

Developing a Camera Application with i.MX RT Series

Contents

1. Introduction

This application note describes how to develop an HD camera application with the NXP i.MX RT1050 processor.

For this demo application, the RT1050 receives video frames from a camera device with resolution of HD-720 (1280x720 pixels, also called 720P), displays on an LCD panel, and optionally saves to a microSD card with compressed JPEG format.

The i.MX RT1050 is a processor with single ARM Cortex-M7 core, which operates at speed up to 600 MHz. The great processing capability, real-time feature, and reach integration of abundant peripherals make i.MX RT1050 ideal for lot of high-performance applications, such as industrial computing, motor control, power conversion, smart consumer products, high-end audio systems, home and building automation.

[Section 2](#) of this document introduces the hardware and software platforms of the demo application. [Section 3](#) describes the procedure to develop the HD camera application using i.MX RT1050, based on [MCUXpresso Software Development Kit \(SDK\)](#).

1.	Introduction.....	1
2.	Hardware and software platforms	2
2.1.	i.MX RT1050 processor.....	2
2.2.	i.MX RT1050 EVK board.....	2
2.3.	MT9M114 image sensor	3
2.4.	RK043FN02H-CT TFT LCD Panel.....	3
2.5.	MCUXpresso SDK	4
3.	Develop an HD camera application	4
3.1.	System structure analysis	4
3.2.	Create the demo project	5
3.3.	Video capture.....	8
3.4.	Display frames on LCD	10
3.5.	Store frames to SD card.....	12
3.6.	The software structure.....	12
3.7.	Run the demo application	13
4.	Conclusion	13
5.	References.....	13
6.	Revision history	13

2. Hardware and software platforms

This section presents short introductions of the hardware and software platforms of the demo application, including the below items.

- [i.MX RT 1050 processor](#)
- [i.MX RT1050 EVK board](#)
- [MT9M114 image sensor](#)
- [RK043FN02H-CT TFT LCD Panel](#)
- [MCUXpresso SDK](#)

2.1. i.MX RT1050 processor

The [i.MX RT1050](#) offered by NXP with single ARM Cortex-M7 core can operate at speed up to 600 MHz. It has 512 KB on-chip RAM, which can be flexibly configured as core Tightly-Coupled Memory (TCM) or general-purpose RAM. It provides various interfaces for connecting various external memories, and a wide range of serial communication interfaces, such as USB, Ethernet, SDIO, CAN, UART, I2C, and SPI. It also has rich audio and video features, including LCD display, basic 2D graphics, camera interface, SPDIF and I2S audio interface. Other notable features including various modules for security, motor control, analog signal processing, and power management.

The CMOS Sensor Interface (CSI) module enables the chip to connect directly to external CMOS image sensors. CSI supports for CCIR656 video interface as well as traditional sensor interface, with 8/10/16-bit width data port. The embedded DMA controller has the capability to transfer data from receive FIFO or statistic FIFO through AHB bus to memory buffers.

The enhanced Liquid Crystal Display Interface (eLCDIF) is a RGB interface display controller, which supports 8/16/18/24-bit width data port and up to 800x480 WXGA resolution. To transfer frame data for display refresh, the eLCDIF can acts as a bus master, or a bus slave working in coordination with the SoC integrated DMA engine. CPU is liberated from handling the frame data in both conditions.

The Pixel Pipeline (PXP) module integrates several 2D graphics processing functions, including scaling, color space conversion (CSC), and rotation.

2.2. i.MX RT1050 EVK board

The [i.MX RT1050 EVK](#) board is a platform designed to showcase the most commonly used features of the i.MX RT1050 processor. The EVK board offers the below features:

- Memory: 256Mbit SDRAM, 64Mbit Quad SPI Flash, 512Mbit Hyper Flash, TF Card Slot;
- Communication interfaces: USB 2.0 OTG connector, USB 2.0 host connector, 10/100 Mbit/s Ethernet connector, CAN bus connector;
- Multimedia interfaces: CMOS sensor connector, LCD connector;
- Audio interfaces: 3.5 mm stereo headphone jack, board-mounted microphone, SPDIF connector (not mounted by default);

- Debug interfaces: On-board debug adapter with DAP-Link, JTAG 20-pin connector;
- Arduino interface;
- User button and LEDs.

Figure 1 presents the picture of i.MX RT1050 EVK.

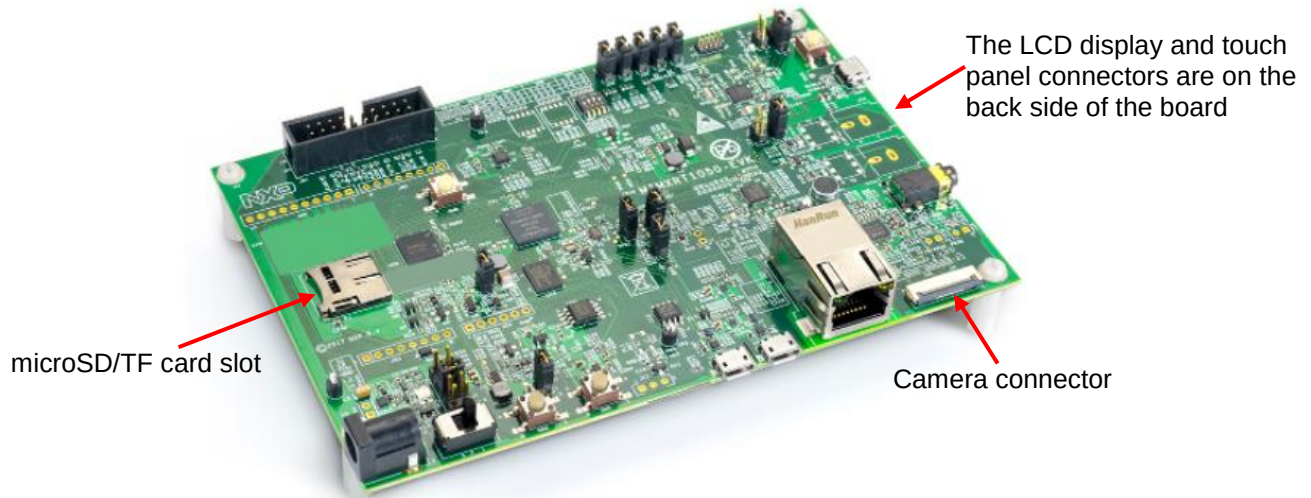


Figure 1. i.MX RT1050 EVK

2.3. MT9M114 image sensor

The MT9M114 from ON Semiconductor (formerly Aptina Imaging Corporation) is a 1/6-inch 1.26 Mp CMOS digital image sensor with an active-pixel array of 1296x976. It includes sophisticated camera functions such as auto exposure control, auto white balance, black level control, flicker avoidance, and defect correction.

This demo application uses a MT9M114 camera module. The module exposes its signals via a 24-pin FPC wire which is compatible with the camera connector on the i.MX RT1050 EVK board.

2.4. RK043FN02H-CT TFT LCD Panel

The characteristics of the RK043FN02H-CT TFT LCD panel are listed below:

- 4.3-inch physical size;
- 480*272 pixels (RGB888);
- 24-bit RGB888 interface, supporting DE or HV mode;
- LED backlight;
- I2C interface capacitive touch;

The LCD panel exposes its signals via a 40-pin FPC wire for the display signals and a 6-pin FPC wire for the touch signals, which are compatible with the connectors on the i.MX RT1050 EVK board

The i.MX RT1050 EVK does not make use of all the 24-bit display data signals but part of them, supporting RGB565 at most.

2.5. MCUXpresso SDK

The MCUXpresso SDK provides comprehensive software support for multiple microcontroller families from NXP. The SDK comprises the below components:

- A flexible set of peripheral drivers.
- A rich set of example applications.
- Various middleware from NXP or incorporated from a third party, such as FreeRTOS, emWin, FatFs, LIBJPEG, LwIP, mbed TLS, USB stack, wolfSSL, and so on.
- The SOC header file, startup files, and linker configuration files for various tool chains.

3. Develop an HD camera application

This section describes the procedure to develop the HD camera application based on the hardware and software platforms presented in [Section 2](#).

3.1. System structure analysis

Firstly, we analyze the system requirements, then figure out the system structure, and draw the system block diagram. So that we can develop each of the software component, and then build the whole system.

[Figure 2](#) presents the hardware block diagram of this demo application, which shows the primary components of the system.

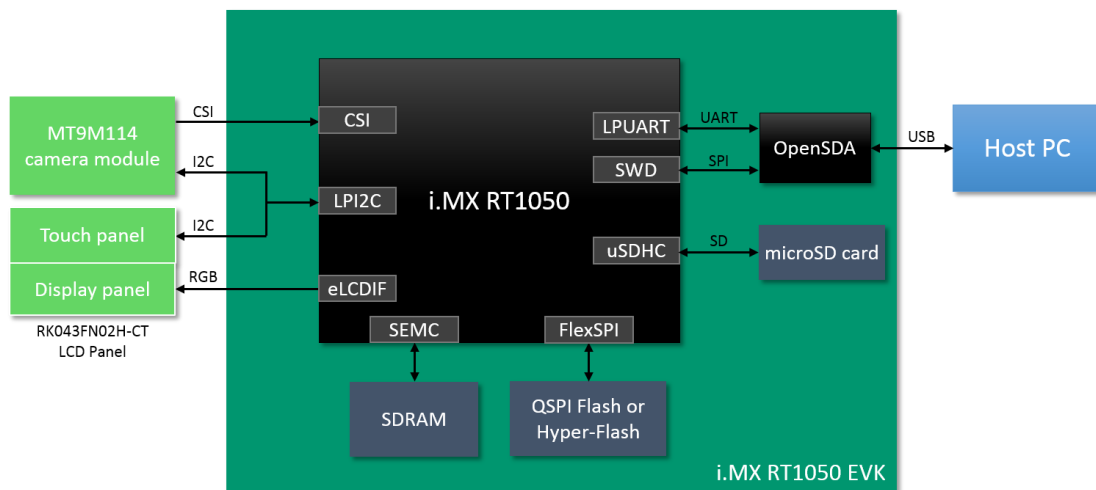


Figure 2. System hardware block diagram

- RT1050 configures MT9M114 via an I2C bus, and captures the video frame data via the CSI interface.
- RT1050 transfers frame data to LCD panel via RGB interface, and receives touch events via I2C bus.
- A microSD card connecting with RT1050 uSDHC module is used to store JPEG files of compressed video frames.

- SDRAM provides data space for frame buffer and/or code space. RT1050 connects with the SDRAM device via the Smart External Memory Controller (SEMC) module.
- QSPI flash or hyper-flash provides code space for non-debugging running configuration. The i.MX RT1050 connects both with via the FlexSPI controller.
- The Open-Standard Serial Debug Adapter (*OpenSDA*) provides SWD debug access, debug UART bridge, and power supply for the board. *OpenSDA* communicates with the host PC via a USB port.

Figure 3 shows the frame data flow diagram of this demo application.

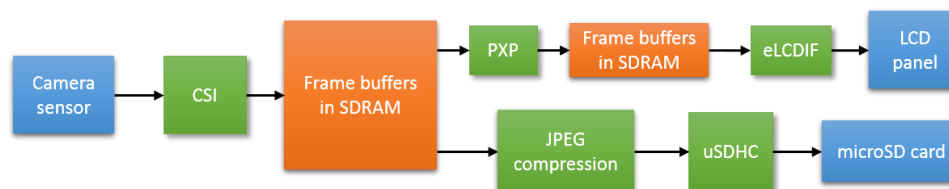


Figure 3. Frame data flow diagram

CSI receives frame data and stores in the frame buffers located in SDRAM. On one hand, PXP is introduced to resize the frames from original size 1280x720 to the LCD resolution size 480x272, then eLCDIF transfers the resized frames to the LCD display panel. On the other hand, frames can be optionally compressed to JPEG format and stored in the microSD card.

3.2. Create the demo project

As we described in [Section 2](#), the MCUXpresso SDK provides rich set of peripheral drivers, middleware, examples applications, and other necessary components for an embedded application. It is recommended to develop applications based on these SDK components, in order to speed up and simplify the development process.

You can create your application project by porting an SDK example project, then delete, add, and customize the components; or create an empty project, then add and customize the components. Choose your own favorite way. In this demo, we start with creating an empty project for the application, using the IAR Embedded Workbench IDE, version 8.20.

The below subsections describe the steps of creating the application project based on MCUXpresso SDK. For detailed information, please refer to the software package along with this application note, and the IDE help documents.

3.2.1. Download MCUXpresso SDK

Download the SDK package from <https://mcuxpresso.nxp.com>. Login is required. Register an NXP account if you do not have one. Download the latest version of the SDK package SDK_2.3.0_EVK-MIMXRT1050.zip for the EVK board, and extract the archive file to your local disk.

3.2.2. Create an empty IAR project

Follow the below steps to create an empty IAR project.

1. Create new directories *user_apps\camera_demo\iar* in the *boards\evkmimxrt1050* directory for the project.
2. Launch the IAR Embedded Workbench, and create a new project “*camera_demo.ewp*” in the created “*iar*” directory. And save the workspace as “*camera_demo.eww*”.
3. As SDRAM will be used to save data and code in the project, so we create a configuration target named “*sdram_txt*”. And remove the default “Debug” and “Release” configurations.

This documents mainly describes the SDRAM configuration which is used for debugging session. Besides this configuration, the software package offers a FlexSPI NOR flash configuration which is used for offline running purpose. The only difference between the two configurations lies in the memory space allocation.

3.2.3. Import the needed source files

Follow the below steps to import the needed source files to the project.

1. Import the used driver files from the “*devices\MIMXRT1052\drivers*” directory to the project, including drivers for cache, clock, CSI, eLCDIF, GPIO, IOMUX, LPI2C, LPUART, PXP, and uSDHC.
2. Import startup files from the “*devices\MIMXRT1052*” and “*devices\MIMXRT1052\iar*” directories.
3. Import debug UART files from the “*devices\MIMXRT1052\utilities*” directory and its subdirectories.
4. Import FatFs source files from the “*middleware\fatfs\src*” directory.
5. Import SD/MMC Card management files from the “*middleware\sdmmc*” directory.
6. Import LIBJPEG source files from the “*middleware\libjpeg*” directory.
7. From an example directory, such as “*sdcard_fatfs*” copy board specific files and to the project directory; and import the files to the project.

Besides these existing files from SDK, there are some project specific source files need to create for this demo project, for the sake of configuring various modules, capturing and displaying frames, processing the touch events, and so on. The rest of this document does not detail those project specific files, but illustrate the methods and principles of the implementation. For more details, please refer to the software package.

3.2.4. Project configuration

Open the project option dialog in the IDE, and configure the project.

1. Specify the target device as “*NXP MIMXRT1052xxx6A*”.
2. Add the various include paths, including header file directories of CMSIS, peripheral drivers, startup files, debug UART utility, FatFs, LIBJPEG, SDMMC, and the project specific header files.
3. Add necessary project symbol definitions for SDK driver configurations.

4. Specify “CISIS DAP” as the debugger driver, and set SWD as the interface.
5. Copy the debugger setup macro file and script files from an example project to the project directory, which is mainly used for SDRAM initialization. Then import the files to the project.

3.2.5. Memory space allocation

Copy the linker configuration files “MIMXRT1052xxxxx_sdram_txt.icf” and “MIMXRT1052xxxxx_flexspi_nor.icf” from the “devices\ MIMXRT1052\iar” directory to the project directory. Edit this file to specify the memory spaces (RAM, SDRAM, FlexSPI flash) for text, data, stack, heap, and no-cache regions based on the application requirements.

For this demo application, we allocate the memory space with the schemes as shown in [Figure 4](#).

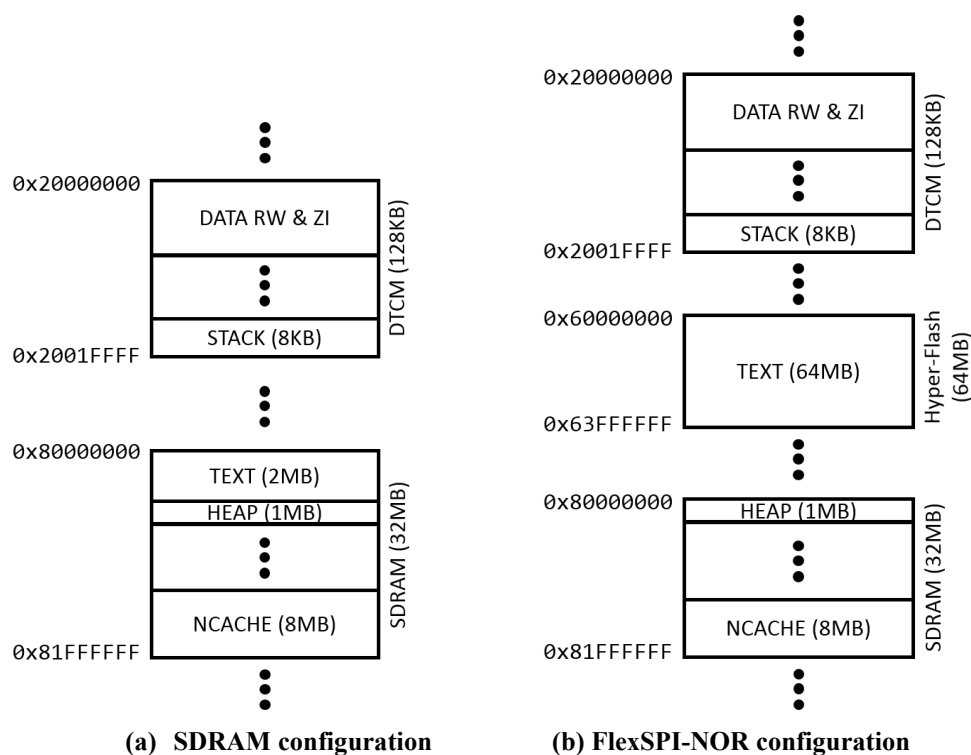


Figure 4. Memory allocation schemes

At last, open the project option dialog, and import the customized linker configuration files to the project for the configurations.

3.2.6. Pin mux configuration

Most pins of i.MX RT1050 are multiplexed by multiple peripheral modules. Before a multiplexed pin is used, the pin’s multiplex must be configured.

For this demo application, we need to configure the pin multiplexes of the debug UART, SEMC, CSI, LPI2C, eLCDIF, and SDHC pins.

3.3. Video capture

For this demo application, the MT9M114 camera device is configured to output 720P video frames with RGB565 pixel format, at 30fps frame rate. Both the camera device and the CSI module need to configure in advance.

3.3.1. The MT9M114 drivers

The MT9M114 drivers offer APIs and parameters for configuring the camera device. Note that MT9M114 has a large number of registers (and the so-called variables) to configure before it works. It is difficult for users to determine the various parameters. Thus, a configuration tool *Register Wizard* is offered by the device vendor, which makes it easy to get the basic parameters for a specific set of video characteristics. Figure 5 shows the *Register Wizard* tool interface.

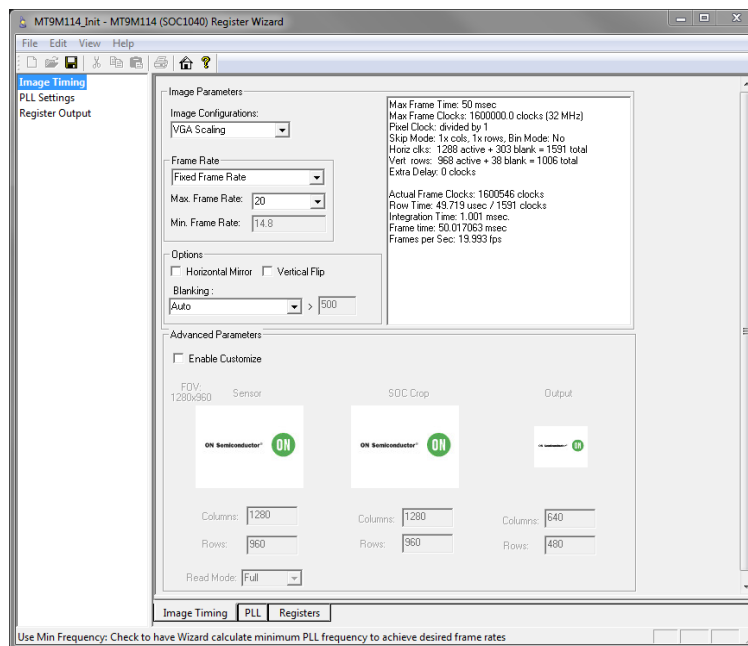


Figure 5. The Register-Wizard tool

To archive or update a configuration, the below steps should be completed:

1. Prepare the parameters for those required registers and variables to be written.
2. Write the registers and variables via the I2C interface.
3. Issue a change-config process by writing a series of commands under prescriptive timing.

3.3.2. Video capture via CSI

CSI module needs to be initialized for a specific video format. The CSI driver refers various video characteristics to initialize CSI, including pixel data format, number of bytes per pixel, frame resolution, frame rate, and so on.

The CSI embedded DMA can write two frame buffer addresses alternately without stop. To improve the performance, especially for a complicated application, the SDK CSI driver extends the supported buffers to up to four, and organizes the frame buffers in a ring queue structure. CSI fills the buffers one by one, so that the data sink, such as LCD interface, can drain the buffers at the same time. Figure 6 illustrates the CSI frame buffer filling scheme.

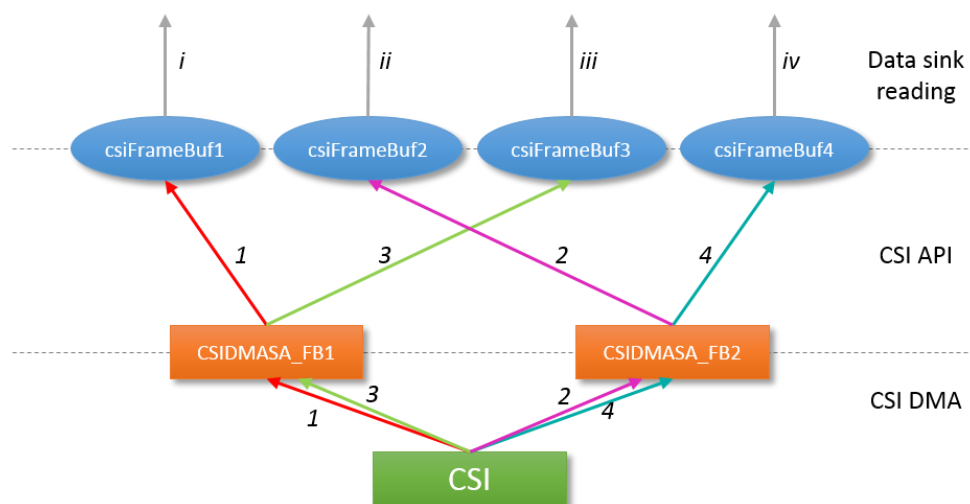


Figure 6. CSI Frame buffer filling scheme

The CSI DMA makes use of two registers *CSIDMASA_FB1* and *CSIDMASA_FB2*, which save the buffer addresses, to access the four frame buffers. The process is:

- a. Set *CSIDMASA_FB1* pointing to frame buffer 1, *CSIDMASA_FB2* pointing to frame buffer 2.
- b. CSI fills frame buffer 1 via *CSIDMASA_FB1* (the data stream 1 in Figure 6).
- c. After frame buffer 1 is full, an interrupt is generated. Set *CSIDMASA_FB1* pointing to frame buffer 3.
- d. CSI starts to fill frame buffer 2 via *CSIDMASA_FB2* (the data stream 2 in Figure 6). At the same time, data sink can drain frame buffer 1 (the data stream *i* in Figure 6).
- e. After frame buffer 2 is full, an interrupt is generated. Set *CSIDMASA_FB2* pointing to frame buffer 4.
- f. CSI starts to fill frame buffer 3 via *CSIDMASA_FB1* (the data stream 3 in Figure 6). At the same time, data sink can drain frame buffer 2 (the data stream *ii* in Figure 6).
- g. After frame buffer 3 is full, an interrupt is generated. Set *CSIDMASA_FB1* pointing to frame buffer 1.
- h. CSI starts to fill frame buffer 4 via *CSIDMASA_FB2* (the data stream 4 in Figure 6). At the same time, data sink can drain buffer 3 (the data stream *iii* in Figure 6).
- i. After frame buffer 4 is full, an interrupt is generated. Set *CSIDMASA_FB2* pointing to frame buffer 2.
- j. Repeats from step 2 again. At the same time, data sink can drain buffer 4 (the data stream *iv* in Figure 6).

The filling and draining operations on the same set of frame buffers can be simultaneous and independent which makes the system performance to be maximized.

3.4. Display frames on LCD

To drain the CSI frame buffers, two buffer pointers *csiFrameBufPtr1* and *csiFrameBufPtr2* are introduced. In conjunction with the two SDK driver functions *CSI_TransferSubmitEmptyBuffer()* and *CSI_TransferGetFullBuffer()*, the top layer application can drain the CSI frame buffers continuously. Figure 7 illustrates the CSI frame buffer draining scheme.

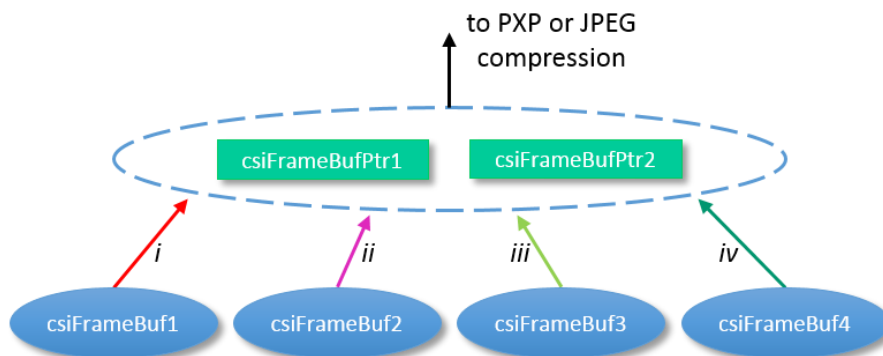


Figure 7. CSI Frame buffer draining scheme

One point to be noted is that the two buffer pointers are designed to point the four frame buffers alternately. The purpose of this policy is to allow the filling and draining of the frames buffers to be continuous and simultaneous.

Due to the LCD's display resolution is different from the camera's output resolution, additional LCD buffers are required and the PXP module is responsible for the inter-buffers scaling.

Figure 8 illustrates the LCD frame buffer accessing scheme.

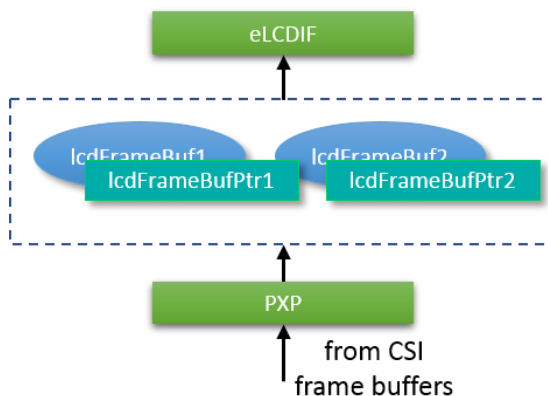


Figure 8. LCD Frame buffers accessing scheme

Two LCD frame buffers and two buffer pointers are introduced. PXP fills the two buffers via the two pointers, and eLCDIF drains the two buffer via the two pointers too. Similarly, the use of two pointers is also to make the filling and draining of the buffers to be continuous and simultaneous.

The frame buffers operation flow is show in [Figure 9](#), including the operations on CSI buffers and LCD buffers. Here we can see how the buffer pointers are operated to realize the filling and draining operations to be continues, simultaneous, and independent.

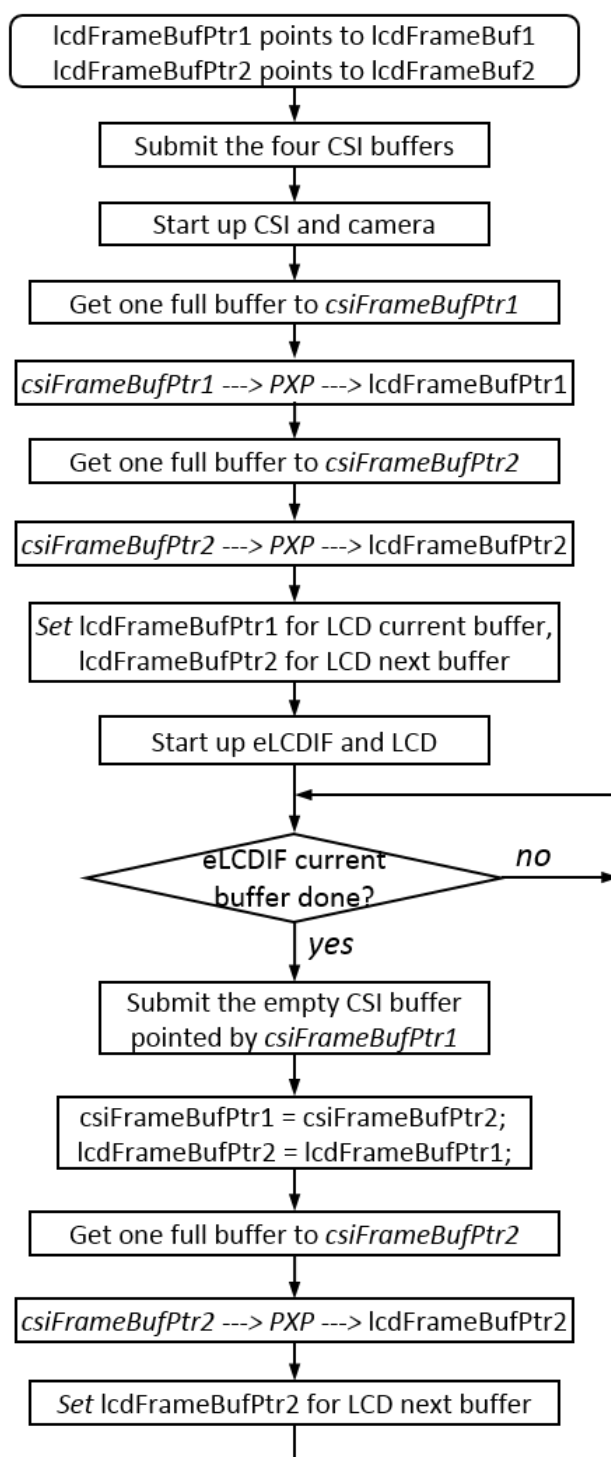


Figure 9. CSI/LCD frame buffers operation flow

3.5. JPEG compression and stored in SD card

For this demo application, when the user touches the LCD panel the current video frame would be compressed to JPEG format and stored in the microSD card.

The method of getting CSI frames for JPEG compression is similar with getting frames for LCD display. Actually, the code used to get frames for compression and display are mixed together. When one CSI frame is received, and handled by *csiFrameBufPtr2* as shown in [Figure 9](#), the touch events is polled. JPEG compression would initiate if the touch-down event is detected.

The open source LIBJPEG and FatFs are incorporated in SDK. This demo application makes use of these two components to compress video frames to JPEG files and store in a microSD card.

3.6. The software structure

[Figure 10](#) presents the overall software structure of this demo application, which shows the program and data flow.

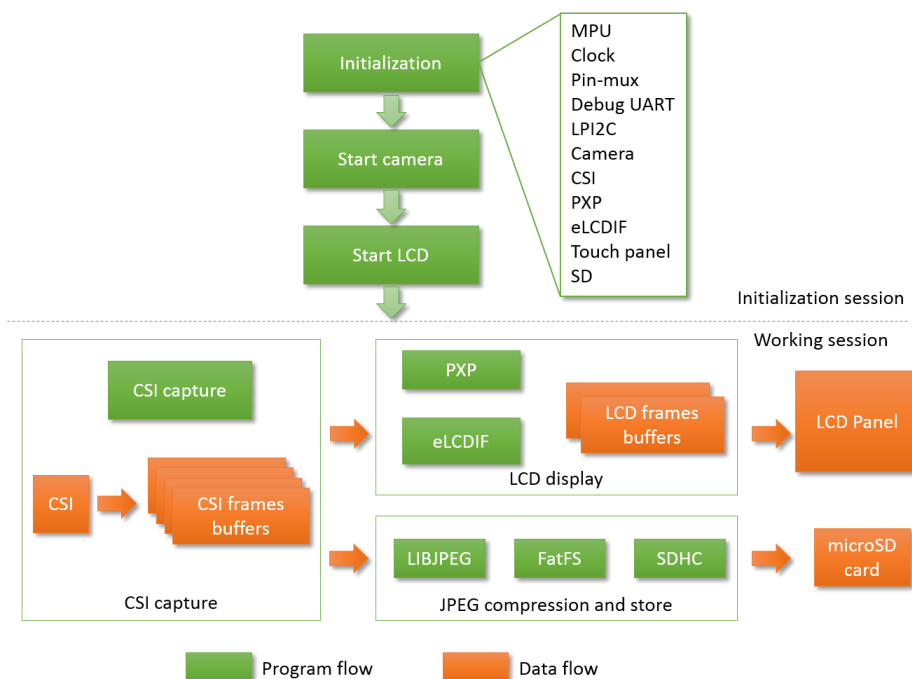


Figure 10. Overall program and data flow

During the initialization session, various components are initialized, like system clock, pin mux, camera device, CSI, eLCDIF, touch panel, and SD card. Then the camera and LCD start to run.

In the endless working session, CSI captures frames with which CSI fills the frames buffers. On the one hand, the CSI frames are scaled to LCD frame size by PXP and transferred to the LCD panel by eLCDIF. On the other hand, the CSI frames can be optionally compressed to JPEG format and stored into a microSD card.

3.7. Run the demo application

The software package along with this document offers the whole source and project files of the demo application. To run the demo:

- Connect a micro USB cable between the host PC and the *OpenSDA* USB port J28 on the EVK-MIMXRT1050 board.
- Optionally, open a serial terminal tool with settings of 115200 baud rate, 8 data bits, no parity bits, and 1 stop bit to display debug logs.
- Open the demo project, and build one of the configurations: "sdram_txt" or "flexspi_nor".
- Start the debug session or download the binary to the processor.
- Launch the debugger in the IDE or press the reset button SW4 to begin running the demo.
- Then the video frames can be seen on the LCD panel. You can touch the LCD panel with your finger, and one frame will be compressed to JPEG format and stored in the microSD card.

4. Conclusion

This application note describes the steps of developing a camera application using the i.MX RT1050 processor based on the MCUXpresso SDK, from creating an empty project to completing the application. The peripheral drivers and the various middleware offered by the SDK make it easy for the whole development process.

Along with this application note, the source code of the demo application is provided, based on which you can develop your own customized camera applications.

5. References

Following documents may offer further reference.

- [i.MX RT1050 Processor Reference Manual, Rev. 0](#)
- [MT9M114 Data Sheet, Rev. 12](#)
- [MT9M114 Register and Variable Reference, Rev. E](#)

6. Revision history

Revision number	Date	Substantive changes
0	12/2017	Initial release

How to Reach Us:

Home Page:

nxp.com

Web Support:

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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners.

Arm, the Arm logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number: AN12110

Rev. 0

12/2017

