the system with a thin client, with containers on the server providing applications. Each user gets a private, sandboxed workspace.

There are two general usage models for containers:

- **application containers:** Running a single application in a container. In this scenario, a single executable program is started in the container.
- **system containers:** Booting an instance of user space in a container. Booting multiple system containers allows multiple isolated instances of user space to run at the same time.

Containers are conceptually different than virtual machine technologies such as QEMU/KVM. Virtual machines emulate a hardware platform and are capable of booting an operating system kernel. A container is a mechanism to isolate Linux applications. In a system using containers there is only one Linux kernel running -- the host Linux kernel.

### 10.2.1.3 Comparing LXC and Libvirt

LXC and the lxc driver in libvirt provide similar capabilities and use the same kernel mechanisms to create containers. This section highlights some of the differences between the two tools.

#### LXC

- Container management is done with local LXC package commands. No remote support.
- Container creation done with **lxc-create**. LXC config file and template govern the creation of the template and the container's rootfs.

#### libvirt

- libvirt abstracts the container and thus a variety of tools can be used to manage containers.
- Remote management is supported.
- Container configuration defined in libvirt XML file.
- No tools to facilitate container creation.
- Same tools can be used to manage containers and KVM/QEMU virtual machines.

### 10.2.1.4 For Further Information

Linux containers is an approach to virtualization similar to OS virtualization solutions such as Linux VServer and OpenVZ that are widely used for virtual private servers. Documentation for these projects has helpful and relevant information:

- <http://linux-vserver.org/Overview>
- [http://wiki.openvz.org/Main\\_Page](http://wiki.openvz.org/Main_Page)

The LXC package is an open source project and much information is available online.

#### General Information

- libvirt LXC driver: <http://libvirt.org/drvlxc.html>
- Getting started with LXC using libvirt : <https://www.berrange.com/posts/2011/09/27/getting-started-with-lxc-using-libvirt/>
- LXC: Official web page for the LXC project: <https://linuxcontainers.org/>
- LXC: Overview article on LXC on IBM developerWorks (2009): <http://www.ibm.com/developerworks/linux/library/l-lxc-containers/>
- LXC manpages: <https://linuxcontainers.org/lxc/manpages/>
- Article on POSIX file capabilities: <http://www.friedhoff.org/posixfilecaps.html>
- SUSE LXC tutorial: [https://www.suse.com/documentation/sles11/singlehtml/lxc\\_quickstart/lxc\\_quickstart.html](https://www.suse.com/documentation/sles11/singlehtml/lxc_quickstart/lxc_quickstart.html)
- LXC Linux Containers, presentation: <http://www.slideshare.net/samof76/lxc-17456998>
- Stephane Graber's LXC 1.0 blog posts: <https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/>
- Linux Plumbers 2013 videos: <https://www.youtube.com/channel/UCIxmRWj3-795FMlrsikd3A/videos>
- Control Groups: <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>

#### Containers and Security

If using containers to sandbox untrusted applications, a thorough understanding is needed of the capabilities granted to a container and the security vulnerabilities they may imply. The following references are helpful for understanding container security:

- Ubuntu's security issues and mitigations with LXC, <https://wiki.ubuntu.com/LxcSecurity>

- Emeric Nasi, Exploiting capabilities, [http://packetstorm.io/papers/attack/exploiting\\_capabilities\\_the\\_dark\\_side.pdf](http://packetstorm.io/papers/attack/exploiting_capabilities_the_dark_side.pdf)
- Secure containers with SELinux and Smack, <http://www.ibm.com/developerworks/linux/library/l-lxc-security/index.html>
- Seccomp and sandboxing, <http://lwn.net/Articles/332974/>

### Mailing Lists

For LXC, there are two mailing lists available which can be subscribed to. Archives of the lists are also available.

<https://lists.linuxcontainers.org/listinfo/lxc-devel>

<https://lists.linuxcontainers.org/listinfo/lxc-users>

## 10.2.2 More Details

### 10.2.2.1 LXC: Command Reference

This section contains links to available open source documentation for the commands in the LXC user space package.

Table 61.

| LXC man page    | Description                                                         | Man Page Link              |
|-----------------|---------------------------------------------------------------------|----------------------------|
| lxc             | lxc overview                                                        | <a href="#">click here</a> |
| lxc-attach      | start a process inside a running container                          | <a href="#">click here</a> |
| lxc-autostart   | start/stop/kill auto-started containers                             | <a href="#">click here</a> |
| lxc-cgroup      | manage the control group associated with a container                | <a href="#">click here</a> |
| lxc-checkconfig | check the current kernel for lxc support                            | <a href="#">click here</a> |
| lxc-clone       | clone a new container from an existing one                          | <a href="#">click here</a> |
| lxc-config      | query LXC system configuration                                      | <a href="#">click here</a> |
| lxc.conf        | a description of all configuration options available                | <a href="#">click here</a> |
| lxc-console     | launch a console for the specified container                        | <a href="#">click here</a> |
| lxc-create      | creates a container                                                 | <a href="#">click here</a> |
| lxc-destroy     | destroy a container previously created with lxc-create              | <a href="#">click here</a> |
| lxc-execute     | run the specified command inside a container                        | <a href="#">click here</a> |
| lxc-freeze      | freeze (suspend) all the container's processes                      | <a href="#">click here</a> |
| lxc-info        | query information about a container                                 | <a href="#">click here</a> |
| lxc-ls          | list the containers existing on the system                          | <a href="#">click here</a> |
| lxc-monitor     | monitor the container state                                         | <a href="#">click here</a> |
| lxc-snapshot    | snapshot an existing container                                      | <a href="#">click here</a> |
| lxc-start       | starts a container previously created with lxc-create               | <a href="#">click here</a> |
| lxc-stop        | stop a container                                                    | <a href="#">click here</a> |
| lxc-unfreeze    | resumes a containers processes suspended previously with lxc-freeze | <a href="#">click here</a> |

Table continues on the next page...

Table 61. (continued)

| LXC man page | Description                              | Man Page Link              |
|--------------|------------------------------------------|----------------------------|
| lxc-unshare  | run a task in a new set of namespaces    | <a href="#">click here</a> |
| lxc-userns   | run task as root in a new user namespace | <a href="#">click here</a> |
| lxc-wait     | wait for a specific container state      | <a href="#">click here</a> |

The following LXC commands are not supported:

- lxc-userns

## 10.2.2.2 LXC: Configuration Files

### NOTE

This section is applicable to LXC only, not to libvirt.

For LXC, configuration files are used to configure aspects of a container at the time it is created. The configuration file defines what resources are private to the container and what is shared. By default the following resources are private to a container:

- process IDs
- sysv ipc mechanisms
- mount points

This means for example, that by default the container will share network resources and the filesystem with the host system, but will have its own private process IDs.

The container configuration file allows additional isolation to be specified through configuration in the following areas:

- network
- console
- mount points and the backing store for the root filesystem
- control groups (cgroups)
- POSIX capabilities

See the <http://man7.org/linux/man-pages/man5/lxc.conf.5.html> for details on each configuration option.

When a container is created a new directory with the container's name is created in /var/lib/lxc. The configuration file for the container is stored in:

```
/var/lib/lxc/[container-name]/config
```

Below is an example of the contents of a minimal configuration file for a container named "foo", which has no networking:

```
$ cat /var/lib/lxc/foo/config
Container with non-virtualized network
lxc.utsname = foo
lxc.tty = 1
lxc.pts = 1
lxc.rootfs = /var/lib/lxc/foo/rootfs
lxc.mount.entry=/lib /var/lib/lxc/foo/rootfs/lib none ro,bind 0 0
lxc.mount.entry=/usr/lib /var/lib/lxc/foo/rootfs/usr/lib none ro,bind 0 0
```

See the [LXC: Getting Started \(with a Busybox System Container\)](#) on page 589 how-to article for an introduction to the container lifecycle and how configuration files are used when creating containers.

Several example configuration files are provided with LXC:

```
/usr/share/doc/lxc-common/examples/lxc-complex.conf
/usr/share/doc/lxc-common/examples/lxc-empty-netns.conf
/usr/share/doc/lxc-common/examples/lxc-macvlan.conf
/usr/share/doc/lxc-common/examples/lxc-no-netns.conf
/usr/share/doc/lxc-common/examples/lxc-phys.conf
/usr/share/doc/lxc-common/examples/lxc-veth.conf
/usr/share/doc/lxc-common/examples/lxc-vlan.conf
/usr/share/doc/lxc-common/examples/seccomp-v1.conf
/usr/share/doc/lxc-common/examples/seccomp-v2-blacklist.conf
/usr/share/doc/lxc-common/examples/seccomp-v2.conf
```

### 10.2.2.3 LXC: Templates

#### NOTE

This section is applicable to LXC only, not to libvirt.

For LXC, When a container is "created" a directory for the container (which has the same name as the container) is created under `/var/lib/lxc`. This is where the container's configuration file is stored and can be edited.

For system containers (containers created with **lxc-create**), the default is for the root filesystem structure of the container to be stored here as well.

Creating containers is simplified by the use of example "templates" provided with the LXC. Template examples are provided for a number of different Linux distributions. A template is a script invoked by **lxc-create** that creates the root filesystem structure and sets up the container's config file.

The following example templates are provided with LXC and can be referred to for the expected template structure:

```
/usr/share/lxc/templates/lxc-alpine
/usr/share/lxc/templates/lxc-altlinux
/usr/share/lxc/templates/lxc-archlinux
/usr/share/lxc/templates/lxc-busybox
/usr/share/lxc/templates/lxc-centos
/usr/share/lxc/templates/lxc-cirros
/usr/share/lxc/templates/lxc-debian
/usr/share/lxc/templates/lxc-download
/usr/share/lxc/templates/lxc-fedora
/usr/share/lxc/templates/lxc-gentoo
/usr/share/lxc/templates/lxc-openmandriva
/usr/share/lxc/templates/lxc-opensuse
/usr/share/lxc/templates/lxc-oracle
/usr/share/lxc/templates/lxc-plamo
/usr/share/lxc/templates/lxc-sshd
/usr/share/lxc/templates/lxc-ubuntu
/usr/share/lxc/templates/lxc-ubuntu-cloud
```

For the NXP LSDK the busybox template is recommended and has been tested with flex-builder created root filesystems.

The how-to examples provided in this user guide that create system containers use the busybox template.

### 10.2.2.4 Containers with Libvirt

This section provides an overview to using libvirt-based containers.

For an general introduction to libvirt, please see the container information available on the libvirt website: <http://libvirt.org/drvlxc.html>.

With libvirt, a container "domain" is specified in an XML file. The XML is used to "define" the container, which then allows the container to be managed with the standard libvirt domain lifecycle.

### Libvirt XML

The XML for the simplest functional container would look like the example below:

```
<domain type='lxc'>
 <name>container1</name>
 <memory>500000</memory>
 <os>
 <type>exe</type>
 <init>/bin/sh</init>
 </os>
 <devices>
 <console type='pty' />
 </devices>
</domain>
```

Refer to the XML reference information available on the libvirt website for detailed reference information: <http://libvirt.org/formatdomain.html>

The <domain> element must specify a type attribute of "lxc" for a container/lxc domain. There are 4 additional sub-nodes required:

- <name> - specifies the name of the container
- <memory> - specifies the maximum memory the container may use
- <os> - identifies the initial program to run. In the example this is /bin/sh. For an application based container this is the name of the application. If booting an instance of Linux user space this would typically be /sbin/init.
- <devices> - specifies any devices, in the above example there is just a console

### Filesystem mounts (from <http://libvirt.org/drvlxc.html>)

In the absence of any explicit configuration, the container will inherit the host OS filesystem mounts. A number of mount points will be made read only, or re-mounted with new instances to provide container specific data. The following special mounts are setup by libvirt:

- /dev a new "tmpfs" pre-populated with authorized device nodes
- /dev/pts a new private "devpts" instance for console devices
- /sys the host "sysfs" instance remounted read-only
- /proc a new instance of the "proc" filesystem
- /proc/sys the host "/proc/sys" bind-mounted read-only
- /sys/fs/selinux the host "selinux" instance remounted read-only
- /sys/fs/cgroup/NNNN the host cgroups controllers bind-mounted to only expose the sub-tree associated with the container
- /proc/meminfo a FUSE backed file reflecting memory limits of the container

Additional filesystem mounts can be created using the <filesystem> node under the <devices> node. See the libvirt.org documentation referenced above for further details.

### Device nodes from <http://libvirt.org/drvlxc.html>

The container init process will be started with CAP\_MKNOD capability removed and blocked from re-acquiring it. As such it will not be able to create any device nodes in /dev or anywhere else in its filesystems. Libvirt itself will take care of pre-

populating the /dev filesystem with any devices that the container is authorized to use. The current devices that will be made available to all containers are:

- /dev/zero
- /dev/null
- /dev/full
- /dev/random
- /dev/urandom
- /dev/stdin symlinked to /proc/self/fd/0
- /dev/stdout symlinked to /proc/self/fd/1
- /dev/stderr symlinked to /proc/self/fd/2
- /dev/fd symlinked to /proc/self/fd
- /dev/ptmx symlinked to /dev/pts/ptmx
- /dev/console symlinked to /dev/pts/0

### 10.2.2.5 Linux Control Groups (cgroups)

Linux control groups (or cgroups) is a feature of the Linux kernel that allows the allocation, prioritization, control, and monitoring of resources such as CPU time, memory, network bandwidth among groups of Linux processes.

Cgroups is one of the underlying Linux kernel features that LXC is built upon. LXC automatically creates a cgroup for each container when it is started. A pre-requisite for using LXC is mounting the cgroup virtual filesystem.

Cgroups encompass a number of different subsystems or "controllers" that are used for managing and controlling different resources. The following subsystems/controllers are supported:

- cpu - controls CPU allocation for tasks in a cgroup;
- cpuset - assigns individual CPUs and memory nodes to tasks in a cgroup;
- cpuacct - generates automatic reports on CPU resources used by the tasks in a cgroup;
- memory - isolates the memory behavior of a group of tasks from the rest of the system;
- devices - allows or denies access to devices by tasks in a cgroup;
- freezer - suspends or resumes tasks in a cgroup;
- net\_cls - tags packets with a class identifier that allows the Linux traffic controller to identify packets originating from a particular cgroup;
- net\_prio - provides a way to dynamically set the priority of network traffic per each network interface for applications within various cgroups;
- blkio - controls and monitors access to I/O on block devices by tasks in cgroups.

Check out the Red Hat documentation on cgroups here: [https://access.redhat.com/site/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html/Resource\\_Management\\_Guide/ch-Subsystems\\_and\\_Tunable\\_Parameters.html](https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch-Subsystems_and_Tunable_Parameters.html).

Cgroup subsystems can be configured within the configuration file used when creating a container. The configuration file accepts cgroup configuration in the following form:

```
lxc.cgroup.[subsystem name] = <value>
```

See the <http://man7.org/linux/man-pages/man5/lxc.conf.5.html> for further details.

Cgroup subsystems can also be displayed or updated while a container is running using the **lxc-cgroup** command:

```
lxc-cgroup -n [container-name] [cgroup-subsystem] [value]
```

For some examples of how to use cgroups to control container configuration, see the article: [LXC: How to use cgroups to manage and control a containers resources](#) on page 602.

## 10.2.2.6 Linux Namespaces

Linux namespaces is a feature in the Linux kernel that allows one to unshare and isolate a processes' resources like UTS, PID, IPC, file system mount and network from their parent. To achieve this the kernel places the resources in different namespaces.

When LXC spawns the container's main process it unshares all these resources except the network. The network is controlled from the configuration file and is shared by default.

A network namespace provides an isolated view of the networking stack (network device interfaces, IPv4 and IPv6 protocol stacks, IP routing tables, firewall rules, the /proc/net and /sys/class/net directory trees, sockets, etc.). A physical network device can live in exactly one network namespace. A virtual network device ("veth") pair provides a pipe-like abstraction that can be used to create tunnels between network namespaces, and can be used to create a bridge to a physical network device in another namespace. When a network namespace is freed (i.e., when the last process in the namespace terminates), its physical network devices are moved back to the initial network namespace (not to the parent of the process).

Each namespace is documented in the Linux **clone** man page. See: [clone \(2\)](#)

## 10.2.2.7 POSIX Capabilities

Linux supports the concept of file "capabilities" which provides fine grained control over what executable programs are permitted to do. Instead of the "all or nothing" paradigm where a super-user or "root" has the power to perform all operations, capabilities provide a mechanism to grant a specific program specific capabilities.

LXC uses this feature of the kernel to implement containers. By default processes running in a container will have **all** capabilities, but this can be configured. Capabilities can be dropped in the container's configuration file. See [LXC: Configuration Files](#) on page 584.

For example, to drop the CAP\_SYS\_MODULE, CAP\_MKNOD, CAP\_SETUID, and CAP\_NET\_RAW capabilities, the following configuration file options would be specified:

```
lxc.cap.drop = sys_module mknod setuid net_raw
```

Each capability is documented in the Linux **capabilities** man page. See: [capabilities \(7\)](#)

In order to fully isolate a container, the capabilities to be dropped must be carefully considered. The Linux Vserver project considers only the following capabilities as **safe** for virtual private servers:

```
CAP_CHOWN
CAP_DAC_OVERRIDE
CAP_DAC_READ_SEARCH
CAP_FOWNER
CAP_FSETID
CAP_KILL
CAP_SETGID
CAP_SETUID
CAP_NET_BIND_SERVICE
CAP_SYS_CHROOT
CAP_SYS_PTRACE
CAP_SYS_BOOT
```



```
CAP_SYS_TTY_CONFIG
CAP_LEASE
```

(see: [http://linux-vserver.org/Paper#Secure\\_Capabilities](http://linux-vserver.org/Paper#Secure_Capabilities))

## 10.2.3 LXC: How To's

### 10.2.3.1 LXC: Getting Started (with a Busybox System Container)

The following article describes steps to run a simple container example. All the command below are issued from a host Linux command prompt.

1. Confirm that your kernel environment is configured correctly using **lxc-checkconfig**. All options should show as 'enabled'.

```
lxc-checkconfig
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
Bridges: enabled
Advanced netfilter: enabled
CONFIG_NF_NAT_IPV4: enabled
CONFIG_NF_NAT_IPV6: enabled
CONFIG_IP_NF_TARGET_MASQUERADE: enabled
CONFIG_IP6_NF_TARGET_MASQUERADE: enabled
CONFIG_NETFILTER_XT_TARGET_CHECKSUM: enabled
FUSE (for use with lxcfs): enabled

--- Checkpoint/Restore ---
checkpoint restore: missing
CONFIG_FHANDLE: enabled
CONFIG_EVENTFD: enabled
CONFIG_EPOLL: enabled
CONFIG_UNIX_DIAG: enabled
CONFIG_INET_DIAG: enabled
CONFIG_PACKET_DIAG: enabled
CONFIG_NETLINK_DIAG: enabled
File capabilities: enabled
```

**Note:** Before booting a new kernel, you can check its configuration

**Usage :** CONFIG=/path/to/config /usr/bin/lxc-checkconfig

If the cgroup namespace option shows as required:

```
Cgroup namespace: required
```

The `/cgroup` directory most likely needs to be created and or mounted.

## 2. Create a container

Create a system container using `lxc-create` and specify the busybox template and `lxc-empty-netns.conf` config file. `lxc-empty-netns.conf` is a simple config file with no networking:

```
lxc-create -n foo -t busybox -f /usr/share/doc/lxc-common/examples/lxc-empty-netns.conf
setting root password to "root"
Password for 'root' changed
#
```

By default, LXC will try to install the dropbear ssh utility, if it's available on the host system. The Busybox template also has support for installing OpenSSH (assuming it's installed on the host Linux) in the container. This needs to be passed explicitly using a command line parameter:

```
lxc-create -n foo -t busybox -f /usr/share/doc/lxc-common/examples/lxc-empty-netns.conf -- -s
openssh
setting root password to "root"
Password for 'root' changed
'OpenSSH' ssh utility installed
#
```

## 3. List containers that exist

```
lxc-ls -f
NAME STATE AUTOSTART GROUPS IPV4 IPV6
foo STOPPED 0 - - -
```

## 4. From a shell on the host Linux, start the container. When prompted, press 'Enter'.

```
lxc-start -n foo -F

Please press Enter to activate this console.

/ #

/ #
```

Note that the shell is now running within the container. Normal Linux commands can be executed.

**Important notice:** while this mode starts the container and directly connects to one of its terminals, there is a minor caveat: the terminal will be stuck in this container console until the container is halted (either from here, by running **halt**, or from another terminal by running **lxc-stop**). In order to avoid this, there is also the possibility to start the container as a daemon and connect to it using **lxc-console** (this is the default mode). This provides better terminal capabilities and the user is not forced to stop the container from another terminal. On the other hand, there is no indication that after running **lxc-start** the container has actually started - no errors are reported. You must check if the container is running yourself, using **lxc-info** - see below.

```
lxc-start -n foo
lxc-console -n foo

Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself

foo login: root
```

```
Password: (root)
~ #
~ #
~ #
~ # (Ctrl+a q)
#
```

This will be the preferred mode of starting and connecting to containers.

##### 5. List processes in the container.

From in the container shell use the `ps` command to list processes:

```
~ # ps
 PID USER VSZ STAT COMMAND
 1 root 2384 S init
 4 root 2384 S /bin/syslogd
 6 root 2388 S -sh
 7 root 2384 S init
 8 root 2388 R ps
```

Note process IDs have a number-space unique to the container.

##### 6. Show the status of the foo container (from a host shell):

```
lxc-info -n foo
Name: foo
State: RUNNING
PID: 4544
CPU use: 0.01 seconds
Memory use: 472.00 KiB
KMem use: 0 bytes
```

##### 7. Look at the files/directories in `/var/lib/lxc` related to the container

```
ls -l /var/lib/lxc/foo
total 2
-rw-r--r-- 1 root root 675 May 30 15:37 config
drwxr-xr-x 16 root root 1024 May 30 15:44 rootfs
```

This shows the containers config file and rootfs backing store.

Look at the contents of the config file:

```
cat /var/lib/lxc/foo/config
Template used to create this container: /usr/share/lxc/templates/lxc-busybox
Parameters passed to the template:
For additional config options, please look at lxc.conf(5)
lxc.utsname = omega
lxc.network.type = empty
lxc.network.flags = up
lxc.rootfs = /var/lib/lxc/foo/rootfs
lxc.haltsignal = SIGUSR1
lxc.utsname = foo
lxc.tty = 1
lxc.pts = 1
lxc.cap.drop = sys_module mac_admin mac_override sys_time

When using LXC with apparmor, uncomment the next line to run unconfined:
#lxc.aaprofile = unconfined
lxc.mount.entry = /lib lib none ro,bind 0 0
```

```
lxc.mount.entry = /usr/lib usr/lib none ro,bind 0 0
lxc.mount.entry = /sys/kernel/security sys/kernel/security none ro,bind,optional 0 0
lxc.mount.auto = proc:mixed sys
```

8. Start a process inside the container using `lxc-attach`. This command will run the process inside the system container's isolated environment. The container has to be running already.

```
lxc-attach -n foo -- /bin/sh
root@foo:/# ps
PID USER TIME COMMAND
 1 root 0:00 init
 6 root 0:00 /bin/syslogd
 8 root 0:00 /bin/getty -L tty1 115200 vt100
 9 root 0:00 init
 10 root 0:00 /bin/sh
 11 root 0:00 ps
root@foo:/# ls -l /dev
total 0
crw-rw-rw- 1 root 5 136, 1 May 26 13:13 console
lrwxrwxrwx 1 root root 13 May 26 13:12 fd -> /proc/self/fd
lrwxrwxrwx 1 root root 7 May 26 13:13 kmsg -> console
srw-rw-rw- 1 root root 0 May 26 13:13 log
crw-rw-rw- 1 root root 1, 3 May 26 13:10 null
lrwxrwxrwx 1 root root 13 May 26 13:12 ptmx -> /dev/pts/ptmx
drwxr-xr-x 2 root root 0 May 26 13:13 pts
brw----- 1 root root 1, 0 May 26 13:10 ram0
drwxrwxrwt 2 root root 40 May 26 13:13 shm
lrwxrwxrwx 1 root root 15 May 26 13:12 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 May 26 13:12 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 May 26 13:12 stdout -> /proc/self/fd/1
crw-rw-rw- 1 root root 5, 0 May 26 13:10 tty
crw-rw-rw- 1 root root 4, 0 May 26 13:10 tty0
crw--w---- 1 root root 136, 0 May 26 13:13 tty1
crw-rw-rw- 1 root root 4, 0 May 26 13:10 tty5
crw-rw-rw- 1 root root 1, 9 May 26 13:10 urandom
crw-rw-rw- 1 root root 1, 5 May 26 13:10 zero
root@foo:/#
```

9. Stop the container (from a host shell)

```
lxc-stop -n foo
#
lxc-info -n foo
Name: foo
State: STOPPED
```

10. Destroy the container. This removes the containers config file and backing store.

```
lxc-destroy -n foo
#
```

### 10.2.3.2 LXC: How to configure non-virtualized networking (lxc-no-netns.conf)

One approach to providing networking capability to a container is to simply allow the container to use existing host network interfaces. To accomplish this, a configuration file is created with no networking setup (i.e. the `lxc.network.type` property is not set) and the default will be to allow the container to access the host's networking interfaces.

With this approach no network namespace is created for the container.

An example config is provided:

```
/usr/share/doc/lxc-common/examples/lxc-no-netns.conf
```

The contents of `lxc-no-netns.conf` look like this:

```
Container with non-virtualized network
lxc.network.type = none
lxc.utsname = delta
```

The example below shows starting an application container (running `bash`) with this config file and shows that the host network interface `enp1s0` is inherited and accessible by the container:

```
lxc-execute -n mytest -f /usr/share/doc/lxc-common/examples/lxc-no-netns.conf -- /bin/bash
root@delta:/root# ifconfig
docker0 Link encap:Ethernet HWaddr 02:42:b0:95:11:e0
 inet addr:172.17.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
 inet6 addr: fe80::42:b0ff:fe95:11e0/64 Scope:Link
 UP BROADCAST MULTICAST MTU:1500 Metric:1
 RX packets:18 errors:0 dropped:0 overruns:0 frame:0
 TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:0
 RX bytes:4568 (4.5 KB) TX bytes:1236 (1.2 KB)

enp1s0 Link encap:Ethernet HWaddr 68:05:ca:36:9d:75
 inet addr:192.168.1.20 Bcast:0.0.0.0 Mask:255.255.248.0
 inet6 addr: fe80::6a05:caff:fe36:9d75/64 Scope:Link
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
 RX packets:94306 errors:0 dropped:0 overruns:0 frame:0
 TX packets:40146 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:138650604 (138.6 MB) TX bytes:2922616 (2.9 MB)
 Interrupt:100 Memory:30460c0000-30460e0000

lo Link encap:Local Loopback
 inet addr:127.0.0.1 Mask:255.0.0.0
 inet6 addr: ::1/128 Scope:Host
 UP LOOPBACK RUNNING MTU:65536 Metric:1
 RX packets:1018 errors:0 dropped:0 overruns:0 frame:0
 TX packets:1018 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:78782 (78.7 KB) TX bytes:78782 (78.7 KB)

lxcbr0 Link encap:Ethernet HWaddr 00:16:3e:00:00:00
 inet addr:10.0.3.1 Bcast:0.0.0.0 Mask:255.255.255.0
 UP BROADCAST MULTICAST MTU:1500 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

### 10.2.3.3 LXC: How to assign a physical network interface to a container (lxc-phys.conf)

One approach to providing networking capability to a container is to directly assign an available, unused network interface to the container. The interface is not shared, it becomes the private resource of the container.

An example LXC configuration file is provided to configure this type of networking:

```
/usr/share/doc/lxc-common/examples/lxc-phys.conf
```

The contents of the default lxc-phys.conf example are show below:

```
Container with network virtualized using a physical network device with name
'eth0'
lxc.utsname = gamma
lxc.network.type = phys
lxc.network.flags = up
lxc.network.link = eth0
lxc.network.hwaddr = 4a:49:43:49:79:ff
lxc.network.ipv4 = 10.2.3.6/24
lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3297
```

**Note:** The network type is set to: **phys**. Make a copy of the example config file and update it with the name of the Ethernet interface to be assigned, an appropriate IP address, and any other appropriate changes (e.g. mac address). For example, the change (in universal diff format) to set the interface `enp1s0` and IP address 192.168.10.3 would look like:

```
/usr/share/doc/lxc-common/examples/lxc-phys.conf
+++ lxc-phys.conf
@@ -3,7 +3,6 @@
 lxc.utsname = gamma
 lxc.network.type = phys
 lxc.network.flags = up
- lxc.network.link = eth0
- lxc.network.hwaddr = 4a:49:43:49:79:ff
- lxc.network.ipv4 = 10.2.3.6/24
- lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3297
+ lxc.network.link = enp1s0
+ lxc.network.hwaddr = 00:e0:0c:00:93:05
+ lxc.network.ipv4 = 192.168.10.3/24
```

A simple way to test the new config file and the network interface is to run `/bin/bash` as a command with `lxc-execute`, which will provide a shell running in the container:

```
lxc-execute -n mytest -f lxc-phys.conf -- /bin/bash
bash-4.2#
```

In the container, use the `fm1-gb4` interface normally:

```
bash-4.3# ifconfig
enp1s0 Link encap:Ethernet HWaddr 00:e0:0c:00:93:05
 inet addr:192.168.10.3 Bcast:192.168.10.255 Mask:255.255.255.0
 UP BROADCAST MULTICAST MTU:1500 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:0 (0.0 B) TX bytes:508 (508.0 B)
 Memory:fe5e8000-fe5e8fff
```

```
lo Link encap:Local Loopback
 inet addr:127.0.0.1 Mask:255.0.0.0
 inet6 addr: ::1/128 Scope:Host
 UP LOOPBACK RUNNING MTU:65536 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:0
 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

bash-4.2# ping -c 3 192.168.10.1
PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data.
64 bytes from 192.168.10.1: icmp_req=1 ttl=64 time=0.385 ms
64 bytes from 192.168.10.1: icmp_req=2 ttl=64 time=0.207 ms
64 bytes from 192.168.10.1: icmp_req=3 ttl=64 time=0.187 ms

--- 192.168.10.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.187/0.259/0.385/0.090 ms
```

### 10.2.3.4 LXC: How to configure networking with virtual Ethernet pairs (lxc-veth.conf)

One approach to providing a virtual network interface to a container is using the "Virtual ethernet pair device" feature of the Linux kernel in conjunction with a network bridge.

See the veth description in <http://man7.org/linux/man-pages/man5/lxc.conf.5.html> for additional details on this approach to networking.

With this approach LXC creates a new network namespace for the container.

The example configuration file lxc-veth.conf demonstrates this approach:

```
/usr/share/doc/lxc-common/examples/lxc-veth.conf
```

The contents of the default lxc-veth.conf example are show below:

```
Container with network virtualized using a pre-configured bridge named br0 and
veth pair virtual network devices
lxc.utsname = beta
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
lxc.network.hwaddr = 4a:49:43:49:79:bf
lxc.network.ipv4 = 10.2.3.5/24
lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3597
```

Note, the network type value is: **veth** and the link property value is **br0**.

First, create a network bridge which is attached to a physical network interface and assign the bridge an IP address. The bridge becomes one endpoint In the example below the bridge br0 is created, interface `enp1s0` is added to it, and the bridge is assigned an IP address of 192.168.20.2.

```
brctl addbr br0
ifconfig br0 192.168.20.2 up
ifconfig enp1s0 up
brctl addif br0 enp1s0
```

Make a copy of the example config file and update it with an appropriate IP address and any other appropriate changes (e.g. mac address). For example, the change (in universal diff format) to update the IP address to 192.168.20.3 would look like:

```
--- /usr/share/doc/lxc-common/examples/lxc-veth.conf
+++ lxc-veth.conf
@@ -5,5 +5,5 @@
 lxc.network.flags = up
 lxc.network.link = br0
 lxc.network.hwaddr = 4a:49:43:49:79:bf
-lxc.network.ipv4 = 10.2.3.5/24
+lxc.network.ipv4 = 192.168.20.3/24
 lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3597
```

A simple way to test the new config file and the network interface is to run `/bin/bash` as a command with `lxc-execute`, which will provide a shell running in the container:

```
lxc-execute -n mytest -f lxc-veth.conf -- /bin/bash
bash-4.2#
```

In the container, use the virtual network interface (eth0 in this example) normally:

```
bash-4.2# ifconfig
eth0 Link encap:Ethernet HWaddr 4a:49:43:49:79:bf
 inet addr:192.168.20.3 Bcast:192.168.20.255 Mask:255.255.255.0
 inet6 addr: fe80::4849:43ff:fe49:79bf/64 Scope:Link
 inet6 addr: 2003:db8:1:0:214:1234:fe0b:3597/64 Scope:Global
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
 RX packets:6 errors:0 dropped:0 overruns:0 frame:0
 TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:468 (468.0 B) TX bytes:586 (586.0 B)

lo Link encap:Local Loopback
 inet addr:127.0.0.1 Mask:255.0.0.0
 inet6 addr: ::1/128 Scope:Host
 UP LOOPBACK RUNNING MTU:65536 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:0
 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

bash-4.2# ping -c 3 192.168.20.1
PING 192.168.20.1 (192.168.20.1) 56(84) bytes of data.
64 bytes from 192.168.20.1: icmp_req=1 ttl=64 time=0.433 ms
64 bytes from 192.168.20.1: icmp_req=2 ttl=64 time=0.221 ms
64 bytes from 192.168.20.1: icmp_req=3 ttl=64 time=0.228 ms

--- 192.168.20.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.221/0.294/0.433/0.098 ms
```

### 10.2.3.5 LXC: How to configure networking with macvlan (lxc-macvlan.conf)

An LXC container can be provided with a virtual network interface using the "MAC-VLAN" feature of the Linux kernel (see kernel config option `CONFIG_MACVLAN`). MAC-VLAN allows virtual interfaces to be created that route packets to or from a MAC address to a physical network interface.



See the macvlan description in <http://man7.org/linux/man-pages/man5/lxc.conf.5.html> for some additional details on this approach to networking.

The example configuration file `lxc-veth.conf` demonstrates this approach:

```
/usr/share/doc/lxc-common/examples/lxc-macvlan.conf
```

The contents of the provided `lxc-phys.conf` example configuration file are show below:

```
Container with network virtualized using the macvlan device driver
lxc.utsname = alpha
lxc.network.type = macvlan
lxc.network.flags = up
lxc.network.link = eth0
lxc.network.hwaddr = 4a:49:43:49:79:bd
lxc.network.ipv4 = 10.2.3.4/24
lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3596
```

Make a copy of the example config file and update it with the physical network interface to be used, an appropriate IP address, and any other appropriate changes (e.g. mac address). For example, the change (in universal diff format) to specify the `enp1s0` interface and update the IP address to `192.168.1.24` would look like:

```
--- /usr/share/doc/lxc-common/examples/lxc-macvlan.conf
+++ lxc-macvlan.conf
@@ -2,7 +2,7 @@
 lxc.utsname = alpha
 lxc.network.type = macvlan
 lxc.network.flags = up
-lxc.network.link = eth0
+lxc.network.link = enp1s0
 lxc.network.hwaddr = 4a:49:43:49:79:bd
-lxc.network.ipv4 = 10.2.3.4/24
+lxc.network.ipv4 = 192.168.10.3/24
 lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3596
```

Put the network interface in promiscuous mode:

```
ifconfig enp1s0 promisc
ifconfig enp1s0
enp1s0 Link encap:Ethernet HWaddr 00:e0:0c:00:93:05
 inet addr:192.168.10.2 Bcast:192.168.10.255 Mask:255.255.255.0
 inet6 addr: fe80::2e0:cff:fe00:9305/64 Scope:Link
 UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
 RX packets:5 errors:0 dropped:0 overruns:0 frame:0
 TX packets:17 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:1000
 RX bytes:344 (344.0 B) TX bytes:1314 (1.2 KiB)
 Memory:fe5e0000-fe5e0fff
```

Test the MAC-VLAN interface by starting an application container running `/bin/bash`:

```
lxc-execute -n mytest -f lxc-macvlan.conf -- /bin/bash
bash-4.2#
```

Note: the shell prompt above ("`bash-4.2`") is in the container.

Test the interface in the now running container:

```
bash-4.2# ifconfig
eth0 Link encap:Ethernet HWaddr 4a:49:43:49:79:bd
 inet addr:192.168.10.3 Bcast:192.168.10.255 Mask:255.255.255.0
 inet6 addr: fe80::4849:43ff:fe49:79bd/64 Scope:Link
 inet6 addr: 2003:db8:1:0:214:1234:fe0b:3596/64 Scope:Global
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:0
 RX bytes:0 (0.0 B) TX bytes:586 (586.0 B)

lo Link encap:Local Loopback
 inet addr:127.0.0.1 Mask:255.0.0.0
 inet6 addr: ::1/128 Scope:Host
 UP LOOPBACK RUNNING MTU:65536 Metric:1
 RX packets:0 errors:0 dropped:0 overruns:0 frame:0
 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
 collisions:0 txqueuelen:0
 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

bash-4.2# ping -c 3 192.168.10.1
PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data.
64 bytes from 192.168.10.1: icmp_req=1 ttl=64 time=0.380 ms
64 bytes from 192.168.10.1: icmp_req=2 ttl=64 time=0.204 ms
64 bytes from 192.168.10.1: icmp_req=3 ttl=64 time=0.201 ms

--- 192.168.10.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.201/0.261/0.380/0.085 ms
```

### 10.2.3.6 LXC: How to configure networking using a VLAN (lxc-vlan.conf)

A container can be provided with a virtual network interface using VLANs.

See the vlan description in <http://man7.org/linux/man-pages/man5/lxc.conf.5.html> for some additional details on this approach to networking.

The example configuration file lxc-veth.conf demonstrates this approach:

```
/usr/share/doc/lxc-common/examples/lxc-vlan.conf
```

The contents of the provided lxc-vlan.conf example configuration file are show below:

```
Container with network virtualized using the vlan device driver
lxc.utsname = alpha
lxc.network.type = vlan
lxc.network.vlan.id = 1234
lxc.network.flags = up
lxc.network.link = eth0
lxc.network.hwaddr = 4a:49:43:49:79:bd
lxc.network.ipv4 = 10.2.3.4/24
lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3596
```

Make a copy of the example config file and update it with the physical network interface to be used and the vlan ID, an appropriate IP address, and any other appropriate changes. For example, the change (in universal diff format) to specify the `enp1s0` interface, a VLAN id of 2, and an IP address of 192.168.30.2 would look like:

```
--- /usr/share/doc/lxc/examples/lxc-vlan.conf 2013-05-30 14:22:14.980406375 +0300
+++ lxc-vlan.conf 2013-06-03 13:26:38.477580000 +0300
@@ -1,9 +1,9 @@
 # Container with network virtualized using the vlan device driver
 lxc.utsname = alpha
 lxc.network.type = vlan
-lxc.network.vlan.id = 1234
+lxc.network.vlan.id = 2
 lxc.network.flags = up
-lxc.network.link = eth0
+lxc.network.link = enp1s0
 lxc.network.hwaddr = 4a:49:43:49:79:bd
-lxc.network.ipv4 = 10.2.3.4/24
+lxc.network.ipv4 = 192.168.30.2/24
 lxc.network.ipv6 = 2003:db8:1:0:214:1234:fe0b:3596
```

In this setup, the host is connected to a test machine through physical interface `enp1s0`. On the test machine, the following commands have been issued (interface `p7p1` on this machine has physical link to `enp1s0`):

```
[root@everest] [~]# modprobe 8021q
[root@everest] [~]# lsmod | grep 8021q
8021q 23476 0
garp 13763 1 8021q
[root@everest] [~]# vconfig add p7p1 2
Added VLAN with VID == 2 to IF -:p7p1:-
[root@everest] [~]# ifconfig p7p1.2 192.168.30.1 up
```

Test the VLAN interface by starting an application container running `/bin/bash`:

```
lxc-execute -n mytest -f lxc-vlan.conf -- /bin/bash
bash-4.2#
```

Test the interface in the now running container:

```
bash-4.2# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
 inet 192.168.30.2 netmask 255.255.255.0 broadcast 192.168.30.255
 inet6 fe80::21e:c9ff:fe49:bb93 prefixlen 64 scopeid 0x20<link>
 ether 00:1e:c9:49:bb:93 txqueuelen 0 (Ethernet)
 RX packets 0 bytes 0 (0.0 B)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 6 bytes 468 (468.0 B)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 16436
 inet 127.0.0.1 netmask 255.0.0.0
 inet6 ::1 prefixlen 128 scopeid 0x10<host>
 loop txqueuelen 0 (Local Loopback)
 RX packets 4 bytes 200 (200.0 B)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 4 bytes 200 (200.0 B)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

bash-4.2# ping -c 3 192.168.30.1
PING 192.168.30.1 (192.168.30.1) 56(84) bytes of data.
```

```

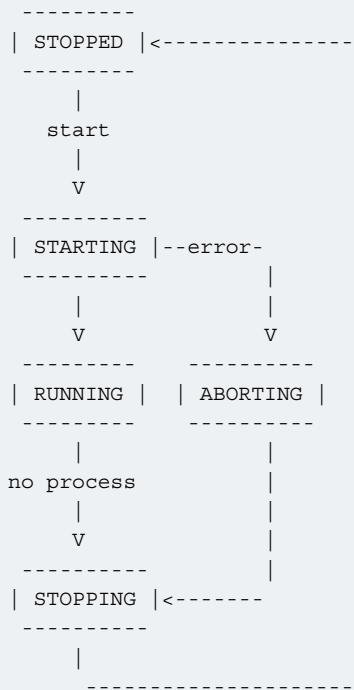
64 bytes from 192.168.30.1: icmp_req=1 ttl=64 time=0.338 ms
64 bytes from 192.168.30.1: icmp_req=2 ttl=64 time=0.372 ms
64 bytes from 192.168.30.1: icmp_req=3 ttl=64 time=0.355 ms

--- 192.168.30.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.338/0.355/0.372/0.013 ms

```

### 10.2.3.7 LXC: How to monitor containers

Containers transition through a set of well defined states. After a container is created it is in the "stopped" state.



A number of commands are available in LXC to monitor the state of a container. The following examples provide an introduction and demonstrate the capabilities of these commands.

#### 1. lxc-info

The `lxc-info` command shows the current state of the container.

In the example below, a container called "foo" has already been created but not started and the container is stopped:

```

lxc-info -n foo
Name: foo
State: STOPPED
After the container is started lxc-info shows the container in the running state:

```

```

lxc-start -n foo
lxc-info -n foo
Name: foo
State: RUNNING
PID: 5075
CPU use: 0.01 seconds
Memory use: 508.00 KiB
KMem use: 0 bytes

```

## 2. lxc-monitor

The **lxc-monitor** command can monitor the state of one or more containers, the command continues to run until it is killed.

In this example **lxc-monitor** monitors the state of a container named "foo":

```
lxc-monitor -n foo
```

In a separate shell, start and then stop the container foo:

```
lxc-start -n foo
lxc-stop -n foo
```

The running **lxc-monitor** command displays the state changes as they occur:

```
'foo' changed state to [STARTING]
'foo' changed state to [RUNNING]
'foo' changed state to [STOPPING]
'foo' changed state to [STOPPED]
```

## 3. lxc-wait

The **lxc-wait** command will wait for a container state change and then exit. This can be useful for scripting and synchronizing the start or exit of a container.

For example, to wait until the container named "foo" stops:

```
lxc-wait -n foo -s STOPPED
```

# 10.2.3.8 LXC: How to modify the capabilities of a container to provide additional isolation

As described in [POSIX Capabilities](#) on page 588, by default processes running in a container will have all capabilities. And the configuration for a container can further restrict these capabilities.

This example shows how to remove the ability for a container to issue the **mknod** command.

By default a container can issue the **mknod** command:

```
~ # mknod zero c 1 5
~ # ls -l zero
crw-r--r-- 1 root root 1, 5 Jun 3 17:08 zero
```

In this example we modify the config file of a container named "foo" (/var/lib/lxc/foo/config) and specify in the **lxc.cap.drop** property that the **mknod** capability (CAP\_MKNOD) should be removed:

```
@@ -5,6 +5,7 @@
lxc.utsname = foo
lxc.tty = 1
lxc.pts = 1
+lxc.cap.drop = mknod
lxc.rootfs = /var/lib/lxc/foo/rootfs
lxc.mount.entry=/lib /var/lib/lxc/foo/rootfs/lib none ro,bind 0 0
lxc.mount.entry=/usr/lib /var/lib/lxc/foo/rootfs/usr/lib none ro,bind 0 0
```

Now restart the container and the `mknod` operation is no longer permitted:

```
~ # mknod zero c 1 5
mknod: zero: Operation not permitted
```

### 10.2.3.9 LXC: How to use cgroups to manage and control a containers resources

This example demonstrates how to use control groups to control which CPU's a container is scheduled on and the percentage of CPU time allocated to a container.

In this example we'll examine and change:

- the **cpuset** subsystem's **cpus** parameter which controls which physical CPUs the container's processes will run on
- the **cpu** subsystem's **shares** parameter which controls the percentage of the CPU to be allocated to the container

1. Start two application containers each running `/bin/bash`:

First container:

```
lxc-execute -n foo1 -f lxc-no-netns.conf -- /bin/bash
bash-4.2#
```

Second container:

```
lxc-execute -n foo2 -f lxc-no-netns.conf -- /bin/bash
bash-4.2#
```

2. In both containers start a process that will put a 100% load on the CPUs:

```
(while true; do true; done) &
```

3. The **cpuset.cpus** subsystem/value specifies which physical CPUs the container's processes run on. From a host shell, examine this with the **lxc-cgroup** command:

```
lxc-cgroup -n foo1 cpuset.cpus
0-7
```

In this example the host system has 4 CPUs.

This can also be seen directly through the `/cgroup` filesystem:

```
cat /sys/fs/cgroup/cpuset/lxc/foo1/cpuset.cpus
0-7
```

4. Change both containers to run only on CPU 2:

```
lxc-cgroup -n foo1 cpuset.cpus 2
lxc-cgroup -n foo2 cpuset.cpus 2
#
```

The `top` command now shows CPU 2 with 100% utilization. The bash commands running in each container, each have about 50% of the CPU:

```
top - 17:14:41 up 10 min, 4 users, load average: 1.64, 0.61, 0.23
Tasks: 100 total, 3 running, 97 sleeping, 0 stopped, 0 zombie
Cpu0 : 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
```

```
Cpu2 :100.0%us, 0.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu4 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu6 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 3996400k total, 189836k used, 3806564k free, 1652k buffers
Swap: 0k total, 0k used, 0k free, 26180k cached
```

| PID  | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+   | COMMAND |
|------|------|----|----|------|-----|-----|---|------|------|---------|---------|
| 2875 | root | 20 | 0  | 3624 | 416 | 164 | R | 50   | 0.0  | 1:28.12 | bash    |
| 2874 | root | 20 | 0  | 3624 | 424 | 168 | R | 50   | 0.0  | 1:31.06 | bash    |

5. The `cpu.shares` subsystem/value specifies the percentage of the CPU allocated to the cgroup/container. By default each container has a shares value of 1024:

```
lxc-cgroup -n foo1 cpu.shares
1024
lxc-cgroup -n foo2 cpu.shares
1024
```

6. Change container "foo2" to have about 10% of the CPU:

```
lxc-cgroup -n foo2 cpu.shares 100
lxc-cgroup -n foo1 cpu.shares 900
```

Now the `top` command output reflects the new CPU allocation:

| PID  | USER | PR | NI | VIRT | RES | SHR | S | %CPU      | %MEM | TIME+   | COMMAND |
|------|------|----|----|------|-----|-----|---|-----------|------|---------|---------|
| 2874 | root | 20 | 0  | 3624 | 424 | 168 | R | <b>90</b> | 0.0  | 2:53.63 | bash    |
| 2875 | root | 20 | 0  | 3624 | 416 | 164 | R | <b>10</b> | 0.0  | 2:11.36 | bash    |

7. Stop the containers

```
lxc-stop -n foo1 -k
lxc-stop -n foo2 -k
#
```

## 10.2.3.10 LXC: How to run an application in a container with `lxc-execute`

The **`lxc-execute`** command allows a single application to be run in a container (as contrasted with a system container which boots an instance of Linux user space starting with System V style init).

In the example below an instance of a QEMU/KVM virtual machine is started in a container called `foo`.

Note, it is not required to explicitly create (and destroy) a container when running application containers with `lxc-execute`. The containers will automatically created and destroyed.

1. Start QEMU in the container with `lxc-execute`:

```
lxc-execute -n foo -f lxc-no-netns.conf -- qemu-system-ppc -enable-kvm -smp 2 -m 256M -nographic
-M ppc64 -kernel uImage -initrd rootfs.ext2.gz -append "root=/dev/ram rw console=ttyS0,115200" -
serial tcp::4445,server,telnet
```

NOTE: For 64bit platforms, please replace `qemu-system-ppc` with `qemu-system-ppc64`.

Some notes:

- The QEMU command line follows the double dash ("--") specified on the lxc-execute command line and distinguishes argument to lxc-execute from arguments to qemu-system-ppc.
- Using the specified configuration file, QEMU will run in the network namespace of the host system, meaning the TCP ports for serial and the monitor (ports 4445 and 4446) can be accessed from the host. However, lxc-execute will accept a configuration file as an argument allowing customization of the degree of isolation of the container.
- In this example there are 2 virtual cpus specified, which results in a total of 3 QEMU processes/threads. So we expect to see 3 QEMU processes in the container.

2. Examine the state of the container with lxc-ls and lxc-info:

```
lxc-ls --active
foo

lxc-info -n foo
Name: foo
State: RUNNING
PID: 3205
IP: 192.168.2.80
CPU use: 3.96 seconds
Memory use: 140.46 MiB
KMem use: 0 bytes
```

3. In the QEMU console look at the CPU status which shows the process IDs for the two virtual CPUs in the virtual machine:

```
(qemu) info cpus
* CPU #0: nip=0x00000000c001450c thread_id=4
 CPU #1: nip=0x00000000c001450c thread_id=5
(qemu)
```

Note that the process/thread IDs as viewed from within the container (thread IDs 4 and 5) are different than from the host, since they are in a different namespace.

5. Using the container's cgroup restrict the physical CPUs on which the virtual machine is allowed to run.

By default all 4 CPUs can be used by the container :

```
by default all 4 CPUs can be used by the container
cat /cgroup/lxc/foo/cpuset.cpus
0-3
```

Restrict the containers processes to CPUs 2 and 3:

```
echo 2-3 > /cgroup/lxc/foo/cpuset.cpus
cat /cgroup/lxc/foo/cpuset.cpus
2-3
```

### 10.2.3.11 LXC: How to run an unprivileged container

With the addition of the user namespace in the Linux kernel, a normal user on a Linux host can create and run container instances. This feature has been integrated in the LXC package, starting from version 1.0.

The steps below detail the necessary steps required in order to configure and manage an unprivileged container.

**NOTE:** Before running these steps, make sure that the host is properly configured for container use, by running **lxc-checkconfig** (cgroups, namespaces, etc.).

1. Create the **/etc/subuid** and **/etc/subgid** file on the Linux host. These will be used to store the unprivileged user's subordinate UIDs and GIDs. The unprivileged user has the ability to manage users on his own in his user namespace,



and their IDs will be mapped to corresponding ranges on IDs on the host system. The subordinate IDs will correspond to the ranges defined in these files.

```
for file in '/etc/subuid' '/etc/subgid'; do
 touch $file
 chown root:root $file
 chmod 644 $file
done
```

2. Add a user in the system - **lxc-user**.

```
useradd lxc-user -p $(echo test | openssl passwd -1 -stdin)
```

3. Check the contents of **/etc/subuid** and **/etc/subgid**. If they contain the following entries, the user has been automatically assigned a default set of subordinate IDs.

```
root@t4240qds:~# cat /etc/sub*
lxc-user:100000:65536
lxc-user:100000:65536
root@t4240qds:~#
```

If the files are empty, you need to manually assign a set of subordinate IDs to the user.

```
usermod --add-subuids 100000-165536 lxc-user
usermod --add-subgids 100000-165536 lxc-user
```

4. The container will have a virtual interface linked to a bridge on the host. Use the following command to create the bridge.

```
brctl addbr br0 && ifconfig br0 10.0.0.1
```

5. You must create and edit the **/etc/lxc/lxc-usernet** file. This file specifies how many interfaces the lxc-user will be allowed to have linked in this bridge.

```
echo "lxc-user veth br0 10" > /etc/lxc/lxc-usernet
```

6. Create the **/home/lxc-user/.config/lxc** directory on the host. This will hold the default configuration for unprivileged containers belonging to the lxc-user.

```
mkdir -p /home/lxc-user/.config/lxc
```

7. **Create** the default container configuration file, **/home/lxc-user/.config/lxc/default.conf**, and paste the following lines.

```
lxc.network.type = veth
lxc.network.link = br0
lxc.network.flags = up
lxc.id_map = u 0 100000 65536
lxc.id_map = g 0 100000 65536
```

8. Change the ownership of the newly created files and folders to lxc-user.

```
chown -R lxc-user:lxc-user /home/lxc-user/.config
```

9. For each of the mounted cgroup controllers, created a directory in the top called **lxc-user**, and change its ownership to lxc-user. Be sure to enable the cgroup.clone\_children and memory.use\_hierarchy flags.

```
echo 1 > /sys/fs/cgroup/memory/memory.use_hierarchy

for c in `ls /sys/fs/cgroup/`; do
```

```
echo 1 > /sys/fs/cgroup/$c/cgroup.clone_children
mkdir /sys/fs/cgroup/$c/lxc-user
chown -R lxc-user:lxc-user /sys/fs/cgroup/$c/lxc-user
done
```

#### 10. Login as the new user in a new console.

```
t4240qds login: lxc-user
Password:
t4240qds:~$
```

#### 11. Copy the shell PID in the lxc-user cgroups.

```
for c in `ls /sys/fs/cgroup/`; do
 echo $$ > /sys/fs/cgroup/$c/lxc-user/tasks
done
```

#### 12. From the same shell as before, create a Busybox container. You can pass it a custom config file using the **-f** cmdline parameter. Otherwise, it will pick the default config from **/home/lxc-user/.config/default.conf**.

```
t4240qds:~$ lxc-create -n foo -t busybox
setting root password to "root"
Password for 'root' changed
t4240qds:~$
```

#### 13. Start the container.

```
t4240qds:~$ lxc-start -n foo -F

Please press Enter to activate this console.
/ #
/ #
/ # whoami
root
/ #
```

Now you can interact with the container as you would with one created by root. **Make sure that all container commands are run as lxc-user.**

## 10.2.3.12 LXC: How to run containers with Seccomp protection

A large number of system calls are exposed to every userland process with many of them going unused for the entire lifetime of the process. As system calls change and mature, bugs are found and eradicated. A certain subset of userland applications benefit by having a reduced set of available system calls. The resulting set reduces the total kernel surface exposed to the application. System call filtering is meant for use with those applications.

Seccomp (short for *secure compute*) is a system call filtering mechanism present in the kernel. [Initially](#) it has been thought to be a sandboxing mechanism that would allow userspace processes to issue a very limited set of system calls - read(), write(), exit() and sigreturn(). This has been further known to be **seccomp mode 1**, and while it is strong on the security side, it doesn't leave much room for flexibility.

The next addition to seccomp was to allow [filtering](#) (or **seccomp mode 2**) based on the kernel [BPF](#) (Berkeley Packet Filter) infrastructure. This allows the system administrator to define complex and granular policies, per system call and its arguments. This is an extension to the BPF mechanism, that allows filtering to apply to system call numbers and their arguments, besides its original purpose (socket packets). The defined filter results in a seccomp policy which is attached to the userspace process in the form of a BPF program. Each time the process issues a system call, it is checked against this policy in order to determine how it will be handled:

- **SECCOMP\_RET\_KILL** - the task exits immediately without executing the system call.

- **SECCOMP\_RET\_TRAP** - the kernel sends a SIGSYS to the triggering task without executing the system call.
- **SECCOMP\_RET\_ERRNO** - a custom errno is returned to userspace without executing the system call.
- **SECCOMP\_RET\_TRACE** - causes the kernel to attempt to notify a ptrace-based tracer prior to executing the system call. The tracer can skip the system call or change it to a valid syscall number.
- **SECCOMP\_RET\_ALLOW** - results in the system call being executed.

In order to make the secure computing facility more userspace-friendly, the [libseccomp](#) library has been developed, which is meant to make it easier for applications to take advantage of the packet-filter-based seccomp mode. Prior to this, userspace applications had to [define the BPF filter themselves](#). libseccomp restructures this approach into [a simple and straightforward API](#) which userspace applications can use. [The latest version of libseccomp](#) adds support for Python bindings as well, and is designed to work on multiple architectures (ARM, MIPS). [PowerPC support has also been merged](#) on a separate branch, and is expected to be included in future releases.

### Using seccomp with LXC containers

Refer to [Linux Containers \(LXC\) for NXP QorIQ User's Guide](#) on page 580 for information on how to build LXC with seccomp support in the SDK.

**Note:** Currently LXC seccomp support is not available for ARM64 architectures.

Seccomp filtering integrates well with processes sandboxed as containers, as they can be assigned to untrusted users and exposed with a limited set of allowed system calls. This is a portable and granular low-level security mechanism which can be used to increase container security. The seccomp policy file needs to be applied only to the init process in the container, and will be inherited by all its children.

The seccomp policy for the container is specified using the [container configuration file](#), in the form of a single line containing:

```
lxc.seccomp = /var/lib/lxc/lxc_seccomp.conf
```

An example lxc\_seccomp policy file can look as follows:

```
2
blacklist
[ppc64]
mknod errno 120
sched_setscheduler trap
fchmodat kill
[ppc]
mknod
```

The elements in the policy file represent the following:

1. Version number (1/2) - a single integer containing a single number, 1 or 2. Version 1 only allows to define a set of system calls which are allowed (whitelisted) in the container, specified by syscall number. This version is limited in configurability and portability, since it's only used to specify allowed syscall numbers, which may differ from arch to arch. Version 2 allows the policies to be either a whitelist (default deny, except mentioned syscalls) or a blacklist (default allow, except mentioned syscalls), and the syscalls can be expressed by name.
2. Policy type (whitelist/blacklist) - with an option of a default policy action (errno #, trap, kill, allow). The policy type is per seccomp context, and can be either whitelist or blacklist, not both.
3. Architecture tag [optional] - mentions that the following set of system calls will only be applied to a specific architecture. There can be multiple architecture tags and associated syscalls. These tags allow the same seccomp policy file to be used on multiple platforms, treating each one differently with respect to the set of system calls.

4. System calls - which can be expressed by number (in version 1) or name (in version 2). Optionally, an action can be expressed after the system call (errno #, trap, kill, allow), specifying the desired seccomp behavior. If this is omitted, the default rule action of the policy will be applied (allow for whitelist policies, kill for blacklist policies).

When running a container with the previous policy file on a PowerPC 64-bit platform, the `mknod`, `sched_setscheduler` (`chrt`) and `fchmodat` (`chmod`) system calls will be denied, with mentioned behaviors: `mknod` will return errno 120 without executing, `chrt` will trap and `chmod` will result in the process executing it being killed. On PowerPC platforms, only `mknod` will be denied, resulting in the process being killed. All other system calls will be allowed.

#### Notes:

- Containers can still be started without loading a seccomp policy file, simply by omitting the `lxc.seccomp` line in the config file. No seccomp policy is loaded by default.
- If a container process has a seccomp policy loaded, this can be seen in `/proc/PID/status`, on the `seccomp` line. This line will contain "Seccomp: 2" when using seccomp filter (mode 2). "Seccomp: 0" means there is no seccomp policy in effect.
- Seccomp policies of a process are automatically inherited by its children.
- Currently LXC supports only system call based filtering, with no support for system call arguments.
- The performance degradation of the processes running with a seccomp policy applied is directly proportional with the policy file size: normally, the system calls are listed as rules in the BPF filter program, and they all need to be parsed and matched at each system call. The longer the list, the more time this will take.
- The LXC package comes shipped with a set of example policy files which can be found at `/share/doc/lxc/examples/seccomp-*`. There's also a policy file, [common.seccomp](#), which denies common security syscall threats in the container, such as kernel module manipulation, `kexec` and `open_by_handle_at` (the vector for the [Shocker exploit](#)).

## 10.2.4 Libvirt

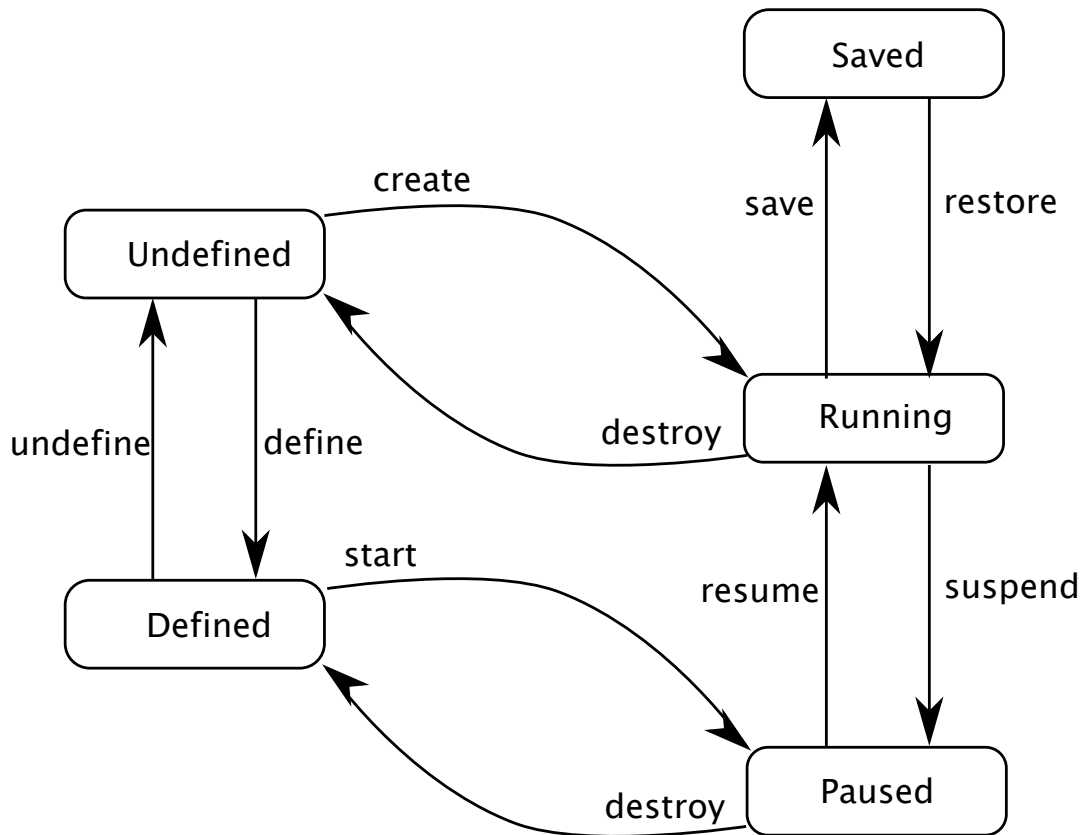
This document is a guide and tutorial to using libvirt on NXP SoCs. Libvirt is an open source toolkit that enables the management of Linux-based virtualization technologies such as KVM/QEMU virtual machines and Linux containers. The goal of the libvirt project (see <https://libvirt.org>) is to provide a stable, standard, hypervisor-agnostic interface for managing *virtualization domains* such as virtual machines and containers. Domains can be remote and libvirt provides full security for managing remote domains over a network. Libvirt is a layer intended to be used as a building block for higher level management tools and applications.

Libvirt provides:

- An interface to remotely manage the lifecycle of virtualization domains – provisioning, start/stop, monitoring
- Support for a variety of hypervisors – KVM/QEMU and Linux Containers are supported in the NXP SDK
- `libvirtd` – a Linux daemon that runs on a target node/system and allows a libvirt management tool to manage virtualization domains on the node
- `virsh` – a basic command shell for managing libvirt domains
- A standard XML format for defining domains

#### Libvirt Domain Lifecycle

Two types of libvirt domains are supported – KVM/QEMU virtual machines and Linux containers. The following state diagram illustrates the lifecycle of a domain, the states that domains can be in and the `virsh` commands that move the domain between states.



### Domain States

- **Undefined.** There are two types of domains – persistent and transient domains. All domains begin in the *undefined* state where they are defined in XML definition file, and libvirt is unaware of them.
- **Defined.** Persistent domains begin with being *defined*. This adds the domain to libvirt, but it is not running. This state can also be conceptually thought of as *stopped*. The output of `virsh list --all` shows the domain as being *shut off*.
- **Running.** The *running* state is the normal state of an active domain after it has been started. The `start` command is used to move persistent domains into this state. Transient domains go from being undefined to *running* through the `create` command.
- **Paused.** The domain execution has been suspended. The domain is unaware of being in this state.
- **Saved.** The domain state has been saved and could be restored again.

### Libvirt URIs

Because libvirt supports managing multiple types of virtualization domains (possibly remote) it uses uniform resource identifiers (URIs) to describes the target *node* to manage and the type of domain being managed.

An URI is specified when tools such as `virsh` makes a connection to a target node running libvirtd. Two types of URIs are supported – QEMU/KVM and LXC.

QEMU/KVM URIs are in the form:

- for a local node: *qemu:///system*
- for a remote node: *qemu[+transport]://[hostname]/system*

Linux containers URIs:

- for a local node: *lxc:///*
- for a remote node: *lxc[+transport]://[hostname]/*

A default URI can be specified using the environment variable *LIBVIRT\_DEFAULT\_URI* or in the */etc/libvirt/libvirtd.conf* config file.

For further information on URIs:

- <https://libvirt.org/uri.html>
- [https://libvirt.org/remote.html#Remote\\_URI\\_reference](https://libvirt.org/remote.html#Remote_URI_reference)

## Virsh

The virsh command is a command line tool provided with the libvirt package for managing libvirt domains. It can be used to create, start, pause, shutdown domains. The general command format is:

```
virsh [OPTION]... <command> <domain> [ARG]...
```

## Libvirt XML

The libvirt XML format is defined at <http://libvirt.org/format.html>.

## Running libvirtd

The libvirtd daemon is installed as part of a libvirt packages installation. By default the target system init scripts should start libvirtd. Running libvirtd on the target system is a pre-requisite to running any management tools such as virsh. The libvirtd daemon can be manually started like this:

```
$ systemctl start libvirtd
```

In some circumstances the daemon may need to be restarted, such as after mounting cgroups or hugetlbfs. Daemon restart can be done like this:

```
$ systemctl restart libvirtd
```

The libvirtd daemon can be configured in */etc/libvirt/libvirtd.conf*. The file is self-documented and has detailed comments on the configuration options available.

The libvirt daemon logs data to */var/log/libvirt/*:

- General libvirtd log messages are in: */var/log/libvirt/libvirtd.log*
- QEMU/KVM domain logs are in: */var/log/libvirt/qemu/[domain-name].log*
- LXC domains logs are in: */var/log/libvirt/lxc/[domain-name].log*

The verbosity of logging can be controlled in */etc/libvirt/libvirtd.conf*.

In order to be able to start virtual machines the user used to manage virtual machines need to be added to the `libvirt` group:

```
sudo adduser <USER> libvirt
```

## Examples

### Libvirt KVM/QEMU Examples

#### Virtio Block scenario

1. We begin with a simple QEMU command line in a text file named `kvm_virtio_blk.args`:

```
$ echo "/usr/bin/qemu-system-aarch64 -name kvm_virtio_blk -smp 2 -enable-kvm -m 1024 -nographic
-cpu host -machine type=virt -kernel /boot/Image -serial pty -drive if=virtio,index=0,file=/
root/ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -append 'root=/dev/vda rw
console=ttyAMA0 rootwait earlyprintk'" > kvm_virtio_blk.args
```

*Note:* The serial console is a tty, not a telnet server. The `-name` option is required and specifies the name of the virtual machine.

2. Before defining the domain, the QEMU command line must be converted to libvirt XML format:

```
$ virsh domxml-from-native qemu-argv kvm_virtio_blk.args > kvm_virtio_blk.xml
```

3. Now the domain can be defined:

```
$ virsh define kvm_virtio_blk.xml
Domain kvm_virtio_blk defined from kvm_virtio_blk.xml

$ virsh list --all
Id Name State

- kvm_virtio_blk shut off
```

4. Start the domain. This starts the VM and boots the Linux Guest from the `ubuntu_bionic_arm64_rootfs.ext4.img` image.

```
$ virsh start kvm_virtio_blk
Domain kvm_virtio_blk started

$ virsh list
Id Name State

16 kvm_virtio_blk running
```

5. The `virsh console` command can be used to connect to the console of the running Linux domain.

```
$ virsh console kvm_virtio_blk
Connected to domain kvm_virtio_blk
Escape character is ^]

Ubuntu 16.04.3 LTS localhost ttyAMA0

localhost login: root
Password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.9.62 aarch64)

*Documentation: https://help.ubuntu.com
*Management: https://landscape.canonical.com
*Support: https://ubuntu.com/advantage
```

6. To stop the domain use the *destroy* command:

```
$ virsh destroy kvm_virtio_blk
Domain kvm_virtio_blk destroyed

$ virsh list --all
Id Name State

- kvm_virtio_blk shut off
```

7. To remove the domain from libvirt, use the *undefine* command:

```
$ virsh undefine kvm_virtio_blk
Domain kvm_virtio_blk has been undefined

$ virsh list --all
Id Name State

```

*Note:* One can find the full XML for this configuration in *Annex 1*.

**Virtio Net scenario**

This example uses a *virtio* model NIC card and a tap network backend. The virtual network interface is bridged via a TAP interface to the physical network.

Perform the following steps:

1. Enable virtio networking in the host and guest Linux kernels.
2. On the host, create a bridge to the physical network interface to be used by the virtual network interface in the virtual machine using the *brctl* command. In this example the physical interface being used is *enp1s0*:

```
$ brctl addbr br0
$ ifconfig br0 192.168.1.10 netmask 255.255.248.0
$ ifconfig enp1s0 0.0.0.0
$ brctl addif br0 enp1s0
```

3. Create a *qemu-ifup* script on the host Linux system:

```
#!/bin/sh

#TAP interface will be passed in $1
bridge=br0
guest_device=$1
ifconfig $guest_device 0.0.0.0 up
brctl addif $bridge $guest_device
```

4. Create a *args* file and convert it to the libvirt xml:

```
$ echo "/usr/bin/qemu-system-aarch64 -name kvm_virtio_net -smp 2 -enable-kvm -m 1024 -nographic
-cpu host -machine type=virt -kernel /boot/Image -serial pty -drive if=virtio,index=0,file=/
root/ubuntu_bionic_arm64_rootfs.ext4.img,id=foo,format=raw -netdev tap,id=tap0,script=/root/
qemu-ifup,downscript=no,ifname=tap0 -device virtio-net-pci,netdev=tap0 -append 'root=/dev/vda
rw console=ttyAMA0 rootwait earlyprintk'" > kvm_virtio_net.args

$ virsh domxml-from-native qemu-argv kvm_virtio_net.args > kvm_virtio_net.xml
```



## 5. Define and start the domain. Check if the virtual network interface is created.

```
$ virsh define kvm_virtio_net.xml
Domain kvm_virtio_net defined from kvm_virtio_net.xml

$ virsh start kvm_virtio_net
Domain kvm_virtio_net started

$ virsh console kvm_virtio_net
Connected to domain kvm_virtio_net
Escape character is ^]

Ubuntu 16.04.3 LTS localhost ttyAMA0

localhost login: root
Password:

$ dmesg | grep virtio_net
[4.121280] virtio_net virtio1 enp0s2: renamed from eth0

$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: enp0s2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
 link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1
 link/sit 0.0.0.0 brd 0.0.0.0
4: docker0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
 link/ether 02:42:81:50:d5:f5 brd ff:ff:ff:ff:ff:ff
 inet 172.17.0.1/16 scope global docker0
 valid_lft forever preferred_lft forever
```

The libvirt XML generated and used in this scenario differs from the previous one by the following lines:

```
<qemu:commandline>
 <qemu:arg value='-netdev' />
 <qemu:arg value='tap,id=tap0,script=/root/qemu-ifup,downscript=no,ifname=tap0' />
 <qemu:arg value='-device' />
 <qemu:arg value='virtio-net-pci,netdev=tap0' />
</qemu:commandline>
```

**Note:** Currently libvirt has no support for PCI transport, but it can be used using passthrough QEMU command line arguments (as seen in the previous xml).

**Note:** If you get the following error when starting the domain please use the steps from this [thread](#) to fix it.

```
could not open /dev/net/tun: Operation not permitted
```

**Note:** One can find the full XML for this configuration in *Annex 2*.

### Virtio Block Dataplane

*Virtio-blk-dataplane* was developed for high performance disk I/O, especially for high IOPS devices. QEMU performs the disk I/O in a dedicated thread that is optimized for I/O performance.

Even though the scenario can use also a block device on the Linux host, the next steps will show how to implement this using a raw disk file.

*Note:* A direct translation between the qemu args are not possible using virsh that is why in this example we will start from the XML used in the previous scenario and build on it.

1. Create the raw disk file:

```
$ dd if=/dev/zero of=/root/fake-dev0-backstore.img bs=1M count=300
```

2. Copy the libvirt XML file from the previous example:

```
$ cp kvm_virtio_net.xml kvm_virtio_blk_dataplane.xml
```

3. Change the *name* and *uuid* of the new domain. Define the number of *IOTthreads* to be assigned to the domain and used by the new storage device. Add the storage disk and assign it to the *iothread='1'*.

```
$ diff kvm_virtio_blk_dataplane.xml kvm_virtio_net.xml
```

```
2,3c2,3
< <name>kvm_virtio_blk_dataplane</name>
< <uuid>5c30747a-a2c9-485e-b814-2a503fef8657</uuid>

> <name>kvm_virtio_net</name>
> <uuid>5c30747a-a2c9-485e-b814-2a503fef8653</uuid>
22d21
< <iothreads>1</iothreads>
29,34d27
< </disk>
< <disk type='file' device='disk'>
< <driver name='qemu' type='raw' cache='none' io='native' iothread='1'/>
< <source file='/root/fake-dev0-backstore.img'/>
< <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
< <target dev='vdb' bus='virtio'/>
```

4. Start the new domain and check if virtio-blk-dataplane works properly.

```
$ virsh define kvm_virtio_blk_dataplane.xml
Domain kvm_virtio_blk_dataplane defined from kvm_virtio_blk_dataplane.xml

$ virsh start kvm_virtio_blk_dataplane
Domain kvm_virtio_blk_dataplane started

After the guest boots, the virtual disk is visible as a block device with the name vdb.
$ virsh console kvm_virtio_blk_dataplane

root@localhost:~# ls -la /dev/vd*
brw-rw---- 1 root disk 254, 0 Aug 23 12:00 /dev/vda
brw-rw---- 1 root disk 254, 16 Aug 23 12:00 /dev/vdb

We can also check if the IOThread is correctly assigned to the domain.
$ virsh iothreadinfo kvm_virtio_blk_dataplane
IOThread ID CPU Affinity

1 0-7
```

*Note:* One can find the full XML for this configuration in *Annex 3*.

## Libvirt KVM/QEMU FAQ

1. What if I get "error: XML error: No PCI buses available" error when trying to convert QEMU arguments to XML?

If you are using a 32 bit QEMU and you are trying a command like:

```
echo -e '/usr/bin/qemu-system-arm -enable-kvm -name demo1 -enable-kvm -m 512 -nographic -cpu
host -machine type=virt -mem-path /dev/hugepages/libvirt/qemu -kernel /media/ram/zImage -
initrd /media/ram/fsl-image-core-ls1021atwr.ext2.gz -append "root=/dev/ram rw
console=ttyAMA0,115200" -serial pty' >> demo.args
root@localhost:~# virsh domxml-from-native qemu-argv demo.args > demo1.xml
error: XML error: No PCI buses available
```

### NOTE

The above QEMU command is just an example, your command can be completely different.

The reason the above mentioned command does not work is that `domxml-from-native` expects that the suffix of the `qemu-system-` to be a canonical architecture name and `arm` is not. For ARM32 bit, little endian, the canonical name is `armv7l`. The solution could be either to manually create the xml file or to create a symbolic link `qemu-system-armv7l` to point to `qemu-system-arm` and then use the symbolic link in `demo.args`:

```
echo -e '/usr/bin/qemu-system-armv7l -enable-kvm -name demo1 -enable-kvm -m 512 -nographic -cpu
host -machine type=virt -mem-path /dev/hugepages/libvirt/qemu -kernel /media/ram/zImage -initrd /
media/ram/fsl-image-core-ls1021atwr.ext2.gz -append "root=/dev/ram rw console=ttyAMA0,115200" -
serial pty' >> demo.args
```

## Libvirt LXC Examples

### Basic Example

The following example shows the lifecycle of a simple LXC libvirt domain called `lxc_basic`.

1. Confirm the host Linux configuration. Begin by confirming that the host kernel is configured correctly and that rootfs setup such as mounting cgroups has been done. This can be done with the `lxc-checkconfig` command.
2. Create a libvirt XML file defining the container. The example below shows a very simple container defined in `lxc_basic.xml` that runs the command `/bin/sh` and has a console:

```
$ cat lxc_basic.xml
<domain type='lxc'>
 <name>lxc_basic</name>
 <memory>500000</memory>
 <os>
 <type>exe</type>
 <init>/bin/sh</init>
 </os>
 <devices>
 <console type='pty' />
 </devices>
</domain>
```

```
$ virsh -c lxc:/// define lxc_basic.xml
Domain lxc_basic defined from lxc_basic.xml
```

```
$ virsh -c lxc:/// list --all
Id Name State

- lxc_basic shut off
```

```
$ virsh -c lxc:/// start lxc_basic
Domain lxc_basic started

$ virsh -c lxc:/// console lxc_basic
Connected to domain lxc_basic
Escape character is ^]
#ps -ef
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 13:14 ? 00:00:00 /bin/sh
root 3 1 0 13:14 ? 00:00:00 ps -ef
```

**Note:** The processes inside the container are running in a separate namespace, hence the different process hierarchy since no network configuration for the domain is explicitly specified, all networking interfaces are shared with the host (all the other interfaces are present too - br0 is mentioned as an example) since no filesystem configuration is specified for the domain, the filesystem is shared with the host— all host mounts are present in the container as well.

### Further Information

Libvirt is an open source project and a great deal of technical and usage information is available on the [libvirt.org](http://libvirt.org) website:

Additional references:

- Architecture: <http://libvirt.org/intro.html>
- Deployment: <http://libvirt.org/deployment.html>XML
- Format: <http://libvirt.org/format.html>
- Virsh command reference: <http://linux.die.net/man/1/virsh>
- User generated content: [http://wiki.libvirt.org/page/Main\\_Page](http://wiki.libvirt.org/page/Main_Page)

Mailing Lists. There are three libvirt mailing lists available which can be subscribed to. Archives of the lists are also available:

- <https://www.redhat.com/archives/libvir-list>
- <https://www.redhat.com/archives/libvirt-users>
- <https://www.redhat.com/archives/libvirt-announce>

### Annex 1: kvm\_virtio\_blk.xml

```
<domain type='kvm'>
 <name>kvm_virtio_blk</name>
 <uuid>b8ec80c1-4fd6-4e08-aec7-02150fab316d</uuid>
 <memory unit='KiB'>1048576</memory>
 <currentMemory unit='KiB'>1048576</currentMemory>
 <vcpu placement='static'>2</vcpu>
 <os>
 <type arch='aarch64' machine='virt'>hvm</type>
 <kernel>/boot/Image</kernel>
 <cmdline>root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk</cmdline>
 </os>
 <features>
 <gic version='3' />
 </features>
 <cpu mode='custom' match='exact'>
 <model fallback='allow'>host</model>
 </cpu>
 <clock offset='utc' />
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>destroy</on_crash>
```

```
<devices>
 <emulator>/usr/bin/qemu-system-aarch64</emulator>
 <disk type='file' device='disk'>
 <driver name='qemu' type='raw'>
 <source file='/root/ubuntu_bionic_arm64_rootfs.ext4.img'>
 <target dev='vda' bus='virtio'>
 </disk>
 <controller type='pci' index='0' model='pcie-root'>
 <controller type='pci' index='1' model='dmi-to-pci-bridge'>
 <controller type='pci' index='2' model='pci-bridge'>
 <serial type='pty'>
 <target port='0'>
 </serial>
 <console type='pty'>
 <target type='serial' port='0'>
 </console>
 <memballoon model='none'>
</devices>
</domain>
```

## Annex 2: kvm\_virtio\_net.xml

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
 <name>kvm_virtio_net</name>
 <uuid>5c30747a-a2c9-485e-b814-2a503fef8653</uuid>
 <memory unit='KiB'>1048576</memory>
 <currentMemory unit='KiB'>1048576</currentMemory>
 <vcpu placement='static'>2</vcpu>
 <os>
 <type arch='aarch64' machine='virt'>hvm</type>
 <kernel>/boot/Image</kernel>
 <cmdline>root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk</cmdline>
 </os>
 <features>
 <gic version='3'>
 </features>
 <cpu mode='custom' match='exact'>
 <model fallback='allow'>host</model>
 </cpu>
 <clock offset='utc'>
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>destroy</on_crash>
 <devices>
 <emulator>/usr/bin/qemu-system-aarch64</emulator>
 <disk type='file' device='disk'>
 <driver name='qemu' type='raw'>
 <source file='/root/ubuntu_bionic_arm64_rootfs.ext4.img'>
 <target dev='vda' bus='virtio'>
 </disk>
 <controller type='pci' index='0' model='pcie-root'>
 <controller type='pci' index='1' model='dmi-to-pci-bridge'>
 <controller type='pci' index='2' model='pci-bridge'>
 <serial type='pty'>
 <target port='0'>
 </serial>
 <console type='pty'>
 <target type='serial' port='0'>
 </console>
```

```

 <memballoon model='none' />
</devices>
<qemu:commandline>
 <qemu:arg value='-netdev' />
 <qemu:arg value='tap,id=tap0,script=/root/qemu-ifup,downscript=no,ifname=tap0' />
 <qemu:arg value='-device' />
 <qemu:arg value='virtio-net-pci,netdev=tap0' />
</qemu:commandline>
</domain>

```

### Annex 3: kvm\_virtio\_blk\_dataplane.xml

```

<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
 <name>kvm_virtio_blk_dataplane</name>
 <uuid>5c30747a-a2c9-485e-b814-2a503fef8657</uuid>
 <memory unit='KiB'>1048576</memory>
 <currentMemory unit='KiB'>1048576</currentMemory>
 <vcpu placement='static'>2</vcpu>
 <os>
 <type arch='aarch64' machine='virt'>hvm</type>
 <kernel>/boot/Image</kernel>
 <cmdline>root=/dev/vda rw console=ttyAMA0 rootwait earlyprintk</cmdline>
 </os>
 <features>
 <gic version='3' />
 </features>
 <cpu mode='custom' match='exact'>
 <model fallback='allow'>host</model>
 </cpu>
 <clock offset='utc' />
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>destroy</on_crash>
 <iothreads>1</iothreads>
 <devices>
 <emulator>/usr/bin/qemu-system-aarch64</emulator>
 <disk type='file' device='disk'>
 <driver name='qemu' type='raw' />
 <source file='/root/ubuntu_bionic_arm64_rootfs.ext4.img' />
 <target dev='vda' bus='virtio' />
 </disk>
 <disk type='file' device='disk'>
 <driver name='qemu' type='raw' cache='none' io='native' iothread='1' />
 <source file='/root/fake-dev0-backstore.img' />
 <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x00' />
 <target dev='vdb' bus='virtio' />
 </disk>
 <controller type='pci' index='0' model='pcie-root' />
 <controller type='pci' index='1' model='dmi-to-pci-bridge' />
 <controller type='pci' index='2' model='pci-bridge' />
 <serial type='pty'>
 <target port='0' />
 </serial>
 <console type='pty'>
 <target type='serial' port='0' />
 </console>
 <memballoon model='none' />
 </devices>
 <qemu:commandline>

```

```

 <gemu:arg value='-netdev' />
 <gemu:arg value='tap,id=tap0,script=/root/qemu-ifup,downscript=no,ifname=tap0' />
 <gemu:arg value='-device' />
 <gemu:arg value='virtio-net-pci,netdev=tap0' />
 </gemu:commandline>
</domain>

```

## 10.3 Docker Containers

### 10.3.1 Introduction to Docker Containers

#### 10.3.1.1 Overview

This section is a guide and tutorial to building and using Docker Containers. Docker Containers are only available on ARM64 platforms, with the exception of LS1043 Big Endian.

Docker is a different set of userspace tools implementing Linux containers and focusing on a different set of use cases. The highlights of this open source project are ease of use, shared contributions and fast deployment. In the Docker ecosystem, containers are application environment packages, which can be easily distributed and developed collaboratively, and are guaranteed to be reproducible on any supporting platform, from the development stage to production. Currently, Docker containers are mainly targeting cloud environments.

Docker can be viewed as a set of separate components:

- **Images** - the "build" component of Docker. These are read-only copies of container root filesystems, consisting of the designed application and its userspace dependencies. For example, an image can contain an Ubuntu application, an Apache server and a user web app. This image can be used to get a webserver running.
- **Registries** - the "distribution" component of Docker. These are public or private stores where users can upload / download images. The images are versioned, and are built from layers. When sharing images, the layers are first downloaded separately, and the image is assembled at runtime. Each layer corresponds to a specific user commit. Images can also be built using buildfiles. The most representative registry example is the [Docker Hub](#). The current Docker installation does not support registry configuration.
- **Containers** - the "run" component of Docker. These are very similar to the containers provided by the LXC package. The main difference is that Docker containers use an overlay filesystem as container support. The layers are taken as is from the image and marked read-only, with a topmost read-write layer on top. This means that no container makes any persistent changes to the image by default - these need to be explicitly committed by the user when the environment is in the desired state. Docker containers are designed to work as application containers by default.

Docker uses a [client-server architecture](#). The client takes the user commands and talks to a daemon, which does the entire container management work. A Linux host running the daemon is called a Docker Host. The client and daemon can run on the same machine, or on different ones, communicating through sockets or a RESTful API.

The [Docker official page](#) advertises a set of use cases, mostly relevant in cloud environments: continuous integration, continuous delivery, devops, big data and infrastructure optimization. These can be easily adapted to embedded distributions as well. As for the containers themselves, the [Linux Containers](#) chapter use cases apply, with a focus on ease of use, fast deployment and distributed usage.

### 10.3.2 Docker How To's

#### 10.3.2.1 Running a webserver container

The following article describes the necessary steps to deploy a web server service using a Docker container. This is based on downloading a prepared image from the Docker hub and using it to start a container.

1. Verify if the docker daemon is running. Make sure that the board has Internet access - this will be required to download the image from the Docker Hub. The daemon will configure a Linux bridge for the containers with a private network and NAT. One can verify if the docker daemon is running by using one of the following commands:

```
$ docker info
$ docker version
```

In this case, the docker daemon is configured to start at boot time, but if for any reason the daemon is not running just issue the following command:

```
root@localhost:~# dockerd
```

2. You can search the registry for available **arm64** images, or using any other keyword.

```
root@localhost:~# docker search arm64
```

| NAME                                    | DESCRIPTION                                   | STARS | OFFICIAL | AUTOMATED |
|-----------------------------------------|-----------------------------------------------|-------|----------|-----------|
| ericvh/arm64-ubuntu                     | Base image for arm64 (armv8 aka aarch64) U... | 6     |          |           |
| owlab/alpine-arm64                      | This is Alpine Linux for arm64 (or aarch64)   | 3     |          |           |
| necrose99/gentoo-arm64                  | Arm64 with qemu-arm64 static AMD64 host h...  | 1     |          |           |
| [OK]                                    |                                               |       |          |           |
| mickaelguene/arm64-debian               | Arm64 debian base with umeq install so you... | 1     |          |           |
| [OK]                                    |                                               |       |          |           |
| markusk/arm64-crosscompile              | A debian image with the necessary tools in... | 1     |          |           |
| [OK]                                    |                                               |       |          |           |
| snappcraft/zesty-arm64                  | Docker image for building Ubuntu snaps        | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| mickaelguene/arm64-debian-jenkins-slave | arm64 with java and sshd with umeq so you ... | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| containerstack/alpine-arm64             | Alpine Linux (arm64/aarch64) Docker image     | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| arm64el/helloworld-arm64el              | hello world for arm64 el platform             | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| arm64el/busybox-arm64el                 | busybox image for arm64                       | 0     |          | [OK]      |
| eqw3rty/minecraft-server-arm64          | Dockerized Minecraft server for arm64         | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| arm64el/unshare-arm64el                 | unshare image for arm64el platform            | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| mickaelguene/arm64-debian-dev           | arm64 debian images with development tool ... | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| necrose99/gentoo-arm64-chroot           | base Gentoo AMD64 + ARM64 CHROOT volume. ...  | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| marcust/jessie-arm64-rust               | Debian Jessie (arm64) image containing a R... | 0     |          |           |
| ip4368/node-arm64                       | Node.js is a JavaScript-based platform for... | 0     |          |           |
| marcust/bionic-arm64-rust               | Ubuntu bionic (arm64) image containing a R... | 0     |          |           |
| snappcraft/bionic-arm64                 | Docker image for building Ubuntu snaps        | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| jefby/arm64                             | arm64 develop                                 | 0     |          |           |
| dil001/nginx-arm64                      | These are the arm64 version of the officia... | 0     |          |           |
| knjcode/arm64-node                      | arm64-compatible Docker base image with No... | 0     |          |           |
| parity/rust-arm64                       | RUST for GitLab CI runner (ARM64 architect... | 0     |          |           |
| [OK]                                    |                                               |       |          |           |
| thenatureofsoftware/mc-arm64            | Minio client for arm64                        | 0     |          |           |
| thenatureofsoftware/ubuntu-arm64        | Ubuntu slim images for arm64                  | 0     |          |           |
| dil001/fluentd-arm64                    | arm64 fork of the offical docker images       | 0     |          |           |



3. In this example, `goriq/arm64-ubuntu` is used. It is a standard Ubuntu compiled for ARM64, with a `lighttpd` webserver installed and with a home page configured to display some information on the board, processes and networking in the container. First download the image.

```
root@localhost:~# docker pull goriq/arm64-ubuntu
Using default tag: latest
latest: Pulling from goriq/arm64-ubuntu
a3ed95caeb02: Pull complete
9025035f8d16: Pull complete
d54663dfcaf9: Pull complete
b940f6a4f33c: Pull complete
688957367bc4: Pull complete
88ca67eab938: Pull complete
f5f1c1a40562: Pull complete
688957367bc4: Pull complete
88ca67eab938: Pull complete
f5f1c1a40562: Pull complete
357cdf8f1a01: Pull complete
de8e5d34ebd8: Pull complete
811aa6d4eba3: Pull complete
0dc75b6c54d0: Pull complete
654cadd8a53b: Pull complete
40d300e17719: Pull complete
ce42abd87d1e: Pull complete
Digest: sha256:eaef3a08336f59155e6cfb61bf55688711214561ddf00817b5c848211ac66b00
Status: Downloaded newer image for goriq/arm64-ubuntu:latest
```

You can check the image is available using `docker images`:

```
root@localhost:~# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
goriq/arm64-ubuntu latest 903eae3b724 12 months ago 326.4 MB
root@localhost:~#
```

4. Start a container using the following command:

```
root@localhost:~# docker run -d -p 30081:80 --name=sandbox1 \
-h sandbox1 goriq/arm64-ubuntu \
bash -c "lighttpd -f /etc/lighttpd/lighttpd.conf -D"
```

- `run` - create and start the container. Optionally, download the image if not available on the host.
- `-d` - start the container as a daemon.
- `-p 30081:80` - forward port 80 in the container to port 30081 on the board.
- `--name=sandbox1` - the name of the container (as visible to Docker).
- `-h sandbox1` - the hostname inside the container.
- `goriq/arm64-ubuntu` - the base image for the container.
- `bash -c "lighttpd -f /etc/lighttpd/lighttpd.conf -D"` - the command to execute as PID 1 in the container.

The command will return a unique SHA for the container. You can check that the webserver is up and running by accessing `http://BOARD_IP:30081/` from a browser. You can also check the container is running using `docker`:

```
root@localhost:~# docker ps -a
```

| CONTAINERID | IMAGE | COMMAND | CREATED | STATUS | PORTS |
|-------------|-------|---------|---------|--------|-------|
| NAMES       |       |         |         |        |       |

|                     |                                |                         |              |                    |
|---------------------|--------------------------------|-------------------------|--------------|--------------------|
| b5b8a45db81c<br>ago | goriq/arm64-ubuntu<br>sandbox1 | "bash -c 'lighttpd -f'" | 16 hours ago | Exited (0)16 hours |
|---------------------|--------------------------------|-------------------------|--------------|--------------------|

##### 5. Stopping and deleting the container are easy operations:

```
root@localhost:~# docker stop sandbox1
sandbox1
root@localhost:~# docker rm sandbox1
sandbox1
```

##### 6. A similar command can be used to delete the image from the board.

```
root@localhost:~# docker rmi goriq/arm64-ubuntu
Untagged: goriq/arm64-ubuntu:latest
Untagged: goriq/arm64-ubuntu@sha256:eaef3a08336f59155e6cfb61bf55688711214561ddf00817b5c848211ac66b00
Deleted: sha256:903eaef3b7240612111df4308f4d598ae1dee14b696a4b01654175b6771520f1
Deleted: sha256:48e73c491543279a59d202470394f0f91acd9b3a8a6f5f9bfa933bc4cf4776a
Deleted: sha256:e21b9d6aa0007e242abb10948b13c93e4471694695a91a47d639f45927f25eb6
Deleted: sha256:7ec2184e81ef396a206e965e6dae42a122c4348dd7cfee1b731aa59931a5ec82
Deleted: sha256:0b081c8c711c2d14522ealb5763e5ead19ab2975e4c28864a0ee2c0942ebae43
Deleted: sha256:b256d9ce72b40a1dc9dfdb13003a44976ba81e4fb31e774e913ed57241424231
Deleted: sha256:e07c8e0adb08295db7e3f2e13f41be622d5b8590575f87813922dd4ef0914e8f
Deleted: sha256:09ec9672e9e6d30855f1274415edf6a023b86764261b6cd88fc2b692f997977d
Deleted: sha256:d29d57006e3df9a03fb3d430183166c9337378404c1ad66db391251ea24592fd
Deleted: sha256:84be8839209cbbecd3b3f064b9593e16d30468d71c788fc3ab8f3125990002bf
Deleted: sha256:09be261c306e6c01756d16c31e2a9d4b638e8d205a068b767cb0a078480633a9
Deleted: sha256:47d9e04c91309d23f8135f579a302c2309b206cb392c42c55ec13b2c26fb317f
Deleted: sha256:8495eed3352e7d2a237f179e3a3a6e449a56821a77e2efd943bc9ccf8d6d964c
Deleted: sha256:423a2c50f96dad2f267bbbe11a8a9efc21e776419fbd618ec1a9a21e918c918b
Deleted: sha256:67629909bfc67e60ba87451caf1f98b375e8b81f21a87bab5f5e2740a78c025b
Deleted: sha256:f821f1edfff4c38033e84024e844e503d5e0e470155c4bd69ec3f0af04f01b6b
Deleted: sha256:837a3e2cff861610e7672192dac0342041c30b2548a3a63a47b92d964a862c8a
Deleted: sha256:129149fe5b4dc97f940c38cd37cfa3fc06bbdc12a8d9d22e4aa3b3e4ff709346
```

# Chapter 11

## NFV OpenStack Compute Node

### 11.1 Network Function Virtualization OpenStack Compute Node

The OpenStack project is an open source cloud computing platform that supports all types of cloud environments.

#### 11.1.1 OpenStack Nova overview

OpenStack provides an Infrastructure-as-a-Service (IaaS) solution through a variety of complementary services. Each service offers an Application Programming Interface (API) that facilitates this integration.

#### 11.1.2 Build OpenStack

This document contains instructions to build the root file system with all the packages required to install the OpenStack Nova compute. OpenStack Nova Newton release component installation will be done after booting the device with this root file system. Those instructions are given in the next section.

Add below line to `<sdk-install path>/build_lx2160ardb/conf/local.conf` file to include Openstack kernel configuration.

```
DELTA_KERNEL_DEFCONFIG_append = " <sdk-install path>/sources/meta-qorik-demos/recipes-kernel/
linux/files/openstack_config"
```

Execute `bitbake` command to generate root file system as shown below:

```
#> bitbake fsl-image-oscompute
```

### 11.2 Board bring up

This section provides the procedure to bring up the LX2160ARDB board. NFV requires the device to boot from a storage medium (USB/SD Card).

#### 11.2.1 Bringing up Openstack Controller (OS) Container

This section provides the steps to prepare the USB disk, like format the disk, create a partition and copying the rootfs onto the partition. These steps have to be done in a linux machine. Insert USB disk in the slot.

List the attached disks by issuing command `fdisk -l` and identify the name of USB disk attached to the server. Check whether the USB is mounted automatically by issuing the command `mount`. If USB is mounted, then make sure that the USB disk is unmounted before doing the below steps.

Delete the existing partitions in USB disk and create a single primary partition as shown below:

```
#> fdisk /dev/<name of the USB disk> # for example sda or sdb or sdc
Then input following
d # delete existing partition
n # add new partition
p # Partition type: primary
1 # Partition number: 1
```

```
<Enter> # First sector: default
<Enter> # Last sector: default
w # write table to disk and exit
```

Format the partition created above as shown below:

```
#> mkfs.ext2 /dev/<name of partition> # for example sda1 or sdb1 or sdc1
```

Mount the partition. Create /mnt directory, if it does not exist.

```
#> cd /#> mkdir mnt
#> mount /dev/sda1 /mnt
```

Copy root file system tar file (fsl-image-oscompute-lx2160ardb.tar.gz) to the mount point (/mnt) and untar it as given in the below steps.

```
#> cp fsl-image-oscompute-lx2160ardb.tar.gz /mnt
#> cd /mnt/
#> tar -zxvf fsl-image-oscompute-lx2160ardb.tar.gz
#> rm fsl-image-oscompute-lx2160ardb.tar.gz
#> cd -
#> umount /mnt
```

Remove the USB disk from linux machine. Now USB disk has the rootfs required to boot LX2160ARDB board.

## 11.2.2 LX2160ARDB board bring up

Insert USB disk in the slot on board.

Before executing the below commands, make sure to copy the respective files to the TFTP server.

Connect console to the board. Power **ON/reset the LX2160 board**.

Flash the images on DEV #0 bank using following commands. Save the environment and boot the board..

```
#> i2c mw 66 50 20;sf probe 0:0;
#> tftp 90000000 firmware.itb;
#> sf erase 800000 +1c000;sf write 90000000 800000 1c000;
#> tftp a0000000 rcw_1900_600_2600_19_5_2.bin;
#> sf erase 0 +200;sf write a0000000 0 200;
#> tftp b0000000 u-boot.bin;
#> sf erase 100000 +100000;sf write b0000000 100000 100000;
#> tftp c0000000 mc.itb;
#> sf erase a00000 +200000;sf write c0000000 a00000 200000;
#> tftp d0000000 dpl-eth.19.dtb;
#> sf erase d00000 +c0000;sf write d0000000 d00000 c0000;
#> tftp e0000000 dpc-warn.dtb;
#> sf erase e00000 +c0000;sf write e0000000 e00000 c0000;
#> tftp f0000000 ppa.itb;
#> sf erase 400000 +100000;sf write f0000000 400000 100000;
#> setenv bootcmd qixis_reset altbank
#> saveenv
#> boot
```

Stop boot prompt at DEV #1 bank, Execute below commands. Detect the USB drive attached to the device. Set the boot arguments to boot from usb drive. Kernel image and the dtb files will be loaded from the usb drive. Set the bootcmd to 'run usbboot'. Save the environment and boot the board.

```
#> setenv bootargs 'console=ttyAMA0,115200 root=/dev/sda1 rw rootdelay=10 earlycon=pl011,mmio32,
0x21c0000 ramdisk_size=0x2000000 default_hugepagesz=2m hugepagesz=2m hugepages=3072 mem=9216M
```

```
pmem=2048M smem=1024M isolcpus=8-15'
#> setenv usbboot 'fsl_mc apply dpl 0x20d00000;usb start;ext2load usb 0:1 0xa0000000 /boot/
uImage;ext2load usb 0:1 0x90000000 /boot/Image-fsl-lx2160a-rdb.dtb;bootm 0xa0000000 - 0x90000000'
#> setenv bootcmd 'run usbboot'
#> saveenv
#> boot
```

LX2160ARDB board boots from USB and login prompt appears. Login as root. The above steps need to be done only once. Next time when the box rebooted, it boots automatically as the u-boot configuration and commands are saved.

## 11.2.3 Installing OpenStack Nova (Newton)

### NOTE

*Make sure that Compute Node is connected to Internet*

After booting the device, download Openstack source form git and build it in the device.

```
#> git clone https://github.com/openstack/nova.git -b newton-eol
#> cd nova
#> pip install -r requirements.txt
#> python setup.py build
#> python setup.py install
#> cd /home/root
```

# Chapter 12

## Power Management

### 12.1 Power Management User Manual

#### Linux SDK for QorIQ Processors

##### Description

QorIQ Processors have features to minimize power consumption at several different levels. All processors support a sleep mode (LPM20). Some processors, such as T1040 and LS1021 also support a deep sleep mode (LPM35).

The following power management features are supported on various QorIQ processors:

- Dynamic power management
- Shutting down unused IP blocks
- Cores support low power modes (such as PW15)
- Processors enter low power state (LPM20, LPM35)
  - LPM20 mode: most part of processor clocks are shut down
  - LPM35 mode: power is removed to cores, cache and IP blocks of the processor such as DIU, eLBC, PEX, eTSEC, USB, SATA, eSDHC etc.
- CPU hotplug: If cores are down at runtime, they will enter low power state.

The wake-up event sources caused quitting from low power mode are listed as below:

- Wake on LAN (WoL) using magic packet
- Wake by MPIC timer or FlexTimer
- Wake by Internal and external interrupts

For more information on a specific processor, refer to processor Reference Manual.

##### Kernel Configure Tree View Options

For ARM platforms

| Kernel Configure Tree View Options                                                                                      | Description                                    |
|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Power management options --><br>[*] Suspend to RAM and standby                                                          | Enable sleep feature                           |
| Device Drivers ---><br>SOC (System On Chip) specific Drivers ---><br>[*] Layerscape Soc Drivers<br>[*] FTM alarm driver | Enable the FTM alarm (FlexTimer module) driver |
| Table continues on the next page...                                                                                     |                                                |

Table continued from the previous page...

| Kernel Configure Tree View Options                                                                                                                                                                                                                  | Description                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| <pre> CPU Power Management  ---&gt; CPU Idle  ---&gt; [*] CPU idle PM support [*]  Ladder governor (for periodic timer tick) -*  Menu governor (for tickless system) ARM CPU Idle Drivers  ---&gt;     [*] Generic ARM/ARM64 CPU idle Driver </pre> | Enable the CPU Idle driver |
| Table continues on the next page...                                                                                                                                                                                                                 |                            |

### Compile-time Configuration Options

| Linux Framework | Hardware Feature  | Platform                                             | Kernel Config      |
|-----------------|-------------------|------------------------------------------------------|--------------------|
| Suspend         | LPM20             | LS1012, LS1021,<br>LS1046, LS1043,<br>LS1088, LS2088 | CONFIG_SUSPEND     |
|                 | wake by Flextimer | LS1012, LS1021,<br>LS1046                            | CONFIG_FTM_ALARM   |
| CPU idle        | PW15              | LS1012, LS1021,<br>LS1046                            | CONFIG_ARM_CPUIDLE |

### Device Tree Binding

| Property | Type | Description |
|----------|------|-------------|
| N/A      | N/A  | N/A         |

### Source Files

The source files are maintained in the Linux kernel source tree.

| Source File                           | Description                                    |
|---------------------------------------|------------------------------------------------|
| drivers/soc/fsl/layercape/ftm_alarm.c | the FTM timer driver worked as a wakeup source |
| drivers/cpuidle/cpuidle-arm.c         | the cpuidle driver for ARM core                |

### Verification in Linux

#### • Cpuidle Driver

The cpuidle driver can switch CPU state according to the idle policy (governor). For more information, please see "Documentation/cpuidle/sysfs.txt" in kernel source code.

```

/* Check the cpuidle driver which is currently used. */
cat /sys/devices/system/cpu/cpuidle/current_driver

```

```
/* Check the following directory to see the detailed statistic information of each state on
each CPU. */
/sys/devices/system/cpu/cpu0/cpuidle/state0/
/sys/devices/system/cpu/cpu0/cpuidle/state1/
```

- CPU SLEEP

```
Cpu can enter sleep which reduce the power consumption dramatically.
echo mem > /sys/power/state
```

- Sleep and Wake up by FTM timer

```
/* Start a FTM timer. It will trigger an interrupt to wake up the system in 5 seconds. */
echo 5 > /sys/devices/platform/soc/29d0000.ftm0/ftm_alarm && echo mem > /sys/power/state
```

## Supporting Documentation

- QorlQ processor reference manuals

# 12.2 Power Management User Manual

## Linux SDK for QorlQ Processors

### Description

QorlQ Processors have features to minimize power consumption at several different levels. All processors support a sleep mode (LPM20). Some processors, such as T1040, LS1021, LS1046, also support a deep sleep mode (LPM35).

The following power management features are supported on various QorlQ processors:

- Dynamic power management
- Shutting down unused IP blocks
- Cores support low power modes (such as PW15)
- Processors enter low power state (LPM20, LPM35)
  - LPM20 mode: most part of processor clocks are shut down
  - LPM35 mode: power is removed to cores, cache and IP blocks of the processor such as DIU, eLBC, PEX, eTSEC, USB, SATA, eSDHC etc.
- CPU hotplug: If cores are down at runtime, they will enter low power state.

The wake-up event sources caused quitting from low power mode are listed as below:

- Wake on LAN (WoL) using magic packet
- Wake by MPIC timer or FlexTimer
- Wake by Internal and external interrupts

For more information on a specific processor, refer to processor Reference Manual.

### Kernel Configure Tree View Options

For ARM platforms



| Kernel Configure Tree View Options                                                                                                                                                                                                       | Description                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| <pre>Power management options --&gt; [*] Suspend to RAM and standby</pre>                                                                                                                                                                | Enable sleep feature                           |
| <pre>Device Drivers ---&gt; SOC (System On Chip) specific Drivers ---&gt; [*] Layerscape Soc Drivers [*] FTM alarm driver</pre>                                                                                                          | Enable the FTM alarm (FlexTimer module) driver |
| <pre>CPU Power Management ---&gt; CPU Idle ---&gt; [*] CPU idle PM support [*] Ladder governor (for periodic timer tick) *- Menu governor (for tickless system) ARM CPU Idle Drivers ---&gt; [*] Generic ARM/ARM64 CPU idle Driver</pre> | Enable the CPU Idle driver                     |
| Table continues on the next page...                                                                                                                                                                                                      |                                                |

### Compile-time Configuration Options

| Linux Framework | Hardware Feature  | Platform               | Kernel Config      |
|-----------------|-------------------|------------------------|--------------------|
| Suspend         | LPM20             | LS1012, LS1021, LS1046 | CONFIG_SUSPEND     |
|                 | wake by Flextimer | LS1012, LS1021, LS1046 | CONFIG_FTM_ALARM   |
| CPU idle        | PW15              | LS1012, LS1021, LS1046 | CONFIG_ARM_CPUIDLE |

### Device Tree Binding

| Property               | Type         | Description                                                       |
|------------------------|--------------|-------------------------------------------------------------------|
| fsl,#rcpm-wakeup-cells | unsigned int | the number of cells in "rcpm-wakeup" except the pointer to "rcpm" |
| rcpm-wakeup            | unsigned int | require if the IP block can work as a wakeup source               |

For processors integrated RCPM

```
rcpm: rcpm@1ee2000 {
 compatible = "fsl,ls1046a-rcpm", "fsl,qoriq-rcpm-2.1";
 reg = <0x0 0x1ee2000 0x0 0x1000>;
 fsl,#rcpm-wakeup-cells = <1>;
};
```

```
ftm0: ftm0@29d0000 {
 compatible = "fsl,ftm-alarm";
 reg = <0x0 0x29d0000 0x0 0x10000>;
 interrupts = <0 86 0x4>;
 big-endian;
 rcpm-wakeup = <&rcpm 0x0 0x20000000>;
 status = "okay";
};
```

Refer to the Linux document: Documentation/devicetree/bindings/soc/fsl/rcpm.txt

**Source Files**

The source files are maintained in the Linux kernel source tree.

| Source File                           | Description                                    |
|---------------------------------------|------------------------------------------------|
| drivers/soc/fsl/layercape/rcpm.c      | the RCPM driver needed by the sleep feature    |
| drivers/soc/fsl/layercape/ftm_alarm.c | the FTM timer driver worked as a wakeup source |
| drivers/cpuidle/cpuidle-arm.c         | the cpuidle driver for ARM core                |

**Verification in Linux**

- Cpuidle Driver

The cpuidle driver can switch CPU state according to the idle policy (governor). For more information, please see "Documentation/cpuidle/sysfs.txt" in kernel source code.

```
/* Check the cpuidle driver which is currently used. */
cat /sys/devices/system/cpu/cpuidle/current_driver

/* Check the following directory to see the detailed statistic information of each state on
each CPU. */
/sys/devices/system/cpu/cpu0/cpuidle/state0/
/sys/devices/system/cpu/cpu0/cpuidle/state1/
```

**Supporting Documentation**

- QorIQ processor reference manuals

# 12.3 CPU Frequency Switching User Manual

**Linux SDK for QorIQ Processors**

**Abbreviations and Acronyms**

DFS: Dynamic Frequency Scaling

**Description**

QorIQ Processors support DFS (Dynamic Frequency Switching) feature, also known as CPU Frequency Switch, which can change the frequency of cores dynamically.

For more information on a specific processor, refer to processor Reference Manual.

| Kernel Configure Tree View Options                                                                                                                                                                                                                                                                                                                                                                        | Description                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| <pre> CPU Power Management --&gt;   CPU Frequency scaling --&gt;     [*] CPU Frequency scaling     &lt;*&gt; CPU frequency translation statistics       Default CPUFreq governor (userspace) --&gt;         *- 'userspace' governor for userspace frequency scaling           ARM CPU frequency scaling drivers --&gt;             &lt;*&gt; CPU frequency scaling driver for Freescale QorIQ SoCs </pre> | Enable the CPU frequency driver |

### Compile-time Configuration Options

| Linux Framework | Hardware Feature | Platform   | Kernel Config                                             |
|-----------------|------------------|------------|-----------------------------------------------------------|
| cpufreq         | DFS              | ALL        | CONFIG_CPU_FREQ,<br>CONFIG_CPU_FREQ_DEFAULT_GOV_USERSPACE |
| cpufreq         | DFS              | Layerscape | CONFIG_QORIQ_CPUFREQ                                      |

### User Space Application

Simply using command "cat" and "echo" can verify this feature.

### Device Tree Binding

| Property                            | Type         | Status   | Description                              |
|-------------------------------------|--------------|----------|------------------------------------------|
| #clock-cells                        | unsigned int | Required | The number of cells in a clock-specifier |
| clocks                              | handle       | Required | Clock source handle                      |
| compatible                          | String       | Required | Compatible strings                       |
| reg                                 | unsigned int | Required | register address range                   |
| Table continues on the next page... |              |          |                                          |

```

clockgen: clocking@1ee1000 {
 compatible = "fsl,ls1012a-clockgen";
 reg = <0x0 0x1ee1000 0x0 0x1000>;
 #clock-cells = <2>;
 clocks = <&sysclk>;
};

```

### Source Files

The driver source is maintained in the Linux kernel source tree.

Table continued from the previous page...

| Source File                     | Description                                  |
|---------------------------------|----------------------------------------------|
| drivers/cpufreq/qoriq_cpufreq.c | CPU frequency scaling driver for qoriq chips |

## Verification in Linux

- CPU frequency mode

In order to test the CPU frequency scaling feature, we need to enable the CPU frequency feature on the menuconfig and choose the USERSPACE governor.  
You can learn more about CPU frequency scaling feature by referring to the kernel documents. They all are put under Documentation/cpu-freq/ directory.  
For example: all the information about governors is put in Documentation/cpu-freq/governors.txt.

Test step:

1. list all the frequencies a core can support (take cpu 0 for example) :  
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_available\_frequencies  
1199999 599999 299999 799999 399999 199999 1066666 533333 266666

2. check the CPU's current frequency  
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_cur\_freq  
1199999

3. change the CPU's frequency we expect:  
# echo 799999 > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_setspeed

You can check the CPU's current frequency again to confirm if the frequency transition is successful.

Please note that if the frequency you want to change to doesn't support by current CPU, kernel will round up or down to one CPU supports.

## 12.4 Thermal Management User Manual

### Description

The thermal management function is based on TMU (Thermal Monitoring Unit).

The driver sets two thresholds for management function. If the CPU temperature crosses the first one (75 C for LS2080. 85 C for other platforms), the driver will trigger CPU frequency limitation auto-scaling according to the temperature trend; If the CPU temperature crosses the second one (85 C for LS2080, 95 C for other platforms, critical for core) the driver will shut down the system.

User could also get current temperature through sysfs interface.

## Specifications

|                   |                                                                             |
|-------------------|-----------------------------------------------------------------------------|
| Target boards:    | T1040RDB, T1042RDB, T1023RDB, T1024RDB, LS1021ATWR, LS1043ARDB, LS2080ARDB. |
| Operating system: | Linux 3.12+                                                                 |

## Kernel Configure Tree View Options (For PowerPC platform)

| Kernel Configure Tree View Options                                                                                                                                                    | Description                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre>Platform support ---&gt;   CPU Frequency scaling ---&gt;     PowerPC CPU frequency scaling drivers ---&gt;       &lt;*&gt; CPU frequency scaling driver for NXP QorIQ SoCs</pre> | Enable CPUfreq driver.                                                                    |
| <pre>Device Drivers ---&gt;   [*] Generic Thermal sysfs driver ---&gt;     [*] generic cpu cooling support     [*] Freescale QorIQ Thermal Monitoring Unit</pre>                      | Enable thermal management framework, cpu cooling device support and QorIQ thermal driver. |

## Kernel Configure Tree View Options (For ARM platform)

| Kernel Configure Tree View Options                                                                                                                                                    | Description                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre>CPU Power Management ---&gt;   CPU Frequency scaling ---&gt;     ARM CPU frequency scaling drivers ---&gt;       &lt;*&gt; CPU frequency scaling driver for NXP QorIQ SoCs</pre> | Enable CPUfreq driver.                                                                    |
| <pre>Device Drivers ---&gt;   [*] Generic Thermal sysfs driver ---&gt;     [*] generic cpu cooling support     [*] Freescale QorIQ Thermal Monitoring Unit</pre>                      | Enable thermal management framework, cpu cooling device support and QorIQ thermal driver. |

## Compile-time Configuration Options

| Option               | Values | Default Value | Description                       |
|----------------------|--------|---------------|-----------------------------------|
| CONFIG_QORIQ_CPUFREQ | y/n    | n             | Enable QorIQ CPUfreq driver       |
| CONFIG_THERMAL       | y/m/n  | n             | Enable thermal management support |
| CONFIG_CPU_THERMAL   | y/m/n  | n             | Enable cpu cooling device support |
| CONFIG_QORIQ_THERMAL | y/m/n  | n             | Enable QorIQ thermal driver       |

## Device Tree Binding

```
tmu: tmu@f0000 {
 compatible = "fsl,qoriq-tmu";
 reg = <0xf0000 0x1000>;
 interrupts = <18 2 0 0>;
```

```
fsl,tmu-range = <0x000a0000 0x00090026 0x0008004a 0x0001006a>;
fsl,tmu-calibration = <0x00000000 0x00000025
 0x00000001 0x00000028
 0x00000002 0x0000002d
 0x00000003 0x00000031
 0x00000004 0x00000036
 0x00000005 0x0000003a
 0x00000006 0x00000040
 0x00000007 0x00000044
 0x00000008 0x0000004a
 0x00000009 0x0000004f
 0x0000000a 0x00000054

 0x00010000 0x0000000d
 0x00010001 0x00000013
 0x00010002 0x00000019
 0x00010003 0x0000001f
 0x00010004 0x00000025
 0x00010005 0x0000002d
 0x00010006 0x00000033
 0x00010007 0x00000043
 0x00010008 0x0000004b
 0x00010009 0x00000053

 0x00020000 0x00000010
 0x00020001 0x00000017
 0x00020002 0x0000001f
 0x00020003 0x00000029
 0x00020004 0x00000031
 0x00020005 0x0000003c
 0x00020006 0x00000042
 0x00020007 0x0000004d
 0x00020008 0x00000056

 0x00030000 0x00000012
 0x00030001 0x0000001d>;
};
```

## Source Files

The driver source is maintained in the Linux kernel source tree.

| Source File                     | Description           |
|---------------------------------|-----------------------|
| drivers/thermal/qoriq_thermal.c | QoriQ thermal driver. |

## Verification in Linux

**There are two parts for verification: management and monitor.**

[Management:]

1. When CPU temperature cross the first threshold, CPU frequency may be reduced by changing frequency limitation, use the following command to check the current frequency:

```
~$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

2. When CPU temperature cross the first threshold, system will shutdown.

[Monitor:]

```
You can manually read the thermal interfaces in sysfs:
~$ cat /sys/class/thermal/thermal_zone0/temp
35000
This means the current temperature is 35 C.
```

## 12.5 System Monitor

### 12.5.1 Power Monitor User Manual

#### Description

There are two methods currently we can use to measure the power consumption which are called online and offline power monitoring respectively. The difference between them is that offline power monitoring support measuring power consumption during sleep or deep sleep.

The Power Monitor can be supported on P4080DS/P5020DS/P5040DS/T4240QDS/LS1043QDS/LS1046QDS/LS1088QDS/LS2088QDS board.

This User guide uses the LS240QDS board as an example.

#### Online Power Monitoring

The Lm-sensors tool ( download from <http://dl.lm-sensors.org/lm-sensors/releases>) will be used to read the power/temperature from on-boards sensors. The drivers vary from sensor to sensor. Basically they would be INA220, ZL6100 and ADT7461 etc.

The device driver support either a built-in kernel or module loading.

#### Kernel Configure Tree View Options

| Option                                                                                                                                                                                                                                                                  | Description                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| <pre>Device Drivers ---&gt; &lt;*&gt; Hardware Monitoring support ---&gt;     &lt;*&gt;   Texas Instruments INA219 and compatibles</pre>                                                                                                                                | Enables INA220                          |
| <pre>Device Drivers ---&gt;     [*]   Enable compatibility bits for old user-space     &lt;*&gt;   I2C device interface     [*]   Autoselect pertinent helper modules     I2C Hardware Bus support ---&gt;         &lt;*&gt; MPC107/824x/85xx/512x/52xx/83xx/86xx</pre> | Enables I2C block device driver support |
| <pre>Device Drivers ---&gt;     &lt;*&gt;   I2C bus multiplexing support     Multiplexer I2C Chip support ---&gt;         &lt;*&gt; Philips PCA954x I2C Mux/switches</pre>                                                                                              | Enables I2C bus multiplexing PCA9547    |

#### Compile-time Configuration Options

| Option                 | Values | Default Value | Description                      |
|------------------------|--------|---------------|----------------------------------|
| CONFIG_I2C_MPC         | y/n    | y             | Enable I2C bus protocol          |
| SENSORS_INA2XX         | y/n    | y             | Enables INA220                   |
| CONFIG_I2C_MUX_PCA954x | y/n    | y             | Enables I2C multiplexing PCA9547 |

## Device Tree Binding

| Property   | Type    | Status   | Description                       |
|------------|---------|----------|-----------------------------------|
| compatible | String  | Required | "Philips,pca9547" for pca9547     |
| reg        | integer | Required | reg = <0x77>                      |
| compatible | String  | Required | "ti,ina220" for ina220            |
| reg        | integer | Required | reg = <the i2c address of ina220> |

```

Default node:
 i2c@118000 {
 pca9547@77 {
 compatible = "philips,pca9547";
 reg = <0x77>;
 #address-cells = <1>;
 #size-cells = <0>;

 channel@2 {
 #address-cells = <1>;
 #size-cells = <0>;
 reg = <0x2>;

 ina220@40 {
 compatible = "ti,ina220";
 reg = <0x40>;
 shunt-resistor = <1000>;
 };

 ina220@41 {
 compatible = "ti,ina220";
 reg = <0x41>;
 shunt-resistor = <1000>;
 };

 ina220@44 {
 compatible = "ti,ina220";
 reg = <0x44>;
 shunt-resistor = <1000>;
 };

 ina220@45 {
 compatible = "ti,ina220";
 reg = <0x45>;
 shunt-resistor = <1000>;
 };

 ina220@46 {
 compatible = "ti,ina220";

```



```

 reg = <0x46>;
 shunt-resistor = <1000>;
 };

 ina220@47 {
 compatible = "ti,ina220";
 reg = <0x47>;
 shunt-resistor = <1000>;
 };
};
};

```

## Source Files

The driver source is maintained in the Linux kernel source tree.

| Source File                         | Description    |
|-------------------------------------|----------------|
| drivers/i2c/muxes/i2c-mux-pca954x.c | PCA9547 driver |
| drivers/hwmon/ina2xx.c              | ina220 driver  |

## Test Procedure

Do the following to validate under the kernel

1. The bootup information is displayed:

```

.....
i2c /dev entries driver
mpc-i2c ffe118000.i2c: timeout 1000000 us
mpc-i2c ffe118100.i2c: timeout 1000000 us
mpc-i2c ffe119000.i2c: timeout 1000000 us
mpc-i2c ffe119100.i2c: timeout 1000000 us
i2c i2c-0: Added multiplexed i2c bus 6
i2c i2c-0: Added multiplexed i2c bus 7
i2c i2c-0: Added multiplexed i2c bus 8
i2c i2c-0: Added multiplexed i2c bus 9
i2c i2c-0: Added multiplexed i2c bus 10
i2c i2c-0: Added multiplexed i2c bus 11
i2c i2c-0: Added multiplexed i2c bus 12
i2c i2c-0: Added multiplexed i2c bus 13
pca954x 0-0077: registered 8 multiplexed busses for I2C mux pca9547
ina2xx 8-0040: power monitor ina220 (Rshunt = 1000 uOhm)
ina2xx 8-0041: power monitor ina220 (Rshunt = 1000 uOhm)
ina2xx 8-0045: power monitor ina220 (Rshunt = 1000 uOhm)
ina2xx 8-0046: power monitor ina220 (Rshunt = 1000 uOhm)
ina2xx 8-0047: power monitor ina220 (Rshunt = 1000 uOhm)
ina2xx 8-0044: power monitor ina220 (Rshunt = 1000 uOhm)
.....

```

```

root@LS1046ARDB:~# sensors
ina220-i2c-0-40
Adapter: 2180000.i2c
in0: +0.01 V
in1: +1.04 V
power1: 6.82 W
curr1: +6.48 A

adt7461-i2c-0-4c

```

```
Adapter: 2180000.i2c
temp1: +29.0°C (low = +0.0°C, high = +85.0°C)
 (crit = +85.0°C, hyst = +75.0°C)
temp2: +47.8°C (low = +0.0°C, high = +85.0°C)
 (crit = +85.0°C, hyst = +75.0°C)
```

#### NOTE

Please make sure to include the "sensors" command in your rootfs

## 12.5.2 Thermal Monitor User Manual

### Description

The Temperature Monitoring function is provided by the chip ADT7461.

This driver exports the values of Temperature to SYSFS. The user space lm-sensors tools can get and display these values.

### Kernel Configure Tree View Options

| Kernel Configure Tree View Options                                                                                                                                              | Description                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| <pre>Device Drivers  ---&gt;   [*] Hardware Monitoring support  ---&gt;     [*] National Semiconductor LM90 and compatibles</pre>                                               | Enable thermal monitor chip driver like ADT7461. |
| <pre>Device Drivers  ---&gt;   &lt;*&gt; I2C bus multiplexing support  ---&gt;     Multiplexer I2C Chip support  ---&gt;       &lt;*&gt; Philips PCA954x I2C Mux/switches</pre> | Enable I2C PCA954x multiplexer support           |

### Compile-time Configuration Options

| Option                 | Values | Default Value | Description                         |
|------------------------|--------|---------------|-------------------------------------|
| CONFIG_HWMON           | y/m/n  | n             | Enable Hardware Monitor             |
| CONFIG_SENSORS_LM90    | y/m/n  | n             | Enable ATD7461 driver               |
| CONFIG_I2C_MUX         | y/m/n  | n             | Enable I2C bus multiplexing support |
| CONFIG_I2C_MUX_PCA954x | y/m/n  | n             | Enable PCA954x driver               |

### Device Tree Binding

```
adt7461@4c {
 compatible = "adi,adt7461";
 reg = <0x4c>;
};

pca9547@77 {
 compatible = "philips,pca9547";
 reg = <0x77>;
};
```

## Source Files

The driver source is maintained in the Linux kernel source tree.

| Source File                 | Description                   |
|-----------------------------|-------------------------------|
| drivers/hwmon/hwmon.c       | Linux hwmon subsystem support |
| drivers/hwmon/lm90.c        | ADT7461 chip driver           |
| drivers/i2c/i2c-mux.c       | I2C bus multiplexing support  |
| drivers/i2c/muxes/pca954x.c | PCA954x chip driver           |

## Verification in Linux

There are two ways to get temperature results.

```
1. You can manually read the thermal interfaces in sysfs:
~$ ls /sys/class/hwmon/hwmon1/devices
alarms temp1_crit temp1_min_alarm temp2_max_alarm
driver temp1_crit_alarm temp2_crit temp2_min
hwmon temp1_crit_hyst temp2_crit_alarm temp2_min_alarm
modalias temp1_input temp2_crit_hyst temp2_offset
name temp1_max temp2_fault uevent
power temp1_max_alarm temp2_input update_interval
subsystem temp1_min temp2_max

~$ cat /sys/class/hwmon/hwmon1/devices/temp1_input
29000

2. You can use lm_sensors tools as follows.
~ # sensors

adt7461-i2c-1-4c
Adapter: MPC adapter
temp1: +34.0 C (low = +0.0 C, high = +85.0 C)
 (crit = +85.0 C, hyst = +75.0 C)
temp2: +48.5 C (low = +0.0 C, high = +85.0 C)
 (crit = +85.0 C, hyst = +75.0 C)
```

**"lm\_sensors is integrated into rootfs file system by default. If there is no "sensors" command in your rootfs just add lmsensors-sensors package and build your own rootfs."**

# Chapter 13

## Benchmarking guidelines

### 13.1 Coremark

#### 13.1.1 Test Environment

##### Objectives

The Coremark benchmarking guideline aims to do the following:

- Baseline the Coremark performance on QorIQ Layerscape platforms
- Identify any optimizations and ensure they are implemented on the QorIQ Layerscape platforms.
- Investigate other changes that may improve performance

##### Hardware Platform Identification

| Board      | Silicon Revision | Default Frequency(Core/CCB/DDR) in MHz | Core Type  |
|------------|------------------|----------------------------------------|------------|
| LS1021ATWR | Rev2.0           | 1000/300/1600                          | cortex A7  |
| LS1043ARDB | Rev1.1           | 1600/400/1600                          | cortex A53 |
| LS1046ARDB | Rev1.0           | 1800/700/2100                          | cortex A72 |
| LS1088ARDB | Rev1.0           | 1600/700/2100                          | cortex A53 |
| LS2088ARDB | Rev1.0           | 2000/800/2133                          | cortex A72 |

For more information on each boards switch settings, refer to the boards's Reference Manual or Getting Started Guide on <http://www.nxp.com/>

##### Software Platforma Identification

All software was built from Layerscape SDK.

##### Boot Loader

U-boot 2017.03 with NXP-specific patches on top.

##### Coremark Application

###### • Source Code Download:

Coremark Source code can be downloaded from <http://www.eembc.org/coremark/index.php>

Toolchain version is gcc-5.4 with glibc-2.23

###### • Build Coremark

1. If you are compiling Coremark on a 64-bit Linux machine (machine on which you have the intended compiler), go to `coremark_v1.0/linux64` directory, else go to `coremark_v1.0/linux` directory.

2. Give the complete compiler path under "CC" flag in `core_portme.mak` file.
3. For a 32-bit ARM platform, the toolchain path is `/usr/bin/arm-linux-gnueabi-gcc` by default. Change the "CC" flag in `linux/core_portme.mak` as the following:

```
CC = /usr/bin/arm-linux-gnueabi-gcc
```

4. For a 64-bit ARM platform, the toolchain path is `/usr/bin/aarch64-linux-gnu-gcc`. Change the "CC" flag in `linux64/core_portme.mak` as the following:

```
CC = /usr/bin/aarch64-linux-gnu-gcc
```

5. Go back to the `coremark_v1.0` directory. Perform the following commands on the command line:

- For a 64-bit ARM platform (LS1043A/LS1046A/LS1088A/LS2088A):

Single Thread:

```
make PORT_CFLAGS="-O3 -funroll-all-loops --param max-inline-insns-auto=550"
```

Multithread:

```
make PORT_CFLAGS="-O3 -funroll-all-loops --param max-inline-insns-auto=550 -
DMULTITHREAD=<Thread_number> -DUSE_FORK=1"
```

- For a 32-bit ARM platform (LS1021A and LS1043A/LS1046A 32-bit)

Single Thread:

```
make PORT_CFLAGS="-O3 -march=armv7-a -mfloat-abi=hard -mcpu=cortex-a7 -funroll-
all-loops --param max-inline-insns-auto=300 -static"
```

Multithread:

```
make PORT_CFLAGS="-O3 -march=armv7-a -mfloat-abi=hard -mcpu=cortex-a7 -funroll-
all-loops --param max-inline-insns-auto=300 -static -DMULTITHREAD=<Thread_number> -
DUSE_FORK=1"
```

6. The command will first compile, generate the executable file (`coremark.exe`) and try to run the benchmark. Transfer the executable file to the target.

## 13.1.2 Test Procedure

### Running test and result collection

1. Deploy the target board with corresponding software mentioned in the previous section.
2. Put coremark binary compiled with optimized flags mentioned in section test environment on target board
3. Run the benchmark:

```
coremark.exe
```

Check the log below for the results:

```
2K performance run parameters for coremark.
CoreMark Size : 666
Total ticks : 16663
Total time (secs): 16.663000
Iterations/Sec : 6601.452320
```

```

Iterations : 110000
Compiler version : GCC5.4.0 20160609
Compiler flags : -mcpu=cortex-a53 -O3 -funroll-all-loops --param max-inline-insns-auto=550 -
DPERFORMANCE_RUN=1 -lrt
Memory location : Please put data memory location here
 (e.g. code in flash, data on heap etc)
seedcrc : 0xe9f5
[0]crclist : 0xe714
[0]crcmatrix : 0x1fd7
[0]crcstate : 0x8e3a
[0]crcfinal : 0x33ff
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 6601.452320 / GCC5.4.0 20160609 -mcpu=cortex-a53 -O3 -funroll-all-loops --param max-
inline-insns-auto=550 -DPERFORMANCE_RUN=1 -lrt / Heap

```

This test measures the variation of the benchmark results, so an average across 5 runs was taken for every result.

## 13.2 Dhrystone

### 13.2.1 Test Environment

#### Objectives

The Dhrystone benchmarking guideline aims to do the following:

- Baseline the Dhrystone performance on QorIQ Layerscape platforms.
- Identify any optimizations and ensure they are implemented on the QorIQ Layerscape platforms.
- Investigate other changes that may improve performance.

#### Hardware Platform Identification

| Board      | Silicon Revision | Default<br>Frequency(Core/CCB/<br>DDR) in MHz | Core Type  |
|------------|------------------|-----------------------------------------------|------------|
| LS1021ATWR | Rev2.0           | 1000/300/1600                                 | cortex A7  |
| LS1043ARDB | Rev1.1           | 1600/400/1600                                 | cortex A53 |
| LS1046ARDB | Rev1.0           | 1800/700/2100                                 | cortex A72 |
| LS1088ARDB | Rev1.0           | 1600/700/2100                                 | cortex A53 |
| LS2088ARDB | Rev1.0           | 2000/800/2133                                 | cortex A72 |

For more information on each boards switch settings, refer to the boards's Reference Manual or Getting Started Guide on <http://www.nxp.com/>

#### Software Platforma Identification

All software was built from Layerscape SDK.

#### Boot Loader

U-boot 2017.03 with NXP-specific patches on top.

## Dhrystone Application

Dhrystone is a synthetic computing benchmark program intended to be representative of system (integer) programming. It is a simple program that is carefully designed to statistically mimic the processor usage of some common set of programs. It also has some pitfalls, for the performance will be affected by many factors such as the compiler, libraries etc.

- **Source Code Download:**

Dhrystone 1.0 Source code can be downloaded from: <http://www.xanthos.se/~joachim/vaxmips.html> (Dhrystone-src.tar.gz)

Toolchain version is gcc-5.4 with glibc-2.23.

- **Build Dhrystone**

1. Download the source code from <http://www.xanthos.se/~joachim/vaxmips.html> Dhrystone-src.tar.gz
2. Unpack the package
3. Go back to the dhrystone\_v1.0 directory and build dhrystone binary

- For a 64-bit ARM platform:

```
#/usr/bin/aarch64-linux-gnu-gcc -O3 -funroll-all-loops --param max-inline-insns-auto=550 -static dhry21a.c dhry21b.c timers.c -o dhrystone
```

- For 32-bit ARM platform:

```
/usr/bin/arm-linux-gnueabi-gcc -O3 -funroll-all-loops --param max-inline-insns-auto=550 -static dhry21a.c dhry21b.c timers.c -o dhrystone
```

## 13.2.2 Test Procedure

### Running test and result collection

1. Deploy the target board with corresponding software mentioned in the previous section.
2. Put Dhrystone binary compiled with optimized flags mentioned in section 2.2.2.3 on target board
3. Run the benchmark:

```
echo 50000000 | ./dhrystone
```

Check the log below for the results:

```
Dhrystone Benchmark, Version 2.1 (Language: C)

Please give the number of runs through the benchmark:
Execution starts, 50000000 runs through Dhrystone
Execution ends

Final values of the variables used in the benchmark:

Int_Glob: 5
 should be: 5
Bool_Glob: 1
 should be: 1
Ch_1_Glob: A
 should be: A
Ch_2_Glob: B
 should be: B
Arr_1_Glob[8]: 7
```

## Benchmarking guidelines

### Dhrystone

```
 should be: 7
Arr_2_Glob[8][7]: 50000010
 should be: Number_Of_Runs + 10
Ptr_Glob->
 Ptr_Comp: 855702608
 should be: (implementation-dependent)
 Discr: 0
 should be: 0
 Enum_Comp: 2
 should be: 2
 Int_Comp: 17
 should be: 17
 Str_Comp: DHRYSTONE PROGRAM, SOME STRING
 should be: DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
 Ptr_Comp: 855702608
 should be: (implementation-dependent), same as above
 Discr: 0
 should be: 0
 Enum_Comp: 1
 should be: 1
 Int_Comp: 18
 should be: 18
 Str_Comp: DHRYSTONE PROGRAM, SOME STRING
 should be: DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc: 5
 should be: 5
Int_2_Loc: 13
 should be: 13
Int_3_Loc: 7
 should be: 7
Enum_Loc: 1
 should be: 1
Str_1_Loc: DHRYSTONE PROGRAM, 1'ST STRING
 should be: DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc: DHRYSTONE PROGRAM, 2'ND STRING
 should be: DHRYSTONE PROGRAM, 2'ND STRING
```

Register option selected? YES

Microseconds for one run through Dhrystone: 0.1

Dhrystones per Second: 8519121.2

VAX MIPS rating = 4848.675

```
[root@ls1043agw ~]$ echo 50000000|./dhry21
```

Dhrystone Benchmark, Version 2.1 (Language: C)

Please give the number of runs through the benchmark:

Execution starts, 50000000 runs through Dhrystone

Execution ends

Final values of the variables used in the benchmark:

```
Int_Glob: 5
 should be: 5
Bool_Glob: 1
 should be: 1
Ch_1_Glob: A
 should be: A
Ch_2_Glob: B
```



```

 should be: B
Arr_1_Glob[8]: 7
 should be: 7
Arr_2_Glob[8][7]: 50000010
 should be: Number_Of_Runs + 10
Ptr_Glob->
 Ptr_Comp: 1046248528
 should be: (implementation-dependent)
 Discr: 0
 should be: 0
 Enum_Comp: 2
 should be: 2
 Int_Comp: 17
 should be: 17
 Str_Comp: DHRYSTONE PROGRAM, SOME STRING
 should be: DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
 Ptr_Comp: 1046248528
 should be: (implementation-dependent), same as above
 Discr: 0
 should be: 0
 Enum_Comp: 1
 should be: 1
 Int_Comp: 18
 should be: 18
 Str_Comp: DHRYSTONE PROGRAM, SOME STRING
 should be: DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc: 5
 should be: 5
Int_2_Loc: 13
 should be: 13
Int_3_Loc: 7
 should be: 7
Enum_Loc: 1
 should be: 1
Str_1_Loc: DHRYSTONE PROGRAM, 1'ST STRING
 should be: DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc: DHRYSTONE PROGRAM, 2'ND STRING
 should be: DHRYSTONE PROGRAM, 2'ND STRING

Register option selected? YES
Microseconds for one run through Dhrystone: 0.1
Dhrystones per Second: 8498703.9
VAX MIPS rating = 4837.054

```

## 13.3 EEMBC

### 13.3.1 Test Environment

#### Objectives

The EEMBC benchmarking guideline aims to do the following:

- Baseline the EEMBC performance on QorIQ Layerscape platforms.

- Identify any optimizations and ensure they are implemented on the QorIQ Layerscape platforms.
- Investigate other changes that may improve performance.

### Hardware Platform Identification

| Board      | Silicon Revision | Default Frequency(Core/CCB/DDR) in MHz | Core Type  |
|------------|------------------|----------------------------------------|------------|
| LS1021ATWR | Rev2.0           | 1000/300/1600                          | cortex A7  |
| LS1043ARDB | Rev1.1           | 1600/400/1600                          | cortex A53 |
| LS1046ARDB | Rev1.0           | 1800/700/2100                          | cortex A72 |
| LS1088ARDB | Rev1.0           | 1600/700/2100                          | cortex A53 |
| LS2088ARDB | Rev1.0           | 2000/800/2133                          | cortex A72 |

For more information on each boards switch settings, refer to the boards's Reference Manual or Getting Started Guide on <http://www.nxp.com/>

### Software Platforma Identification

All software was built from Layerscape SDK.

### Boot Loader

U-boot 2017.03 with NXP-specific patches on top.

### Endianness

For ARM architecture:

Define correct Endianness by either modifying `th_lite/<platform>/al/thcfg.h` to:

```
#if !defined(EE_BIG_ENDIAN) && !defined(EE_LITTLE_ENDIAN)
#define EE_BIG_ENDIAN (FALSE)
#define EE_LITTLE_ENDIAN (TRUE)
#endif
```

Or by modifying `/util/make/gcc.mak` to:

```
COMPILER_DEFINES += -DEE_BIG_ENDIAN=1 -DEE_LITTLE_ENDIAN=1
```

### Data Types

`th_lite/<platform>/al/eembc_dt.h` has various data types definitions If not already done, do:

```
#define HAVE_64 (1)
```

The default definition for data types is:

```
typedef unsigned long e_u24;
typedef signed long e_s24;

typedef unsigned long e_u32;
typedef signed long e_s32;
```

However, as ppc follows l32LP64, in case of 64 bit execution long will be considered 64 bit. So the data type definitions should be changed to:

```
typedef unsigned int e_u24;
typedef signed int e_s24;

typedef unsigned int e_u32;
typedef signed int e_s32;
```

Build the benchmark with following compiler flags:

```
32bit ARM(LS1021A/LS1043A/LS1046A 32ibt):

TOOLS = /usr/bin
CC = $(TOOLS)/ arm-linux-gnueabi-hf-gcc
AS = $(TOOLS)/arm-linux-gnueabi-hf-as
LD = $(TOOLS)/ arm-linux-gnueabi-hf-gcc
AR = $(TOOLS)/ arm-linux-gnueabi-hf-ar
SIZE = $(TOOLS)/ arm-linux-gnueabi-hf-size

#Both TCPMark and IPMark, use following compiler flags:
COMPILER_FLAGS = -mcpu=cortex-a7 -mtune=cortex-a7 -O3 -funroll-all-loops -ftree-vectorize -flto -fwhole-program -fgcse-las
LINKER_FLAGS = -lm -static --sysroot=/opt/fsl-qoriq/1.9/sysroots/ppce500v2-fsl-linux-gnuspe

64bit ARM(LS1043A/LS1046A/LS1088A/LS2088A):
TOOLS = /usr/bin

CC = $(TOOLS)/ aarch64-linux-gnu-gcc
AS = $(TOOLS)/ aarch64-linux-gnu-as
LD = $(TOOLS)/ aarch64-linux-gnu-gcc
AR = $(TOOLS)/ aarch64-linux-gnu-ar

COMPILER_FLAGS = -O3 -funroll-all-loops -ftree-vectorize
LINKER_FLAGS = -lm -static
SIZE = $(TOOLS)/ aarch64-linux-gnu-size
```

## Generate EEMBC Binary for Target Board

1. Create a working directory.
2. Retrieve EEMBC v2.0 from <http://www.eembc.org/>
3. Extract the EEMBC v2.0 source code.
4. Edit `util/make/gcc.mak` so that the CC variable points to the correct location of your compiler. See the table above for detailed compiler flags and configuration.
5. Build binary using the make command:

```
make VER=v2 TOOLCHAIN=gcc THLITE=_lite all-lite
```

6. After the build is complete, copy binary files under `<EEMBC_2.0_INSTALL_DIR>/networking/gcc/bin_lite` to target board.

## 13.3.2 Test Procedure

### Running test and result collection

1. Deploy the target board with corresponding software mentioned in the previous section.
2. Put EEMBC Netmark binary compiled with optimized flags mentioned in section test environment on target board.  
EEMBC Netmark binary file list are as follows:

```
networking/tcpbulk_lite.exe
networking/tcpjumbo_lite.exe
networking/tcpmixed_lite.exe
networking/ip_pktcheckb1m_lite.exe
networking/ip_pktcheckb2m_lite.exe
networking/ip_pktcheckb4m_lite.exe
networking/ip_pktcheckb512k_lite.exe
networking/ip_reassembly_lite.exe
networking/ip_reassembly_lite.exe
networking/nat_lite.exe -INITTIME
networking/nat_lite.exe
networking/ospfv2_lite.exe
networking/qos_lite.exe
networking/routelookup_lite.exe
```

3. Run the benchmark:

```
networking/tcpbulk_lite.exe -i 200000
networking/tcpjumbo_lite.exe -i 300000
networking/tcpmixed_lite.exe -i 100000
networking/ip_pktcheckb1m_lite.exe -i 50000
networking/ip_pktcheckb2m_lite.exe -i 30000
networking/ip_pktcheckb4m_lite.exe -i 10000
networking/ip_pktcheckb512k_lite.exe -i 100000
networking/ip_reassembly_lite.exe -INITTIME -i 5000
networking/ip_reassembly_lite.exe -i 5000
networking/nat_lite.exe -INITTIME -i 10000
networking/nat_lite.exe -i 10000
networking/ospfv2_lite.exe -i 10000
networking/qos_lite.exe -i 300
networking/routelookup_lite.exe -i 20000
```

Check the log below for the results:

```
root@localhost# ./tcpbulk_lite -i 167000
Configure benchmark for bulk data transfer test
Initialize network buffer pools
INFO: Initializing client and server NIF
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 167000
>> Bench Mark : TCP-BM bulk V2.0R1
-- Non-Intrusive CRC = 0
```

```
-- Iterations = 167000
-- Target Duration = 1600000
-- Target Timer Rate = 1000000
-- v1 = -4280
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 104375.000
-- Total Run Time = 1.600sec
-- Time / Iter = 0.000009581sec
>> DONE!
>> BM: TCP-BM bulk V2.0R1
>> ID: NTW tcp

root@localhost# ./tcpjumbo_lite -i 250500
Configure benchmark for jumbo packet transfer test
Initialize network buffer pools
INFO: Initializing client and server NIF
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 250500
>> Bench Mark : TCP-BM jumbo V2.0R1
-- Non-Intrusive CRC = 0
-- Iterations = 250500
-- Target Duration = 1540000
-- Target Timer Rate = 1000000
-- v1 = -24000
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 162662.338
-- Total Run Time = 1.540sec
-- Time / Iter = 0.000006148sec
>> DONE!
>> BM: TCP-BM jumbo V2.0R1
>> ID: NTW tcp

root@localhost# ./tcpmixed_lite -i 83500
Configure benchmark for mixed packet size test
Initialize network buffer pools
INFO: Initializing client and server NIF
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 83500
>> Bench Mark : TCP-BM mixed V2.0R1
-- Non-Intrusive CRC = 0
```

## EEMBC

```

-- Iterations = 83500
-- Target Duration = 1940000
-- Target Timer Rate = 1000000
-- v1 = -2736
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 43041.237
-- Total Run Time = 1.940sec
-- Time / Iter = 0.000023234sec
>> DONE!
>> BM: TCP-BM mixed V2.0R1
>> ID: NTW tcp

root@localhost# ./ip_pktcheckb1m_lite -i 41750
>> Datagram buffer size : 0x0100000
>> Datagram alignment : 4
>> Descriptor padd size : 8
>> Number of Datagrams allocated : 720
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 41750
>> Bench Mark : Networking: IP Packet Check Benchmark, 1.0M V2.0R1
-- Non-Intrusive CRC = e3b5
-- Iterations = 41750
-- Target Duration = 2260000
-- Target Timer Rate = 1000000
-- v1 = 30060000
-- v2 = 1670000
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 18473.451
-- Total Run Time = 2.260sec
-- Time / Iter = 0.000054132sec
>> DONE!
>> BM: Networking: IP Packet Check Benchmark, 1.0M V2.0R1
>> ID: NTW ip_pktchec

root@localhost# ./ip_pktcheckb2m_lite -i 25050
>> Datagram buffer size : 0x0200000
>> Datagram alignment : 4
>> Descriptor padd size : 8
>> Number of Datagrams allocated : 1412
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10

```

```
>> Recommended Iterations : 25050
>> Bench Mark : Networking: IP Packet Check Benchmark, 2.0M V2.0R1
-- Non-Intrusive CRC = 48b
-- Iterations = 25050
-- Target Duration = 2720000
-- Target Timer Rate = 1000000
-- v1 = 35370600
-- v2 = 1828650
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 9209.559
-- Total Run Time = 2.720sec
-- Time / Iter = 0.000108583sec
>> DONE!
>> BM: Networking: IP Packet Check Benchmark, 2.0M V2.0R1
>> ID: NTW ip_pktchec
```

```
root@localhost# ./ip_pktcheckb4m_lite -i 8350
>> Datagram buffer size : 0x0400000
>> Datagram alignment : 4
>> Descriptor padd size : 8
>> Number of Datagrams allocated : 2824
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 8350
>> Bench Mark : Networking: IP Packet Check Benchmark, 4.0M V2.0R1
-- Non-Intrusive CRC = d86c
-- Iterations = 8350
-- Target Duration = 1800000
-- Target Timer Rate = 1000000
-- v1 = 23580400
-- v2 = 1244150
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 4638.889
-- Total Run Time = 1.800sec
-- Time / Iter = 0.000215569sec
>> DONE!
>> BM: Networking: IP Packet Check Benchmark, 4.0M V2.0R1
>> ID: NTW ip_pktchec
```

```
root@localhost# ./ip_pktcheckb512k_lite -i 83500
>> Datagram buffer size : 0x0080000
>> Datagram alignment : 4
>> Descriptor padd size : 8
>> Number of Datagrams allocated : 374
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
```

## Benchmarking guidelines

### EEMBC

```
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 83500
>> Bench Mark : Networking: IP Packet Check Benchmark, 0.5M V2.0R1
-- Non-Intrusive CRC = 3e1d
-- Iterations = 83500
-- Target Duration = 2160000
-- Target Timer Rate = 1000000
-- v1 = 31229000
-- v2 = 1753500
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 38657.407
-- Total Run Time = 2.160sec
-- Time / Iter = 0.000025868sec
>> DONE!
>> BM: Networking: IP Packet Check Benchmark, 0.5M V2.0R1
>> ID: NTW ip_pktchec

root@localhost# ./ip_reassembly_lite -INITTIME -i 4175
*** Initialization Timing Run, Subtract from normal run for score ***
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 4175
>> Bench Mark : INITIALIZATION Networking: IP Reassembly Benchmark V2.0R1
-- Non-Intrusive CRC = 0
-- Iterations = 4175
-- Target Duration = 640000
-- Target Timer Rate = 1000000
-- v1 = 200
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 6523.438
-- Total Run Time = 0.640sec
-- Time / Iter = 0.000153293sec
>> DONE!
>> BM: INITIALIZATION Networking: IP Reassembly Benchmark V2.0R1
>> ID: NTW ip_reasmIT

root@localhost# ./ip_reassembly_lite -i 4175
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 4175
```



```
>> Bench Mark : Networking: IP Reassembly Benchmark V2.0R1
-- Non-Intrusive CRC = 678e
-- Iterations = 4175
-- Target Duration = 1940000
-- Target Timer Rate = 1000000
-- v1 = 200
-- v2 = 4939025
-- v3 = 835000
-- v4 = 0
-- Iterations/Sec = 2152.062
-- Total Run Time = 1.940sec
-- Time / Iter = 0.000464671sec
>> DONE!
>> BM: Networking: IP Reassembly Benchmark V2.0R1
>> ID: NTW ip_reasm

root@localhost# ./nat_lite -INITTIME -i 8350
*** Initialization Timing Run, Subtract from normal run for score ***
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 8350
>> Bench Mark : INITIALIZATION Network Address Translation V2.0R1
-- Non-Intrusive CRC = 0
-- Iterations = 8350
-- Target Duration = 960000
-- Target Timer Rate = 1000000
-- v1 = 1000
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 8697.917
-- Total Run Time = 0.960sec
-- Time / Iter = 0.000114970sec
>> DONE!
>> BM: INITIALIZATION Network Address Translation V2.0R1
>> ID: NTW NATIT

root@localhost# ./nat_lite -i 8350
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 8350
>> Bench Mark : Network Address Translation V2.0R1
-- Non-Intrusive CRC = 9d46
-- Iterations = 8350
-- Target Duration = 2320000
```

```
-- Target Timer Rate = 1000000
-- v1 = 1000
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 3599.138
-- Total Run Time = 2.320sec
-- Time / Iter = 0.000277844sec
>> DONE!
>> BM: Network Address Translation V2.0R1
>> ID: NTW NAT

root@localhost# ./ospfv2_lite -i 8350
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 8350
>> Bench Mark : Networking: OSPF Benchmark V2.0R1
-- Non-Intrusive CRC = 7f12
-- Iterations = 8350
-- Target Duration = 1480000
-- Target Timer Rate = 1000000
-- v1 = 400
-- v2 = 4
-- v3 = 8
-- v4 = 32000
-- Iterations/Sec = 5641.892
-- Total Run Time = 1.480sec
-- Time / Iter = 0.000177246sec
>> DONE!
>> BM: Networking: OSPF Benchmark V2.0R1
>> ID: NTW ospf

root@localhost# ./qos_lite -i 250
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 250
>> Bench Mark : Networking: QoS V2.0R1
-- Non-Intrusive CRC = fa81
-- Iterations = 250
-- Target Duration = 1000000
-- Target Timer Rate = 1000000
-- v1 = 100
-- v2 = 100
-- v3 = 0
```

```
-- v4 = 0
-- Iterations/Sec = 250.000
-- Total Run Time = 1.000sec
-- Time / Iter = 0.004000000sec
>> DONE!
>> BM: Networking: QoS V2.0R1
>> ID: NTW QoS

root@localhost# ./routelookup_lite -i 16700
>> self-check completed ok.
>>-----
>> EEMBC Component : EEMBC Portable Test Harness V4.100
>> EEMBC Member Company : EEMBC
>> Target Processor : HOST EXAMPLE
>> Target Platform : 32 Bit
>> Target Timer Available : YES
>> Target Timer Intrusive : YES
>> Target Timer Rate : 1000000
>> Target Timer Granularity : 10
>> Recommended Iterations : 16700
>> Bench Mark : Networking: Route Lookup Benchmark V2.0R1
-- Non-Intrusive CRC = 407d
-- Iterations = 16700
-- Target Duration = 2100000
-- Target Timer Rate = 1000000
-- v1 = 0
-- v2 = 0
-- v3 = 0
-- v4 = 0
-- Iterations/Sec = 7952.381
-- Total Run Time = 2.100sec
-- Time / Iter = 0.000125749sec
>> DONE!
>> BM: Networking: Route Lookup Benchmark V2.0R1
>> ID: NTW routelookup EEMBC
```

#### NOTE

For other platforms, use the ratio (CPU\_Freq\_target\_platform/2000) to get the corresponding parameter.

There is a run to run variation, so an average across 5 runs was taken for every result.

### Networking Version 2.0 Calculation

Networking Version 2.0 produces two aggregate "mark" scores: the TCPmark™ and the IPmark™. The IPmark is intended for developers of infrastructure equipment, while the TCPmark, which includes the TCP benchmark, focuses on client- and server-based network hardware.

The IPmark is the geometric mean of the scores for QoS, Route Lookup, OSPF, IP Reassembly, Network Address Translation, and the geometric mean of the individual scores for IP Packet check, all divided by 10:

```
IPmark = Geomean ((Geomean (IP Packet Check [0.5MB], IP Packet Check [1MB], IP Packet Check [2MB],
IP Packet Check [4MB]), QoS, Route Lookup, OSPF, IP Reassembly, NAT))/10
```

The TCPmark is the geometric mean of the scores for TCP Jumbo, TCP Bulk, and TCP Mixed divided by 100:

```
TCPmark = Geomean (TCP jumbo, TCP bulk, TCP Mixed)/100
```

To calculate a geometric mean, multiply all the results of the tests together and take the nth root of the product, where n equals the number of tests.

#### NAT and IP Reassembly Benchmarks

To calculate the iterations per second for the NAT and IP Reassembly benchmarks, it's necessary to run the benchmarks twice:

1. Run the benchmark with the flag -INITTIME supplied on the command line.
2. Run the benchmark without the flag -INITTIME supplied on the command line. The same executable must be run both times and the number of iterations must be identical.
3. Subtract the time of the first run from the time of the second run and calculate the iterations per second based on the calculated time.

The score reported for each device is a single-number figure of merit calculated by taking the geometric mean of the individual Networking scores and dividing by 395.184. This normalization factor is derived from the lowest score in this category on December 5, 2000. Scores for each of the individual benchmarks within this suite allow designers to weight and aggregate the benchmarks to suit specific application requirements.

To calculate a geometric mean, multiply all the results (\*) of the tests together and take the nth root of the product, where n equals the number of tests. (\*) Scores included in geometric mean:

- OSPF
- Route Lookup
- Packet Flow - 512 kbytes
- Packet Flow - 1 Mbyte
- Packet Flow - 2 Mbytes

#### NOTE

This calculation can also be found on EEMBC website: <http://eembc.org/benchmark/reports/mark.php?suite=NT2>

## 13.4 LMBench

### 13.4.1 Test Environment

#### Objectives

The LMBench benchmarking guideline aims to do the following:

- Baseline the LMBench performance on QorIQ Layerscape platforms.
- Identify any optimizations and ensure they are implemented on the QorIQ Layerscape platforms.
- Investigate other changes that may improve performance.

#### Hardware Platform Identification

| Board                               | Silicon Revision | Default Frequency(Core/CCB/DDR) in MHz | Core Type |
|-------------------------------------|------------------|----------------------------------------|-----------|
| LS1021ATWR                          | Rev2.0           | 1000/300/1600                          | cortex A7 |
| Table continues on the next page... |                  |                                        |           |

Table continued from the previous page...

| Board      | Silicon Revision | Default Frequency(Core/CCB/DDR) in MHz | Core Type  |
|------------|------------------|----------------------------------------|------------|
| LS1043ARDB | Rev1.1           | 1600/400/1600                          | cortex A53 |
| LS1046ARDB | Rev1.0           | 1800/700/2100                          | cortex A72 |
| LS1088ARDB | Rev1.0           | 1600/700/2100                          | cortex A53 |
| LS2088ARDB | Rev1.0           | 2000/800/2133                          | cortex A72 |

For more information on each board's switch settings, refer to the boards's Reference Manual or Getting Started Guide on <http://www.nxp.com/>

### Software Platforma Identification

All software was built from Layerscape SDK.

### Boot Loader

U-boot 2017.03 with NXP-specific patches on top.

If the target platform support chip select interleaving and address hash, please enable chip select interleaving and address hash, disabling ecc by hwconfig with syntax:

```
"hwconfig=fsl_ddr:bank_intlv=cs0_cs1,addr_hash=true,ecc=off"
```

### LMBench Application

The rootfs includes LMBench binaries which were built without optimization. For general latency performance, the default LMBench binary file in the root file system will be OK. To get better bandwidth performance result, modify the compiler flags to enable "O3" optimization.

1. Download lmbench source code from following link: <https://sourceforge.net/projects/lmbench/files/latest/download>
2. Change "CC" and "CFLAGS" in file `src/Makefile` as the following:

```
CC = /usr/bin/aarch64-linux-gnu-gcc
CFLAGS = -O3
```

3. Build code with new optimized compiler flags:

```
#make
```

4. Transfer the `bw_mem` binary file to target board.

## 13.4.2 Test Procedure

### Running Test and Result Collection

Separate U-Boot image was flashed in alternate U-Boot flash bank and the DUT is booted out of that bank. Binaries compiled with different flags were run separately and then the data was collected for the binary showing the best results. To get full LMBench test result, run `lmbench-run` to get the full test result.

There is a run to run variation in, so an average across 5 runs was taken for every result

## Scripts for LMBench Test

### Execution latency test script:

```
for i in `seq 1 5`
do
echo "Integer, integer64,float,double float execution latency"
lat_ops
done
```

### Memory read latency test script:

```
for i in `seq 1 5`
do
echo "L1, L2, L3 and DDR read latency"
lat_mem_rd 100M
done
```

### Memory bandwidth test script:

```
#!/bin/sh

for opt in rd wr rdwr cp frd fwr fcp bzero bcopy
do
echo "L1 cache bandwidth $opt test with #$(nproc) process"
#8k is fit for all platform

for idx in `seq 1 5`
do
bw_mem -P 1 8k $opt
done
echo "L2 cache bandwidth $opt test"
For Layerscape platform, each platform has more than 256K L2 cache, so chose 128k as L2 cache
size.
for idx in `seq 1 5`
do
bw_mem -P 1 128k $opt
done

echo "Main mem bandwidth $opt test"
for idx in `seq 1 5`
do
bw_mem -P 1 100m $opt
done
done
```



**How To Reach Us****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, Freescale, the Freescale logo, CodeWarrior, Layerscape, PowerQUICC, QorIQ, CoreNet, and QUICC Engine are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

© NXP B.V. 2018.

LX2160A\_BSP  
Rev. 0  
22/01/2019

