AN13275

How to Enable Linux BSP L5.4 on a New i.MX8/8X Board

Rev. 1 — 26 May 2023 Application note

Document Information

Information	**Content**
Keywords	iMX8, Linux BSP, port
Abstract	This application note introduces a general procedure of how to enable standard Linux BSP L5.4 on a new customized i.MX8/8X board.

1 Introduction

1.1 Purpose

This application note introduces a general procedure of how to enable standard Linux BSP L5.4 on a new customized i.MX 8/8X board. This document can help users, who have designed a customized i.MX 8/8X board, to quickly port standard Linux BSP release code running on their board and be aware of those key parts that need modifications.

1.2 Example board

This application note uses an i.MX 8QXP reference board for automotive as an example board, because the standard Linux BSP release does not support this board. For more details, contact NXP representative.

The hardware design of the board is based on i.MX 8QXP MEK board, but with the following changes:

- i.MX 8QXP C0 Silicon
- Samsung auto LPDDR4 and eMMC5.1
- MIPI-CSI with NVP6324 automotive AHD solution
- LVDS display with TI DS90UB947/948 SerDes (through FPD-Link III) for automotive application
- MIPI-DSI display with Maxim 96752/96755 SerDes (through GMSL2) for automotive application
- NXP TJA1101 automotive 100 Mbps Ethernet PHY
- USB3.0 host for Carplay/AA and USB2.0 OTG for debug

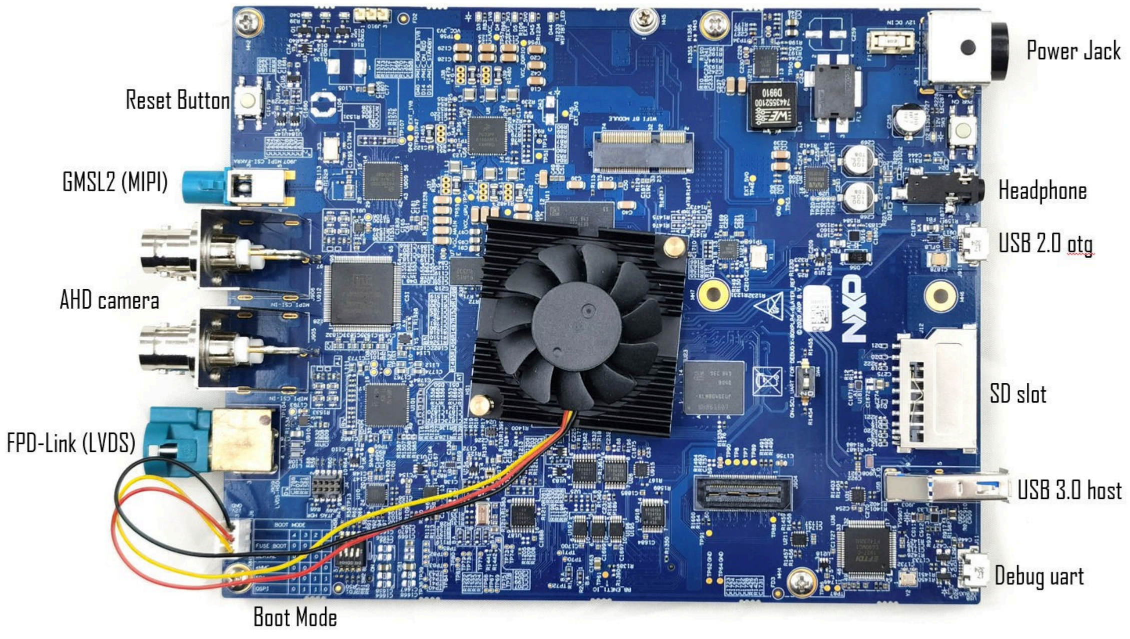


Figure 1. i.MX 8QXP reference board for automotive

1.3 Linux BSP releases

This application note uses the L5.4.47_2.2.0 Linux BSP release as example. For all i.MX Linux BSP releases, see Embedded Linux for i.MX Applications Processors.

The following chapters introduce the general procedure for porting SCFW, ATF, U-Boot, and Linux Kernel. Each of them can be compiled independently, and the release package or source code can be downloaded from following links:

- SCFW
  https://www.nxp.com/webapp/Download?colCode=L5.4.47_2.2.0_SCFWKIT-1.6.0&appType=license
- Arm Trusted Firmware (ATF)
  Git clone https://github.com/nxp-imx/imx-atf -b rel_imx_5.4.47_2.2.0
- U-Boot
  Git clone https://github.com/nxp-imx/uboot-imx -b rel_imx_5.4.47_2.2.0
- imx-mkimage
  Git clone https://github.com/nxp-imx/imx-mkimage -b rel_imx_5.4.47_2.2.0
- Linux Kernel
  Git clone https://github.com/nxp-imx/linux-imx -b rel_imx_5.4.47_2.2.0

2 Generating DDR configuration files

The i.MX 8/8X DDR Register Programming Aid (RPA) is an Excel spreadsheet tool used to develop DDR initialization for the specific DDR configuration (DDR device type, density, and so on). The RPA generates the DDR initialization in two formats (in separate Excel worksheet tabs):

- DDR Stress Test script
  This format is used specifically with the DDR stress test by first copying the contents on the DDR Stress Test Script CBT tab and then pasting it to a text file, naming the document with the `.ds` file extension. Use this file when executing the DDR stress test.
- DCD CFG file
  This format is the configuration file used specifically by the SCU Firmware (SCFW). In this scenario, the user copies the contents on the DCD CFG file CBT tab and pastes it to a text file, naming the document with the `.cfg` file extension and placing this file in the appropriate SCFW board file directory.

2.1 Downloading RPA tools

Note: *In all cases, the RPA revision is aligned to a minimum SCFW version as shown in the table on i.MX 8/8X Family DDR Tools Release. In some cases, the BSP alignment is provided as extra details.*

To obtain the latest RPAs, see the following links:

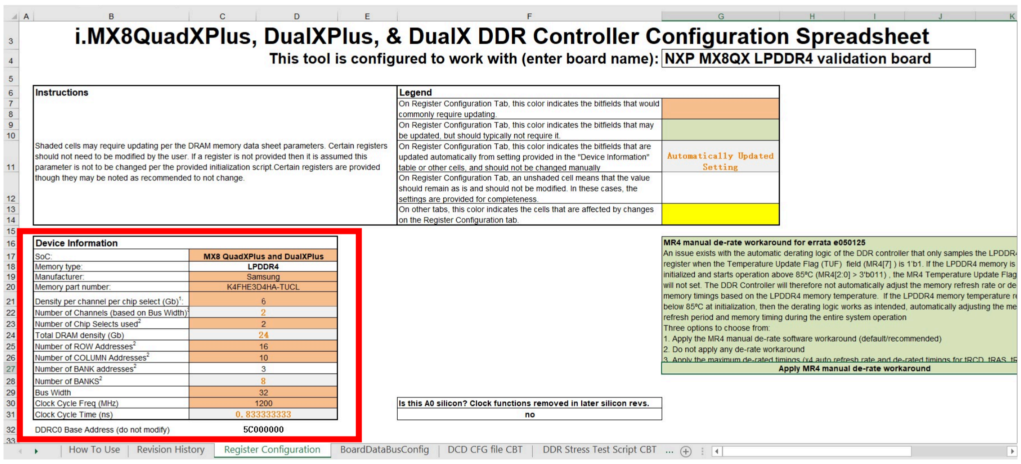
- i.MX8QM DDR Register Programming Aid (RPA)
- i.MX8QXP/DXP/DX DDR Register Programming Aid (RPA)

To align with the L5.4.47_2.2.0 BSP and SCFW 1.6.0, use *MX8QXP_C0_B0_LPDDR4_RPA_1.2GHz_v14.xlsx* RPA version in the below steps.

2.2 Using RPA tools

To use RPA tools to generate a new DDR stress test script and DCD CFG file for the specific DDR on user-customized board, perform the following steps.

1. Obtain the desired DDR data sheet from the DDR vendor
   To fill the DRAM parameters in the RPA tools, use the DDR data sheet from the DDR vendor. Usually this data sheet can be downloaded from the website of the DDR vendor. Users can also contact DDR vendor directly to request this data sheet.
2. Update the Device Information table on the **Register Configuration** tab.



i.MX8QuadXPlus, DualXPlus, & DualX DDR Controller Configuration Spreadsheet

This tool is configured to work with (enter board name): NXP MX8QX LPDDR4 validation board

Instructions

Shaded cells may require updating per the DRAM memory data sheet parameters. Certain registers should not need to be modified by the user. If a register is not provided then it is assumed this parameter is not to be changed per the provided initialization script.Certain registers are provided though they may be noted as recommended to not change.

Legend	
On Register Configuration Tab, this color indicates the bitfields that would commonly require updating.	🟧
On Register Configuration Tab, this color indicates the bitfields that may be updated, but should typically not require it.	🟩
On Register Configuration Tab, this color indicates the bitfields that are updated automatically from setting provided in the "Device Information" table or other cells, and should not be changed manually	Automatically Updated Setting
On Register Configuration Tab, an unshaded cell means that the value should remain as is and should not be modified. In these cases, the settings are provided for completeness.	
On other tabs, this color indicates the cells that are affected by changes on the Register Configuration tab.	🟨

Device Information	
SoC:	MX8 QuadXPlus and DualXPlus
Memory type:	LPDDR4
Manufacturer:	Samsung
Memory part number:	K4FHE3D4HA-TUCL
Density per channel per chip select (Gb)^1:	6
Number of Channels (based on Bus Width)^1	2
Number of Chip Selects used^2	2
Total DRAM density (Gb)	24
Number of ROW Addresses^2	16
Number of COLUMN Addresses^2	10
Number of BANK addresses^2	3
Number of BANKS^2	8
Bus Width	32
Clock Cycle Freq (MHz)	1200
Clock Cycle Time (ns)	0.833333333
DDRC0 Base Address (do not modify)	5C000000

Is this A0 silicon? Clock functions removed in later silicon revs.
no

MR4 manual de-rate workaround for errata e050125

An issue exists with the automatic derating logic of the DDR controller that only samples the LPDDR4 register when the Temperature Update Flag (TUF) field (MR4[7]) is 1'b1. If the LPDDR4 memory is initialized and starts operation above 85ºC (MR4[2:0] > 3'b011) , the MR4 Temperature Update Flag will not set. The DDR Controller will therefore not automatically adjust the memory refresh rate or de memory timings based on the LPDDR4 memory temperature. If the LPDDR4 memory temperature re below 85ºC at initialization, then the derating logic works as intended, automatically adjusting the me refresh period and memory timing during the entire system operation
Three options to choose from:
1. Apply the MR4 manual de-rate software workaround (default/recommended)
2. Do not apply any de-rate workaround
3. Apply the maximum de-rated timings (x4 auto refresh rate and de-rated timings for tRCD, tRAS, tF

Apply MR4 manual de-rate workaround

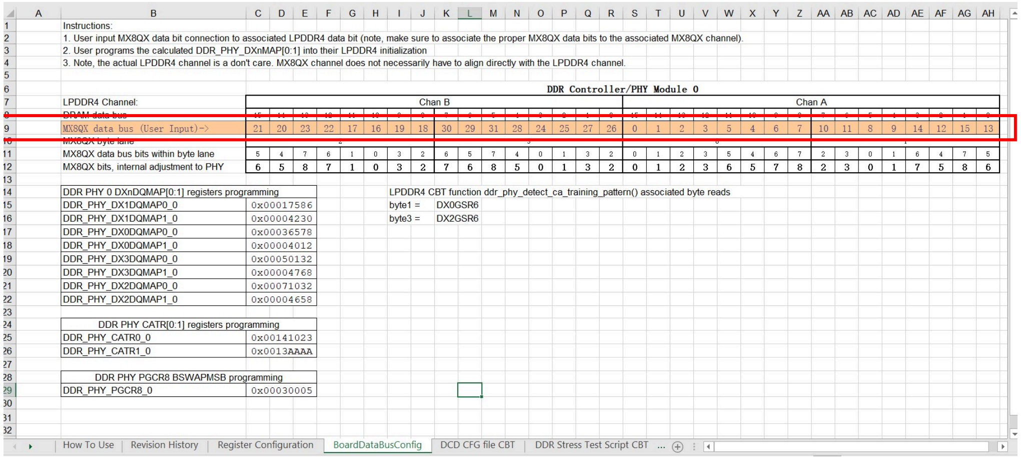
Figure 2. i.MX8QXP RPA tool register configurations

In the Device information table highlighted in Figure 2, update the following information:

- Manufacturer
- Memory part number
- Density per channel per chip select (GB)
- Number of Chip Selects used
- Number of ROW Addresses
- Number of COLUMN Addresses
- Number of BANK addresses
- Bus Width
- Clock cycle Freq (MHz)

Other parameters are automatically calculated and filled in the table using the information above.

3. Update data bus mapping on the **BoardDataBusConfig** tab.



Instructions:
1. User input MX8QX data bit connection to associated LPDDR4 data bit (note, make sure to associate the proper MX8QX data bits to the associated MX8QX channel).
2. User programs the calculated DDR_PHY_DXnMAP[0:1] into their LPDDR4 initialization
3. Note, the actual LPDDR4 channel is a don't care. MX8QX channel does not necessarily have to align directly with the LPDDR4 channel.

DDR Controller/PHY Module 0

LPDDR4 Channel:	Chan B																Chan A															
DRAM data bus	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MX8QX data bus (User Input)->	21	20	23	22	17	16	19	18	30	29	31	28	24	25	27	26	0	1	2	3	5	4	6	7	10	11	8	9	14	12	15	13
MX8QX byte lane	2								3								0								1							
MX8QX data bus bits within byte lane	5	4	7	6	1	0	3	2	6	5	7	4	0	1	3	2	0	1	2	3	5	4	6	7	2	3	0	1	6	4	7	5
MX8QX bits, internal adjustment to PHY	6	5	8	7	1	0	3	2	7	6	8	5	0	1	3	2	0	1	2	3	6	5	7	8	2	3	0	1	7	5	8	6

DDR PHY 0 DXnDQMAP[0:1] registers programming	
DDR_PHY_DX1DQMAP0_0	0x00017586
DDR_PHY_DX1DQMAP1_0	0x00004230
DDR_PHY_DX0DQMAP0_0	0x00036578
DDR_PHY_DX0DQMAP1_0	0x00004012
DDR_PHY_DX3DQMAP0_0	0x00050132
DDR_PHY_DX3DQMAP1_0	0x00004768
DDR_PHY_DX2DQMAP0_0	0x00071032
DDR_PHY_DX2DQMAP1_0	0x00004658

LPDDR4 CBT function ddr_phy_detect_ca_training_pattern() associated byte reads

byte1 =	DX0GSR6
byte3 =	DX2GSR6

DDR PHY CATR[0:1] registers programming	
DDR_PHY_CATR0_0	0x00141023
DDR_PHY_CATR1_0	0x0013AAAA

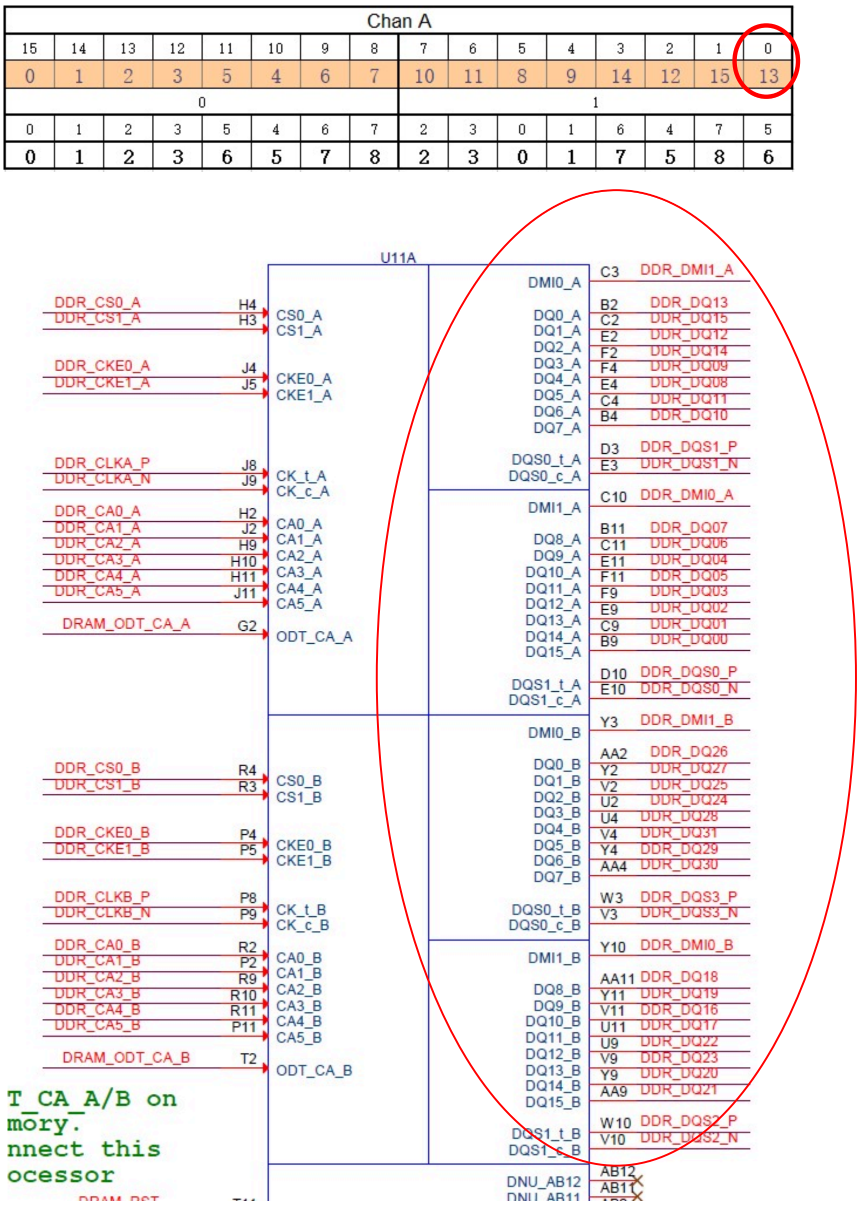
DDR PHY PGCR8 BSWAPMSB programming	
DDR_PHY_PGCR8_0	0x00030005

Figure 3. i.MX 8QXP RPA tool BoardDataBusConfig

Usually the physical connection of data pins between DDR device and SOC is not a direct match due to physical layout constraint. Therefore, we need a mapping table to record the physical connection of DDR data pins, and put this information into the register of DDR controller, so that it can make correct logic connections of DDR data pins.

In the row highlighted in Figure 3, update the physical mapping of data pins between DDR device and SOC, according to their hardware schematic. Other parameters are updated automatically according to user input.

For example, from the example board schematic in Figure 4, we can find that the `DQ0_A` pin on DDR devices is connected to `DDR_DQ13` pin on iMX8QXP, so we type **13** in the circled cell and for others, follow the same method.



Chan A															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	2	3	5	4	6	7	10	11	8	9	14	12	15	13
0								1							
0	1	2	3	5	4	6	7	2	3	0	1	6	4	7	5
0	1	2	3	6	5	7	8	2	3	0	1	7	5	8	6

Figure 4. Connection of DDR data pins in example board schematic

4. Copy text on the DCD CFG file CBT and DDR Stress Test Script CBT tabs to file.
   Click the DCD CFG file CBT tab in RPA tool and copy all the text into a file, naming it as *BOARD_NAME.cfg*. This file is used later when porting SCFW.
   Click the DDR Stress Test Script CBT tab and copy all the text into a file, naming it as *BOARD_NAME.ds*. This file is used in the DDR stress test later.

3 SCFW porting

The System Controller Unit (SCU) provides an abstraction to many underlying features of the hardware. The software running on SCU is known as SC firmware (SCFW). SCFW provides the following features and services.

- System Initialization and Boot
- System Controller Communication
- Power Management
- Resource Management
- Pad Configuration
- Timers
- Interrupts
- Security
- Miscellaneous

Most SCFW codes are provided only in the object file format in SCFW porting kit and users cannot modify. But for board-related settings, SCFW porting kit has provided the source code of *board.c* file, which includes board-related initialization functions and customized features. This chapter focuses on how to port the *board.c* file for a new board.

3.1 Extracting SCFW code

To extract SCFW code, perform the following steps.

1. Download the SCFW 1.6.0 package, *imx-scfw-porting-kit-1.6.0.tar.gz*, from Apps.
2. Unzip the file.
3. Go to the packages folder.
4. Extract SCFW code with the following command.

```
$chmod a+x imx-scfw-porting-kit-1.6.0.bin
$./imx-scfw-porting-kit-1.6.0.bin
```

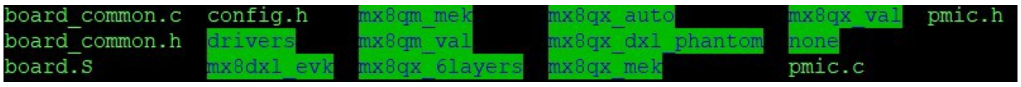
5. After reading and accepting the license, extract the SCFW code in the *imx-scfw-porting-kit-1.6.0* folder. Besides the code, there are release documents extracted in the *imx-scfw-porting-kit-1.6.0/doc/pdf* folder, including release note, api user guide, and a more detailed porting guide. For new users and developers of i.MX8/8X product, these documents are very useful. It is highly recommended to check these documents first when you have questions about SCFW.
6. Use the following command to extract SCFW code specific for i.MX8QXP.

```
$cd imx-scfw-porting-kit-1.6.0/src/
$tar zxvf scfw_export_mx8qx_b0.tar.gz
```

The code is in the path of *imx-scfw-porting-kit-1.6.0/src/scfw_export_mx8qx_b0/*.
We can set this path as *SCFW_DIR*.

3.2 Creating a board file

Each board has its own special hardware design and may have different board operations in SCFW level. Therefore SCFW provides a board folder under *SCFW_DIR/platform/board/* for every supported board.



```
board_common.c  config.h    mx8qm_mek      mx8qx_auto          mx8qx_val  pmic.h
board_common.h  drivers     mx8qm_val      mx8qx_dxl_phantom   none
board.S         mx8dxl_evk  mx8qx_6layers  mx8qx_mek           pmic.c
```

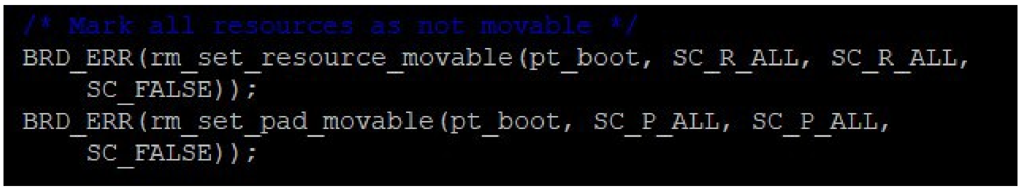
The board folder contains following components:

- `board.bom`: Includes PMIC driver info
- `board.c`: Board related operations
- `board.h`: The header file of `board.c`, including macro definitions used in `board.c`
- `Makefile`: The makefile for compiling of `board.c`
- `dcd/`: The folder of DDR configuration files, usually containing at least two scripts as below:
  - `ddr_stress_test_parser.cfg`: Used for compiling SCFW for DDR stress test.
  - `BOARD_NAME.cfg`: The `ddr` script generated in Section 2.2 and used for compiling SCFW for the normal system use.

To simplify the porting effort, users can directly copy those files from the reference board folder, *mx8qx_mek*, and make modifications according to their own specific requirement.

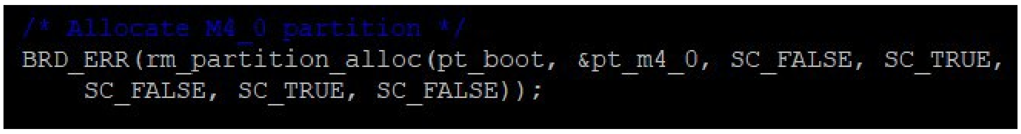
Since the modification is highly related to board design and the use case of the final product, this document does not list detailed modifications. The following three examples are often customized in the *board.c* file of the user.

1. In the `board_system_config()` function
   One major feature that the SCFW provides is resource partitioning. It partitions resources into different domains to protect system. By default, we create a partition for the M4 core, performed in the `board_system_config()` in `board.c`.
   In general, the resource partitioning of M4 follows these steps:
   a. Mark all resources as not movable.



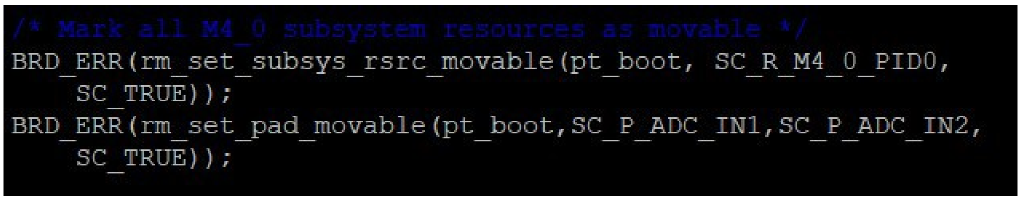
```
/* Mark all resources as not movable */
BRD_ERR(rm_set_resource_movable(pt_boot, SC_R_ALL, SC_R_ALL,
    SC_FALSE));
BRD_ERR(rm_set_pad_movable(pt_boot, SC_P_ALL, SC_P_ALL,
    SC_FALSE));
```

   b. Create a partition for M4 core.



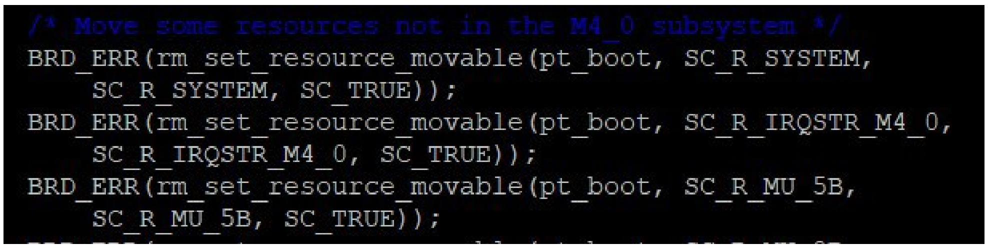
```
/* Allocate M4_0 partition */
BRD_ERR(rm_partition_alloc(pt_boot, &pt_m4_0, SC_FALSE, SC_TRUE,
    SC_FALSE, SC_TRUE, SC_FALSE));
```

   c. Mark all resources and pad that belong to M4 subsystem as movable.

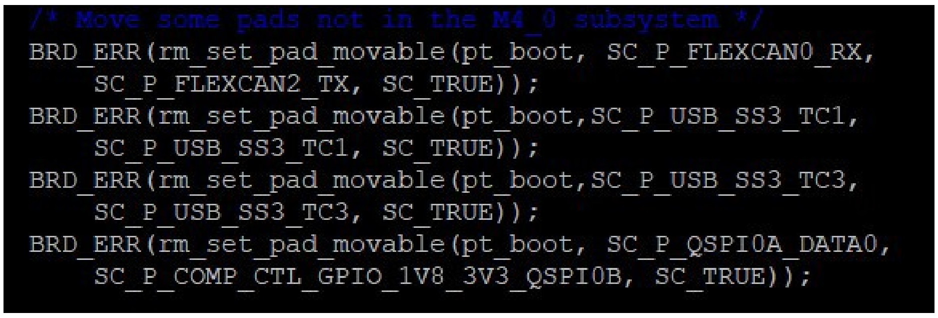


```
/* Mark all M4_0 subsystem resources as movable */
BRD_ERR(rm_set_subsys_rsrc_movable(pt_boot, SC_R_M4_0_PID0,
    SC_TRUE));
BRD_ERR(rm_set_pad_movable(pt_boot,SC_P_ADC_IN1,SC_P_ADC_IN2,
    SC_TRUE));
```

   d. Mark resources and pad that M4 core must use as movable.
   Usually this part need modification according to board design and use case.

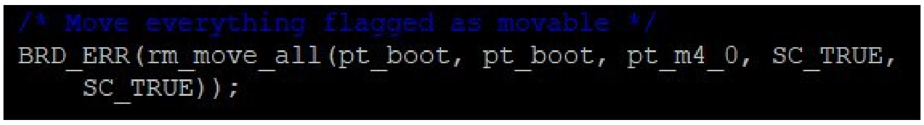


```
/* Move some resources not in the M4_0 subsystem */
BRD_ERR(rm_set_resource_movable(pt_boot, SC_R_SYSTEM,
    SC_R_SYSTEM, SC_TRUE));
BRD_ERR(rm_set_resource_movable(pt_boot, SC_R_IRQSTR_M4_0,
    SC_R_IRQSTR_M4_0, SC_TRUE));
BRD_ERR(rm_set_resource_movable(pt_boot, SC_R_MU_5B,
    SC_R_MU_5B, SC_TRUE));
```



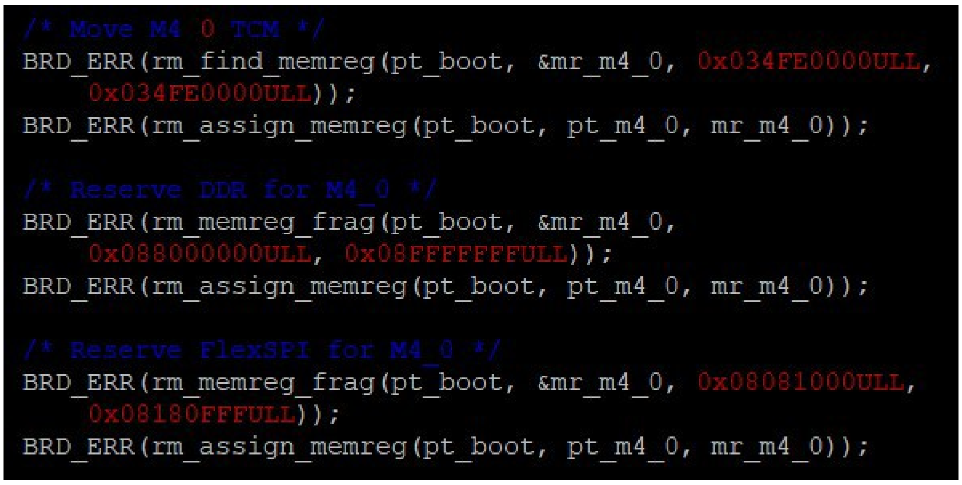
```
/* Move some pads not in the M4_0 subsystem */
BRD_ERR(rm_set_pad_movable(pt_boot, SC_P_FLEXCAN0_RX,
    SC_P_FLEXCAN2_TX, SC_TRUE));
BRD_ERR(rm_set_pad_movable(pt_boot,SC_P_USB_SS3_TC1,
    SC_P_USB_SS3_TC1, SC_TRUE));
BRD_ERR(rm_set_pad_movable(pt_boot,SC_P_USB_SS3_TC3,
    SC_P_USB_SS3_TC3, SC_TRUE));
BRD_ERR(rm_set_pad_movable(pt_boot, SC_P_QSPI0A_DATA0,
    SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0B, SC_TRUE));
```

e. Move all resources and pads that have been marked as movable to M4 partition.



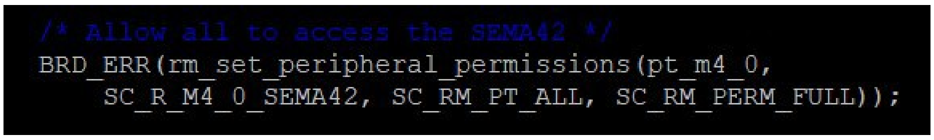
```
/* Move everything flagged as movable */
BRD_ERR(rm_move_all(pt_boot, pt_boot, pt_m4_0, SC_TRUE,
    SC_TRUE));
```

f. Assign memory region for M4 partition.
Adjust the memory region in DDR according to the DRAM size of the board.



```
/* Move M4_0 TCM */
BRD_ERR(rm_find_memreg(pt_boot, &mr_m4_0, 0x034FE0000ULL,
    0x034FE0000ULL));
BRD_ERR(rm_assign_memreg(pt_boot, pt_m4_0, mr_m4_0));

/* Reserve DDR for M4_0 */
BRD_ERR(rm_memreg_frag(pt_boot, &mr_m4_0,
    0x088000000ULL, 0x08FFFFFFFULL));
BRD_ERR(rm_assign_memreg(pt_boot, pt_m4_0, mr_m4_0));

/* Reserve FlexSPI for M4_0 */
BRD_ERR(rm_memreg_frag(pt_boot, &mr_m4_0, 0x08081000ULL,
    0x08180FFFULL));
BRD_ERR(rm_assign_memreg(pt_boot, pt_m4_0, mr_m4_0));
```

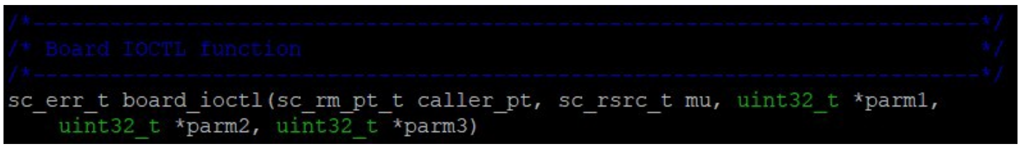
g. Grant permissions for other partitions to access resources of M4.
This part can also be customized depending on use case.



```
/* Allow all to access the SEMA42 */
BRD_ERR(rm_set_peripheral_permissions(pt_m4_0,
    SC_R_M4_0_SEMA42, SC_RM_PT_ALL, SC_RM_PERM_FULL));
```

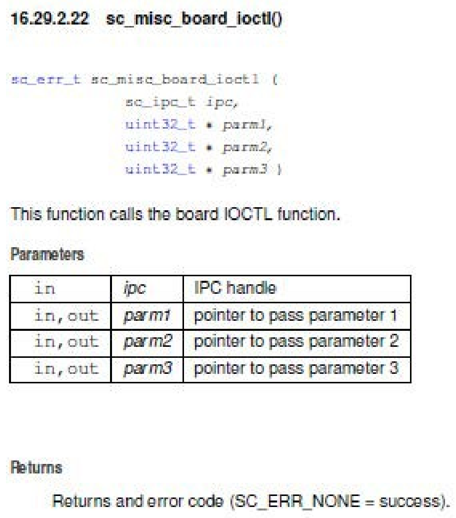
For the detailed descriptions of the SCFW API used above, see Chapter 16 in ***sc_fw_port.pdf***.

2. In the `board_ioctl()` function
In certain use case, add the board level function or feature implementation of the user in SCFW. The `board_ioctl()` function in `board.c` is a good place to do so.



```
/*--------------------------------------------------------------------------*/
/* Board IOCTL function                                                     */
/*--------------------------------------------------------------------------*/
sc_err_t board_ioctl(sc_rm_pt_t caller_pt, sc_rsrc_t mu, uint32_t *parm1,
    uint32_t *parm2, uint32_t *parm3)
```

Use the SCFW API, `sc_misc_board_ioctl`, from Linux or M4 to get into `board_ioctl()` function in `board.c`.



16.29.2.22 sc_misc_board_ioctl()

```
sc_err_t sc_misc_board_ioctl (
            sc_ipc_t ipc,
            uint32_t * parm1,
            uint32_t * parm2,
            uint32_t * parm3 )
```

This function calls the board IOCTL function.

Parameters

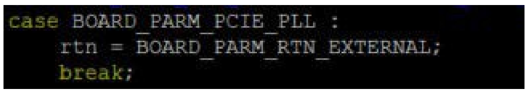
in	*ipc*	IPC handle
in,out	*parm1*	pointer to pass parameter 1
in,out	*parm2*	pointer to pass parameter 2
in,out	*parm3*	pointer to pass parameter 3

Returns

Returns and error code (SC_ERR_NONE = success).

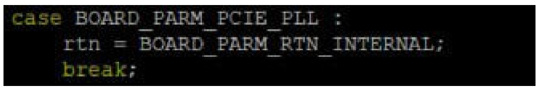
The `sc_misc_board_ioctl()` function is passed almost directly to the `board_ioctl()` function. Three parameters are passed and returned by pointer and an error code is returned. Users can define meaning for these three parameters, and implement their own features in the `board_ioctl()` function. This call is not associated to any resource so there is no security. The MU the API call came from is passed in and the partition number that owns that MU is also passed in. These can be used to implement some kind of security.

3. In the `board_parameter()` function
   The `board_parameter()` function in `board.c`, as its name implies, is used for configuring board level parameters. It includes PCIe PLL clock source, settings for KS1 mode and spread spectrum feature for display.
   Modify the return value for each parameter to choose desired configuration for their board. For example, to use external clock as the source clock of PCIe PLL, set *board_parameter()* as below:



```
case BOARD_PARM_PCIE_PLL :
    rtn = BOARD_PARM_RTN_EXTERNAL;
    break;
```

   To use internal clock as the source clock of PCIe PLL, set `board_parameter()` as below:



```
case BOARD_PARM_PCIE_PLL :
    rtn = BOARD_PARM_RTN_INTERNAL;
    break;
```

   For all available parameter settings, see **Chapter 4.4.1 Board Parameters** in *sc_fw_port.pdf*, or the header file in *SCFW_DIR/platform/main/board.h*.

3.3 Compiling SCFW

To compile SCFW, perform the following steps:

1. Set building environment
   SCFW builds are compiled with a cross compiler in Linux environment. The toolchain for compiling should be obtained from GNU Arm Embedded Toolchain Downloads. The version used is the GNU Arm Embedded Toolchain: 8-2018-q4-major December 20, 2018. Download the toolchain source and follow instructions to install the toolchain on the host Linux machine.
   After the installation, the environment variable, `TOOLS`, must be set to the directory containing the compiler directory. For example, if using the GCC 4.9 cross-compile tools chain and installing to */home/example/gcc-arm-none-eabi-8-2018- q4-major*, set TOOLS to /home/example. Building also requires `srec_cat`, found in the Linux `srecord` package. Optionally the `cppcheck` package is also useful.

If using bash, then set the TOOLS environment variable as follows:

```
$export TOOLS=<your path to dir holding the toolchain>
```

2. Compile the code
The SC firmware can be fully compiled using the Makefile. The command format is:

```
Usage: make TARGET OPTIONS
```

SCFW targets are based on the die, not the part number. Some parts are phantoms of other die (for example QP is a phantom of QM) created by fusing options. The SCFW supports phantoms at runtime, reading the fuses and adapting. There are three primary die targets:

- `qm`: i.MX8QM die
- `qx`: i.MX8QX die
- `dxl`: i.MX8DXL die

They generate the image (*scfw_tcm.bin*) in their respective build directory. Table 1 lists options that can be specified on the `make` command line.

Table 1. Options on make command line

Option	Action
V=0	quiet output (default)
V=1	verbose output
D=0	configure for no debug
D=1	configure for debug (default)
DL=<level>	configure debug level (0-5)
M=0	no debug monitor (default)
M=1	include debug monitor
B=<board>	configure board (default=val)
U=<uart>	configure debug UART (default=0)
DDR_CON=<file>	specify DDR configuration file w/o extension
R=<srev>	silicon revision
T=<test>	run tests rather than boot next core

This application note uses an i.MX8QXP board for automotive as an example board, so the board folder name is mx8qx_auto and DDR script name is imx8qxp_auto_samsung3GB_1.2GHz_v14.cfg. The compile command is:

```
$make qx R=b0 B=auto M=1 U=2 DDR_CON=imx8qxp_auto_samsung3GB_1.2GHz_v14
```

Note: *For MX8QX, the R=b0 applies to both B0 and C0 silicon revisions. In other words, even if you are building for and using C0 silicon, you still use R=b0.*
If the compilation is successful, the SCFW binary can be found in the path of *SCFW_DIR/build_mx8qx_b0/scfw_tcm.bin*.
Note:
DDR stress test in Section 4, an SCFW with special DDR script `ddr_stress_test_parser.cfg` *is needed. Therefore, besides the standard SCFW, users must compile a special SCFW for DDR stress test with following command:*

```
$make qx R=b0 B=auto DDR_CON=ddr_stress_test_parser
```

4 Running a DDR stress test

MX8 DDR stress test is a software application to verify DDR interface on i.MX8 series boards. It is a program running on the PC side which downloads a test image to the internal RAM of i.MX series processors through a USB connection. To access Windows Registry, user must run it in administrator mode. The test image running on the target board executes the DDR stress test. The result is sent to the PC via the A-core UART and is displayed in the log window. There is also an option to save the output to a log file.

MX8 DDR Stress Test can help verify DDR stability on the board in a non-OS environment.

To run the DDR Stress Test Tool, perform the following steps:

1. Download the DDR stress test tool from i.MX 8/8X Family DDR Tools Release.
   After being downloaded and installed, the tool can be found in the *mx8_ddr_stress_test_ER14* folder under the install path. For more details about DDR stress test tool, see *MX8_DDR_Tool_User_Guide.pdf*.
   For more details about DDR stress test tool, see *MX8_DDR_Tool_User_Guide.pdf*.

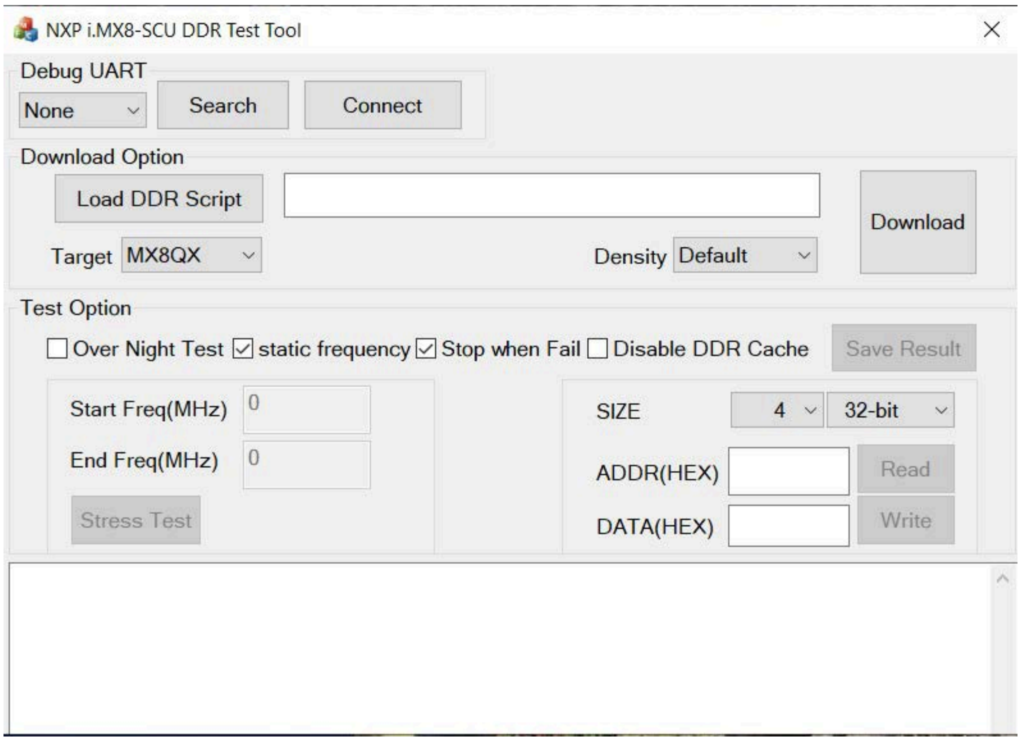
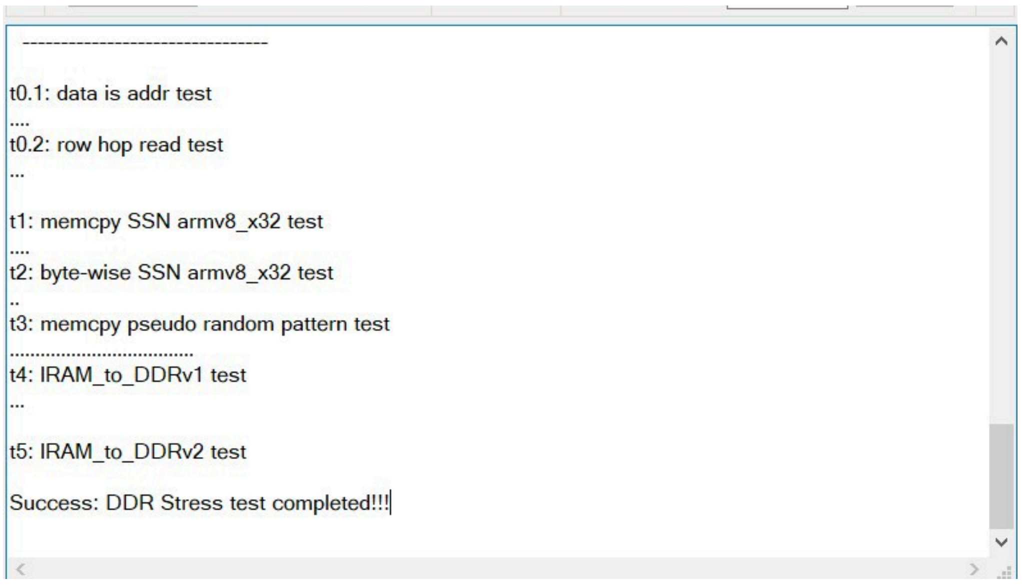


Figure 5. MX8 DDR Stress Test Tool user interface

2. Prepare the following two files:
   - The DDR script for DDR stress test
     This file is generated in Step 4 in Section 2.2. In this case, the name of the DDR script is *imx8qxp_auto_board.ds* and the file can be put in the *mx8_ddr_stress_test_ER14\script\mx8qx\imx8qxp_auto_board.ds* folder.
   - The special SCFW
     This file is generated in Section 3.3, with DDR script, `ddr_stress_test_parser.cfg`. Rename the SCFW binary file from *scfw_tcm.bin* to *mx8qxb0_scfw_download.bin* and replace the file in the tool folder, *mx8_ddr_stress_test_ER14\bin\mx8qxb0_scfw_download.bin*.

Note: For MX8QX, this name applies to both B0 and C0 silicon revisions. In other words, even if you are building for and using C0 silicon, you must rename the scfw_tcm.bin as mx8qxb0_scfw_download.bin.

3. Connect the target board to PC host.
   a. Configure the i.MX target board to boot in serial download mode/Manufacture mode and power up the board.
   b. Connect a UART cable from the host computer to the MX8 debug UART. For Win10, may require manually installing COM port driver (FTDI, SiLabs,…)
   c. Connect a USB cable from the host computer to the USB OTG port on the MX8 target board. An HID-compliant device or USB input device is shown in the **Device Manager**. For the MX8 USB OTG connection, the USB cable must be connected directly to the Host PC USB port and not through a USB HUB.
4. Launch the *MX8_DDR_Tester.exe* in the tool folder.
   ***Note:** For Win10, right-click MX8_DDR_Tester.exe and select **Run as administrator** to view and select the available COM ports.*
5. Press the **Search** button in the **Debug UART** area, choose the correct UART port connected to the MX8 Cortex A-Core Debug UART, and press the **Connect** button.
   ***Note:** To view and select the available COM port, run the DDR stress test in administrator mode.*
6. Load DDR initialization script and choose correct downloading options.
   In this example, we choose the script in the path of *mx8_ddr_stress_test_ER14\script\mx8qx\imx8qxp_auto_board.ds*.
7. Press the **Download** button and wait for target board to be ready.
   If the target board boots successfully, DDR initialization information is present on the console of the tool.
8. Press the **Stress Test** button, with all default settings: default DDR frequency, cache enabled, one loop DDR stress test, stop when encounter error.
   If the board passes the DDR stress test successfully, the **Success: DDR Stress test completed!!!** log is shown as below:



```
 --------------------------------

t0.1: data is addr test
....
t0.2: row hop read test
...

t1: memcpy SSN armv8_x32 test
....
t2: byte-wise SSN armv8_x32 test
..
t3: memcpy pseudo random pattern test
.................................
t4: IRAM_to_DDRv1 test
...

t5: IRAM_to_DDRv2 test

Success: DDR Stress test completed!!!
```

9. Select **Over Night Test**, press the **Stress Test** button again, and the infinite loop of DDR stress test starts. To increase the confidence on the DDR stability, the board must pass DDR stress test for more than 12 hours and repeat the same test in high/low temperature.

5 ATF porting

ATF is a reference implementation of secure world software for Arm A-Profile architectures (Armv8-A and Armv7-A), including an Exception Level 3 (EL3) Secure Monitor. It provides a suitable starting point for production of secure world boot and runtime firmware, in either the AArch32 or AArch64 execution states.

ATF implements Arm interface standards, including:

- Power State Coordination Interface (PSCI)
- Trusted Board Boot Requirements CLIENT (TBBR-CLIENT)
- SMC Calling Convention
- System Control and Management Interface (SCMI)
- Software Delegated Exception Interface (SDEI)

The code is portable and reusable across hardware platforms and software models that are based on the Armv8-A and Armv7-A architectures. Users are encouraged to do their own security validation, including penetration testing, on any secure world code derived from ATF.

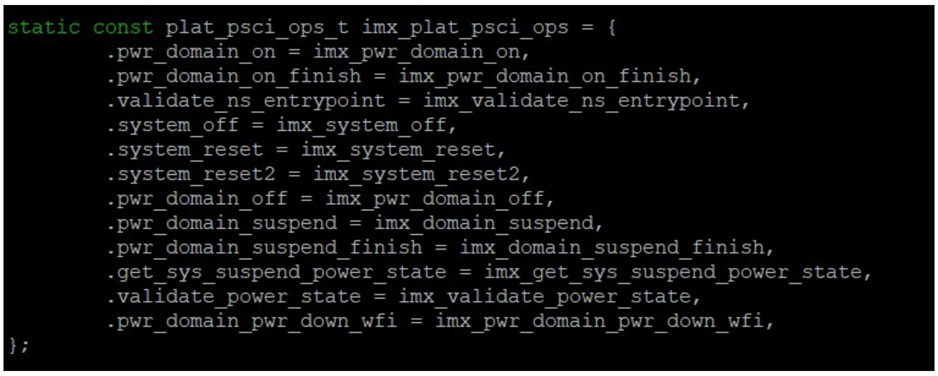
For i.MX8 chips, ATF is required for all i.MX8 boards. Usually two parts might need customization when porting for a new board: power management and resource partitioning.

5.1 Power management

As mentioned above, ATF provides PSCI for Linux system to call as power management mechanism. Each SOC platform can have its own platform-specific PSCI implementation. Below takes the PSCI implementation of i.MX8QXP as an example.

After following Section 1.3 to download ATF source code to the arm-trusted-firmware folder, the PSCI implementation code of i.MX8QXP is in `arm-trusted-firmware/plat/imx/imx8qx/imx8qx_psci.c`.

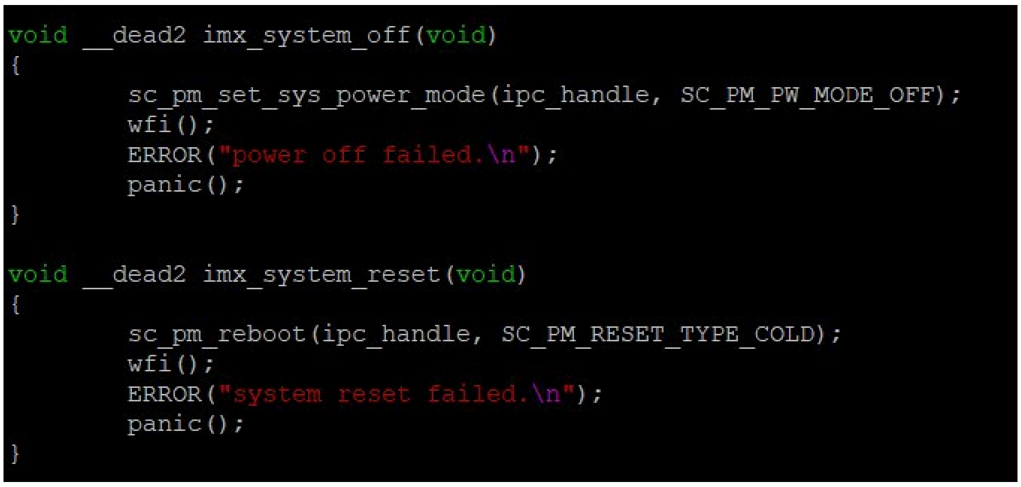
As shown below, all PSCI operations specific to iMX8QXP platform is defined in the `imx_plat_psci_ops` structure and mapped to each implementation function.



```
static const plat_psci_ops_t imx_plat_psci_ops = {
        .pwr_domain_on = imx_pwr_domain_on,
        .pwr_domain_on_finish = imx_pwr_domain_on_finish,
        .validate_ns_entrypoint = imx_validate_ns_entrypoint,
        .system_off = imx_system_off,
        .system_reset = imx_system_reset,
        .system_reset2 = imx_system_reset2,
        .pwr_domain_off = imx_pwr_domain_off,
        .pwr_domain_suspend = imx_domain_suspend,
        .pwr_domain_suspend_finish = imx_domain_suspend_finish,
        .get_sys_suspend_power_state = imx_get_sys_suspend_power_state,
        .validate_power_state = imx_validate_power_state,
        .pwr_domain_pwr_down_wfi = imx_pwr_domain_pwr_down_wfi,
};
```

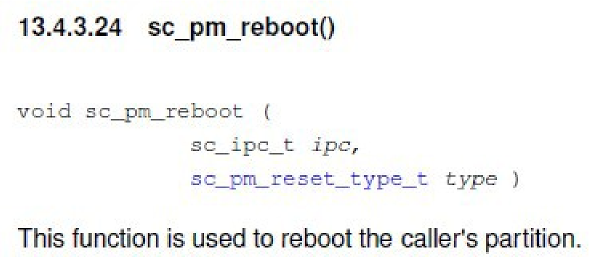
Most functions are implemented in this file. Most operations are calling SCFW API to do the power-related operations, because SCU in i.MX8 architecture controls the power domain of all subsystems. If users have specific requirement for a certain power mode, they can modify the implementation function here.

For some other i.MX8 common functions like `system_off` and `system_reset`, the implementation function can be found in `arm-trusted-firmware/plat/imx/common/imx8_psci.c`.



```
void __dead2 imx_system_off(void)
{
        sc_pm_set_sys_power_mode(ipc_handle, SC_PM_PW_MODE_OFF);
        wfi();
        ERROR("power off failed.\n");
        panic();
}

void __dead2 imx_system_reset(void)
{
        sc_pm_reboot(ipc_handle, SC_PM_RESET_TYPE_COLD);
        wfi();
        ERROR("system reset failed.\n");
        panic();
}
```

One possible modification that users may need is in the `imx_system_reset` function. By default, the `imx_system_reset` function calls `sc_pm_reboot` SCFW API to do partition reboot, which means only the partition of A core is rebooted and the partition of M4 is not affected.

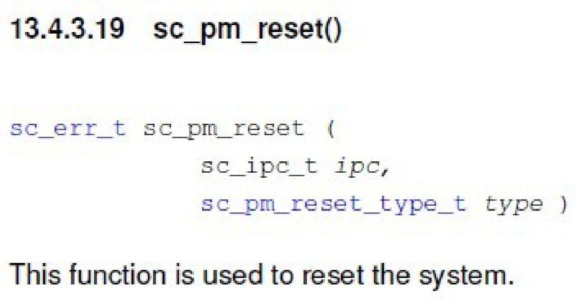


13.4.3.24 sc_pm_reboot()

```
void sc_pm_reboot (
            sc_ipc_t ipc,
            sc_pm_reset_type_t type )
```

This function is used to reboot the caller's partition.

But in some use cases, user may need the whole board to be reset when Linux system is reset. In such situation, use `sc_pm_reset` SCFW API instead of `sc_pm_reboot` in the `imx_system_reset` function. However, only the owner of the `SC_R_SYSTEM` resource or a partition with access permissions to `SC_R_SYSTEM` can call `sc_pm_reset` to reset the whole board.



13.4.3.19 sc_pm_reset()

```
sc_err_t sc_pm_reset (
            sc_ipc_t ipc,
            sc_pm_reset_type_t type )
```

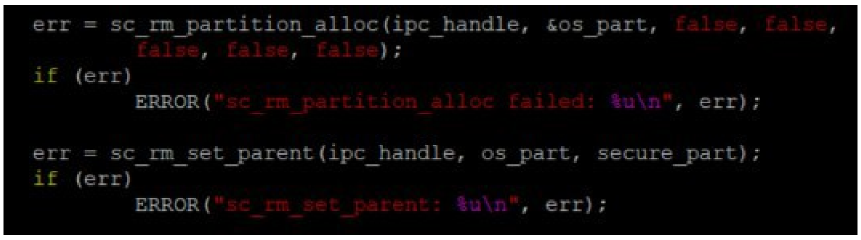
This function is used to reset the system.

5.2 Resource partitioning

For i.MX8 chips, besides power management, another important role of ATF is to create resource partitions for non-secure world of A cores. Take i.MX8QXP as an example, this work is done in *arm-trusted-firmware/plat/imx/imx8qx/imx8qx_bl31_setup.c*, in the *imx8_partition_resources*function.

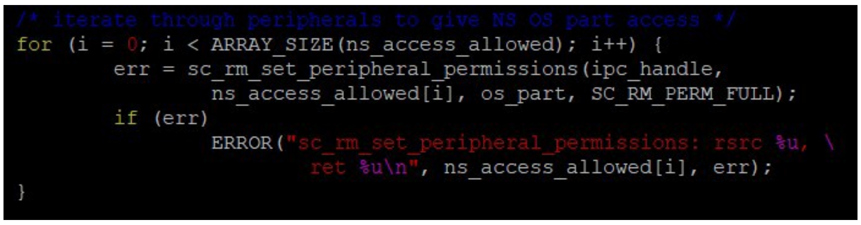
The process of creating and assigning resources to non-secure world partitions in ATF is similar to the process for M4 in Section 3.2. In general, performs the following steps:

1. Create a partition for non-secure world and set the parent as ATF partition.



```
 err = sc_rm_partition_alloc(ipc_handle, &os_part, false, false,
        false, false, false);
 if (err)
        ERROR("sc_rm_partition_alloc failed: %u\n", err);

 err = sc_rm_set_parent(ipc_handle, os_part, secure_part);
 if (err)
        ERROR("sc_rm_set_parent: %u\n", err);
```

2. Mark all resources that must be kept in ATF partition as non-movable.

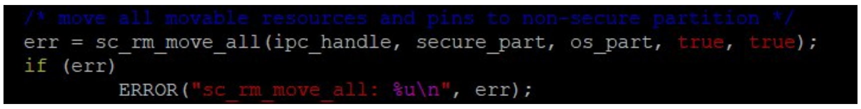


```
/* iterate through peripherals to give NS OS part access */
for (i = 0; i < ARRAY_SIZE(ns_access_allowed); i++) {
        err = sc_rm_set_peripheral_permissions(ipc_handle,
                ns_access_allowed[i], os_part, SC_RM_PERM_FULL);
        if (err)
                ERROR("sc_rm_set_peripheral_permissions: rsrc %u, \
                        ret %u\n", ns_access_allowed[i], err);
}
```

   The `secure_rsrcs[]` array is defined in *arm-trusted-firmware/plat/imx/imx8qx/include/sec_rsrc.h* and contains resources that are going to stay in ATF partition. If users have specific requirement, it can be modified.

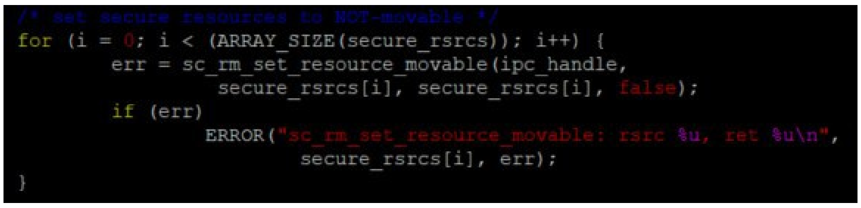
3. Allocate memory region for non-secure world. Depending on whether there is OP-TEE or Trusty implemented, the memory region changes accordingly.
   Note:
   When OP-TEE is implemented,
   - *If the DRMA size is equal to or greater than 2 GB, by default, the memory region `0xFE000000` – `0xFFFFFFFF` is used by OP-TEE. This value is set in arm-trusted-firmware/ plat/imx/imx8qx/ platform.mk with `BL32_BASE` and `BL32_SIZE`.*
   - *If the DRAM size of the new board is less than 2 GB, modify `BL32_BASE` to the highest memory address – `BL32_SIZE`. For example, if DRAM size is 1 GB, `BL32_BASE` is `0xBE000000`.*

4. Move all movable resources and pins to non-secure world partition.



```
 /* move all movable resources and pins to non-secure partition */
 err = sc_rm_move_all(ipc_handle, secure_part, os_part, true, true);
 if (err)
        ERROR("sc_rm_move_all: %u\n", err);
```

5. Grant access of certain sources to non-secure world partition.



```
/* set secure resources to NOT-movable */
for (i = 0; i < (ARRAY_SIZE(secure_rsrcs)); i++) {
        err = sc_rm_set_resource_movable(ipc_handle,
                secure_rsrcs[i], secure_rsrcs[i], false);
        if (err)
                ERROR("sc_rm_set_resource_movable: rsrc %u, ret %u\n",
                        secure_rsrcs[i], err);
}
```

5.3 Compiling ATF

To compile ATF, perform the following steps:

1. Set building environment
   The toolchain used to compile ATF is same cross-compile toolchain used for compiling U-Boot and Linux kernel in the later Chapter. For how to generate and install the toolchain, see **Chapter 4.5.12 How to build U-Boot and Kernel in standalone environment** in *i.MX Linux User's Guide* (document IMXLUG).

https://www.nxp.com/webapp/Download?colCode=L5.4.47_2.2.0_LINUX_DOCS

2. Compile the code
   - Use the following command to compile ATF if OP-TEE is not implemented.
     ```
     $ make PLAT=imx8qx bl31
     ```
   - Use the following command to compile ATF if OP-TEE is implemented.
     ```
     $ make PLAT=imx8qx SPD=opteed bl31
     ```

   If the compilation is successful, the binary file is located in *arm-trusted-firmware/build/imx8qx/release/bl31.bin*.
3. Enable debug print
   By default, the debug print is not enabled in ATF. To enable debug print, change `DEBUG_CONSOLE and DEBUG_CONSOLE_A35` to be defined as **1** in *arm-trusted-firmware/ plat/imx/imx8qx/include/platform_def.h*. Compile ATF with the following command.
   ```
   $ make DEBUG=1 PLAT=imx8qx SPD=opteed bl31
   ```
   The binary file is in *arm-trusted-firmware/build/imx8qx/debug/bl31.bin*.

6 U-Boot porting

The Universal Bootloader (shortened as U-Boot) is an open source and primary bootloader used in embedded devices. It packs the instructions to boot the operating system kernel of the device.

U-Boot is both a first-stage and second-stage bootloader. It is loaded by the ROM of the system or the BIOS from a supported boot device, such as, an SD card, SATA drive, NOR flash (such as, using SPI or I^2C), or NAND FLASH. If there are size constraints, U-Boot splits into stages:

- The platform loads a small Secondary Program Loader (SPL), which is a stripped-down version of U-Boot.
- The SPL initializes hardware configuration and loads the larger, fully featured version of U-Boot.

Regardless of whether the SPL is used, U-Boot performs both first-stage (such as, configuring memory controllers and SDRAM) and second-stage booting (performing multiple steps to load a modern operating system from various devices that must be configured, presenting a menu for users to interact with and control the boot process, and so on).

i.MX8 chips support both SPL or non-SPL U-Boot. In Linux BSP L5.4.47_2.2.0 and later release, the SPL is enabled as default.

6.1 Creating files for a new board

After following steps in Section 1.3, download the U-Boot source code to the *uboot-imx* folder.

To port U-Boot for a new board, create files listed in Table 2 for the new board. To save the porting effort, users can copy files from those files for MEK reference board and make modifications according to their own specific requirement.

In the following context, use imx8qxp_auto as the board name for files created for our new board. Users can modify the filename for their own board.

Table 2. Files needed for a new board in U-Boot

File location	Description
configs/imx8qxp_auto_defconfig	The `defconfig` file for auto board
board/freescale/imx8qxp_auto/	The board folder for auto board

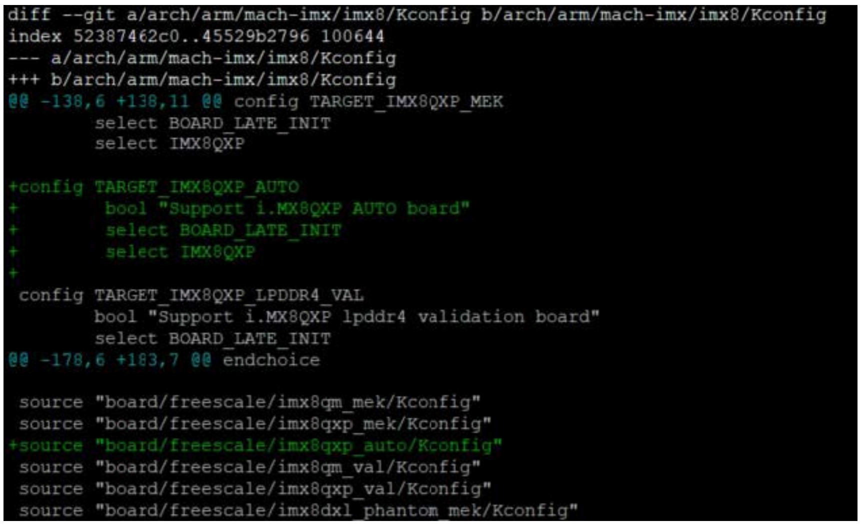
Table 2. Files needed for a new board in U-Boot*...continued*

File location	Description
board/freescale/imx8qxp_auto/Kconfig	The `Kconfig` file for auto board
board/freescale/imx8qxp_auto/Makefile	The `makefile` for C files in auto board folder
board/freescale/imx8qxp_auto/imx8qxp_auto.c	The board file that implements board-related initialization functions, such as, `board_init()`
board/freescale/imx8qxp_auto/imximage.cfg	The file used for configuring imx8 boot image
board/freescale/imx8qxp_auto/spl.c	The file for board-related implementation for SPL boot
board/freescale/imx8qxp_auto/uboot-container.cfg	The file to create a container image that SPL can load
include/configs/imx8qxp_auto.h	The header file for auto board
arch/arm/mach-imx/imx8/ snvs_security_sc_conf_8qxp_auto.h	The header file of `snvs` security configuration of auto board
arch/arm/dts/fsl-imx8qxp-auto.dts	The device tree file for auto board
arch/arm/dts/fsl-imx8qxp-auto-u-boot.dtsi	The device tree including files used for generating `dts` for SPL. For details, see *doc/README.SPL*.

6.2 Modifying files for a new board

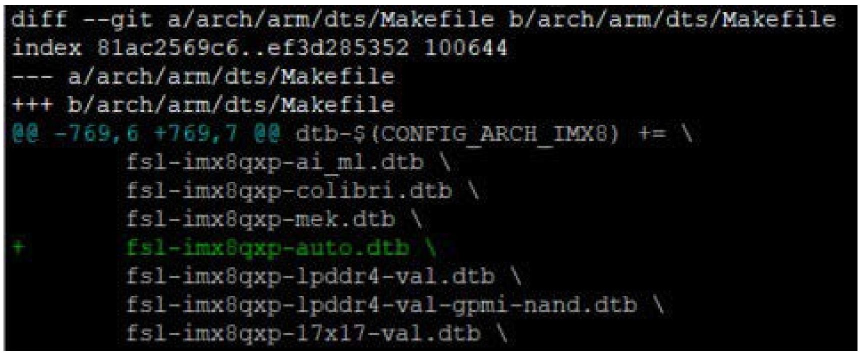
To modify files for a new board, perform the following steps:

1. Modify a few existing files to include new board in U-Boot.
   a. In *arch/arm/mach-imx/imx8/Kconfig*, add `TARGET_IMX8QXP_AUTO` and the `Kconfig` file of the auto board.



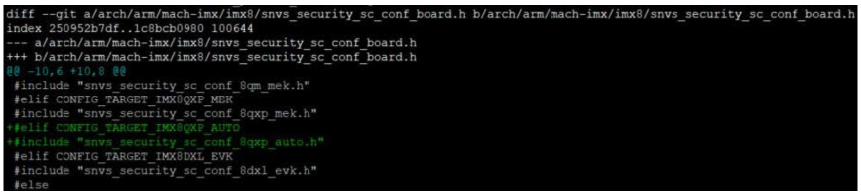
```
diff --git a/arch/arm/mach-imx/imx8/Kconfig b/arch/arm/mach-imx/imx8/Kconfig
index 52387462c0..45529b2796 100644
--- a/arch/arm/mach-imx/imx8/Kconfig
+++ b/arch/arm/mach-imx/imx8/Kconfig
@@ -138,6 +138,11 @@ config TARGET_IMX8QXP_MEK
        select BOARD_LATE_INIT
        select IMX8QXP

+config TARGET_IMX8QXP_AUTO
+       bool "Support i.MX8QXP AUTO board"
+       select BOARD_LATE_INIT
+       select IMX8QXP
+
 config TARGET_IMX8QXP_LPDDR4_VAL
        bool "Support i.MX8QXP lpddr4 validation board"
        select BOARD_LATE_INIT
@@ -178,6 +183,7 @@ endchoice

 source "board/freescale/imx8qm_mek/Kconfig"
 source "board/freescale/imx8qxp_mek/Kconfig"
+source "board/freescale/imx8qxp_auto/Kconfig"
 source "board/freescale/imx8qm_val/Kconfig"
 source "board/freescale/imx8qxp_val/Kconfig"
 source "board/freescale/imx8dxl_phantom_mek/Kconfig"
```

   b. In *arch/arm/dts/Makefile*, add *dtb* file for auto board.



```
diff --git a/arch/arm/dts/Makefile b/arch/arm/dts/Makefile
index 81ac2569c6..ef3d285352 100644
--- a/arch/arm/dts/Makefile
+++ b/arch/arm/dts/Makefile
@@ -769,6 +769,7 @@ dtb-$(CONFIG_ARCH_IMX8) += \
        fsl-imx8qxp-ai_ml.dtb \
        fsl-imx8qxp-colibri.dtb \
        fsl-imx8qxp-mek.dtb \
+       fsl-imx8qxp-auto.dtb \
        fsl-imx8qxp-lpddr4-val.dtb \
        fsl-imx8qxp-lpddr4-val-gpmi-nand.dtb \
        fsl-imx8qxp-17x17-val.dtb \
```

c. In *arch/arm/mach-imx/imx8/snvs_security_sc_conf_board.h*, add `include` for the `snvs` security configuration header file of the auto board.



```
diff --git a/arch/arm/mach-imx/imx8/snvs_security_sc_conf_board.h b/arch/arm/mach-imx/imx8/snvs_security_sc_conf_board.h
index 250952b7df..1c8bcb0980 100644
--- a/arch/arm/mach-imx/imx8/snvs_security_sc_conf_board.h
+++ b/arch/arm/mach-imx/imx8/snvs_security_sc_conf_board.h
@@ -10,6 +10,8 @@
 #include "snvs_security_sc_conf_8qm_mek.h"
 #elif CONFIG_TARGET_IMX8QXP_MEK
 #include "snvs_security_sc_conf_8qxp_mek.h"
+#elif CONFIG_TARGET_IMX8QXP_AUTO
+#include "snvs_security_sc_conf_8qxp_auto.h"
 #elif CONFIG_TARGET_IMX8DXL_EVK
 #include "snvs_security_sc_conf_8dxl_evk.h"
 #else
```

2. Make necessary modifications in those created files. In some files, the modification effort is minor, such as, changing board names and file path. This document does not show all detailed modifications but focuses on aspects that users usually need more attention and consideration for their own board implementation.

   a. In *configs/imx8qxp_auto_defconfig*
   The *defconfig* file is an important configuration file during compilation, which defines modules that the board requires.
   Taking our auto as example, comparing to the `defconfig` file of the MEK board. Make the following changes in the `defconfig` file of auto board.

```
-CONFIG_IMX_CONTAINER_CFG="board/freescale/imx8qxp_mek/uboot-
container.cfg"
-CONFIG_TARGET_IMX8QXP_MEK=y
-CONFIG_SYS_EXTRA_OPTIONS="IMX_CONFIG=board/freescale/imx8qxp_mek/
imximage.cfg"
-CONFIG_DEFAULT_DEVICE_TREE="fsl-imx8qxp-mek"
+CONFIG_IMX_CONTAINER_CFG="board/freescale/imx8qxp_auto/uboot-
container.cfg"
+CONFIG_TARGET_IMX8QXP_AUTO=y
+CONFIG_SYS_EXTRA_OPTIONS="IMX_CONFIG=board/freescale/imx8qxp_auto/
imximage.cfg"
+CONFIG_DEFAULT_DEVICE_TREE="fsl-imx8qxp-auto"
```

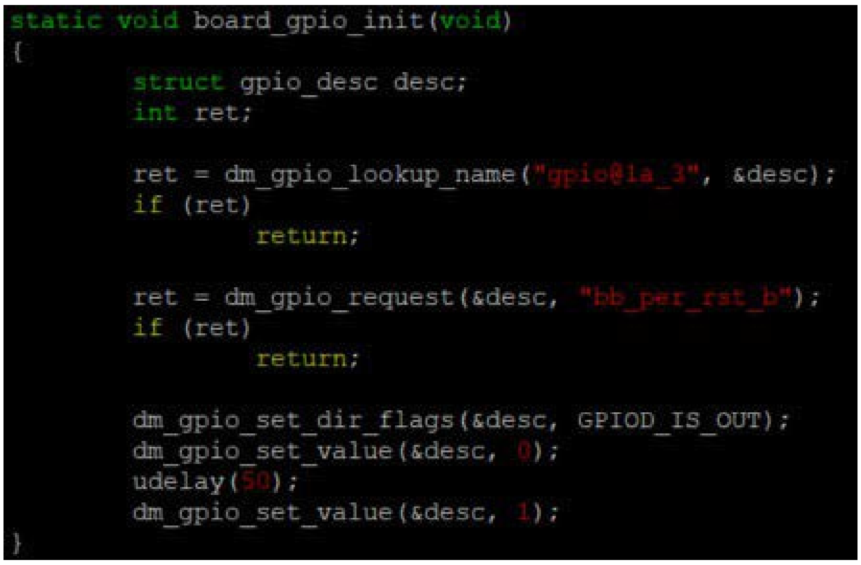
   Besides these changes, there are also changes related to board design or requirement.
   For example, on our auto board, the USB 3.0 port is only used as Host mode. Remove its support for Gadget mode in *defconfig* as below:

```
-CONFIG_USB_CDNS3_GADGET=y
+#CONFIG_USB_CDNS3_GADGET=y
```

   Note: *By default, the U-Boot reserves 128 MB memory region for M4 program to use, from `0x88000000` to `0x90000000`. This value is defined by `CONFIG_BOOTAUX_RESERVED_MEM_BASE` and `CONFIG_BOOTAUX_RESERVED_MEM_SIZE` in `defconfig`. To change the size of memory region for M4 program to use, modify this CONFIG and the memory region assignment in Section 3.2.*
   For features and drivers to enable on their new board, add related CONFIG to the `defconfig` file of the board.

   b. In *board/freescale/imx8qxp_auto/imx8qxp_auto.c*

This file contains board-related initialization functions like `board_init()`. Users can implement their own board-specific initialization function here or modify existing initialization function for various modules.
For example, if users have GPIO pins to be set during initialization, add these GPIO pins in `board_gpio_init()` function. Then users must use `dm_gpio_lookup_name()` to find the target GPIO, use `dm_gpio_request()` to request the GPIO, use `dm_gpio_set_dir_flags()` to set direction and flags of the GPIO pin, and finally use `dm_gpio_set_value()` to set the output value of target GPIO.



```
static void board_gpio_init(void)
{
        struct gpio_desc desc;
        int ret;

        ret = dm_gpio_lookup_name("gpio@1a_3", &desc);
        if (ret)
                return;

        ret = dm_gpio_request(&desc, "bb_per_rst_b");
        if (ret)
                return;

        dm_gpio_set_dir_flags(&desc, GPIOD_IS_OUT);
        dm_gpio_set_value(&desc, 0);
        udelay(50);
        dm_gpio_set_value(&desc, 1);
}
```

c. In *include/configs/imx8qxp_auto.h*
This header file of board contains many board-related macro definitions. Two most common parts to be modified are DRAM size and ENV settings.
The DRAM size is defined by:

```
#define PHYS_SDRAM_1 0x80000000
#define PHYS_SDRAM_2 0x880000000
#define PHYS_SDRAM_1_SIZE 0x80000000 /* 2 GB */
#define PHYS_SDRAM_2_SIZE 0x40000000 /* 1 GB */
```

The `PHYS_SDRAM_1` defines lower base address of DRAM, and `PHYS_SDRAM_2` defines higher base address. The total DRAM size is `PHYS_SDRAM_1_SIZE + PHYS_SDRAM_2_SIZE`.
If the DRAM size is less than 2 GB, then `PHYS_SDRAM_2_SIZE` is 0.
Users must modify the define value of `PHYS_SDRAM_1_SIZE` and `PHYS_SDRAM_2_SIZE` according to the DDR device on their board.
As for ENV settings, there are many ENV settings already been defined in the header file. Users can add new ENV settings in `CONFIG_EXTRA_ENV_SETTINGS`, and modify existing ENV settings, such as, `fdt_file` and `mmcargs` depending on their own requirement.
The ENV settings can also be changed dynamically in U-Boot console with `setenv` and `saveenv` command.

d. In *arch/arm/dts/fsl-imx8qxp-auto.dts*
The device tree architecture is first introduced in Linux kernel and implemented in U-Boot. The `dts` file is a data structure describing the hardware components of the board so that the system can use and manage those components.
Users can add or delete device node according to their board design to modify parameters of some device nodes, such as, clock frequency and control pins.

For SPL, there is also a device tree. To reduce the size of SPL, keep only the nodes with pre-relocation properties (`u-boot, dm-pre-reloc`, `u-boot, dm-spl`) in their device trees. Users can check `arch/arm/dts/fsl-imx8qxp-mek-u- boot.dtsi` as an example.

6.3 Compiling U-Boot

1. Set building environment
   The toolchain used to compile U-Boot is same cross-compile toolchain in Compiling ATF. For how to generate and install the toolchain, see **Chapter 4.5.12 How to build U-Boot and Kernel** in *i.MX Linux User's Guide* (document IMXLUG).
   https://www.nxp.com/webapp/Download?colCode=L5.4.47_2.2.0_LINUX_DOCS
2. Compile the code
   To build the U-Boot for the target board, perform the following steps:
   a. Use the following command to generate configuration file for the board. The *imx8qxp_auto* board is used as an example.
   ```
   $ make imx8qxp_auto_defconfig
   ```
   b. Use the following command to generate U-Boot for the target board. The SPL image is generated if CONFIG_SPL is selected in the configuration file.
   ```
   $ make -j8
   ```

The U-Boot image is put in *uboot-imx/u-boot.bin* and SPL image is in *uboot-imx/spl/u-boot-spl.bin*.

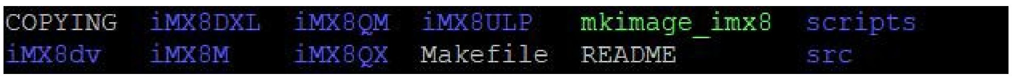
7 Building flash.bin image

For i.MX8 chips, the `flash.bin` image, which is actually the boot image container set, can include SCFW, SECO FW, M4 image, ATF image, U-Boot image, and SPL image. ROM code reads the `flash.bin` image from the boot device and loads it to different memory address according to settings.

To simplify the process of generating flash.bin image, use the `imx-mkimage` tool to combine the images above to produce the final `flash.bin` boot image and burn to the boot device.

7.1 Copying images to mkimage

After following steps in Section 1.3, download the `imx-mkimage` source code in the *imx-mkimage* folder. The content includes folders for all supported i.MX8 chips.


```
COPYING  iMX8DXL  iMX8QM  iMX8ULP   mkimage_imx8  scripts
iMX8dv   iMX8M    iMX8QX  Makefile  README        src
```

To generate the `flash.bin` image, copy all required images into the folder of the target chip. Taking our imx8qxp auto board as an example, follow the steps as below.

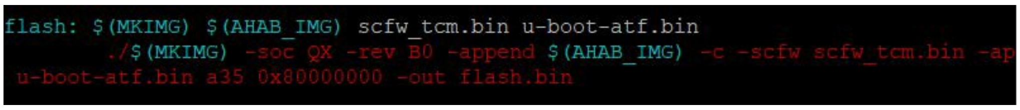
1. Copy SCFW `scfw_tcm.bin` generated in Section 3.3 to the *imx-mkimage/iMX8QX/* folder.
2. Copy SECO FW `mx8qxc0-ahab-container.img` to the *imx-mkimage/iMX8QX/* folder. Use the following command to download the image:
   ```
   $wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/imx-seco-3.7.1.bin
   $chmod a+x imx-seco-3.7.1.bin
   $cd imx-seco-3.7.1/firmware/seco/
   ```
   For each BSP release, there is an SECO FW version coupled with the release. For how to get the correct SECO FW version, see *i.MX Linux Release Notes* (document IMXLXRN).
3. Copy ATF image `bl31.bin` generated in Section 5.3 to the *imx-mkimage/iMX8QX/* folder.

4. Copy U-Boot image `u-boot.bin` and SPL image `u-boot-spl.bin` generated in Section 6.3 to the *imx-mkimage/ iMX8QX/* folder.
5. Copy M4 image to the *imx-mkimage/iMX8QX/* folder, if there is M4 program, and name it as *m4_image.bin*. For how to compile M4 images, see MCUXpresso SDK Builder.

7.2 Checking makefile

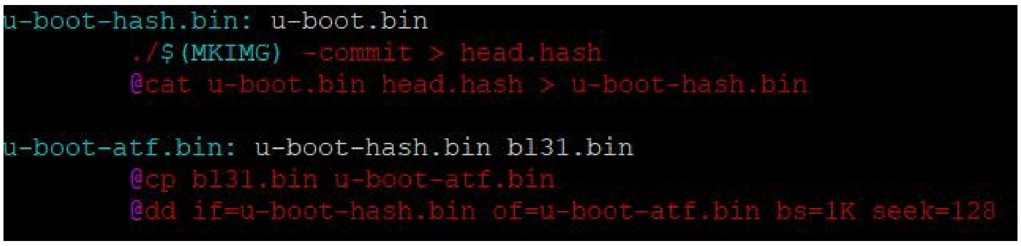
The makefile for `flash.bin` image is in *imx-mkimage/iMX8QX/soc.mak*. By default, many targets have been defined for common use case. If the target meets their requirement, user can use these targets to build their `flash.bin` image directly, or modify the options in these targets, or create a target for their specific need. The below are some examples.

- flash



```
flash: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf.bin
        ./$(MKIMG) -soc QX -rev B0 -append $(AHAB_IMG) -c -scfw scfw_tcm.bin -ap
 u-boot-atf.bin a35 0x80000000 -out flash.bin
```

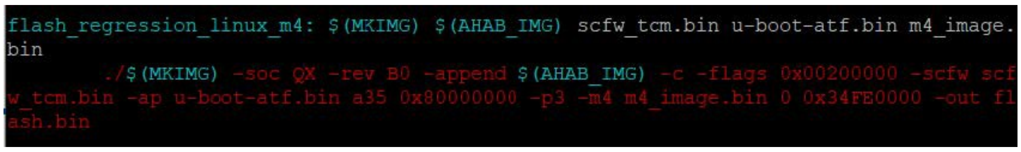
  The flash target includes SCFW, SECO FW, ATF, and U-Boot image. The ATF image `bl31.bin` combined with `u-boot.bin` image generate the *u-boot-atf.bin*.



```
u-boot-hash.bin: u-boot.bin
        ./$(MKIMG) -commit > head.hash
        @cat u-boot.bin head.hash > u-boot-hash.bin

u-boot-atf.bin: u-boot-hash.bin bl31.bin
        @cp bl31.bin u-boot-atf.bin
        @dd if=u-boot-hash.bin of=u-boot-atf.bin bs=1K seek=128
```

  Since the flash target does not include SPL image, the **A** core booting address is `0x80000000`, as shown above.

- `flash_regression_linux_m4`



```
flash_regression_linux_m4: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf.bin m4_image.
bin
        ./$(MKIMG) -soc QX -rev B0 -append $(AHAB_IMG) -c -flags 0x00200000 -scfw scf
w_tcm.bin -ap u-boot-atf.bin a35 0x80000000 -p3 -m4 m4_image.bin 0 0x34FE0000 -out fl
ash.bin
```

  The `flash_regression_linux_m4` target adds M4 image comparing to the flash target. Take care of the following two parts for this target.
  - The `-flags 0x00200000` option
    This option is the boot flag that is passed to SCFW during boot. The definition of the flags can be found in *sc_fw_port.pdf*.

Table 3. Flag definition

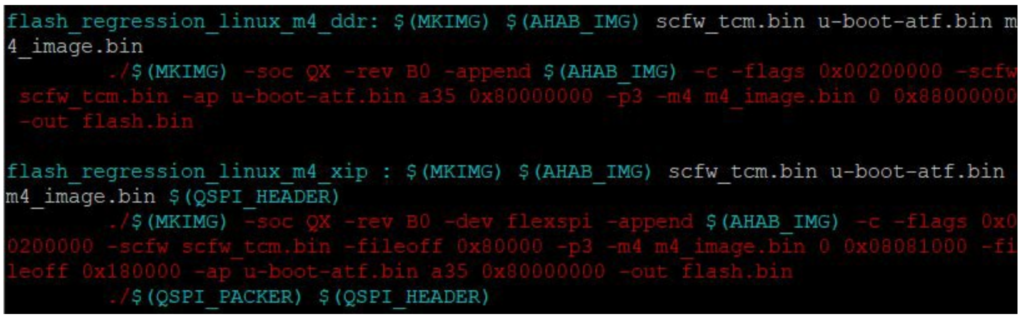
Flag	Bit	Meaning
`SC_BD_FLAGS_NOT_SECURE`	16	Initial boot partition is not secure
`SC_BD_FLAGS_NOT_ISOLATED`	17	Initial boot partition is not isolated
`SC_BD_FLAGS_RESTRICTED`	18	Initial boot partition is restricted
`SC_BD_FLAGS_GRANT`	19	Initial boot partition grants access to the SCFW
`SC_BD_FLAGS_NOT_COHERENT`	20	Initial boot partition is not coherent
`SC_BD_FLAGS_ALT_CONFIG`	21	Alternate SCFW configuration (passed to `board.c`)
`SC_BD_FLAGS_EARLY_CPU_START`	22	Start some CPUs early
`SC_BD_FLAGS_DDRTEST`	23	Configured for DDR stress test

Table 3. Flag definition...*continued*

Flag	Bit	Meaning
`SC_BD_FLAGS_NO_AP`	24	Do not boot AP even if requested by ROM

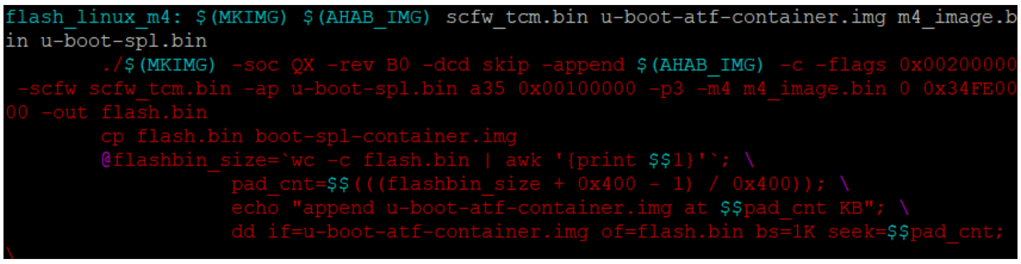
As shown in Table 3, `-flags 0x00200000` means that `SC_BD_FLAGS_ALT_CONFIG` is set. This flag is used in the *board_system_config()* function in *board.c*. SCU creates partition and assigns resources for M4 only when this flag is set.

– The booting address of M4 core
  Based on the user requirement, M4 core can boot from internal memory TCM, external memory DRAM, or external device like NOR flash. When compiling the M4 image with M4 SDK, there are specific link files for each method. Users must choose the correct link files according to the boot method they have chosen and align the booting address here with the address defined in the link files.
  - For the `flash_regession_linux_m4` target, assume that the boot method is booting from TCM. Therefore, the booting address is `0x34FE0000`.
  - For targets ending with `m4_ddr`, assume that the boot method is booting from DDR and the booting address is `0x88000000`.
  - For targets ending with `m4_xip`, assume that the boot method is booting from QSPI NOR flash and the booting address is `0x08081000`, as shown below.



```
flash_regression_linux_m4_ddr: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf.bin m
4_image.bin
	./$(MKIMG) -soc QX -rev B0 -append $(AHAB_IMG) -c -flags 0x00200000 -scfw
 scfw_tcm.bin -ap u-boot-atf.bin a35 0x80000000 -p3 -m4 m4_image.bin 0 0x88000000
 -out flash.bin

flash_regression_linux_m4_xip : $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf.bin
m4_image.bin $(QSPI_HEADER)
	./$(MKIMG) -soc QX -rev B0 -dev flexspi -append $(AHAB_IMG) -c -flags 0x0
0200000 -scfw scfw_tcm.bin -fileoff 0x80000 -p3 -m4 m4_image.bin 0 0x08081000 -fi
leoff 0x180000 -ap u-boot-atf.bin a35 0x80000000 -out flash.bin
	./$(QSPI_PACKER) $(QSPI_HEADER)
```

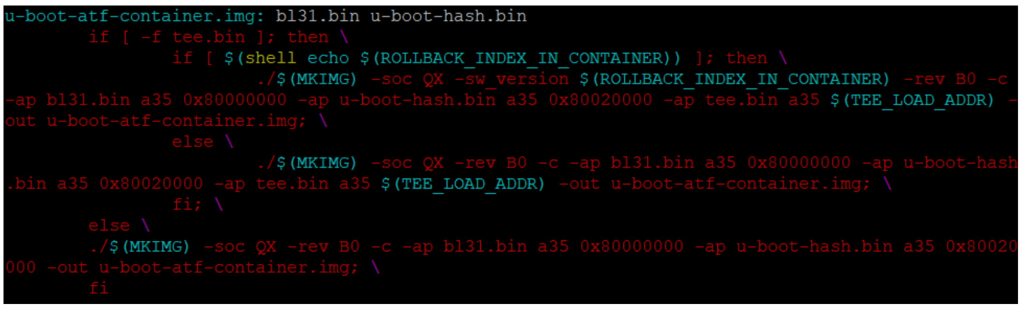
- `flash_linux_m4`



```
flash_linux_m4: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf-container.img m4_image.b
in u-boot-spl.bin
	./$(MKIMG) -soc QX -rev B0 -dcd skip -append $(AHAB_IMG) -c -flags 0x00200000
 -scfw scfw_tcm.bin -ap u-boot-spl.bin a35 0x00100000 -p3 -m4 m4_image.bin 0 0x34FE00
00 -out flash.bin
	cp flash.bin boot-spl-container.img
	@flashbin_size=`wc -c flash.bin | awk '{print $$1}'`; \
		pad_cnt=$$(((flashbin_size + 0x400 - 1) / 0x400)); \
		echo "append u-boot-atf-container.img at $$pad_cnt KB"; \
		dd if=u-boot-atf-container.img of=flash.bin bs=1K seek=$$pad_cnt;
\
```

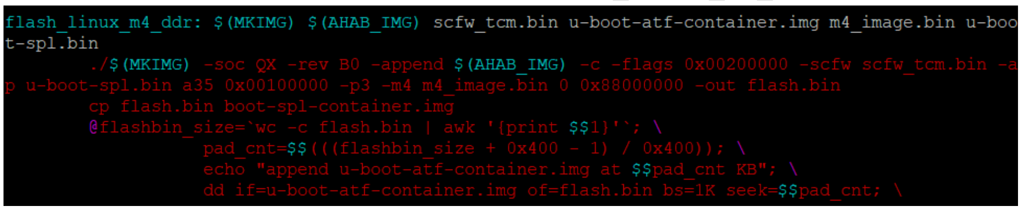
The difference between `flash_linux_m4` and `flash_regession_linux_m4` is that SPL image is added. For SPL boot, the ROM code only loads SPL image to OCRAM. After SPL image boots from OCRAM, it tries to read remained image (such as, `u-boot-aft-container.img`) from boot device and load to DDR. Take care of the following three parts for this target.

– The `u-boot-aft-container.img` image
  This image is generated from ATF image `bl31.bin` and U-Boot image `u-boot-hash.bin` as below. If it exists in the folder, it also includes TEE image `tee.bin`.



```
u-boot-atf-container.img: bl31.bin u-boot-hash.bin
        if [ -f tee.bin ]; then \
                if [ $(shell echo $(ROLLBACK_INDEX_IN_CONTAINER)) ]; then \
                        ./$(MKIMG) -soc QX -sw_version $(ROLLBACK_INDEX_IN_CONTAINER) -rev B0 -c
-ap bl31.bin a35 0x80000000 -ap u-boot-hash.bin a35 0x80020000 -ap tee.bin a35 $(TEE_LOAD_ADDR) -
out u-boot-atf-container.img; \
                else \
                        ./$(MKIMG) -soc QX -rev B0 -c -ap bl31.bin a35 0x80000000 -ap u-boot-hash
.bin a35 0x80020000 -ap tee.bin a35 $(TEE_LOAD_ADDR) -out u-boot-atf-container.img; \
                fi; \
        else \
        ./$(MKIMG) -soc QX -rev B0 -c -ap bl31.bin a35 0x80000000 -ap u-boot-hash.bin a35 0x80020
000 -out u-boot-atf-container.img; \
        fi
```

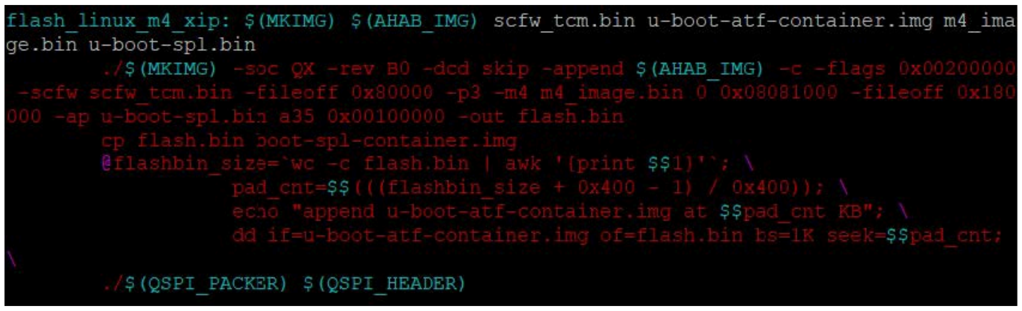
– The `-dcd` skip option
  In non-SPL boot method, the ROM code loads A core booting image from boot device to DRAM, so the ROM code must initialize DDR before loading. But in SPL boot method, the ROM code only loads SPL image into OCRAM. Therefore, the DDR initialization can be skipped in ROM code and be done later in SCFW. The `-dcd` skip option sets a flag in image container, so when ROM code read the image container, it knows how it configures.
  Note: *If the M4 must boot from DDR, ROM code still loads M4 image into DRAM. In such cases, the* `-dcd` *skip is not applicable, as shown below in the* `flash_linux_m4_ddr` *target.*



```
flash_linux_m4_ddr: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf-container.img m4_image.bin u-boo
t-spl.bin
        ./$(MKIMG) -soc QX -rev B0 -append $(AHAB_IMG) -c -flags 0x00200000 -scfw scfw_tcm.bin -a
p u-boot-spl.bin a35 0x00100000 -p3 -m4 m4_image.bin 0 0x88000000 -out flash.bin
        cp flash.bin boot-spl-container.img
        @flashbin_size=`wc -c flash.bin | awk '{print $$1}'`; \
                   pad_cnt=$$(((flashbin_size + 0x400 - 1) / 0x400)); \
                   echo "append u-boot-atf-container.img at $$pad_cnt KB"; \
                   dd if=u-boot-atf-container.img of=flash.bin bs=1K seek=$$pad_cnt; \
```

  For more details about `-dcd` skip option and DDR initialization flow, see **Chapter 4.6 DDR Configuration** in *sc_fw_port.pdf*.
– The booting address of A core
  In SPL boot method, since the ROM code loads SPL image into OCRAM, the booting address of A core also changes to OCRAM address `0x00100000`.

• `flash_linux_m4_xip`



```
flash_linux_m4_xip: $(MKIMG) $(AHAB_IMG) scfw_tcm.bin u-boot-atf-container.img m4_ima
ge.bin u-boot-spl.bin
        ./$(MKIMG) -soc QX -rev B0 -dcd skip -append $(AHAB_IMG) -c -flags 0x00200000
 -scfw scfw_tcm.bin -fileoff 0x80000 -p3 -m4 m4_image.bin 0 0x08081000 -fileoff 0x180
000 -ap u-boot-spl.bin a35 0x00100000 -out flash.bin
        cp flash.bin boot-spl-container.img
        @flashbin_size=`wc -c flash.bin | awk '{print $$1}'`; \
                pad_cnt=$$(((flashbin_size + 0x400 - 1) / 0x400)); \
                echo "append u-boot-atf-container.img at $$pad_cnt KB"; \
                dd if=u-boot-atf-container.img of=flash.bin bs=1K seek=$$pad_cnt;
\

        ./$(QSPI_PACKER) $(QSPI_HEADER)
```

The main change between the `flash_linux_m4_xip` and `flash_linux_m4` is that M4 is booting from QSPI NOR FLASH device instead of TCM. Besides the M4 core booting address mentioned above, take care of the QSPI header file.
For flash.bin image to boot from QSPI/FSPI device, the header file is needed in the image for ROM code to configure QSPI/FSPI device. The sample QSPI/FSPI header file is provided in *imx-mkimage/scripts/fspi_header*. To choose QSPI/FSPI device as boot device, modify the header file to fit the devices.
For example, in the sample header file, we can see that the value for offset `0x44-0x47` is `0x01010200`.

```
01010200 /* Serial Nor, Single/Dual/Quad/Octal, SerialClkFreq 1 - 20MHz, 2 -
 50MHz... */
```

To translate this setting, see **Chapter 5.8.3.3 FlexSPI configuration parameters** in *i.MX 8QuadMax Applications Processor Reference Manual* (document IMX8QMRM).

Table 4. FlexSPI configuration parameters

`devicetype`	`0x044`	1	1 - Serial NOR
`sflashPadType`	`0x045`	j	1 - Single pad
			2 - Dual pads
			4 - Quad pads
			8 - Octal pads
`serialClkFreq`	`0x046`	1	Chip-specific value, for this silicon
			1 - 20 MHz
			2 - 50 MHz
			3 - 62 MHz for SDR and 200 MHz for DDR
			4 - 80 MHz
			5 - 100 MHz
			6 - 133 MHz
			7 - 166 MHz for SDR and 200 MHz for DDR
			Other values: 20 MHz
`IutCustomSeqEnable`	0x047	1	0 - Use predefined LUT sequence index and number
			1 - Use LUT sequence parameters provided in this block

Here we can see that, in the sample header file,
– The value of `deviceType` is `0x01`, which is Serial NOR.
– The value of `sflashPadType` is `0x1`, which is Single pad.
– The value of `serialClkFreq` is `0x02`, which is 50 MHz.
– The value of `lutCustomSeqEnable` is `0x00`, which is Use pre-defined LUT sequence index and number.
For all parameters in the header file, check the definition in **Chapter 5.8.3.3 FlexSPI configuration parameters** in *i.MX 8QuadMax Applications Processor Reference Manual* (document IMX8QMRM) and set the correct value for the FSPI device.

7.3 Generating flash.bin image

To generate flash.bin image, compile an internal tool, `mkimage_imx8`, using the `gcc` in the Linux host environment of the user. The source code of `mkimage_imx8` is in *imx-mkimage/src*.

Since it is automatically compiled when using the following command to generate flash.bin image, there is no need to compile *mkimage_imx8* separately.

The command to generate `flash.bin` image is:

```
$make SOC=<SOC_TARGET> REV=<SOC_REV> [TARGET]
```

The `SOC_TARGET` is the target chips, `SOC_REV` is the reversion of chips, and TARGET is introduced in Section 7.2.

Note:

For iMX8QXP B0 and C0 chips, the SECO FW is not compatible with each other. Therefore, the REV is used to specify the version of SECO FW.

- *For iMX8QXP B0 chips, use the SECO FW, `mx8qxb0-ahab-container.img`.*

- *For iMX8QXP C0 chips, use the SECO FW, `mx8qxc0-ahab-container.img`.*

Taking our `imx8qxp` auto board as an example, to build a `flash.bin` image with M4 image and SPL image, use the command:

```
$make SOC=iMX8QX REV=C0 flash_linux_m4
```

The `flash.bin` binary file is generated in *imx-mkimage/iMX8QX/flash.bin*.

7.4 Burning flash.bin images

For i.MX8/8X chips, the ROM code supports to boot from following boot devices:

- SD/MMC
- NAND FLASH
- FlexSPI NOR flash
- Serial downloader support on USB 2.0 OTG and USB 3.0 (as 2.0)

Usually SD card is the most efficient way to verify if the generated `flash.bin` image can boot the board successfully.

To burn `flash.bin` image into SD card, insert the SD card on a Linux host PC. Assuming the SD card is recognized as */dev/sdx*, use the following command to burn flash.bin into SD card:

```
$sudo umount /dev/sdx*
$sudo dd if=flash.bin of=/dev/sdx bs=1k seek=32 conv=fsync && sync
```

For how to burn flash.bin image into other boot device, see **Chapter 4.4 Downloading images** in *i.MX Linux User's Guide* (document IMXLUG).

8 Linux kernel porting

Comparing to previous chapters, users are more familiar to Linux kernel porting. This chapter focuses on the porting effort for our `imx8qxp` auto reference board to illustrate the process of adding new device tree file and drivers for a new customized board.

For Linux file system porting such as adding new user space tools or package, see *i.MX_Yocto_Project_User's_Guide.pdf*.

8.1 Creating files for a new board

To create files for a new board, perform the following steps:

1. Add a board *defconfig* in *arch/arm64/configs/*.
   The *defconfig* file defines the component that is included in the Linux kernel. In general, each board has a specific *defconfig* file according to the hardware design and software requirement.
   The imx8qxp auto board, it is similar to the MEK reference board. Therefore, use the default *defconfig* file in *arch/ arm64/configs/imx_v8_defconfig* directly as the *defconfig* file. For other iMX8 customized boards, use the *imx_v8_defconfig* as reference and add modification accordingly.
2. Add a board *dts* in *arch/arm64/boot/dts/freescale/*.
   For the imx8qxp auto board, create the following *dts/dtsi* files, which are copied from files for MEK board to save the efforts.

Table 5. Device tree files created for auto board in Linux kernel

File location	Description
arch/arm64/boot/dts/freescale/imx8qxp-auto.dts	The *dts* files for auto board without *rpmsg*, which includes *imx8qxp.dtsi* and *imx8x-auto.dtsi*.
arch/arm64/boot/dts/freescale/imx8qxp-auto-rpmsg.dts	The *dts* file for auto board with *rpmsg*, which includes *imx8qxp-auto.dts* and *imx8x-auto-rpmsg.dtsi*.
arch/arm64/boot/dts/freescale/imx8x-auto.dtsi	The *dtsi* file for auto board without *rpmsg*, which enables the device node that needed.
arch/arm64/boot/dts/freescale/imx8x-auto-rpmsg.dtsi	The *dtsi* file for auto board with *rpmsg*, which enables the device node that needed.
arch/arm64/boot/dts/freescale/imx8qxp-auto-enet-tja1101.dts	The *dts* file for auto board with TJA1101 Ethernet PHY enabled, which includes *imx8qxp-auto.dts* and *imx8qxp- auto-enet-tja1101.dtsi*.
arch/arm64/boot/dts/freescale/imx8qxp-auto-enet-tja1101.dtsi	The *dtsi* file for auto board with TJA1101 Ethernet PHY enabled.

The difference between *dts* with and without *rpmsg* is on resources related to M4 core. If M4 core is used, assign some of the peripheral interface resources to M4 partition in SCFW as mentioned before, like I^2C and FlexCAN. Therefore, disable these resources in *dts* or use virtual driver like *imx_rpmsg_i2c* for I2C interface.

3. Add new drivers for devices on the board.
   For an imx8qxp auto board, add the following three hardware components in the board design:
   - MIPI-CSI with NVP6324 automotive AHD solution
   - LVDS display with TI DS90UB947/948 SerDes (through FPD-Link III) for automotive application
   - MIPI-DSI display with Maxim 96752/96755 SerDes (through GMSL2) for automotive application

   Table 6 lists the corresponding drivers to be added in the kernel code.

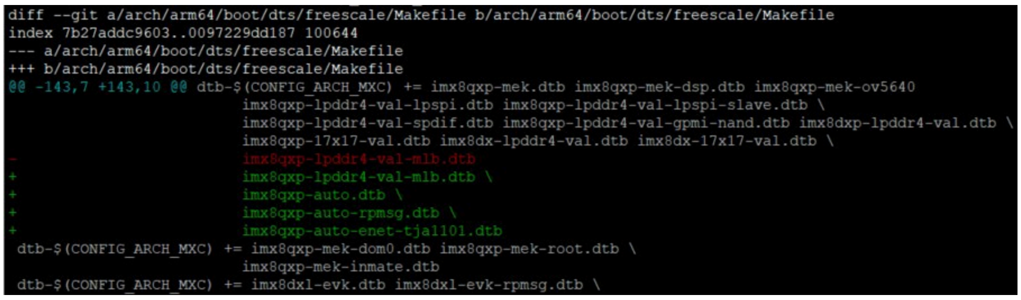
Table 6. Driver files added for auto board in Linux kernel

File location	Description
drivers/gpu/drm/bridge/ds90ub94x.c	Driver of TI DS90UB947/948 SerDes
drivers/gpu/drm/bridge/mx9675x.c	Driver of Maxim 96752/96755 SerDes
drivers/media/platform/imx8/nvp6324/	New folder for NVP6324
drivers/media/platform/imx8/nvp6324/Kconfig	Kconfig of NVP6324 driver
drivers/media/platform/imx8/nvp6324/Makefile	Makefile of NVP6324 driver
drivers/media/platform/imx8/nvp6324/nvp6324.h	Header file of NVP6324 driver
drivers/media/platform/imx8/nvp6324/nvp6324_core.c	Core function file of NVP6324 driver
drivers/media/platform/imx8/nvp6324/nvp6324_mipi.c	MIPI settings of NVP6324 driver
drivers/media/platform/imx8/nvp6324/nvp6324_video.c	Video mode settings of NVP6324 driver
drivers/media/platform/imx8/nvp6324/nvp6324_video_eq.c	Video event queue settings of NVP6324 driver

8.2 Modifying files for a new board

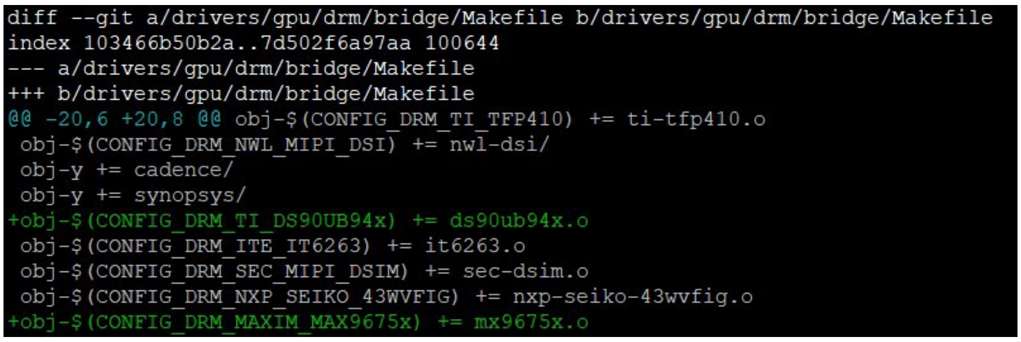
To modify files for a new board, perform the following steps:

1. Edit related makefiles to be compiled into kernel image. In the `imx8qxp` auto board example, the related makefiles include:
   - *arch/arm64/boot/dts/freescale/Makefile*
     Add *dtb* files of `imx8qxp` auto board.



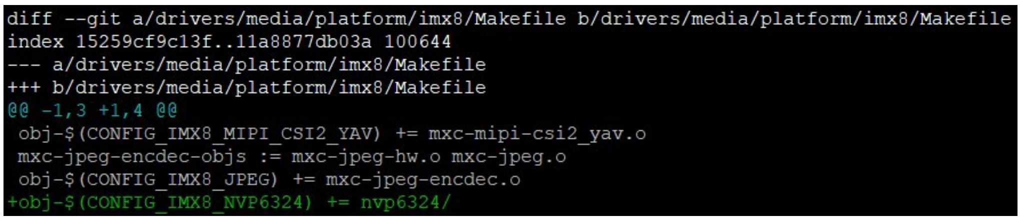
```
diff --git a/arch/arm64/boot/dts/freescale/Makefile b/arch/arm64/boot/dts/freescale/Makefile
index 7b27addc9603..0097229dd187 100644
--- a/arch/arm64/boot/dts/freescale/Makefile
+++ b/arch/arm64/boot/dts/freescale/Makefile
@@ -143,7 +143,10 @@ dtb-$(CONFIG_ARCH_MXC) += imx8qxp-mek.dtb imx8qxp-mek-dsp.dtb imx8qxp-mek-ov5640
                          imx8qxp-lpddr4-val-lpspi.dtb imx8qxp-lpddr4-val-lpspi-slave.dtb \
                          imx8qxp-lpddr4-val-spdif.dtb imx8qxp-lpddr4-val-gpmi-nand.dtb imx8dxp-lpddr4-val.dtb \
                          imx8qxp-17x17-val.dtb imx8dx-lpddr4-val.dtb imx8dx-17x17-val.dtb \
-                         imx8qxp-lpddr4-val-mlb.dtb
+                         imx8qxp-lpddr4-val-mlb.dtb \
+                         imx8qxp-auto.dtb \
+                         imx8qxp-auto-rpmsg.dtb \
+                         imx8qxp-auto-enet-tja1101.dtb
 dtb-$(CONFIG_ARCH_MXC) += imx8qxp-mek-dom0.dtb imx8qxp-mek-root.dtb \
                          imx8qxp-mek-inmate.dtb
 dtb-$(CONFIG_ARCH_MXC) += imx8dxl-evk.dtb imx8dxl-evk-rpmsg.dtb \
```

   - *drivers/gpu/drm/bridge/Makefile*
     Add `ds90ub94x` and `mx9675x` driver.



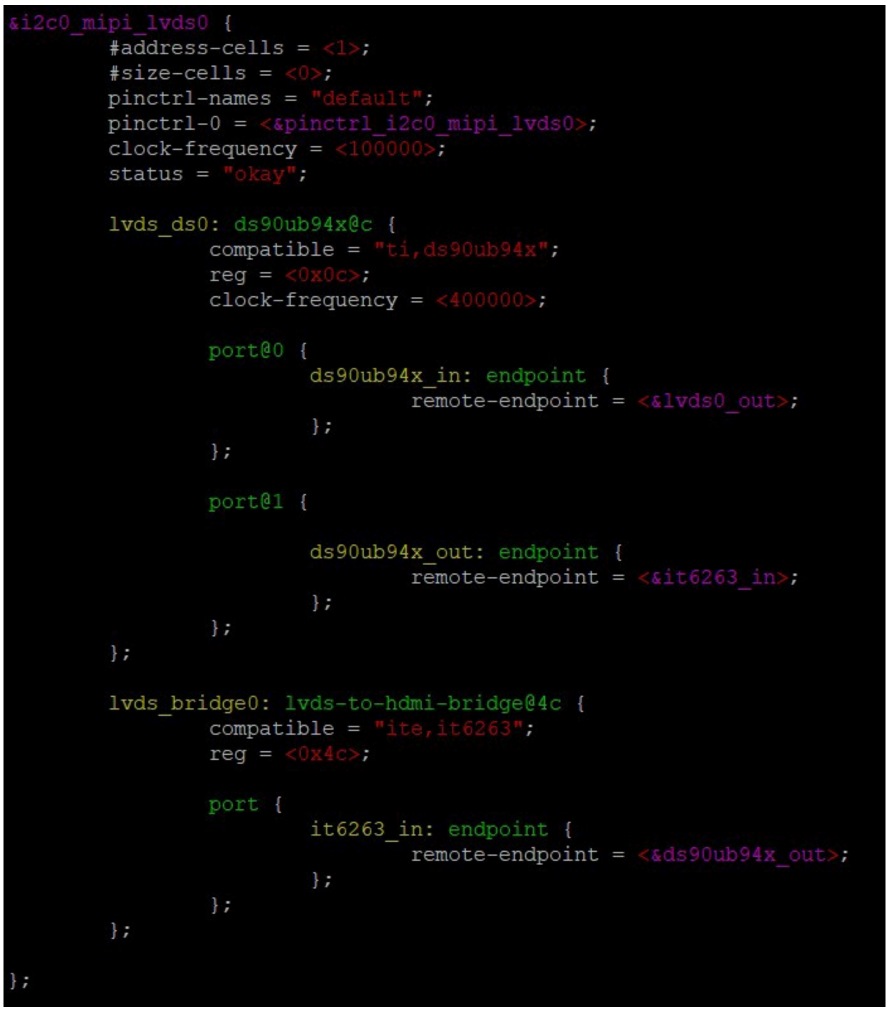
```
diff --git a/drivers/gpu/drm/bridge/Makefile b/drivers/gpu/drm/bridge/Makefile
index 103466b50b2a..7d502f6a97aa 100644
--- a/drivers/gpu/drm/bridge/Makefile
+++ b/drivers/gpu/drm/bridge/Makefile
@@ -20,6 +20,8 @@ obj-$(CONFIG_DRM_TI_TFP410) += ti-tfp410.o
 obj-$(CONFIG_DRM_NWL_MIPI_DSI) += nwl-dsi/
 obj-y += cadence/
 obj-y += synopsys/
+obj-$(CONFIG_DRM_TI_DS90UB94x) += ds90ub94x.o
 obj-$(CONFIG_DRM_ITE_IT6263) += it6263.o
 obj-$(CONFIG_DRM_SEC_MIPI_DSIM) += sec-dsim.o
 obj-$(CONFIG_DRM_NXP_SEIKO_43WVFIG) += nxp-seiko-43wvfig.o
+obj-$(CONFIG_DRM_MAXIM_MAX9675x) += mx9675x.o
```

   - *drivers/media/platform/imx8/Makefile*
     Include *nvp6324* folder for `nvp6324` driver.

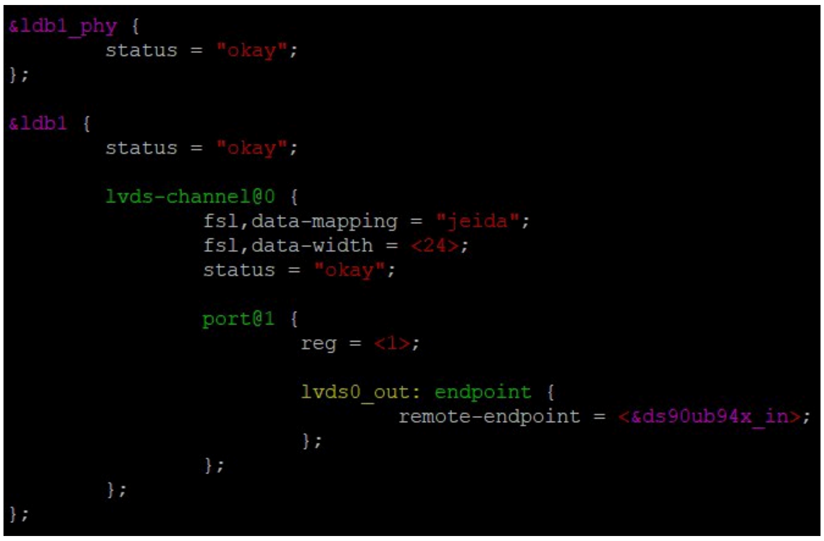


```
diff --git a/drivers/media/platform/imx8/Makefile b/drivers/media/platform/imx8/Makefile
index 15259cf9c13f..11a8877db03a 100644
--- a/drivers/media/platform/imx8/Makefile
+++ b/drivers/media/platform/imx8/Makefile
@@ -1,3 +1,4 @@
 obj-$(CONFIG_IMX8_MIPI_CSI2_YAV) += mxc-mipi-csi2_yav.o
 mxc-jpeg-encdec-objs := mxc-jpeg-hw.o mxc-jpeg.o
 obj-$(CONFIG_IMX8_JPEG) += mxc-jpeg-encdec.o
+obj-$(CONFIG_IMX8_NVP6324) += nvp6324/
```

2. Modify the *dts* files according to the board design. For example, on `imx8qxp` auto board, the connection on LVDS0 is designed in such method:
   ldb1 -> ds90ub947 -> ds90ub948 -> it6263 -> HDMI screen
   Therefore, the `ldb1` and `i2c0_mipi_lvds0` device node in *imx8x-auto.dtsi* is changed as below.



```
&i2c0_mipi_lvds0 {
        #address-cells = <1>;
        #size-cells = <0>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c0_mipi_lvds0>;
        clock-frequency = <100000>;
        status = "okay";

        lvds_ds0: ds90ub94x@c {
                compatible = "ti,ds90ub94x";
                reg = <0x0c>;
                clock-frequency = <400000>;

                port@0 {
                        ds90ub94x_in: endpoint {
                                remote-endpoint = <&lvds0_out>;
                        };
                };

                port@1 {

                        ds90ub94x_out: endpoint {
                                remote-endpoint = <&it6263_in>;
                        };
                };
        };

        lvds_bridge0: lvds-to-hdmi-bridge@4c {
                compatible = "ite,it6263";
                reg = <0x4c>;

                port {
                        it6263_in: endpoint {
                                remote-endpoint = <&ds90ub94x_out>;
                        };
                };
        };

};
```

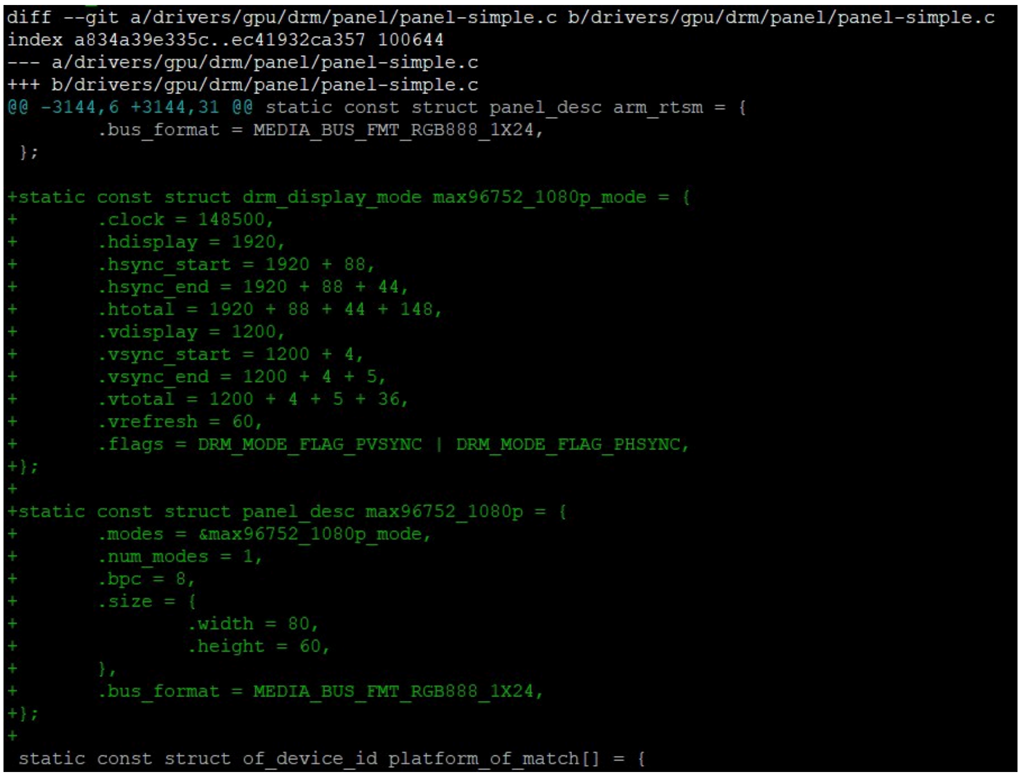


```
&ldb1_phy {
        status = "okay";
};

&ldb1 {
        status = "okay";

        lvds-channel@0 {
                fsl,data-mapping = "jeida";
                fsl,data-width = <24>;
                status = "okay";

                port@1 {
                        reg = <1>;

                        lvds0_out: endpoint {
                                remote-endpoint = <&ds90ub94x_in>;
                        };
                };
        };
};
```

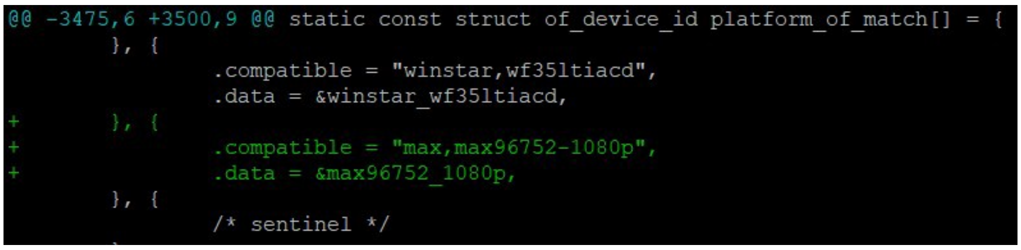
The endpoint matching in the *dts* represents the connection order.
For other board design changes, the related modification is also added in the *dts* in a similar method.

3. To add specific features or implementations for the new board, modify some existing drivers.

For example, in the `imx8qxp` auto board design, to connect an MIPI-DSI panel on Maxim 96755 SerDes, add the settings of this panel in *drivers/gpu/drm/panel/panel-simple.c* as below.



```
diff --git a/drivers/gpu/drm/panel/panel-simple.c b/drivers/gpu/drm/panel/panel-simple.c
index a834a39e335c..ec41932ca357 100644
--- a/drivers/gpu/drm/panel/panel-simple.c
+++ b/drivers/gpu/drm/panel/panel-simple.c
@@ -3144,6 +3144,31 @@ static const struct panel_desc arm_rtsm = {
        .bus_format = MEDIA_BUS_FMT_RGB888_1X24,
 };

+static const struct drm_display_mode max96752_1080p_mode = {
+       .clock = 148500,
+       .hdisplay = 1920,
+       .hsync_start = 1920 + 88,
+       .hsync_end = 1920 + 88 + 44,
+       .htotal = 1920 + 88 + 44 + 148,
+       .vdisplay = 1200,
+       .vsync_start = 1200 + 4,
+       .vsync_end = 1200 + 4 + 5,
+       .vtotal = 1200 + 4 + 5 + 36,
+       .vrefresh = 60,
+       .flags = DRM_MODE_FLAG_PVSYNC | DRM_MODE_FLAG_PHSYNC,
+};
+
+static const struct panel_desc max96752_1080p = {
+       .modes = &max96752_1080p_mode,
+       .num_modes = 1,
+       .bpc = 8,
+       .size = {
+               .width = 80,
+               .height = 60,
+       },
+       .bus_format = MEDIA_BUS_FMT_RGB888_1X24,
+};
+
 static const struct of_device_id platform_of_match[] = {
```



```
@@ -3475,6 +3500,9 @@ static const struct of_device_id platform_of_match[] = {
        }, {
                .compatible = "winstar,wf35ltiacd",
                .data = &winstar_wf35ltiacd,
+       }, {
+               .compatible = "max,max96752-1080p",
+               .data = &max96752_1080p,
        }, {
                /* sentinel */
```

8.3 Compiling Linux kernel

To compile Linux kernel, perform the following steps:

1. Set building environment.
   The toolchain used to compile Linux kernel is the same cross-compile toolchain used for compiling ATF and U-Boot. For how to generate and install the toolchain, see Chapter 4.5.12 How to build U-Boot and kernel in standalone environment in *i.MX Linux User's Guide* (document IMXLUG).
   https://www.nxp.com/webapp/Download?colCode=L5.4.47_2.2.0_LINUX_DOCS
2. Compile the code.
   To build the Linux kernel for the target board, perform as below:
   a. Use the following command to generate configuration file for the board. In this example, the default *defconfig* is used for the auto board.

```
$ make imx_v8_defconfig
```

b. Use the following command to generate Linux kernel for the target board. The related *dtb* files are also generated.

```
$ make -j8
```

c. The compiled Linux kernel image is *arch/arm64/boot/Image* and *dtb* files are located in *arch/arm64/boot/dts/ freescale/* folder.

8.4 Burning Linux kernel

- If the SD/MMC is already partitioned into bootpartition and `rootfs` partition, use the following command to copy kernel image and `dtb` files to bootpartition directly.

```
$sudo mount /dev/sdx1 /mnt/boot/
$sudo cp Image /mnt/boot/
$sudo cp imx8qxp-auto.dtb /mnt/boot/
$sudo umount /dev/sdx1
```

- If the SD/MMC is not partitioned yet, follow the step in **Chapter 4.3 Preparing an SD/MMC card to boot** of *i.MX Linux User's Guide* (document IMXLUG) to partition SD/MMC and then burn kernel image and *dtb* to bootpartition.

9 Revision history

Table 7 summarizes the revisions to this document.

Table 7. Revision history

Rev.	Date	Substantive changes
1	26 May 2023	Updated Section 1.3
0	10 June 2021	Initial release

10 Legal information

10.1 Definitions

10.2 Disclaimers

10.3 Trademarks

Contents

Date of release: 26 May 2023
Document identifier: AN13275