AN12562

Development of H.264 Video Decode on RT Series

Rev. 0 — 28 August, 2019

Application Note

Contents

1 Introduction

This application note describes how to develop an H.264 video decoding application with NXP i.MX RT1050 processor.

For such applications, the i.MX RT1050 receives H.264 video source from the microSD card then invokes the FFMPEG Library to decode the video source and generates YUV data. After scaling and color space conversion of YUV data by pixel processing pipeline(PXP), displays on an LCD panel.

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

Section 2 introduces the hardware and software platforms of the demo application. Section 3 describes the procedure to develop the H.264 video decoding application using i.MX RT1050, based on FFMPEG Library and i.MX RT1050 Software Development Kit (SDK).

2 Hardware and software platforms

This section presents short introductions of the hardware and software platforms of the demo application.

2.1 i.MX RT1050 processor

The NXP i.MX RT1050 cross-over processor has a single Arm Cortex-M7 core at 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 include various modules for security, motor control, analog signal processing, and power management.

The enhanced Liquid Crystal Display Interface (eLCDIF) is an RGB interface display controller, which supports 8/16/18/24-bit width data port and up to 1366x768 resolution. To transfer frame data for display refresh, the eLCDIF acts as a bus master, or a bus slave working in coordination with the SoC integrated DMA engine. CPU is offloaded 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: 256 Mbit SDRAM, 64 Mbit Quad SPI Flash, 512 Mbit 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 hacks, board-mounted microphone, S/PDIF 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 the i.MX RT1050 EVK.

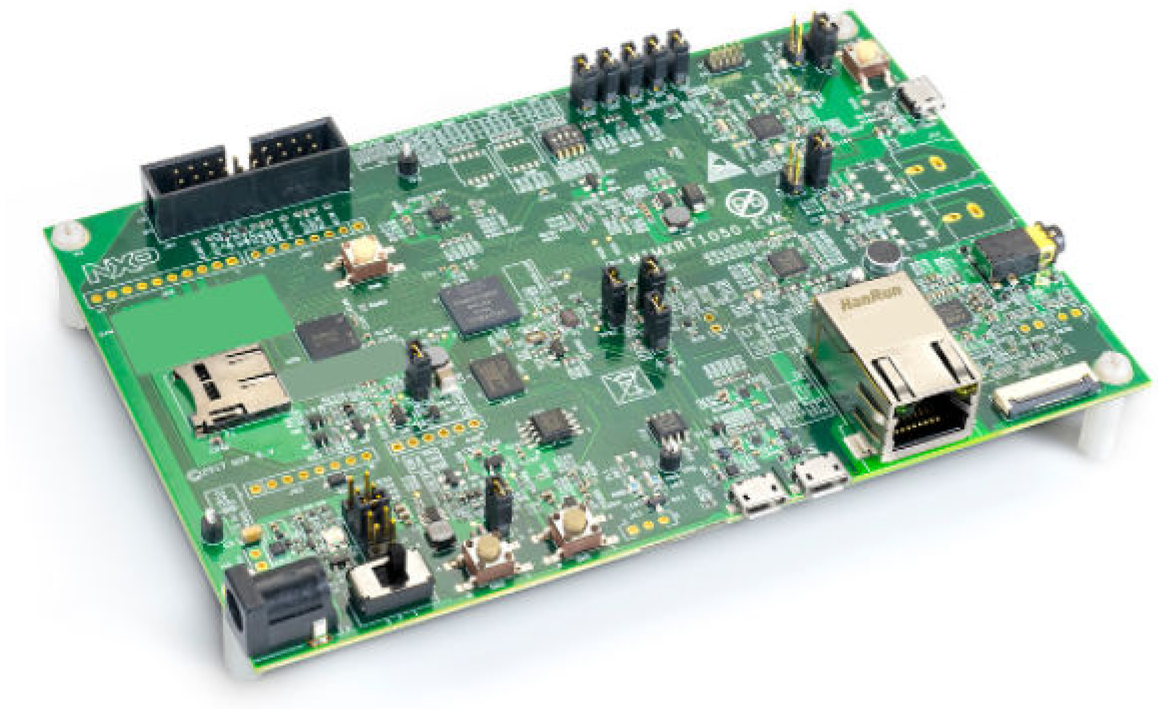


Figure 1. i.MX RT1050 EVK

2.3 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 use all the 24-bit display data signals but part of them, supporting RGB565 at most.

2.4 Pixel Pipeline (PXP)

The pixel processing pipeline is used to perform image processing on image/video buffers before sending to an LCD display. It consists of several pipelined blocks that perform the video source frame scaling, color space conversion, alpha-blending/color key algorithm, secondary CSC, pixel correction.

This application uses a PXP module. PXP is introduced to resize the frames from original size to the LCD resolution size 480x272 and color space conversion from YUV to RGB, then eLCDIF transfers the resized frames to the LCD display panel.

2.5 SDK for i.MX RT1050 EVK board

The 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, start up files, and linker configuration files for various tool chains.

2.6 FFMPEG library

FFMPEG is multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created. It supports the most obscure ancient formats up to the cutting edge. It is also highly portable: FFMPEG compiles, runs, and passes testing infrastructure FATE across Linux, Mac OS X, Microsoft Windows, the BSDs, Solaris, etc. under a wide variety of build environments, machine architectures, and configurations.

It contains libavcodec, libavutil, libavformat, libavfilter, libavdevice, libswscale and libswresample which can be used by applications.

- **Libavcodec:** is a library containing decoders and encoders for audio/video codecs.
- **Libavutil:** is a library containing functions for simplifying programming, including random number generators, data structures, mathematics routines, core multimedia utilities, and much more.
- **Libavformat:** is a library containing demuxers and muxers for multimedia container formats.
- **Libavdevice: i**s a library containing input and output devices for grabbing from and rendering to many common multimedia input/output software frameworks, including Video4Linux, Video4Linux2, VfW, and ALSA.
- **libavfilter :**is a library containing media filters.
- **libswscale :**is a library performing highly optimized image scaling and color space/pixel format conversion operations.

Libswresample: is a library performing highly optimized audio resampling, rematrixing and sample format conversion operations

3 Develop H.264 video decoding application

This section describes the procedure to develop H.264 video decoding application based on the hardware and software platforms presented in Section 2.

3.1 System structure analysis

Figure 2 presents the hardware block diagram of this demo application, which shows the primary components of the system.

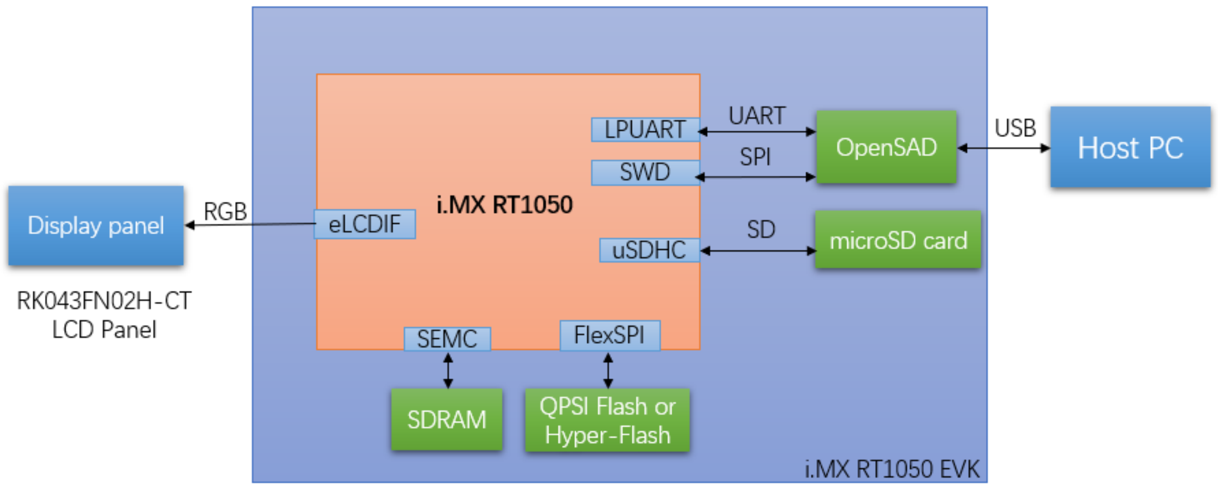


Figure 2. System hardware block diagram

- RT1050 read H.264 video source which save in microSD by uSDHC module
- External SDRAM devices provide data space for frame buffer and/or code space. RT1050 accesses SDRAM devices by the Smart External Memory Controller (SEMC) module.
- External QSPI flash or hyper-flash provides code space for non-debugging running configuration with XIP capability. The i.MX RT1050 accesses flash devices by 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，and implements the “CMSIS-DA” debug protocol
- RT1050 transfers frame data to LCD panel via RGB interface.

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

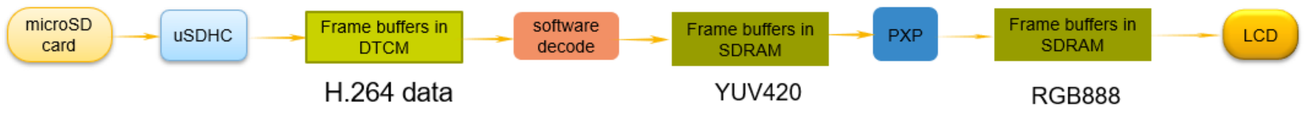


Figure 3. Frame data flow diagram

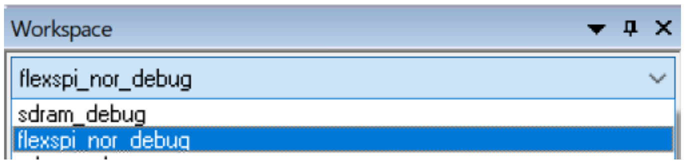
i.MX RT1050 EVK reads video source from microSD card and stores in the frame buffers located in DTCM. Software decodes video data with (a tailored version of) FFMPEG . PXP resizes the frames from original resolution to the LCD resolution and converts color space from YUV format to RGB format, then eLCDIF transfers the frames to the LCD display panel.

3.2 Build the demo project and run

The code package with this AN is self-contained, and you can build the project quite straightforward as below:

3.2.1 Build and run from Flash

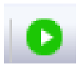
1. Use IAR to open “<h264decode.eww>”, use the default project configuration, which builds the project for flash XIP:



flexspi_nor_debug
sdram_debug
flexspi_nor_debug

2. Press “F7” to build, it may take a minute

3. To enter debug session, press “Ctrl-D” or click



4. After IAR enters debug session and stop at main() function, press “F5” to run.

Since the code is programmed to Flash, you can later reset the board to let it run again.

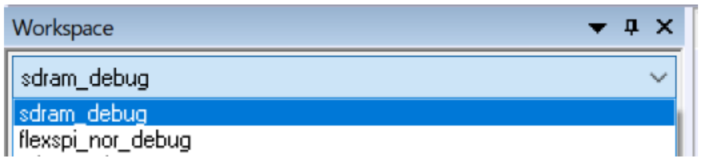
3.2.2 Build and run from SDRAM

If you want to download the program to sdram, continue to the following operation.

NOTE

In SDRAM configuration, we show how to leverage ITCM & DTCM to further improve performance. We reconfigured ITCM to 384 kB and DTCM to 128 kB, which is not included in Flash configuration. It is because if ITCM & DTCM size is reconfigured, most SDK examples which rely on the default ITCM (128 kB), DTCM(128 kB), and OCRAM(256 kB) size could malfunction. Since code in Flash is persistent among power cycles, it may prevent you from running other examples, so we did not enable ITCM/DTCM for flash configuration.

1. select build mode

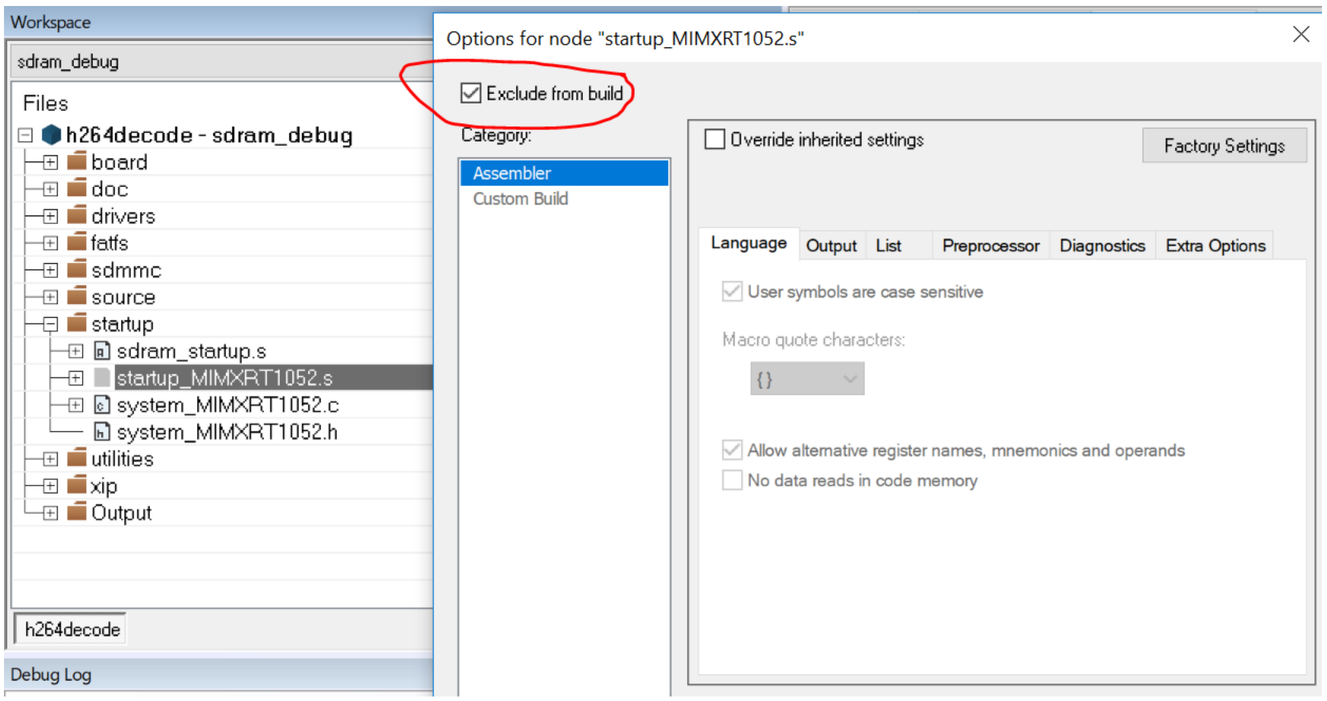


sdram_debug
sdram_debug
flexspi_nor_debug

2. repeat step 2,3,4 in section 3.2.1.

For this demo, The two different builds and run modes of the program have different startup functions. When the program runs in FLASH mode, the startup function is startup_MIMXRT1052.s. When the program runs in SDRAM mode, the startup function is sdram_startup.s.

Figure 4 shows the selection method of the startup file in the sdram mode.



Workspace

sdram_debug

Files

- h264decode - sdram_debug
  - board
  - doc
  - drivers
  - fatfs
  - sdmmc
  - source
  - startup
    - sdram_startup.s
    - startup_MIMXRT1052.s
    - system_MIMXRT1052.c
    - system_MIMXRT1052.h
  - utilities
  - xip
  - Output

h264decode

Debug Log

Options for node "startup_MIMXRT1052.s"

☑ Exclude from build

Category: Assembler / Custom Build

☐ Override inherited settings — Factory Settings

Language | Output | List | Preprocessor | Diagnostics | Extra Options

☑ User symbols are case sensitive

Macro quote characters: { }

☑ Allow alternative register names, mnemonics and operands

☐ No data reads in code memory

Figure 4. Startup files selection

3.3 Memory space allocation

For this demo application, we allocate the memory space with the schemes as shown in Figure 5.

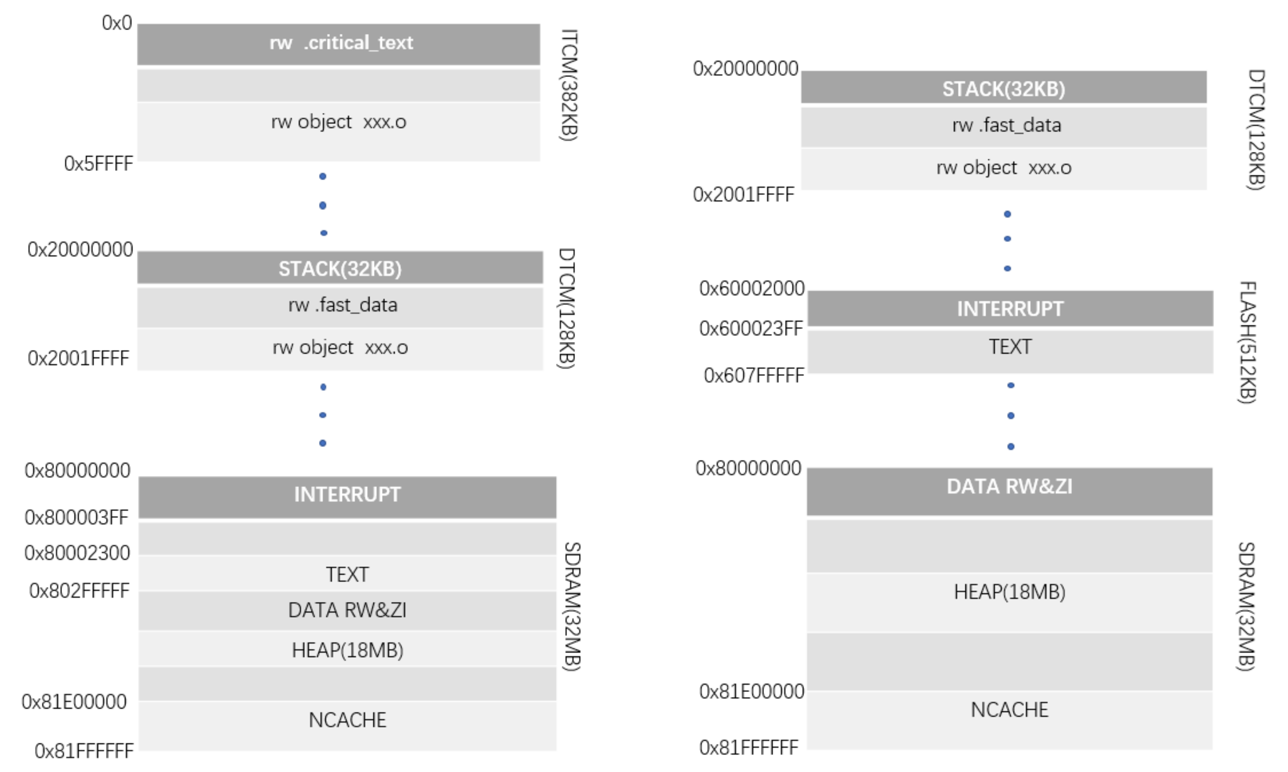


Figure 5. Memory allocation schemes

(a) SDRAM configuration (b) FlexSPI-NOR configuration

3.4 Software decode

The decoding process reads the video source from the microSD card and decode the video data using the clip FFMPEG library(version 3.0.11). The process of FFMPEG video decompression in Figure 6:

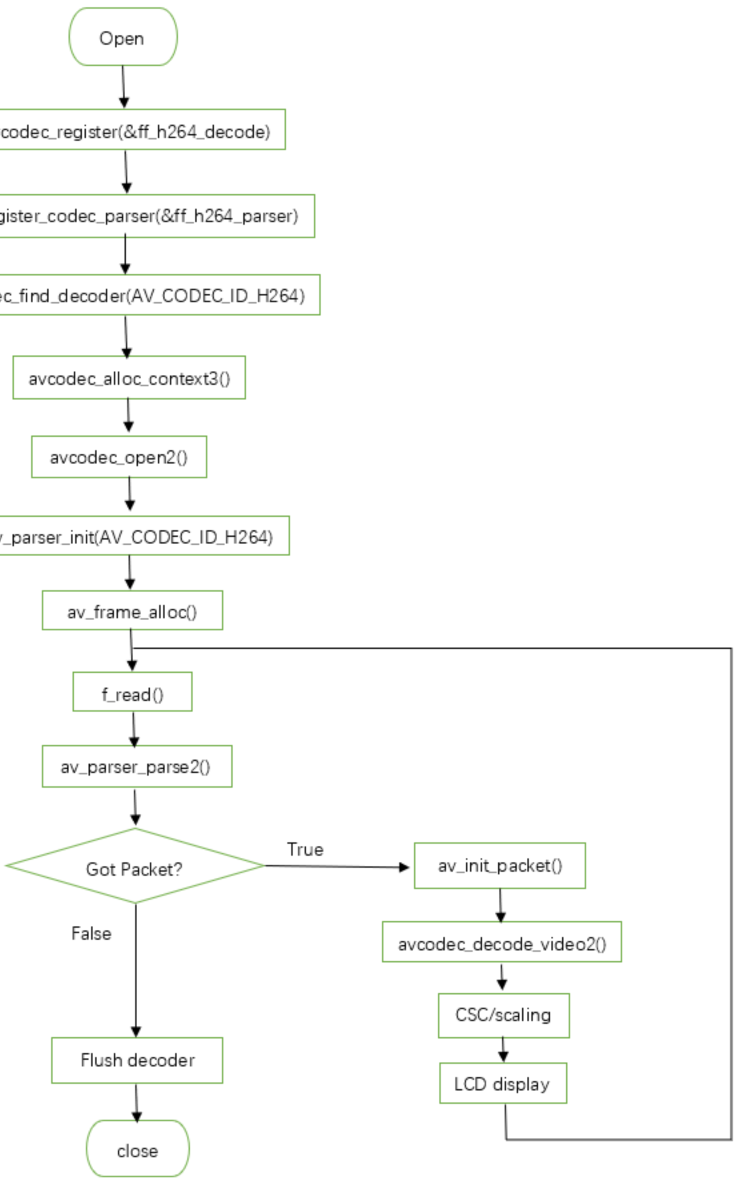


Figure 6. video decoding

The function is explained as follows:

1. avcodec_register(&ff_h264_decoder): Register h264 decoder.
2. av_register_codec_parser(&ff_h264_parser): Register h264 parser.
3. avcodec_find_decoder(AV_CODEC_ID_H264): Find h264 decoder.
4. avcodec_alloc_context3(): Allocate the decoder context and all the content associated with it..
5. avcodec_open2():Open decoder.
6. av_parser_init(AV_CODEC_ID_H264): Select and Init parser .
7. av_frame_alloc():Allocate memory to the AVFrame structure. The data buffer in the AVFrame must be managed in other ways.
8. av_parser_parse2(): Parsing to get a Packet.

9. av_init_packet(): Initialize the value of the packet.
10. avcodec_decode_video2(): Decoding one frame of data.

After the FFMPEG decoding, the original H.264 video source is decompressed into the YUV data format YUV data. To display on LCD panel, it still needs color space conversion(CSC) and scale scaling.

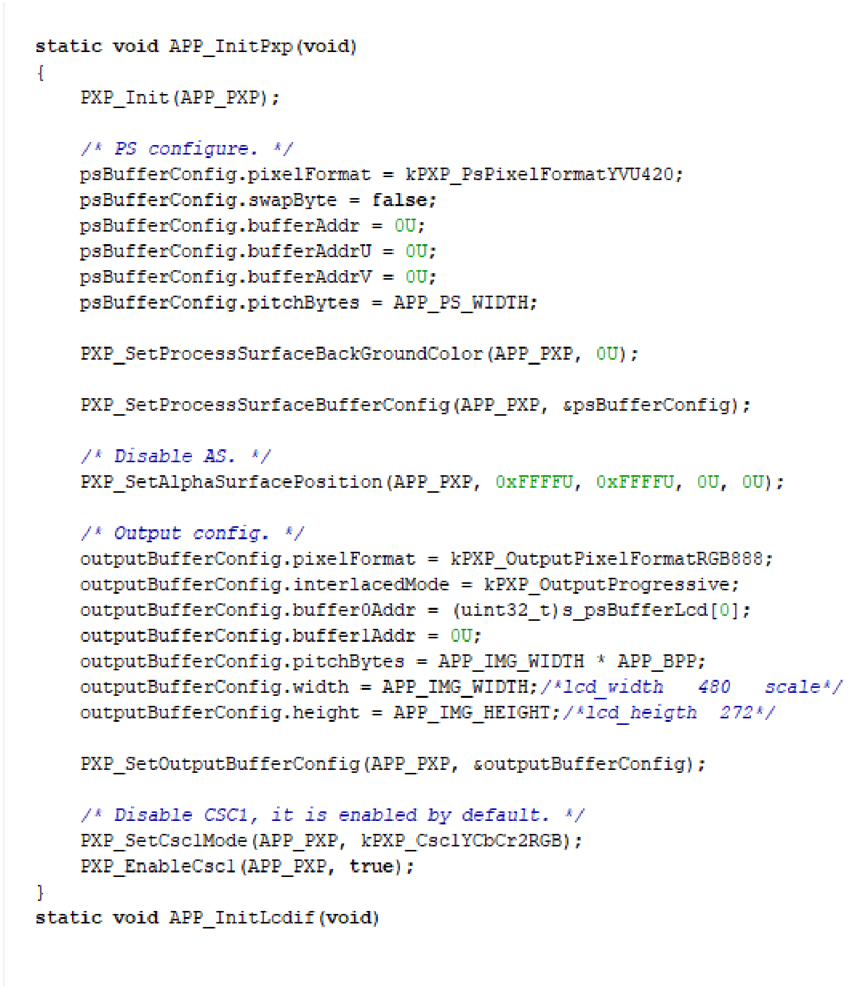
3.5 Scaling and Color Space Conversion (CSC)

The LCD's display resolution is different from the video source resolution and LCD can only display RGB format data, PXP module is responsible for the inter-buffers scaling and CSC.

Due to the different resolution of user video source, the corresponding parameters need to be modified.

```
#define APP_PS_WIDTH 480/* 720,352,image resolution*/
```

Data format configuration in the APP_InitPxp() function, The parameters are configured in Figure 7:



```
static void APP_InitPxp(void)
{
    PXP_Init(APP_PXP);

    /* PS configure. */
    psBufferConfig.pixelFormat = kPXP_PsPixelFormatYVU420;
    psBufferConfig.swapByte = false;
    psBufferConfig.bufferAddr = 0U;
    psBufferConfig.bufferAddrU = 0U;
    psBufferConfig.bufferAddrV = 0U;
    psBufferConfig.pitchBytes = APP_PS_WIDTH;

    PXP_SetProcessSurfaceBackGroundColor(APP_PXP, 0U);

    PXP_SetProcessSurfaceBufferConfig(APP_PXP, &psBufferConfig);

    /* Disable AS. */
    PXP_SetAlphaSurfacePosition(APP_PXP, 0xFFFFU, 0xFFFFU, 0U, 0U);

    /* Output config. */
    outputBufferConfig.pixelFormat = kPXP_OutputPixelFormatRGB888;
    outputBufferConfig.interlacedMode = kPXP_OutputProgressive;
    outputBufferConfig.buffer0Addr = (uint32_t)s_psBufferLcd[0];
    outputBufferConfig.buffer1Addr = 0U;
    outputBufferConfig.pitchBytes = APP_IMG_WIDTH * APP_BPP;
    outputBufferConfig.width = APP_IMG_WIDTH;/*lcd_width   480   scale*/
    outputBufferConfig.height = APP_IMG_HEIGHT;/*lcd_heigth  272*/

    PXP_SetOutputBufferConfig(APP_PXP, &outputBufferConfig);

    /* Disable CSC1, it is enabled by default. */
    PXP_SetCsc1Mode(APP_PXP, kPXP_Csc1YCbCr2RGB);
    PXP_EnableCsc1(APP_PXP, true);
}
static void APP_InitLcdif(void)
```

Figure 7. PXP module initialization configuration

Input video format:

Processed Surface （PS）configure: psBufferConfig.pixelFormat = kPXP_PsPixelFormatYVU420

Output video format:

Output config: outputBufferConfig.pixelFormat = kPXP_OutputPixelFormatRGB888

CSC:

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space only if CSC1 is enabled.

Enable CSC1:

PXP_SetCsc1Mode(APP_PXP, kPXP_Csc1YCbCr2RGB)

PXP_EnableCsc1(APP_PXP, true)

The addresses of Y, U, V data generated after FFMPEG decoding correspond to the Y, U, V addresses of PS configuration.

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 continues and simultaneous.

Figure 8 illustrates the LCD frame buffer accessing scheme.

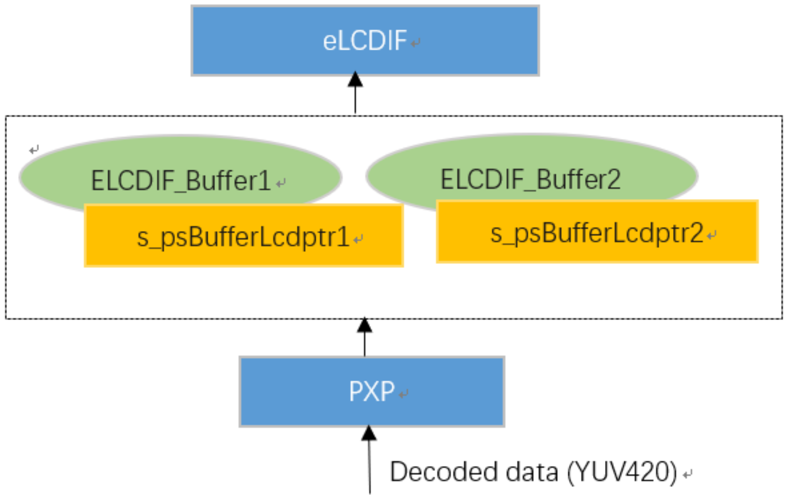


Figure 8. LCD Frame buffers accessing scheme

3.6 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.
- 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 (optional).
- Set boot mode, SW7-1, SW7-2, SW7-3, SW7-4 set to off, on, on, off.
- 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.

4 Performance analysis

In this application, the CPU clock, IPG clock, SDRAM operating frequency, LCD refresh rate and SDRAM working data length are 600 MHz, 150 MHz, 164 MHz, 60 Hz and 16 bit, respectively.

Three different video sources are selected to decode, display and test frame per second(fps) with different build and run methods for this demo.

Table 1 shows the test results for different video sources.

Table 1. Test results(fps)

Video source	Resolution	Total frame	SDRAM(fps)	FLASH(fps)
clown_720x576.h264	720x576	250	21.9	18.5
bigbuckbunny_480x272.h264	480x272	250	34.2	25.4
formen_352x288.h264	352x288	240	41.2	31.7

Table 1 contains the total time of video reading, decoding, and pxp module processing and display.

For restrictions on reading filename and file format, we could configure fatfs to support long filename, just set FF_USE_LFN to 1 in ffconf.h, It is more friendly for customers to test different videos .

PXP converting time is related to storage memory type of src and dest buffer. In previous tests put both src and dest buffer in SDRAM. Now more tests for different memory allocations of the src and dest buffers(by modify .icf files).

Table 2 and Table 3 shows the test time for different videos.

Table 2. Both src and dest video resolutions are 288x180

\Dest buffer Src buffer\	SDRAM	DTCM	ITCM	OCRAM
SDRAM	2.129 ms	1.059 ms	1.059 ms	1.059 ms
DTCM	1.067 ms	0.471 ms	0.471 ms	0.384 ms
ITCM	1.067 ms	0.471 ms	0.470 ms	0.384 ms
OCRAM	1.067 ms	0.384 ms	0.384 ms	0.384 ms

#unique_20 Both src and dest video resolutions are 480x272

Table 3. Both src and dest video resolutions are 480x272

\Dest buffer Src buffer\	SDRAM	DTCM	ITCM	OCRAM
SDRAM	6.172 ms	2.939 ms	2.939 ms	2.939 ms
DTCM	3.394 ms	/	/	0.932 ms
ITCM	3.394 ms	/	/	0.932 ms
OCRAM	3.394 ms	0.932 ms	0.932 ms	0.932 ms

src buffer: Used to storeYUV data generated after decoding

dest buffer: Used to store RGB data generated after pxp module processing.

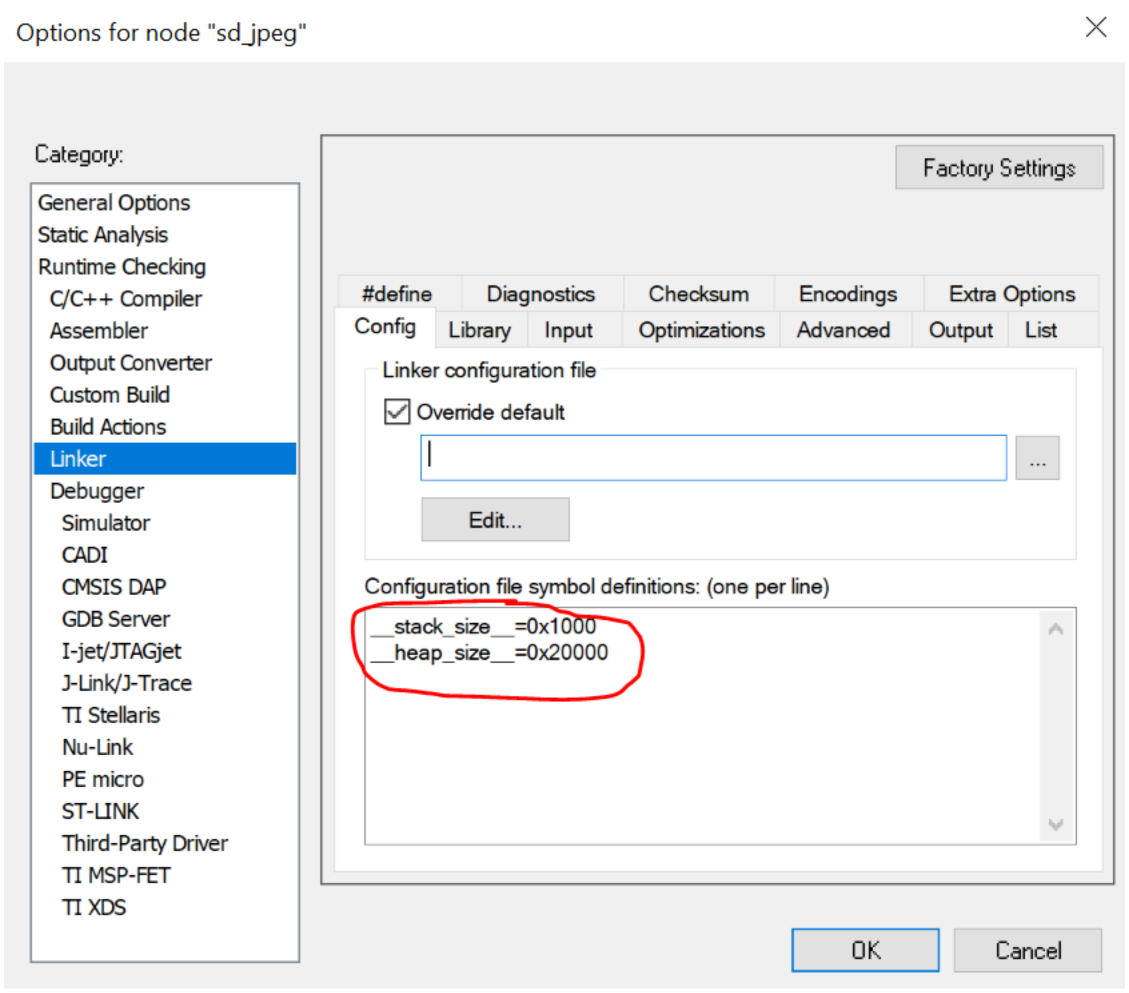
5 Conclusion

This application note describes the steps of how to develop H.264 video decode application with the i.MX RT1050 processor based on the SDK of i.MX RT1050 EVK board, from building 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 H.264 video decode applications.

NOTE

When users use the sdk version of 2.5.0 and above to build project tree, delete the red part of Figure 9, otherwise it overrides the stack and heap configurations/allocations in "icf" linker files, and the demo will fail to run.



Configuration file symbol definitions: (one per line)

```
__stack_size__=0x1000
__heap_size__=0x20000
```

Figure 9. sdk version of 2.5.0 IDE (delete them in red block)

6 References

Following documents may offer further reference.

- i.MX RT1050 Processor Reference Manual, Rev. 0
- FFMPEG video decoding

7 Revision history

Table 4. Revision history

Revision number	Date	Substantive changes
0	09/2019	Initial release

Date of release: 28 August, 2019

Document identifier: AN12562