
SPC58 line - How to use the Memory Protection Layers

Introduction

This technical note explains the concepts and usage of the memory protection mechanisms available in the SPC58 line of automotive microcontrollers.

These devices offer multiple layers of memory protection, i.e.:

- SMPU - System Memory Protection Unit
- CMPU - Core Memory Protection Unit
- PFAPR - Flash Controller Platform Access Control
- MCR/PACR - Peripheral Bridge Access Control
- REGPROT - Register Protection

Each of these mechanisms is in place for a slightly different purpose. Combining all of them gives the user a powerful system to implement access control in their application.

Contents

1	General introduction	7
1.1	Block diagram	7
1.2	Typical use cases	8
1.3	Master assignments and PBRIDGE slot assignments	9
1.4	Differences between SMPU and CMPU	10
1.5	MPU configuration combinations	10
1.5.1	Combinations of different memory protection mechanisms	10
1.5.2	Overlapping regions in System MPU	12
1.5.3	Overlapping regions in Core MPU	12
2	Core Memory Protection Unit (CMPU)	13
2.1	Overview	13
2.2	Types of CMPU Entries and Region Descriptors	13
2.3	Access Matching	15
2.3.1	Access Masks	15
2.3.2	Permissions	16
2.3.3	Memory Attributes (G, I)	17
2.3.4	Access Bypass	17
2.4	Use of Region Descriptor Table Entries for Debug	18
2.5	Software Interface and CMPU Instructions	18
2.6	CMPU Control Registers	18
2.7	CMPU Operations	18
3	System Memory Protection Unit (SMPU)	21
3.1	Overview	21
3.2	Types of SMPU Entries and Region Descriptors	21
3.3	Access Matching	22
3.3.1	Error terminations	23
3.3.2	SMPU functionality on multi-crossbar devices	23
3.4	SMPU Control Registers	23
3.5	SMPU Operations	24
3.5.1	Additional operations	24

4	Flash Controller Platform Access Control	25
4.1	Overview	25
4.2	Access control	25
4.2.1	PFAPR - Platform Flash access protection register	25
4.3	Flash Controller Platform Access Control Registers	25
4.4	Flash Controller Platform Access Control Operations	26
5	Peripheral Bridge Access Control	27
5.1	Overview	27
5.2	Access control	27
5.2.1	Master Privilege Register	27
5.2.2	Peripheral Access Control Register	28
5.3	PBRIDGE Access Control Registers	29
5.4	PBRIDGE Access Control Operations	29
6	Register protection (REGPROT)	31
6.1	Overview	31
6.2	List of protected peripherals	32
6.3	Access control	32
6.3.1	The Soft Lock Bits and the Global Configuration Register	33
6.4	REG_PROT Control Registers	34
6.5	REG_PROT Operations	34
7	Default Configurations	36
7.1	SMPU default configuration	36
7.1.1	Reset behavior	36
7.2	CMPU default configuration	36
7.2.1	Cache-Inhibit and Guarded Control Signals	36
7.2.2	Reset behavior	38
7.3	PFAPR default configuration	38
7.3.1	Reset behavior	38
7.4	MPR/PACR default configuration	38
7.4.1	MPR default configuration	38
7.4.2	(O)PACR default configuration	39
7.4.3	Reset behavior	39

7.5	REGPROT default configuration	39
7.5.1	Reset behavior	40
Appendix A	CMPU configuration code example	41
A.1	Example 1 - Simple process-aware configuration of CMPU	41
Appendix B	SMPU configuration code examples	44
B.1	Example 1 - Dual core system with four bus masters	44
Appendix C	PFlashC PFAPR Access Control Example	47
C.1	Example 1 - Access control in a system w/ multiple Flash controllers ...	47
Appendix D	PBRIDGE Access Control Example	48
D.1	Example 1 - Simple PBRIDGE access control example	48
Appendix E	REGPROT example	49
E.1	Example 1 - Configuration of Register protection for SIUL2 registers. ...	49
Appendix F	Document management	50
Revision history		51

List of tables

Table 1.	Example of AHB master ID assignement on SPC58xGxx microcontroller	9
Table 2.	Example of Pheriperal Slot assignement on SPC58xGxx microcontroller	9
Table 3.	CMPU Region Descriptor Entry Bit Definitions	15
Table 4.	SMPU Region Descriptor Entry Bit Definitions	22
Table 5.	Protection violation definition	22
Table 6.	Configuration bits for Master Privilege Register	27
Table 7.	Configuration bits for Peripheral Access Control Register	28
Table 8.	Memory map SIUL2 registers including register protection	32
Table 9.	Soft Lock Bits vs. Protected Addresses	33
Table 10.	SLBR Configuration options	34
Table 11.	Default values of CI/R control signals on Chorus-10M	37
Table 12.	Default values of CI/R control signals on Chorus-6M, 4M and 2M	37
Table 13.	Default values of CI/R control signals on Chorus-1M	38
Table 14.	MPROTn default configuration	39
Table 15.	Insert title here	39
Table 16.	Configuration of SMPU Example 1	44
Table 17.	Reference documents.	50
Table 18.	Document revision history	51

List of figures

Figure 1.	Block diagram	8
Figure 2.	Protection mechanisms affecting access to ADC on Cluster 0.	11
Figure 3.	CMPU description	14
Figure 4.	Granting of access permission	17
Figure 5.	SMPU protection diagram for multi-crossbar systems SPC58xGx.	23
Figure 6.	REG_PROT Block Diagram	31
Figure 7.	REG_PROT memory diagram	32

1 General introduction

Memory protection units (MPUs) are hardware modules which provide memory protection mechanisms.

Depending on the integration SPC58 devices integrate Core and System MPUs. The main difference among these modules is that the Core MPUs belong to the computational cores and the System MPUs belong to the System on Chip (SoC). Besides, the SPC58 devices embed other layers of protection within the FLASH controller and the peripheral bridges.

The MPUs allow to define memory regions in the address range of the microcontroller and assign access privileges to these regions based on process ID or requestor ID (for example, allowing one specific core to access the memory and deny all other requests).

The user can configure additional memory access parameters. PowerPC-based microcontrollers describe these attributes with the acronym WIMG:

- Write-Through Access (all accesses are Write-Through on SPC58)
- Cache-Inhibited Access
- Memory Coherence (not supported by SPC58)
- Guarded

On SPC58, however, the user can only configure the Cache-Inhibit and Guarded flags 8.

Every device in the SPC58 line of microcontrollers has two types of memory protection units:

- CMPU (Core Memory Protection Unit)
- SMPU (System Memory Protection Unit)

Each of these MPUs allows the user to configure memory protection according to a slightly different set of rules, taking into consideration also the different positions of the MPUs within the system.

Apart from the MPUs, additional access control mechanisms can be used:

- Platform Flash Controller is able to control access to entire Flash based on Master ID
- Peripheral Access Control (PACR) on the Peripheral Bridge (PBRIDGE) is similarly able to control access to its peripherals based on Master ID
- REG_PROT - Register protection mechanism for individual function registers. Only a subset of all registers can be protected

This document aims to summarize all of these mechanisms and to provide simple code examples which help the user to configure their application.

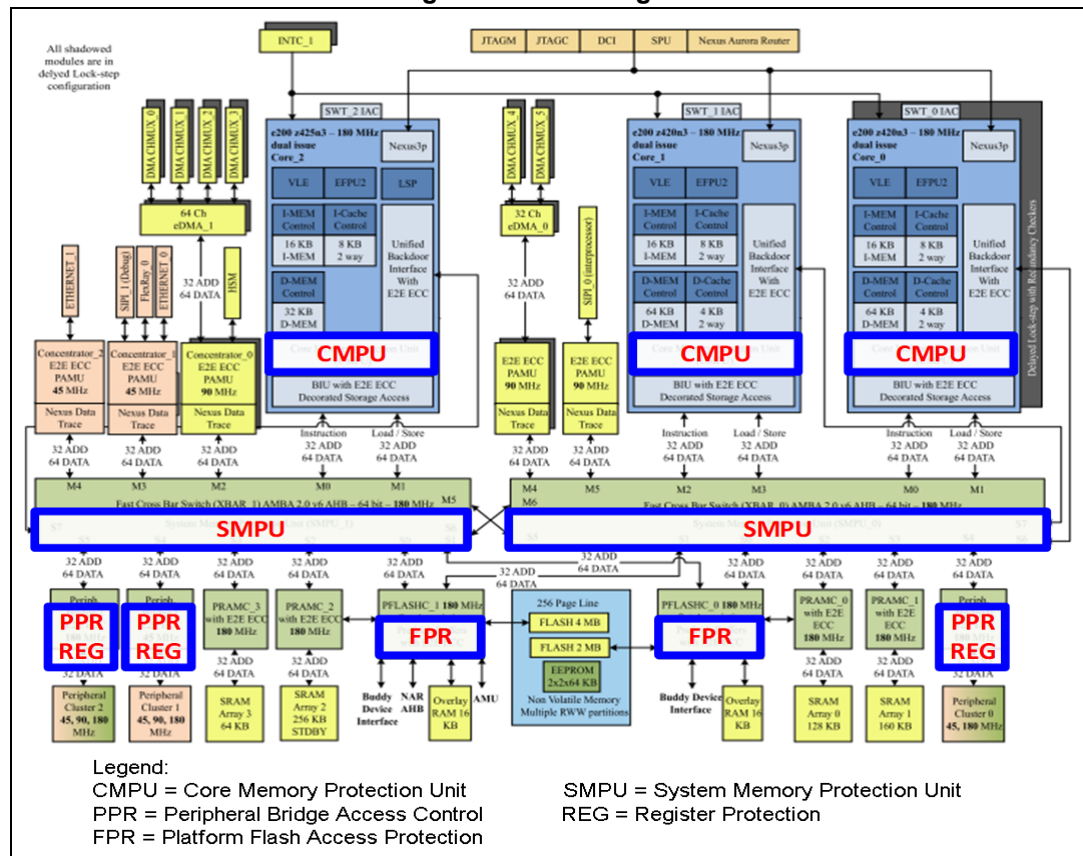
This document serves as an extension of the reference manual for SPC58 line of microcontrollers and to the Core Reference Manual for the Power Architecture e200z4 core, therefore please refer to these documents first.

1.1 Block diagram

The following block diagram visualizes the different memory protection mechanisms and their positions within the SoC. Based on the specific version of the device, there could be one or more instances of each of the memory protection units (for example, the number of CMPUs is identical to the number of cores within the microcontroller).

The different protection mechanisms are organized in a cascaded fashion. Therefore each possible access passes through multiple protection blocks, depending on the configuration.

Figure 1. Block diagram



1.2 Typical use cases

Typically, the different protection mechanisms are used in conjunction with each other, as each of these mechanisms provides protection at a different location of the microcontroller and provides a different set of protection rules.

The CMPU is specific to each core and is therefore usually used to restrict access based on process ID. It is not able to control access by other cores or other master-type peripherals (e.g., DMA engine).

The SMPU is attached to the Crossbar switch (XBAR), which is a hardware interconnect matrix that connects masters (such as microcontroller cores) to slaves (such as memory controllers and peripherals). It controls access to all slaves which are connected to the specific SMPU. Since there can be more than one SMPU in a device, the user must configure the correct SMPU depending on which slave it must protect.

The Peripheral Bridge Access Control mechanism is specific to each Peripheral Bridge (PBRIDGE). Like the SMPU, it can restrict access to all the slaves connected to the PBRIDGE based on Master ID. But it can't restrict access based on address ranges. An "all-or-nothing" principle applies - either the whole peripheral is protected from a specified master, or not.

The Platform Flash Access Control mechanism is almost identical to the PBRIDGE Access Control, only it works over the Flash memory, instead of the peripherals. Access control is again possible based on Master ID, and not on memory range.

Finally, Register Protection is an additional protection mechanism which protects individual registers against writing^(a). It offers a mechanism to protect defined memory-mapped address locations from being written to.

1.3 Master assignments and PBRIDGE slot assignments

This document uses the term "bus master" or "AHB Master" in many places. To find out more about the Master assignments of your particular microcontroller, please refer to the Reference Manual, chapter Device Configuration, subchapter AHB Master ID Assignments.

For example, [Table 1](#) is an extract from the reference manual of the SPC58xGxx microcontroller.

Table 1. Example of AHB master ID assignement on SPC58xGxx microcontroller

SMPU logical bus master number	Bus master
0	Core0 (z420n3_0)
1	Core1 (z420n3_1)
2	Core2 (z425n3)
3	eDMA_0
4	Ethernet 0

Additionally, also the term "PBRIDGE On-/Off-Platform Peripheral Slot" is used, especially in the chapter Peripheral Bridge Access Control. To find out more about the Peripheral Slot assignment, please refer to the Reference Manual, chapter Memory Map, subchapter Peripheral Memory.

For example, [Table 2](#) is an extract from the reference manual of the SPC58xGxx microcontroller.

Table 2. Example of Pheripheral Slot assignement on SPC58xGxx microcontroller

Start Address	End Address	Peripheral	"ON-Platform" Peripheral Slot	"OFF-Platform" Peripheral Slot	PBRIDGE	Size [Byte]
0xF4000000	0xF4003FFF	PBRIDGE_2	0		2	16384
0xF4004000	0xF4007FFF	XBAR_0	1		2	16384
0xF4008000	0xF400BFFF	XBAR_1	2		2	16384

a. The register protection mechanism doesn't protect all registers of every peripheral. Refer to the reference manual to have the list of these registers.

1.4 Differences between S MPU and C MPU

The S MPU lies between the Crossbar switch (XBAR) and the peripherals bridge. Every transaction from a master to a slave goes through the S MPU.

Given its position, the S MPU controls accesses based on:

- Address range
- AHB Master ID of the requestor (unique for each master, even across multiple crossbars)
- Cache-Inhibit/Guarded attributes

The S MPUs prevent access of different bus masters to specific address ranges. The safety application uses this layer of protection to grant freedom of interference among masters (e.g., avoiding unwanted accesses from the DMA accesses to the safety-relevant data of a task running on Core_0).

The C MPU is specific to each microcontroller core. As a consequence, it can't control accesses based on requestor ID.

Given its position, the C MPU controls access based on:

- Address range
- Process ID (PID)
- Supervisor/User execution mode
- Cache-Inhibit/Guarded attributes

The safety application uses this layer of protection to grant freedom of interference among tasks running on the same core.

C MPU and S MPU work independently from each other, and both contain separate descriptor sets. S MPU descriptors are memory mapped inside the S MPU peripheral. C MPU registers are not memory mapped; the user accesses them by using a set of core registers and dedicated instructions.

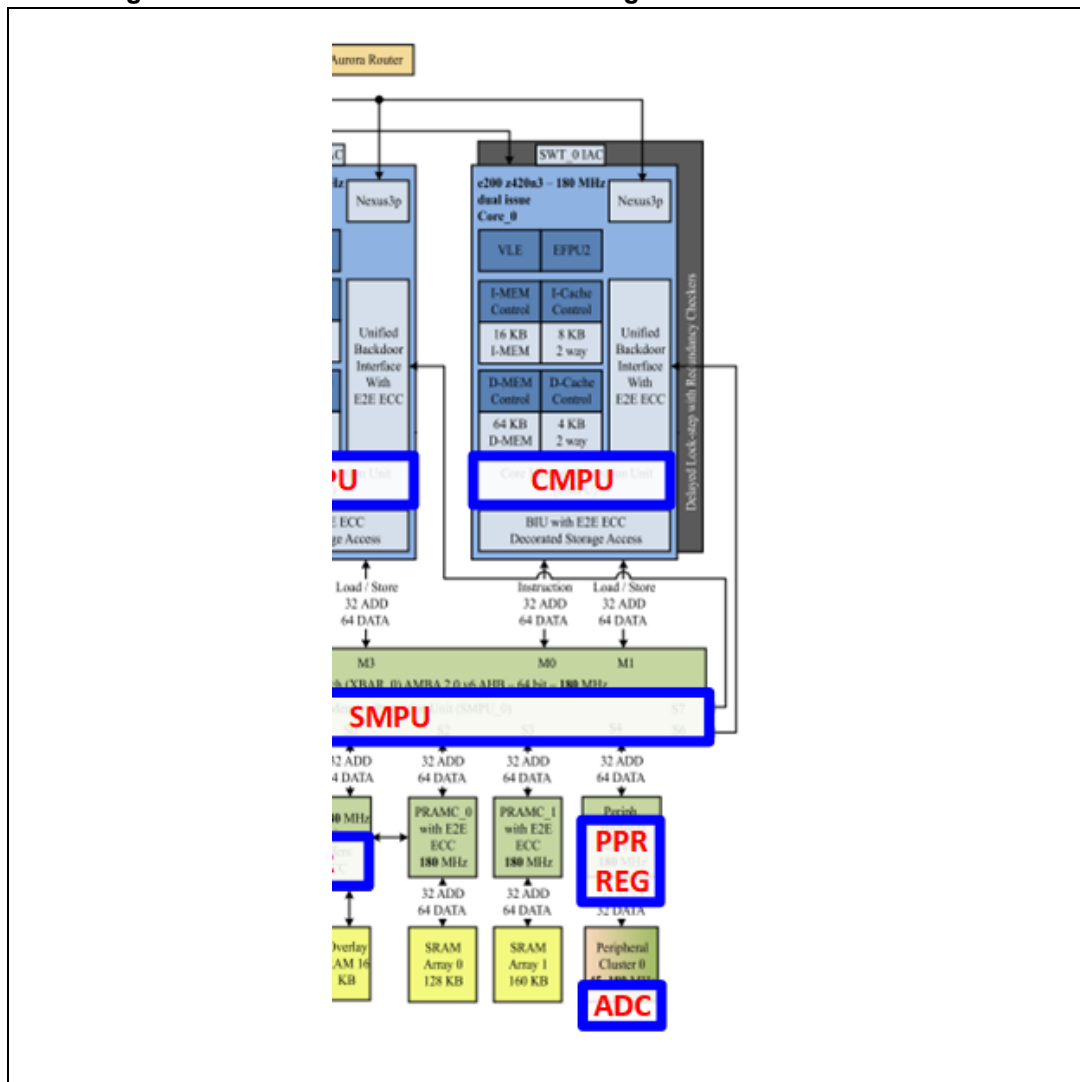
1.5 MPU configuration combinations

Figure 1 shows that the different memory protection mechanisms described in this document are "layered" onto each other.

1.5.1 Combinations of different memory protection mechanisms

The user can reach the most restriction protection by enabling all layers of protection.

Figure 2. Protection mechanisms affecting access to ADC on cluster 0



For example, if we consider the options of protecting certain peripherals on the peripheral cluster 0 (for example, an ADC, see [Figure 2](#)), the application can use the following protection mechanisms:

- Core MPU
- System MPU
- Peripheral Bridge Access Control
- Register Protection

The basic rules for access control are intuitive, as shown below:

1. If any of these protection units is configured to deny access to the peripheral, then the access is denied regardless of the configuration of the other protection mechanisms.
2. Both Core MPU and System MPU can configure certain areas as Cache-Inhibited. As long as at least one of the MPUs is configured to inhibit Caching, then it is inhibited.

For unique access control mechanisms (e.g. Process ID in CMPU), the configuration is applied unless there is a more restrictive configuration set on one of the other mechanisms.

For example, if certain Process IDs are configured as allowed in the C MPU, but the entire area is not allowed in the S MPU, then the access is not allowed.

1.5.2 Overlapping regions in system MPU

For overlapping regions in the S MPU, the following rules are applied:

For Access protection, the least restrictive configuration is used (i.e. descriptors that allow access take precedence before descriptors that deny access)

For Cache-Inhibit, the least restrictive configuration is used (i.e. descriptors that allow caching take precedence before descriptors that deny caching)

1.5.3 Overlapping regions in core MPU

For overlapping regions in the C MPU, the following rules are applied:

For Access protection, the least restrictive configuration is used (i.e. descriptors that allow access take precedence before descriptors that deny access)

For Cache-Inhibit attribute, the least restrictive configuration is used (i.e. descriptors that allow caching take precedence before descriptors that deny caching)

For Guarded attribute, the most restrictive configuration is used (i.e. descriptors that set region as guarded take precedence before descriptors that set it as not guarded).

2 Core Memory Protection Unit (CMPU)

2.1 Overview

The Core (z420/z425/z4256) Memory Protection Unit (MPU) provides the capability of protecting regions of memory, with the following feature set:

- 24-entry region descriptor table with support for
 - arbitrary-sized instruction memory regions
 - 12 arbitrary-sized data memory regions
 - 6 additional arbitrary-sized regions programmable as instruction or data memory regions
- Ability to set access permissions and memory attributes on a per-region basis
- Process ID aware, with per-bit masking of TID values
- Capability for masking upper address bits in the range comparison
- Capability of bypassing permissions checking for selected access types
- Per-entry write-once logic for entry protection
- Hardware flash invalidation support and per-entry invalidation protection controls
- Ability to optionally utilize region descriptors for generating debug events and watchpoints.

The CMPU provides access protection to memory regions using a set of region descriptors. Each region descriptor defines an address range and a set of access protections and memory attributes.

Each Individual region descriptor monitors either instruction or data accesses.

Independent access protection can be configured for supervisor/user mode reads, writes, and instruction accesses. Besides, any subset of these access types can bypass the protection system and instead rely on an external system MPU for enforcing access protections. This methodology allows for an increased number of regions to be supported in the SoC, without requiring excessive processor resources.

Memory attributes of Guarded (for data accesses only) and Cache-Inhibited (for both Data and Instruction accesses) are provided in each descriptor. These attributes are used to control certain aspects of access ordering and buffering, cache behavior, and also to drive external system interface access attributes.

2.2 Types of CMPU Entries and region descriptors

CMPU entries are split in three categories, for a total of 24 fully associative region descriptors:

- Instruction descriptors (max. 6)
- Data descriptors (max. 12)
- Shared descriptors (max. 6, each configurable to be either Instruction or Data type)

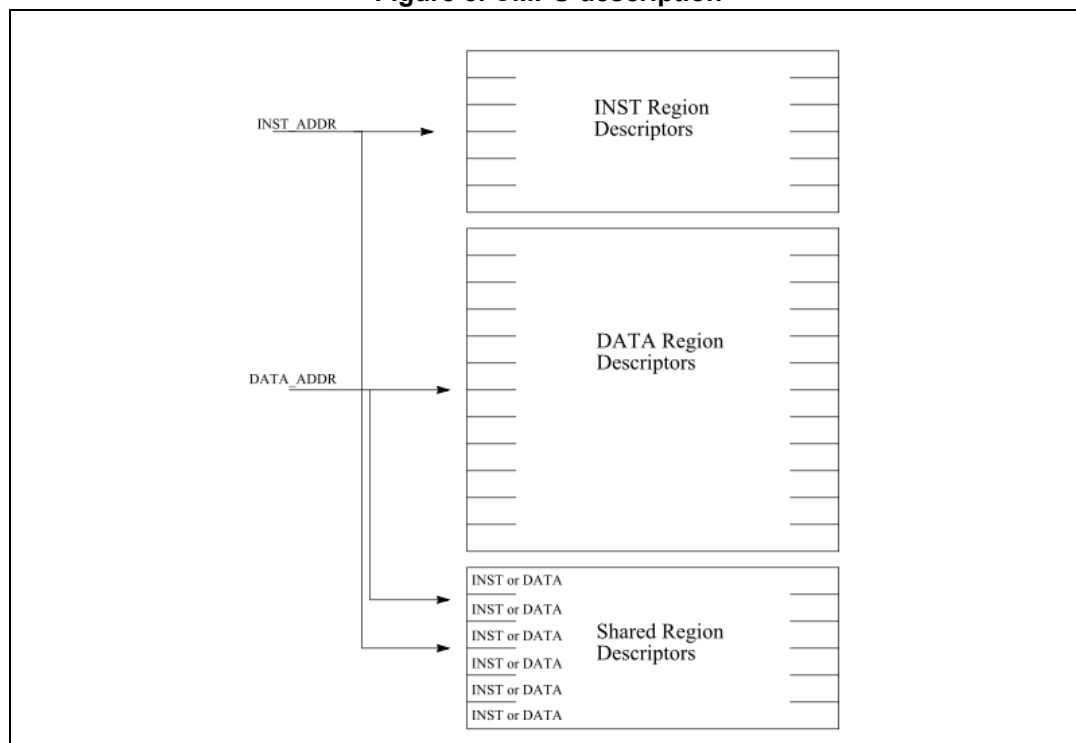
The addressing of the descriptors is done in the MAS0ESEL field. Each of the categories is being addressed starting from zero, i.e.:

- Instruction descriptors: ESEL = 0..5
- Data descriptors: ESEL = 0..11
- Shared descriptors: ESEL = 0..5

The type of the descriptor is given by MAS0INST and MAS0SHD:

- If ((INST = 1) && (SHD = 0)): Entry belongs to INST area
- If ((INST = 0) && (SHD = 0)): Entry belongs to DATA area
- If ((INST = 1) && (SHD = 1)): Entry belongs to Shared area and is used for matching instruction entries
- If ((INST = 0) && (SHD = 1)): Entry belongs to Shared area and is used for matching data entries

Figure 3. CMPU description



Each Region Descriptor contains an Upper and Lower Bound value that defines the size (range) of the region, as well as additional information such as the Region ID, which is compared with the current process ID in the PID register to determine if a match occurs, a Valid bit indicating the region descriptor is valid and should participate in the lookup process, masks for masking upper address bits as well as portions of the region ID, protection information, access attributes, and various control fields. The following table shows the information contained in a region descriptor. The various fields are described further in subsequent sections.

Table 3. CMPU Region descriptor entry bit definitions

Field	Comments
V	Valid bit for entry
UPPER_BOUND	Upper address bound (compared against effective address)
LOWER_BOUND	Lower address bound (compared against effective address)
TID[0:7]	Region ID (compared against PID value or '0')
TIDMSK[0:7]	Region ID Mask
UAMSK	Upper Address Mask control
INST	Instruction or Data Access entry (1=INST)
DEBUG	Entry set for debug event generation
SX, SW, SR	Supervisor execute, write, and read permission bits
UX, UW, UR	User execute, write, and read permission bits
I	Cache-Inhibited region attribute
IOVR	Cache-Inhibited region attribute override
G	Guarded region attribute (not present in dedicated INST entries)
GOVR	Guarded region attribute override (not present in dedicated INST entries)
IROT	Invalidation protect
RO	Read-only control for entry updates

2.3 Access matching

Access matching is performed by comparing the effective address (EA) for the access, and the process ID (PID) value used for the access with values stored in the region descriptors.

A hit to multiple MPU entries is not considered to be a programming error. If this occurs, the MPU will allow the access if any of the matching entries allows the access. Memory attributes are determined as outlined in the Core reference manual section "Memory Attributes (G, I)".

2.3.1 Access masks

Address masking (UAMSK)

The z420/z425/z4256 MPU provides the capability of per-entry masking of a subset of the upper effective address bits to zero prior to performing the address range comparison. Up to 5 MSBs of the access address may be masked to zero prior to the range compare. When using upper-address bit masking, the UPPER_BOUND and LOWER_BOUND are not masked, thus should typically be programmed with '0' in the masked bit positions.

Process ID masking (TIDMSK)

The z420/z425/z4256 MPU provides the capability of per-entry masking of a subset of the TID/PID^(b) bits via the entry TIDMSK field when performing the TID portion of the address range comparison for the entry.

Process ID masking in supervisor mode (MPU0CSR0_{TIDCTL})

The z420/z425/z4256 MPU provides the capability of global masking of the TID bits in all region descriptors to 0 when performing the PID/TID portion of the address range comparison for the entry while in Supervisor mode by means of the TIDCTL bit setting in the MPU0 control and status register (MPU0CSR0). This allows supervisor code to utilize region descriptors loaded for user tasks regardless of the programmed TID value in those descriptors. This capability can be used in certain situations to minimize the number of supervisor region descriptors required to be active, and thus improve efficiency. The actual contents of the descriptors is not changed, the TID values are only masked to 0 at the comparison logic.

Region descriptors used to cover Supervisor code or data areas may be programmed with a TID value of 0 such that they match regardless of the current PID register value, since for system calls and interrupts which originate during a user task, the current PID value may be arbitrary. Supervisor code which needs to access current user regions may do so using the user task's region descriptors, as long as the appropriate supervisor permissions have been set in those descriptors when loaded.

2.3.2 Permissions

An operating system may restrict access to address ranges by selectively granting permissions for user mode read, write, and execute, and supervisor mode read, write, and execute on a per-region basis. These permissions can be set up for a particular system (for example, program code might be execute-only, data structures may be mapped as read/write/no-execute) and can also be changed by the operating system based on application requests and operating system policies.

The UX, SX, UW, SW, UR, and SR access control bits are provided to support selective permissions (access control):

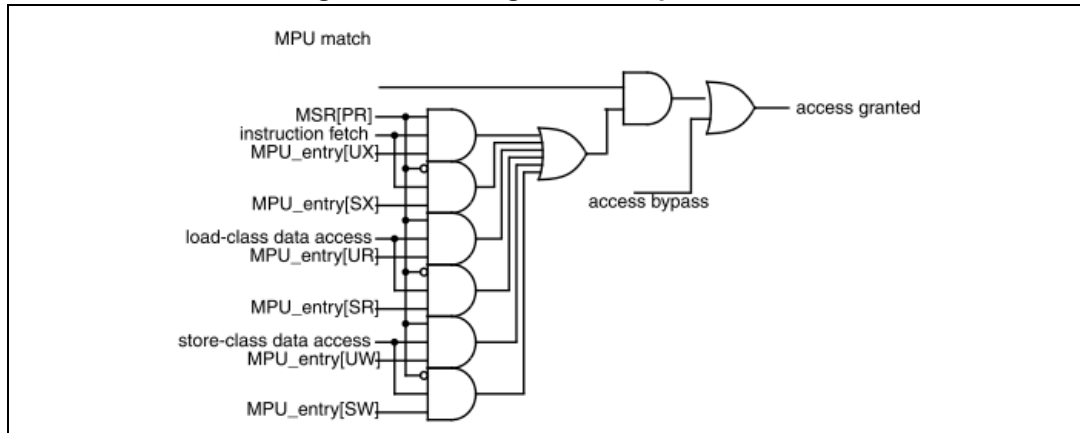
- SR-Supervisor read permission. Allows loads and load-type cache management instructions to access the region while in supervisor mode (MSRPR=0).
- SW-Supervisor write permission. Allows stores and store-type cache management instructions to access the region while in supervisor mode (MSRPR=0).
- SX-Supervisor execute permission. Allows instruction fetches to access the page and instructions to be executed from the region while in supervisor mode (MSRPR=0).
- UR-User read permission. Allows loads and load-type cache management instructions to access the region while in user mode (MSRPR=1).
- UW-User write permission. Allows stores and store-type cache management instructions to access the region while in user mode (MSRPR=1).
- UX-User execute permission. Allows instruction fetches to access the page and instructions to be executed from the region while in user mode (MSRPR=1).

If the access match was successful, the permission bits are checked as shown in [Figure 4](#). If the access is not granted, the processor generates an Instruction or Data Storage interrupt

-
- b. The PowerISA 2.06 architecture defines that a process ID (PID) value is associated with each effective address (instruction or data) generated by the processor. A single PID register is defined as a 32-bit register, and it maintains the value of the PID for the current process. This PID value is included as part of the address in the access protection process. For the z420/z425/z4256, the supported PID values are 8 bits in length. The most-significant 24 bits are unimplemented and read as '0'.

(ISI or DSI), unless the access type is allowed to bypass MPU protections via settings in MPU0CSR0.

Figure 4. Granting of access permission



2.3.3 Memory attributes (G, I)

Each descriptor contains memory attribute information for the corresponding memory address range covered by the descriptor. The G and I bits are provided to indicate whether a matching data access should be treated as guarded and/or cache-inhibited. These attributes are applied on a matching data access.

If more than one descriptor attempts to supply memory attributes, the most restrictive attribute settings are used for the Guarded attribute (ORing of G bits), and the least restrictive access settings are used for cacheability (ANDing of I bits), unless override control is enabled in one of the matching descriptors. If a matching descriptor's address range overlaps with the ranges defined by the guarded address ports or cache-inhibited address ports, the address ports' controls take priority, unless the override bit(s) for G and/or I are set in the region descriptor. The actual values of the address ports can be found in this documents' chapter 8.2.1 - Cache-Inhibit and Guarded Control Signals.

You can find more specific details in the Core Reference Manual chapter Memory Protection Unit.

Memory attribute overrides (GOVR, IOVR)

Each descriptor also contains memory attribute override controls, which can be used to override the setting(s) of other matching descriptor(s). The GOVR (dedicated data and INST=0 entries only) and IOVR bits are provided to indicate whether a matching access use the G and I settings of a descriptor regardless of the settings of any other matching descriptor or of the state of the guarded address or cache-inhibited address ports. Only a single matching descriptor is allowed to have GOVR or IOVR set, although it is allowable for two different matching descriptors to each have one of these set which is not set in another matching descriptor. If multiple GOVR or IOVR bits are set, the result is undefined. No detection mechanism is provided for this case.

2.3.4 Access bypass

Access bypass of the MPU is supported for individual access types by control information located in the MPU0 Control and Status Register 0 (MPU0CSR0). When a specific access

type is enabled to bypass the MPU, no access protection checking is performed for that access type. In addition, no access attributes are supplied, unless a region descriptor match occurs. Control is provided to individually bypass supervisor read, write or instruction fetch accesses using the MPU0CSR0BYPSR, BYPSW, BYPSX control bits, as well as user read, write and instruction fetch accesses using the MPU0CSR0BYPUR, BYPUW, BYPUX control bits. Refer to the reference manual for more details about the MPU0CSR0 register.

Accesses to the DMEM and IMEM may also bypass MPU protections when configured in the DMEMCTL1 or IMEMCTL1 registers respectively. These controls override any MPU-based settings unless configured to use the MPU protections. Note that for MPU region descriptors being used for debug, accesses are still compared for matches, regardless of bypass settings in the local memory control registers or MPU control registers.

2.4 Use of region descriptor table entries for debug

The z4256n3 MPU supports an alternate use of region descriptor table entries for debug purposes if they are not needed to support memory protection. This provides the user with additional debug resources beyond those described in Chapter 14, "Debug Support".

You can find more information about the Use of Region Descriptor Table Entries for Debug in the Core Reference Manual.

2.5 Software interface and CMPU instructions

Please refer to the reference manual of your device.

Chapter: System Modules, subchapter Core Description, subchapter MPU.

2.6 CMPU control registers

Please refer to the reference manual of your device.

Chapter: System Modules, subchapter Core Description, subchapter MPU.

2.7 CMPU operations

Please consult the reference documentation for the description of the individual registers (e.g. MASx, MPU0CSR0, etc.) which are being accessed in the following code snippets.

Reading entries from the CMPU: Individual entries in the MPU region descriptor table can be read by first writing the necessary entry select information into MAS0 using mtspr and then executing the mpure instruction. To read an entry from the MPU, the INST, SHD, and ESEL bits in MAS0 must be set to point to the desired entry. After executing the mpure instruction, MAS0-MAS3 will be updated with the data from the selected MPU entry.

```
#define MAS(x)    624 + x //MAS registers are 624, 625, 626 and 627
```

```
uint32_t MAS0_SPR_Val, MAS1_SPR_Val, MAS2_SPR_Val, MAS3_SPR_Val;
```

```

__mtpspr(MAS(0), 0x01000000); // Prepare to read entry 0 of INST descriptor
                                // table
asm("mpure")                    // Read the MPU entry

MAS0_SPR_Val = __mfspr(MAS(0)); // Read contents of MAS0 .. MAS3
MAS1_SPR_Val = __mfspr(MAS(1));
MAS2_SPR_Val = __mfspr(MAS(2));
MAS3_SPR_Val = __mfspr(MAS(3));

```

Writing Entries into the CMPU: Individual entries in the MPU region descriptor table can be written by first writing the necessary information into MAS0-MAS3 using mtpspr and then executing the mpuwe instruction. To write an entry into the MPU, the INST, SHD, and ESEL fields in MAS0 must be set to point to the desired entry. When the mpuwe instruction is executed, the MPU entry information stored in MAS0-MAS3 will be written into the selected MPU region descriptor.

```

#define MAS(x)    624 + x //MAS registers are 624, 625, 626 and 627

uint32_t MAS0_SPR_Val, MAS1_SPR_Val, MAS2_SPR_Val, MAS3_SPR_Val;

__mtpspr(MAS(0), MAS0_SPR_Val); // Prepare the contents of MAS0 .. MAS3
__mtpspr(MAS(1), MAS1_SPR_Val);
__mtpspr(MAS(2), MAS2_SPR_Val);
__mtpspr(MAS(3), MAS3_SPR_Val);
asm("mpuwe");                    // Write the values to the MPU table

```

MPU invalidation: The CMPU supports hardware flash-invalidation of region descriptor table entries via the MPU0CSR0MPUFI control bit. This provides software with an efficient mechanism for reloading region descriptor table entries on a user task switch, since existing task entries can be flash-invalidated in one control operation, and the entries needed for the new task can be loaded without examining each entry to see if it belongs to the old task.

The MPU invalidation takes three clock cycles to complete.

```

#define MPU0CSR0_SPR 1014

uint32_t val;

__mtpspr(MPU0CSR0_SPR, 0x00000002); // Request invalidation operation

// Read the register repeatedly until INV flag is cleared
do {
    val = __mfpsr(MPU0CSR0_SPR) & 0x00000002;
} while(val!=0);

```

IPROT Invalidation Protection: In case some entries should not be invalidated, the IPROT bit is used to protect MPU region descriptor table entries from invalidation. MPU entries with IPROT set are not invalidated by the MPU0CSR0MPUFI control function. The IPROT bit may be used to protect interrupt vectors/handlers (and optionally the supervisor stack), since the instruction fetch of those vectors must be guaranteed to never take a ISI exception.

```
// For this example, the "Writing Entries into the CMPU" example can be
reused, with the following modification
```

```
// Add the IPROT flag to the entry
__mtspr(MAS(0), (MAS0_SPR_Val | 0x40000000));
```

Enabling CMPU protection for tightly coupled memories (IMEM/DMEM): Accesses to the DMEM and IMEM may also bypass MPU protections when configured in the DMEMCTL1 or IMEMCTL1 registers respectively. These controls override any MPU-based settings unless configured to use the MPU protections.

Note: *Not all members of the Chorus family contain IMEM and DMEM memories. Please consult the datasheet of your device.*

```
#define DMEMCTL1_DCR 498
#define IMEMCTL1_DCR 499

__mtdcr(DMEMCTL1_DCR, 0xAAAAAAAA); //DMEM accesses conditionally allowed
//based on the CMPU logic
__mtdcr(IMEMCTL1_DCR, 0x00AA00AA); //IMEM accesses conditionally allowed
//based on the CMPU logic
```

3 System Memory Protection Unit (SMPU)

3.1 Overview

The System Memory Protection Unit (SMPU) is a memory protection unit which provides hardware access control for system bus memory references. The SMPU concurrently monitors and evaluates system bus transactions using pre-programmed region descriptors that define memory spaces and their access rights. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with an access error response.

The SMPU feature set includes:

- Supports up to 24 program-visible 128-bit region descriptors, accessible as four 32-bit words each. (Specific module instances may support fewer than 24)
 - Each region descriptor defines an arbitrarily sized space, aligned anywhere in memory Region sizes can vary from a minimum of 1 byte to a maximum of (4 GB minus 1 byte)
 - Read/write access control permissions defined in region descriptor
 - Cache-inhibit attribute indicator per region descriptor
 - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
 - Priority given to granting permission over denying access for overlapping region descriptors
- Supports as many as eight crossbar slave ports
- Error registers (per bus master ID) capture the last faulting address, attributes, and other information.

The SPC58 chips contain up to three instances of SMPU:

- All devices contain a Peripheral shell SMPU
- 6MB and larger devices contain a Computational shell SMPU
- Devices with HSM (Hardware Security Module) contain HSM SMPU

3.2 Types of SMPU entries and region descriptors

Unlike CMPU entries, there is only a single common type of SMPU entries. There can be up to 24 SMPU region descriptors for each SMPU instance.

The SMPU does not distinguish between instruction and data accesses, however it is able to restrict access separately for reads and writes. For the purpose of memory protection, instruction fetch operation is considered a "read" operation.

Each Region Descriptor consists of four 32bit registers names RGDn_WORD0-3.

Table 4. SMPU region descriptor entry bit definitions

Register	Field	Comments
RGDn_WORD0	SRTADDR	Start address. Defines the byte start address of the memory region.
RGDn_WORD1	ENDADDR	End address. Defines the byte end address of the memory region. <i>Note: The SMPU does not verify that ENDADDR > SRTADDR</i>
RGDn_WORD2	MxP (M0P, M1P, ...)	Bus master x permissions. 2-bit field to configure read/write permissions for each master.
RGDn_WORD3	FMT	Region Descriptor Format selection (FMT0/FMT1). Fixed value of 0 is applied. FMT1 is not supported.
	RO	Read only. This bit is intended to prevent accidental writes of an SMPU region descriptor. Setting RO locks the descriptor until a system reset.
	CI	Cache Inhibit. Defines the cacheability attribute of the memory region.
	VLD	Valid. Signals that the region descriptor is valid. Any write to RGDn_WORD0–2 clears this bit.

3.3 Access matching

The basic operation of the SMPU is performed in a so-called "Access evaluation macro". This is a hardware structure, which is a part of the SMPU. Details about the Access evaluation macro can be found in the reference manual, chapter "Access evaluation macro".

To determine if the current reference hits in the given region, two comparators are used with the region's start (SRTADDR) and end (ENDADDR) addresses.

Furthermore, the logic evaluates whether the current access is allowed by the permissions defined in the region descriptor. The protection violation logic evaluates the access against the effective permissions, using the specification shown in [Table 5](#).

Table 5. Protection violation definition

Access Type	Read permission	Write permission	Protection violation?
Instruction fetch read	0	not applicable	Yes, no read permission
	1	not applicable	No, access is allowed
Data read	0	not applicable	Yes, no read permission
	1	not applicable	No, access is allowed
Data write	not applicable	0	Yes, no write permission
	not applicable	1	No, access is allowed

3.3.1 Error terminations

For each slave port monitored, the SMPU performs a reduction-AND of all the individual terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports an access error for three conditions:

1. The access does not hit in any region descriptor.
2. The access hits in a single region descriptor and that region has a protection violation.
3. The access hits in multiple (overlapping) regions and all regions have protection violations.

As shown in the third condition, granting permission is a higher priority than denying access for overlapping regions. This approach is more flexible to system software in region descriptor assignments.

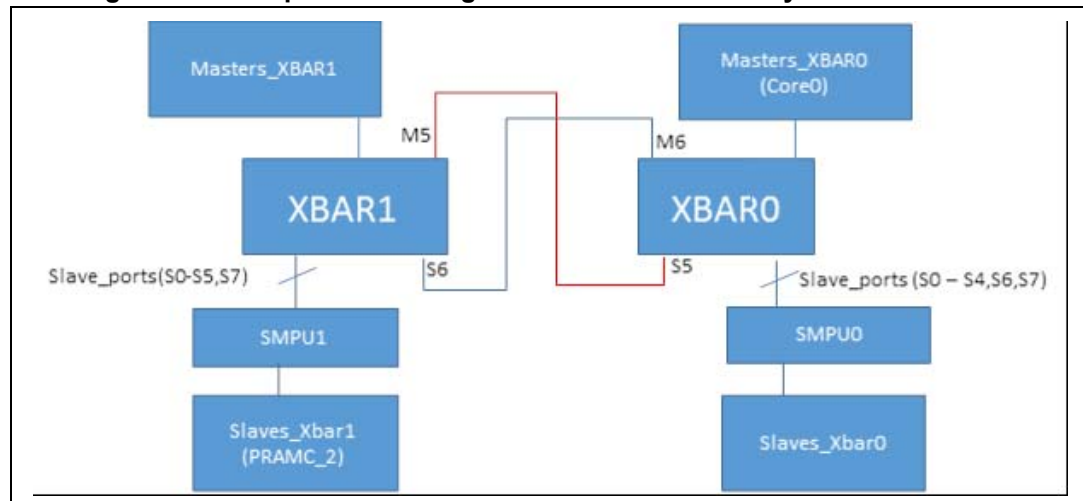
3.3.2 SMPU functionality on multi-crossbar devices

The SMPU protects accesses to the slave ports directly connected to it. This means that in case of a request by a master on one crossbar to access slave on a different crossbar, only the SMPU on the last crossbar in the line is applied (i.e., SMPU_i monitors accesses to the slaves of the XBAR_i).

Figure 5 shows that the SMPU0 doesn't cover the XBAR interconnection (slave 5 of the XBAR0). As a consequence, if the application needs to restrict accesses to PRAMC_2 from Core0 then the user must configure the SMPU1 accordingly. The SMPU0 is not included in the signal path to the RAM controller 2 (or any other slave on XBAR1).

The same applies vice-versa also for masters on XBAR1 and slaves on XBAR0 (the user must configure the SMPU0 in that case).

Figure 5. SMPU protection diagram for multi-crossbar systems SPC58xGx



3.4 SMPU control registers

Please refer to the reference manual of your device.

Chapter: System Modules, subchapter System Memory Protection Unit, subchapter Register Definition.

3.5 SMPU operations

Creating a new memory region: load the appropriate region descriptor into an available RGDn, using four sequential 32-bit writes. The hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues with the multi-cycle descriptor writes.

```
SMPU_x.RGD[n].WORD0.B.SRTADDR = 0x00000000; // Addr. range start
SMPU_x.RGD[n].WORD1.B.ENDADDR = 0xFFFFFFFF; // Addr. Range end
SMPU_x.RGD[n].WORD2_FMT0.R = 0xFFFFFFFF; // All masters allowed
SMPU_x.RGD[n].WORD3.B.VLD = 1; // Entry is valid

SMPU_x.CESR0.B.GVLD = 1; // SMPU module is enabled and also all the
                        // descriptors
```

Modifying a region descriptor: load the updates into the region descriptor using sequential 32-bit writes. Writing to Word3 re-enables the region descriptor valid bit. The hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues with the multi-cycle descriptor writes.

```
SMPU_x.RGD[n].WORD3.B.VLD = 0; // Invalidate the entry for
                        // configuration
SMPU_x.RGD[n].WORD2_FMT0.B.M2P = 0; // Master 2 no longer allowed
SMPU_x.RGD[n].WORD3.B.VLD = 1; // Entry is re-enabled
```

Removing a region descriptor: clearing RGDn_Word3.VLD disables an existing region descriptor.

```
SMPU_x.RGD[n].WORD3.B.VLD = 0; // Invalidate the entry
```

3.5.1 Additional operations

Detecting an access error: the current bus cycle terminates with an error response. The EARN and EDRn status registers capture some details on the error event. The error-terminated bus cycle triggers an error response in the originating bus master.

For example, a core reacts with a bus error exception, and a data movement bus master (e.g., an error during a DMA transfer) reacts with an error interrupt. The processor can retrieve the captured error address and detail information simply by reading E{A,D}Rn.CESR0.MERR registers.

Overlapping region descriptors: the usage of overlapping regions often reduces the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator).

4 Flash controller platform access control

4.1 Overview

The SPC58 line of devices usually implements up to 2 Flash Memory Controllers. The flash memory controllers act as an interface between the system bus (AHB-Lite 2.v6) and the Flash memory arrays.

One feature of the Flash Memory Controller is the ability to configure access control based on Read/Write attributes and AHB Master IDs (similarly to the SMPU). While this feature does not allow to control access based on address range, it is possible to allow or restrict access to the entire Flash area from a specific AHB Master.

This can be particularly useful in microcontrollers with multiple Flash controllers and multiple cores, where one flash controller can be configured to only allow access from one core, and the other flash controller can be configured to only allow access from the other core. Even on single-flash microcontrollers, it might be useful to restrict some Masters from accessing the Flash memory, for example because they do not need to use it as they only use data in RAM.

The PFAPR allows to configure individual access rights for read, write, or both.

4.2 Access control

4.2.1 PFAPR - Platform flash access protection register

The Flash memory controller provides programmable, configurable access protections for both read and write cycles on a per-master basis via the PFlash access protection register (PFAPR). It allows restriction of read and write requests on a per-master basis. Detection of a protection violation results in an error response from the Flash memory controller on the AHB transfer.

The options given for each master are as follows:

Master x Access Protection

Controls whether read and write accesses to the Flash are allowed based on the master ID of a requesting master. These fields are initialized by hardware reset.

- 00 No accesses may be performed by this master
- 01 Only read accesses may be performed by this master
- 10 Only write accesses may be performed by this master
- 11 Both read and write accesses may be performed by this master

For an example of PFAPR configuration, please refer to [Appendix C](#).

4.3 Flash controller platform access control registers

Please refer to the reference manual of your device.

Chapter: Memories and memory interfaces, subchapter Flash memory controller, subchapter Flash memory controller memory map.

4.4 Flash controller platform access control operations

Configure Flash Controller Platform Access Control: configure the flash controller to only accept accesses from a single master.

```
PFLASH_x.PFAPR.R = 0; //Deny access to all
PFLASH_x.PFAPR.B.M8AP = 3; // 0 = Deny all
                        // 1 = Read-only
                        // 2 = Write-only
                        // 3 = Full access
```

Note: *It is recommended to perform all Flash Controller reconfigurations with RAM-code.*

Disable Flash Controller Platform Access Control: configure the flash controller to accept accesses from all masters.

```
PFLASH_x.PFAPR.R = 0xFFFFFFFF; //Full access to all
                        // 0 = Deny all
                        // 1 = Read-only
                        // 2 = Write-only
                        // 3 = Full access
```

5 Peripheral bridge access control

5.1 Overview

The Peripheral Bridge converts the crossbar switch interface to an interface that can access the peripherals on the device. The slave devices connected to the peripheral bridge are modules that contain readable/writable control and status registers. The system masters read and write these registers through the peripheral bridge. The peripheral bridge generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the peripherals. The peripheral bridge captures read data from the peripheral interface and drives it to the crossbar switch.

The peripheral bridge can restrict accesses to the bridge and the underlying peripherals based on AHB Master ID, similarly to the Flash controller Platform Access Control, described in the previous chapter.

On most of the devices, there are multiple Peripheral Bridges present, each with its own set of peripherals attached to it. To correctly configure their application, the user must analyze which peripherals lie on which PBRIDGE, and which Peripheral slot on the particular PBRIDGE they occupy. For this task refer to the Reference manual, chapter "Memory Map", table "Peripheral Memory Map".

5.2 Access control

The access control inside the peripheral bridge is configured by the following two sets of registers, which together form a set of rules for access protection:

5.2.1 Master privilege register

Each MPR specifies eight 4-bit fields defining the access privilege level associated with a bus master in the device to the various peripherals. There are two MPR registers, which provide one MPROTx field per bus master.

Each MPROTx field allows the user to configure:

Table 6. Configuration bits for master privilege register

1 MTR	2 MTW	3 MPL
Master trusted for read Determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.	Master trusted for writes Determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.	Master privilege level Determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user mode. 1 Accesses from this master are not forced to user mode.

For an example of PBRIDGE access control configuration, please refer to [Appendix D](#).

5.2.2 Peripheral access control register

Each of the off-platform and on-platform peripherals has a four-bit (O)PACRn field which defines the access levels supported by the given module. Eight (O)PACR fields are grouped together to form a 32-bit (O)PACRx register.

The peripheral assignments to each (O)PACR field are defined by the memory map slot to which the peripherals are assigned. See the Memory Map chapter for the assignments of the device. If a peripheral is absent, the corresponding PACR field is not implemented. Reads to the location return zeroes, and writes are ignored.

Each (O)PACR field allows the user to configure:

Table 7. Configuration bits for peripheral access control register

1 SP	2 WP	3 TP
Supervisor protect Determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The master privilege level must indicate supervisor access attribute, and the MPROTn[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated.	Write protect Determines whether the peripheral allows write accesses. 0 This peripheral allows write accesses. 1 This peripheral is write-protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated.	Trusted protect Determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated.

The PACRs work together with the MPRs and allow the user to configure access rights individually to each peripheral connected to the particular PBRIDGE. For example, the PACRs can be used to configure some peripherals to only allow accesses from "Trusted" masters - but the MPRs define which masters are considered "Trusted".

The main differences between "On-Platform" and "Off-Platform" are:

- On-Platform peripherals are peripherals, whose data paths are connected directly to the XBAR or are parts of the core itself (e.g. Software Watchdog Timer, Interrupt Controller, DMA controller, etc.). These peripherals expose a set of configuration registers, which are mapped into the address space through the Peripheral Bridge. In other words, by configuring PBRIDGE Access Control for On-Platform Peripheral, only the configuration registers are protected, but the actual data accesses done through the data paths (e.g. DMA transfers) remain unprotected.
- Off-Platform peripherals are peripherals, whose data paths are connected to the XBAR and the rest of the MCU via a Peripheral bridge (e.g. ADCs, LIN, etc.). Similarly to the On-platform peripherals, the configuration registers are also connected to the rest of the microcontroller through the Peripheral Bridge. In other words, by configuring PBRIDGE Access Control for Off-Platform peripherals, both the data paths and the configuration registers are protected.

For an example of PBRIDGE Access Control configuration, please refer to [Appendix D](#).

5.3 PBRIDGE access control registers

Please refer to the reference manual of your device.

Chapter: System Modules, subchapter Peripheral Bridge, subchapter Memory map and register description.

5.4 PBRIDGE access control operations

Configure Master Privilege Register: in this example, master 0 is forced to user mode (i.e. attempted supervisor mode access automatically converted to user mode access) and is not trusted for write or read. Only peripherals configured for access by untrusted masters can be accessed.

```
PBRIDGE_x.MPRm.B.MPROTn = 0;    // Master n untrusted
                                   // 0 = Deny all
                                   // 1 = Accesses not forced to user mode
                                   // (supervisor mode access allowed)
                                   // 2 = Master trusted for write
                                   // 4 = Master trusted for read
                                   // and possible combinations (7 = full
access)
```

Disable Master Privilege Control: configure one or all masters as fully trusted

```
PBRIDGE_x.MPRm.B.MPROTn = 7;    // Master n fully trusted

PBRIDGE_x.MPRA.R = 0x77777777;  // All masters fully trusted
PBRIDGE_x.MPRB.R = 0x77777777;
```

Configure Peripheral Access Control Register: a single on-platform (PACR registers) or off-platform (OPACR registers) is configured to require supervisor type access, to deny write accesses, or to only allow trusted master to access it.

```
PBRIDGE_x.PACRm.B.PACRn = 0// e.g. m = A, n = 1
                                   // 1 = Only trusted Masters allowed
                                   // 2 = Write protected
                                   // 4 = Supervisor access only
                                   // and possible combinations
```

Disable Peripheral Access Control: configure one or all peripherals on the PBRIDGE to accept all accesses.

```
PBRIDGE_x.PACRm.B.PACRn = 0; // No access restrictions for peripheral on  
                        // slot n
```

```
PBRIDGE_x.PACRm.R = 0x00000000; // No access restrictions active (m = A..H)  
PBRIDGE_x.PFAPR.B.M8AP = 3; // 0 = Deny all
```

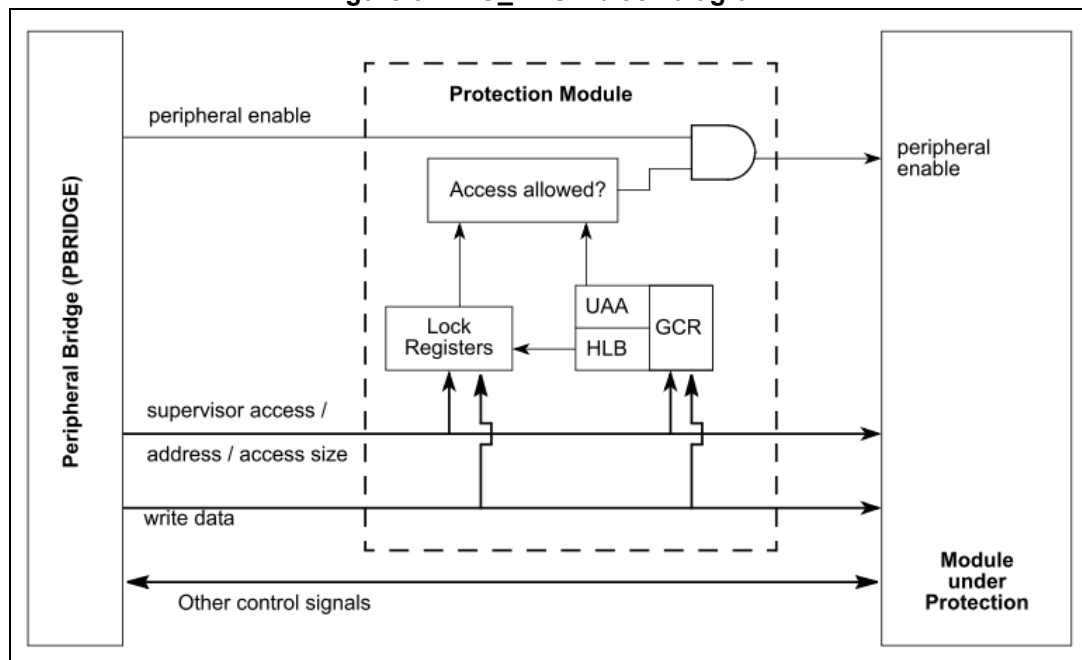
Note: *This is not the default configuration. The default configuration after reset is such that all peripherals require supervisor mode access.*

6 Register protection (REGPROT)

6.1 Overview

Register protection allows individual registers to be "locked" against writing. It offers a mechanism to protect defined memory-mapped address locations from being written to. The functionality is facilitated by the REG_PROT module, which is located between the peripheral bridge (PBRIDGE) and the module under protection, as you can see in [Figure 6](#). The address locations that can be protected are module-specific.

Figure 6. REG_PROT block diagram



Register Protection includes these features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Write to address mirror automatically sets corresponding lock bit
- Once configured, lock bits can be protected from changes

Also, hard lock bit protection can only be cleared by a system reset once set.

The Register Protection module is in operation whenever the module under protection is in operation.

For all addresses that are protected there are SLBRn.SLBm bits which specify whether the address is locked. When an address is locked it can only be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBRn.SLBm bit is always 0b0 no matter what software is writing to it.

Please note that not all of the registers can be protected using the REG_PROT. For a complete list of protectable registers, please refer to the reference manual of your device.

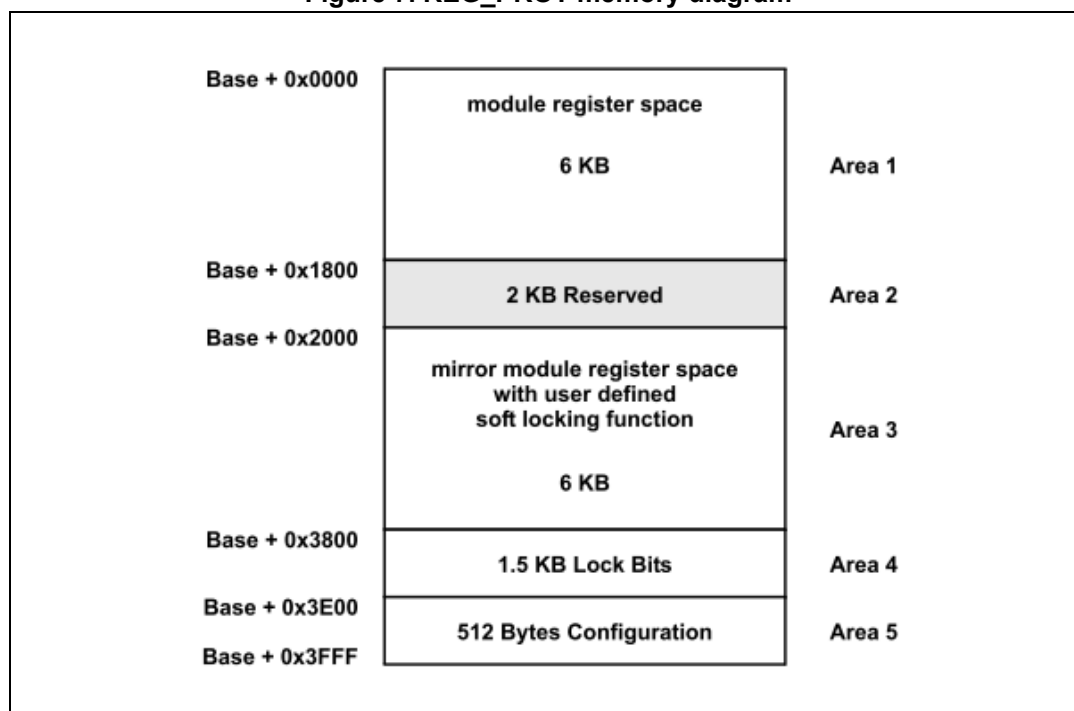
6.2 List of protected peripherals

The list of peripherals, whose registers can be protected by the REG_PROT functionality can be found in the reference manual of the device, chapter Device Configuration, subchapter Safety modules, subchapter Register Protection Configuration.

6.3 Access control

If a peripheral is listed as having Register Protection functionality (see above), then the following register organization applies:

Figure 7. REG_PROT memory diagram



The above diagram can be considered a "template", which is overlaid over the registers of any Register protection capable peripheral. Let's consider the SIUL2 peripheral for an explanation. The SIUL2 peripheral has Register Protection capability and therefore the above "template" works as follows:

The SIUL2 registers start at address 0xF7FC_0000 (see chapter Memory Map). Using the template shown above, we arrive at the following:

Table 8. Memory map SIUL2 registers including register protection

Address	Contents
0xF7FC_0000 – 0xF7FC_17FF	SIUL2 registers (<i>note: the memory range might not be fully utilized</i>)
0xF7FC_1800 – 0xF7FC_1FFF	Reserved area
0xF7FC_2000 – 0xF7FC_37FF	Mirrored SIUL2 registers with user defined soft locking functionality

Table 8. Memory map SIUL2 registers including register protection (continued)

Address	Contents
0xF7FC_3800 – 0xF7FC_3DFF	Soft Lock Bits used for register protection configuration
0xF7FC_3E00 – 0xF7FC_3FFF	General register protection configuration for SIUL2 (Global Configuration Register) (Note: Global Configuration Register is global for all SIUL2 Lock Bits, but not for all Register Protection capable peripherals)

The memory map of all other Register protection capable peripherals can be obtained in the same way.

6.3.1 The soft lock bits and the global configuration register

The Soft Lock Bit Registers hold the Soft Lock Bits for the protected registers in the normal register address space of the protected module (i.e. 0x0000 - 0x17FF range). Each SLB register has a four Soft Lock Bits (SLB0-SLB3), each of which controls write access to a byte in the normal register address space. Each Soft Lock Bit also has a corresponding Write Enable bit in the same register that controls whether the Soft Lock Bit can be written. The following table shows the mapping between the Soft Lock Bits to the bytes in memory area.

Table 9. Soft lock bits vs. protected addresses

Soft Lock Bit	Protected Address
SLBR0.SLB0	MR0 (e.g. SIUL2: 0xF7FC_0000)
SLBR0.SLB1	MR0 (e.g. SIUL2: 0xF7FC_0001)
SLBR0.SLB2	MR0 (e.g. SIUL2: 0xF7FC_0002)
SLBR0.SLB3	MR0 (e.g. SIUL2: 0xF7FC_0003)
SLBR1.SLB0	MR0 (e.g. SIUL2: 0xF7FC_0004)
SLBR1.SLB1	MR0 (e.g. SIUL2: 0xF7FC_0005)
...	...

Important: Clearly, the Soft Lock Bits can be configured for each byte, meaning that each 32b register actually has four soft lock bit sets (SLBRx.SLB0-3).

The Global Configuration Register controls the configuration of the locking of the SLB's and the level of user access allowed to the protected registers. It is able to configure Hard Lock. This means that once the lock configuration is entered and the Hard Lock bit is enabled, the lock configuration cannot be changed by software, and can only be changed by system reset. Additionally, it is able to restrict register access to Supervisor Mode. For more details about the User/Supervisor mode of the MCU, please refer to the reference manual.

6.4 REG_PROT control registers

To find information about the Soft Lock Bits and the Global Configuration Register, please refer to the reference manual of your device.

Chapter: Safety, subchapter Register Protection, subchapter Memory map and register definition.

6.5 REG_PROT operations

Changing lock settings

To change the setting whether an address is locked or unlocked, the corresponding SLBRn.WEm bit needs to be set to 1 to enable Soft Lock Bit modification. Once the corresponding Soft Lock Bit is enabled for writing, the following values can be written into the SLBRn.SLBm field:

Table 10. SLBR configuration options

SLBRn.SLBm	Result
0	Associated MRn byte is locked against write accesses
1	Associated MRn byte is unprotected and writable

The described operation can be programmed as follows (SIUL2.MSCR[0] register selected as an example).

```
base_address = (uint32_t) (& SIUL2);
base_offset = ((uint32_t) (& SIUL2.MSCR[0].R)) - base_address;

// Calculate SLB register address - add 0x3800 and then 1/4 of the offset
// between SIUL2 base and the register to be protected (8-bit SLBR registers
// are used to protect 32-bit register)
slb_address = base_address + 0x3800 + (base_offset / 4);

(*(uint8_t *) slb_address) = 0xff; // Unlock the register for protection
                                   // configuration (unlock all)
                                   // and configure the protection
                                   // (protect all)
```

Enabling locking via mirror module space

An alternative way to lock a register is to use the mirror module space at address offset 0x2000 (see [Table 8](#)). If the user performs a write to the mirror module space, then the register at the original address is locked automatically. The width of the lock depends on the width of the write into the mirror module space.

For example, considering the register SIUL2.MSCR[0], if the user writes a 32b value to SIUL2.MSCR[0] + 0x2000, then the value is updated, and the entire 32bit register is locked automatically.

```
register_address = (uint32_t) (& SIUL2.MSCR[0].R);
```

```
// Write a value to the mirror module space at offset 0x2000 - update the  
// SIUL2 and lock it at the same time.
```

```
*(uint32_t *) (register_address + 0x2000) = register_value;
```

7 Default configurations

7.1 SMPU default configuration

The default configuration of the System Memory Protection Unit is such that all 24 configuration registers are reset. Therefore, by default, the System Memory Protection is not active. In case of multiple crossbars, and, therefore, multiple SMPUs, all of the SMPUs are configured in the same way.

The default SMPU configuration is identical for all Chorus devices.

Be aware that by turning on the SMPU, you should define descriptors to cover the entire memory - accesses to memory locations which have no corresponding memory protection register configured will be denied by the SMPU.

7.1.1 Reset behavior

During any type of reset (Destructive reset, long functional reset and short functional reset), the SMPU configuration is reset to default values.

7.2 CMPU default configuration

The default configuration of the Core Memory Protection Unit is such that all 24 configuration registers (6 instruction, 12 data, 6 shared) are reset. In case of multiple cores, all the CMPUs are configured in the same way. From the CMPU point of view, the memory protection is disabled.

The default CMPU configuration is identical for all Chorus devices.

Similarly to the SMPU, be aware that once the CMPU is enabled, memory regions should be defined for all memory locations within the microcontroller - all accesses to addresses not falling into any memory descriptor will be denied.

7.2.1 Cache-Inhibit and guarded control signals

Even though the CMPU is disabled by default, there are several special "Control Signals" for both Cache-Inhibit and Guarded attributes. These Control Signals define independent regions of memory, which are by default Cache-Inhibited or Guarded. There are two Cache-Inhibited and two Guarded regions defined by these Control Signals on each SPC58 device.

These signals are hardwired inside the SoC. They are not configurable and not changeable. They are not identical on all Chorus devices.

The primary purpose of these signals is to provide basic memory protection functionality, without which the MCU would not work properly. For example, it is necessary to configure the memory-mapped peripheral registers (such as CAN, ADC, etc.) as Cache-Inhibited. Intuitively, the contents of these registers are volatile, because their contents can change without intervention from the MCU, based on external inputs, for example. If these memory areas would be configured as cacheable, then the application could very easily end up with wrong data being read as soon as caching is enabled.

Default values of CI/G control signals on Chorus-10M (Cut 2.0 and onwards)

The regions are defined by their base address, and a bit mask. They are usually given in Verilog notation, for those unfamiliar with the notation, a conversion to a normal memory range is given.

Table 11. Default values of CI/R control signals on Chorus-10M

Region	Base Address	Mask	Corresponds to
Cache Inhibit 1	23'h780000	23'h280000	0x5000_0000 – 0x5FFF_FFFF
Cache Inhibit 2	23'h400000	23'h400000	0x8000_0000 – 0xFFFF_FFFF
Guarded 1	12'h800	12'h800	0x8000_0000 – 0xFFFF_FFFF
Guarded 2	12'h800	12'h800	0x8000_0000 – 0xFFFF_FFFF

Default values of CI/G control signals on Chorus-6M

The regions are defined by their base address, and a bit mask. They are usually given in Verilog notation, for those unfamiliar with the notation, a conversion to a normal memory range is given.

Table 12. Default values of CI/R control signals on Chorus-6M, 4M and 2M

Region	Base Address	Mask	Corresponds to
Cache Inhibit 1	23'h280000	23'h280000	0x5000_0000 – 0x5FFF_FFFF 0x7000_0000 – 0x7FFF_FFFF 0xD000_0000 – 0xDFFF_FFFF 0xF000_0000 – 0xFFFF_FFFF
Cache Inhibit 2	23'h780000	23'h780000	0xF000_0000 – 0xFFFF_FFFF
Guarded 1	12'hF00	12'hF00	0xF000_0000 – 0xFFFF_FFFF
Guarded 2	12'hF00	12'hF00	0xF000_0000 – 0xFFFF_FFFF

Default values of CI/G control signals on Chorus-4M

See [Default values of CI/G control signals on Chorus-6M](#).

The default configuration is identical to Chorus-6M.

Default values of CI/G control signals on Chorus-2M

See [Default values of CI/G control signals on Chorus-6M](#).

The default configuration is identical to Chorus-6M.

Default values of CI/G control signals on Chorus-1M

The regions are defined by their base address, and a bit mask. They are usually given in Verilog notation, for those unfamiliar with the notation, a conversion to a normal memory range is given.

Table 13. Default values of CI/R control signals on Chorus-1M

Region	Base Address	Mask	Corresponds to
Cache Inhibit 1	23'h000000	23'h000000	No regions inhibited
Cache Inhibit 2	23'h000000	23'h000000	No regions inhibited
Guarded 1	12'h000	12'h000	No regions guarded
Guarded 2	12'h000	12'h000	No regions guarded

The default configuration does not define any regions to be Cache-Inhibited or Guarded.

7.2.2 Reset behavior

During a system reset, all MPU region descriptor table entries (except those owned by hardware debugger: entry V=1, DEBUG=1, in EDM mode and EDBRAC0MPU=1) are hardware invalidated by clearing the V, DEBUG, IPROT, and RO bits, and the MPU0CSR0 register is cleared. During a power-on reset, all entries are invalidated, regardless of EDM status. Other fields in the entries are undefined following a reset.

7.3 PFAPR default configuration

The default configuration is such that all masters are allowed access to the Flash controller and the memory:

$PFLASH_x.PFAPR_{MXAP} = 0b11$ (Both read and write accesses may be performed by this master)

The default configuration of the Platform Flash access protection is identical on all Chorus devices.

7.3.1 Reset behavior

During any type of reset (Destructive reset, long functional reset and short functional reset), the SMPU configuration is reset to default values.

7.4 MPR/PACR default configuration

7.4.1 MPR default configuration

Each MPR specifies eight 4-bit fields defining the access privilege level associated with a bus master in the device to the various peripherals. The register provides one field per bus master. Each master is assigned depending on its logical master number. See the Device Configuration chapter and the MPR chapter of this document for details about the master assignments to these registers.

The default configuration of the Master Privilege Registers is as follows:

Table 14. MPROTn default configuration

Register	Value
MPROTn	0x77777777 (0b0111_0111_0111_0111_0111_0111_0111_0111)

This means that each Master has full access:

- MTR (Master Trusted for Read) = 1 (Trusted)
- MTW (Master Trusted for Writes) = 1 (Trusted)
- MPL (Master Privilege Level = 1 (Accesses not forced to user mode)

The default configuration of the Master Privilege registers is identical on all Chorus devices. Some devices might have more peripheral bridges than others, in which case the additional bridges are configured in the same way.

7.4.2 (O)PACR default configuration

Each of the on- and off-platform peripherals has a four-bit (O)PACRn field which defines the access levels supported by the given module. Eight (O)PACR fields are grouped together to form a 32-bit (O)PACRx register.

The default configuration of the Master Privilege Registers is as follows:

Table 15. Insert title here

Register	Value
(O)PACRx	0x44444444 (0b0100_0100_0100_0100_0100_0100_0100_0100)

This means that each peripheral on the peripheral bridge is configured in a least-restrictive way, but is configured to require Supervisor mode to access:

- SP (Supervisor Protect) = 1 (Requires Supervisor privilege level for access)
- WP (Write Protect) = 0 (Allows Write Accesses)
- TP (Trusted Protect) = 0 (Allows accesses from untrusted masters)

The default configuration of the Platform Access Control registers is identical on all Chorus devices. Some devices might have more peripheral bridges than others, in which case the additional bridges are configured in the same way.

7.4.3 Reset behavior

During any type of reset (Destructive reset, long functional reset and short functional reset), the PFAPR configuration is reset to default values.

7.5 REGPROT default configuration

The Register Protection feature is by default disabled on all modules offering register protection. All the soft lock bit registers read zero.

7.5.1 Reset behavior

During different types of reset (Destructive reset, long functional reset and short functional reset), if the peripheral is reset, then the REGPROT configuration is also reset to default values and Register Protection functionality is disabled.

For peripherals that are not reset (for example the RGM Reset Generation Module), the REGPROT functionality is also kept.

Appendix A CMPU configuration code example

A.1 Example 1 - Simple process-aware configuration of CMPU

The following example configured the CMPU on a microcontroller with cores 0 and 2.

The configuration is done on Core2. The configuration is as follows:

- Code can be executed only from the Flash memory. All PIDs.
- Read access everywhere. All PIDs.
- Write access to SRAM. All PIDs.
- Write access to Peripherals. All PIDs.
- Write access to own DMEM (Core2). All PIDs.
- Write access to DMEM of second core. Only one specific PID (7) allowed.

Please take note of the assignments of these descriptors into the INST/DATA/SHARED descriptor type sections. The inline assembly parts can be compiler dependent, therefore please modify this example to suit your environment.

Note for Entry 2 of DATA memory: the tightly coupled memories are usually mapped to a memory area corresponding to 0x5y00_0000 - 0x5y80_FFFF, where y corresponds to the Core ID (e.g. Core 2: 0x5200_0000 - 0x5280_FFFF)

This example was tested using Green Hills ccppc compiler version v201754.

Code example

```
#define PIR_SPR          286    /* Processor ID Register */
#define PIR_MASK        0xFF   /* Bitfield for core ID */

#define MPU0CSR0_SPR     1014   /* SPR register MPU0CSR0 */
#define MAS0_SPR        624    /* SPR register MAS0 */
#define MAS1_SPR        625    /* SPR register MAS0 */
#define MAS2_SPR        626    /* SPR register MAS0 */
#define MAS3_SPR        627    /* SPR register MAS0 */

// These are compiler dependent
#define __mpuwe()        asm("mpuwe")
#define __mpure()        asm("mpure")
#define __mpusync()      asm("mpusync")

void CMPU_Init(void)
{
    uint32_t val;
    uint32_t coreId;

    // Invalidation operation and wait for completion - takes 3 cycles to
    complete
    __mtspr(MPU0CSR0_SPR, 0x00000002);
    do
```

```

{
    val = __mfspr(MPU0CSR0_SPR) & 0x00000002;
} while(val != 0);

__mtspr(MAS0_SPR, 0xE5000330); // code only from flash - Entry 0 of INST
memory
__mtspr(MAS1_SPR, 0x00000000);
__mtspr(MAS2_SPR, 0x013BFFFF);
__mtspr(MAS3_SPR, 0x00FC0000);
__mpuwe();
__mtspr(MAS0_SPR, 0xE4000330); // read access everywhere - Entry 0 of
DATA memory
__mtspr(MAS1_SPR, 0x00000000);
__mtspr(MAS2_SPR, 0xFFFFFFFF);
__mtspr(MAS3_SPR, 0x00000000);
__mpuwe();

__mtspr(MAS0_SPR, 0xE4010C30); // write access only to SRAM - Entry 1 of
DATA
__mtspr(MAS1_SPR, 0x00000000); // memory
__mtspr(MAS2_SPR, 0x40107FFF);
__mtspr(MAS3_SPR, 0x400A8000);
__mpuwe();

// Read own Core ID
coreId = (__mfspr(PIR_SPR) & PIR_MASK) & 0x03;

__mtspr(MAS0_SPR, 0xE4020C30); // write access only to DMEM - Entry 2 of
DATA
__mtspr(MAS1_SPR, 0x00000000); // memory
__mtspr(MAS2_SPR, 0x5080FFFF | (coreId << 24));
__mtspr(MAS3_SPR, 0x50000000 | (coreId << 24));
__mpuwe();

__mtspr(MAS0_SPR, 0xE4030C30); // write access only to peripherals -
Entry 3 of
__mtspr(MAS1_SPR, 0x00000000); // DATA memory
__mtspr(MAS2_SPR, 0xFFFFFFFF);
__mtspr(MAS3_SPR, 0xF0000000);
__mpuwe();

if(coreId != 2)
{
    __mtspr(MAS0_SPR, 0xE4800C30); // write access only to DMEM of core 2
- Entry 0
    __mtspr(MAS1_SPR, 0x00070000); // of SHARED memory - PID7 ONLY!

```

```
__mtspr(MAS2_SPR, 0x5280FFFF);  
__mtspr(MAS3_SPR, 0x52000000);  
__mpuwe();  
}  
__mtspr(MPU0CSR0_SPR, 0x00000009); // CMPU ON  
__mpusync();  
}
```

Appendix B SMPU configuration code examples

B.1 Example 1 - Dual core system with four bus masters

The following dual-core system example contains four bus masters: the two processors (Core0, Core1) and two DMA engines (DMA1, a traditional data movement engine transferring data between RAM and peripherals, and DMA0, a secondary engine transferring data to/from the RAM only). Consider the following region descriptor assignments:

Table 16. Configuration of SMPU example 1

Region description	RGDn		Core0	Core1	DMA0	DMA1	System memory map space
Core0 code	0		r-	--	--	--	Flash
Core1 code	1		--	r-	--	--	
Core0 data & stack	2		rw	--	--	--	RAM
Core0->Core1 shared data	2	3	rw	r-	--	--	
Core1->Core0 shared data	4		r-	rw			
Core1 data & stack	4		--	rw	--	--	
Shared DMA data	5		rw	rw	rw	rw	
Peripherals	6		rw	rw	--	rw	Peripheral space

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and peripheral space). Each region indicates the specific permissions for each of the four bus masters, and this definition provides an assigned set of shared, private, and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 & 3 and 3 & 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to Core0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from Core0 to Core1, and the access controls are defined by the logical OR of the two region descriptors. Thus, Core0 has $(rw- | r--)$ = $(rw-)$ permissions, while Core1 has $(--- | r--)$ = $(r--)$ permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from Core1 to Core0. For this overlapping space, Core0 has $(r-- | ---)$ = $(r--)$ permission, while Core1 has $(rw- | r--)$ = $(rw-)$ permission. The non-overlapped space of RGD4 defines a private data and stack area for Core1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the peripheral bus is partitioned into two regions: one containing the SMPU's programming model accessible only to the two processor cores and the remaining peripheral region accessible to both processors and the traditional DMA1 master.

This simple example is intended to show one possible application of the capabilities of the SMPU in a typical system.

The implementation of this example follows on the next page.

This example was tested using Green Hills ccppc compiler version v201754.

Note: This example will work when all SMPUs in the system are configured the same. Otherwise, please consider [Section 3.3.2](#)

Code example

```
void SMPU_configuration(void)
{
    /*
     * *****
     * SMPUn configuration
     * *****
     */

    /* *** List of masters *** */
    /*      0  = Core0      */
    /*      1  = Core1      */
    /*      3  = eDMA0      */
    /*     11  = eDMA1      */
    /* ***** */

    // Memory protection - Core0 code
    SMPU_n.RGD[0].WORD0.B.SRTADDR = 0x00FC0000;    // Flash Controller 0
start
    SMPU_n.RGD[0].WORD1.B.ENDADDR = 0x013BFFFF;    // Flash Controller 0 end
    SMPU_n.RGD[0].WORD2_FMT0.R= 0x80000000;        // Core0 - r; Core1 - 0
    SMPU_n.RGD[0].WORD3.B.VLD = 1;                 // Valid

    // Memory protection - Core1 code
    SMPU_n.RGD[1].WORD0.B.SRTADDR = 0x013C0000;    // Flash Controller 1
start
    SMPU_n.RGD[1].WORD1.B.ENDADDR = 0x015BFFFF;    // Flash Controller 1 end
    SMPU_n.RGD[1].WORD2_FMT0.R= 0x20000000;        // Core0 - 0; Core1 - r
    SMPU_n.RGD[1].WORD3.B.VLD = 1;                 // Valid

    // Memory Protection - Core0 private data and stack
    SMPU_n.RGD[2].WORD0.B.SRTADDR = 0x40060000;    // PRAMC_0 start
    SMPU_n.RGD[2].WORD1.B.ENDADDR = 0x4007FFFF;    // PRAMC_0 end
    SMPU_n.RGD[2].WORD2_FMT0.R= 0xC0000000;        // Core0 - rw; Core1 - 0
    SMPU_n.RGD[2].WORD3.B.VLD = 1;                 // Valid

    // Memory Protection - Data shared between cores
```

```

    SMPU_n.RGD[3].WORD0.B.SRTADDR = 0x4007E000;    // part of PRAMC_0 (last
    8K)

                                                    // writable only by
    Core0 (RGD2)

    SMPU_n.RGD[3].WORD1.B.ENDADDR = 0x40081FFF;    // + part of PRAMC_1
    (first 8K)

                                                    //writable only by
    Core1 (RGD4)

    SMPU_n.RGD[3].WORD2_FMT0.R= 0xA0000000;        // Core0 - r; Core1 - r
    SMPU_n.RGD[3].WORD3.B.VLD = 1;                  // Valid

    // Memory Protection - Core1 private data and stack
    SMPU_n.RGD[4].WORD0.B.SRTADDR = 0x40080000;    // PRAMC_1 start
    SMPU_n.RGD[4].WORD1.B.ENDADDR = 0x400A7FFF;    // PRAMC_1 end
    SMPU_n.RGD[4].WORD2_FMT0.R= 0x30000000;        // Core0 - 0; Core1 - rw
    SMPU_n.RGD[4].WORD3.B.VLD = 1;                  // Valid

    // Memory Protection - Shared DMA data (DMA0+DMA1)
    SMPU_n.RGD[5].WORD0.B.SRTADDR = 0x400A8000;    // PRAMC_2 start
    SMPU_n.RGD[5].WORD1.B.ENDADDR = 0x400E7FFF;    // PRAMC_2 end
    SMPU_n.RGD[5].WORD2_FMT0.R= 0xF3000300;        // Core0 - rw; Core1 - rw;
                                                    // DMA0 - rw; DMA1- rw
    SMPU_n.RGD[5].WORD3.B.VLD = 1;                  // Valid

    // Memory Protection - Peripherals (DMA1 only)
    SMPU_n.RGD[6].WORD0.B.SRTADDR = 0xF0000000;    // PRAMC_2 start
    SMPU_n.RGD[6].WORD1.B.ENDADDR = 0xFFFFFFFF;    // PRAMC_2 end
    SMPU_n.RGD[6].WORD2_FMT0.R= 0xF0000300;        // Core0 - rw; Core1 - rw;
                                                    // DMA0 - rw; DMA1- rw
    SMPU_n.RGD[6].WORD3.B.VLD = 1;                  // Valid
}

void SMPU_enable(void)
{
    SMPU_1.CESR0.B.GVLD = 1;    // Enable SMPU module and also all the
    descriptors
}

```

Appendix C PFlashC PFAPR access control example

C.1 Example 1 - Access control in a system w/ multiple flash controllers

In this simple example, we are considering a system with two flash controllers and two or more cores.

One core is allowed access into one of the Flash controllers, and the other core is allowed access to the other Flash controller. The other masters (such as DMA) are left disabled for both Flash controllers. Ultimately it is up to the user to decide which peripheral is used in their application, and which peripheral should be allowed to access which Flash controller.

For a complete list of all the PFCRx and PFAPR register bitfields, please refer to the reference manual.

This example was tested using Green Hills ccppc compiler version v201754.

Code example

```
void Flash_Config (uint8_t Waitstates)
{
    uint8_t Waitstates = 6;

    PFLASH_0.PFCR1.R = (0xFFFF0055 | (Waitstates << 8)); // Prefetching
    enabled
    PFLASH_0.PFCR2.R = 0xFFFF0055; // on both ports (0 and 1)
    PFLASH_0.PFCR3.R = 0x00000000; // Fixed prio. arbitration and cache
    way config
    PFLASH_0.PFAPR.R = 0xC0000000; // Access only from Core 0

    PFLASH_1.PFCR1.R = (0xFFFF0055 | (Waitstates << 8)); // Prefetching
    enabled
    PFLASH_1.PFCR2.R = 0xFFFF0055; // on both ports
    (0 and 1)
    PFLASH_1.PFCR3.R = 0x00000000; // Fixed prio. arbitration and cache
    way config
    PFLASH_1.PFAPR.R = 0x30000000; // Access only from Core 1
}
```

Appendix D PBRIDGE access control example

D.1 Example 1 - Simple PBRIDGE access control example

In this simple example, one of the PBRIDGE controllers is configured. Each Chorus device has varying amount of Peripheral Bridges, with various peripherals attached to it. For this example, the PBRIDGE_2 of the Chorus-6M (SPC58xGxx) is considered.

Let's imagine that for any reason that we wish to:

- Restrict access to HSM (Hardware Security Module) Host Interface and to SWT_2 and SWT_3 (Software Watchdog Timer) only from Trusted Masters.
- We consider Core0 to be the only trusted master on the PBRIDGE_2, on account of it being the only core with lock-step partner.
- All other peripherals shall be accessible by untrusted masters.

This example was tested using Green Hills ccppc compiler version v201754.

Code example

```
void PBRIDGE_Init(void)
{
    // PBRIDGE0
    PBRIDGE_0.MPRA.R = 0x71111111; // Only Core0 considered trusted, others
    // untrusted, but allowed supervisor access
    PBRIDGE_0.MPRB.R = 0x11111111; // 1 = Allow Supervisor Access
    // 2 = Trusted for Writes
    // 4 = Trusted for Reads
    // and combinations (7 = full trust)

    // Default configuration for PACRs is such that peripherals do not require
    trusted
    // access, but require supervisor privilege level.
    // HSM Host I/F = Off-Platform Nr. 51
    // SWT_2          = On-Platform Nr. 22
    // SWT_3          = On-Platform Nr. 23

    PBRIDGE_2.PACRC.B.PACR22 = 0x5; // SWT_2 - Supervisor Access only, Trusted
    only
    PBRIDGE_2.PACRC.B.PACR23 = 0x5; // SWT_3 - Supervisor Access only, Trusted
    only

    PBRIDGE_2.OPACRG.B.OPACR51 = 0x5; // HSM Host I/F - Supervisor Access
    only,
    // Trusted only
}
```


Appendix E REGPROT example

E.1 Example 1 - Configuration of register protection for SIUL2 registers

In this example, a register protection mechanism is applied to a register of SIUL2.

A single SIUL2 pad is configured in the usual way, before being configured for register protection.

This example was tested using Green Hills ccppc compiler version v201754.

Code example

```
void SIUL2_init_and_protect (uint32_t pinNr)
{
    base_address = (uint32_t) (& SIUL2);
    base_offset = ((uint32_t) (& SIUL2.MSCR[pinNr].R)) - base_address;

    slb_address = base_address + 0x3800 + (base_offset / 4);

    // Configure pin
    SIUL2.MSCR_IO[pinNr].R = MEDIUM_DRIVE | PUSH_PULL | MSCR_SMC;

    // Protect Register - by writing to the Soft Lock Registers.
    // 0xFF - All Write Enable Bits and Soft Lock Bits are set.
    (*(uint8_t *) slb_address) = 0xFF;

    // Try to change the now protected register - an error is generated
    SIUL2.MSCR_IO[pinNr].R = 0;
}
```

Appendix F Document management

Table 17. Reference documents

Document Name	Document Number	Version
SPC58xEx/SPC58xGx Reference manual	RM0391	4.0

Revision history

Table 18. Document revision history

Date	Revision	Changes
27-May-2019	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved