NXP Semiconductors
Application Note

Document Number: AN5317
Rev. 0, 08/2016

Loading Code on Cortex-M4 from Linux for the i.MX 6SoloX and i.MX 7Dual/7Solo Application Processors

1. Introduction

There are a growing number of embedded use cases which require concurrent execution of isolated and secure software environments. Multiple software execution environments are useful for:

• Off-loading tasks and improving real-time performance

• Increasing system integrity and security

• Optimizing Power consumption

The i.MX 7Dual/Solo and i.MX 6SoloX application processors offer Heterogeneous Multi-Core Processing (HMP) with both ARM® Cortex®-A processors and a Cortex-M4 micro-controller on a single SoC. The cores can be partitioned into two respective processing domains that can be programmed to run a different OS to cater for the real time latency and application processing requirements.

In some applications it is very useful to have the ARM Cortex-A processors re-load code onto the Cortex-M4 micro-controller. This application note provides the details of re-loading code on the Cortex-M4 from the Linux shell using the ARM Cortex-A7 processor and the ARM Cortex-A9 processor for i.MX 7Dual/Solo and i.MX 6SoloX SoC's respectively. The same method can be used for any other OS or bare metal implementation.

Contents

This application note provides the details of loading code on the Cortex-M4 using the Cortex-A7 processor and the Cortex-A9 processor for i.MX 7Dual/Solo and i.MX 6SoloX application processors respectively.

2. Overview of i.MX 7Dual/7Solo and i.MX 6SoloX Implementations

There are many similarities between the i.MX 7Dual/7Solo and i.MX 6SoloX application processors in terms of where the TCM_U and TCM_L (Tightly Coupled Memory - Upper/Lower) memories are located. However, the boot vector and the specific register to issue the platform reset and to reset the Cortex-M4 are different on the two application processors. Moreover, the bit locations inside the registers for both i.MX 7Dual/7Solo and i.MX 6SoloX application processors are also different.

The i.MX 7Dual/7Solo application processor provides a multicore solution of ARM Cortex-A7 cores (Dual or Single) and a single ARM Cortex-M4 core.

The i.MX 6SoloX application processor provides a single ARM Cortex-A9 and a single ARM Cortex-M4. The ARM Cortex-A7 on i.MX 7Dual/7Solo and the ARM Cortex-A9 on i.MX 6SoloX both are capable of booting using different interfaces and are also responsible for bringing up the different interfaces of the chip. It is the responsibility of Cortex-A7 on i.MX 7Dual/7Solo application processor and Cortex-A9 on i.MX 6SoloX application processor to enable the Cortex-M4 core.

The current U-Boot source code for both i.MX 7Dual/7Solo and i.MX 6SoloX application processors provides a bootaux and a fatload command that helps with loading the code to the Cortex-M4 core and bringing it up: For example "*fatload mmc 0:1 0x7f8000 sensor_demo.bin*" will load the file sensor_demo.bin to the location 0x7f8000 and "*bootaux 0x7f80000"* will boot the Cortex-M4 core with the image loaded at *0x007f_8000*.

Although this feature is useful to bring the Cortex-M4 core up as soon as possible after the boot up or power-on-reset, with the existing implementation every time a user wants to modify the application, the U-Boot would need to be reconfigured. Moreover, this implementation does not allow users to reload another application while a task is already running on the Cortex-M4 core. This application note will detail the registers that are required to be programmed to reload the application from the Linux shell.

For this application note, we will assume that the Cortex-M4 core is compiled to execute from the TCM_L and TCM_U on chip memories. We also assume that the Cortex-M4 clock is enabled. Users using U-Boot can enable the clock with the *bootaux* command by loading a primary image which on the Cortex-M4 side tells the Linux kernel not to disable the Cortex-M4 clock when the kernel takes over. If users are not running the U-Boot and want to enable the clock of the Cortex-M4 then refer to the respective clock chapters in the i.MX 7Dual Applications Processor Reference Manual (document IMX7DRM) and the i.MX 6SoloX Applications Processor Reference Manual (document IMX6SXRM)

3. Reloading Code on the i.MX 7Dual/7Solo

3.1. On-chip memory view from each ARM core on the i.MX 7Dual/7Solo

The memory view of different peripherals is different between the Cortex-A7 and the Cortex-M4 side. Table 1, below shows only the relevant memory areas for this application note. For more details, refer to the Memory Map chapter in the i.MX 7Dual Applications Processor Reference Manual (document IMX7DRM)

NOTE

On i.MX 7Dual/7Solo, the boot vector for the Cortex-M4 core is located at the **start of the OCRAM_S** (On Chip RAM - Secure) whose address is 0x0018_0000 from Cortex-A7.

Table 1. Start and end addresses of different memories from Cortex-A7 and Cortex-M4 side

Peripheral	Start Address Cortex-A7	End Address Cortex-A7	Start Address Cortex-M4 Side	End Address Cortex-M4 Side	Size
OCRAM_S	0x0018_0000	0x0018_7FFF	0x2018_0000	0x2018_7FFF	32 KB
TCM_L	0x007F_8000	0x007F_7FFF	0x1FFF_8000	0x1FFF_FFFF	32 KB
TCM_H	0x0080_0000	0x0080_7FFF	0x2000_0000	0x2000_7FFF	32 KB

3.2. Detailed procedure

In order to reload the code on the Cortex-M4 core using the Cortex-A7 processor on **i.MX 7Dual/7Solo** users have to follow the steps below:

1. Issue a software platform reset by setting up **SW_M4P_RST** (Bit 2) in the **SRC_M4RCR** (SRC_M4RCR[2]) register**.** Issuing a platform reset, resets the Cortex-M4 cores and associated memories. The address of SRC_M4RCR register is 0x3039_000C for i.MX 7Dual/7Solo SoC.
2. Load the code for the Cortex-M4 processor into the TCM_L memory. For this application, we assume that the Cortex-M4 code is compiled to execute from TCM_L memory. From Table 1, the TCM_L address from the Cortex-A7 side is 0x007F_8000, therefore users need to program the binary file generated by the FreeRTOS to that address.
3. Once the file has been loaded, the next step is to setup the Stack and PC pointer in the OCRAM_S memory, because after reset the processor uses the OCRAM_S start address (0x0018_0000) as the first instruction. For this implementation, the stack value is the first four bytes found in the binary file generated for the Cortex-M4 processor using FreeRTOS source. The PC value is also four bytes long and is located at an offset of 0x4 in the binary file. This PC value is written to the OCRAM_S base address plus four which for this platform is (0x0018_0004). Table 2 further clarifies the Stack and PC addresses.

Table 2. Showing boot vectors location for Cortex-M4 core

Boot Vectors	OCRAM_S location for boot vectors	Location of boot vectors in binary file
Stack Pointer	0x0018_0000	First four bytes
Program Counter	0x0018_0004	Four bytes after first four bytes

4. Once the startup address in the OCRAM_S have been adjusted according to the binary file and the file has been loaded into the memory. The next step is to set the **ENABLE_M4** (Bit 3) in the **SRC_M4RCR** (SRC_M4RCR[3]) register. Since the *bootaux* already booted a primary image, this bit should be 1. Performing a platform reset using **SW_M4P_RST** (Bit 2) in the **SRC_M4RCR** (SRC_M4RCR[2]) register does not clear this bit. The last step is to set the **SW_M4C_RST (**Bit 1) in the **SRC_M4RCR** (SRC_M4RCR[1]) register which will boot the new code on the Cortex-M4 processor.
5. Repeat steps 1-3 for reloading a new image. More details about the **SRC_M4RCR** register are shown in Figure 1.

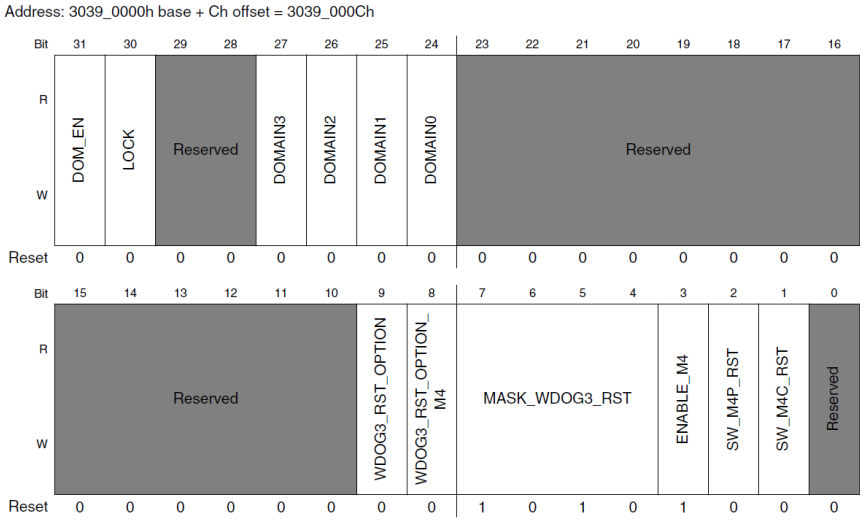

Figure 1. Cortex-M4 reset control register (SRC_M4RCR)

3.3. Steps for reloading code on the i.MX 7Dual/7Solo

A summary of the steps required to reload code on i.MX 7Dual/7Solo are as follows:

1. Issue a platform reset by setting the SRC_M4RCR[2] bit in SRC_MRCR register
2. Wait for the SRC_M4RCR[2] to be cleared
3. Set the Stack pointer to the first four bytes of the binary file
4. Set the PC pointer to next four bytes after the first four bytes of the binary file
5. Load the binary file starting at the address 0x007F_8000
6. Reset the Cortex-M4 microcontroller by setting the SRC_M4RCR[1] bit in the SRC_M4RCR register.

4. Reloading Code on the i.MX 6SoloX

4.1. On-chip memory view from each ARM core on the i.MX 6SoloX

The memory view of different peripherals is different between the Cortex-A9 and the Cortex-M4 side. Table 3, below shows only the relevant memory areas for this application note. For more details, refer to the Memory Map chapter in the i.MX 6SoloX Applications Processor Reference Manual (document IMX6SXRM)

Table 3. Start and end addresses of different memories from Cortex-A9 and Cortex-M4 side

Peripheral	Start address Cortex-A9	End address Cortex-A9	Start address Cortex-M4 side	End address Cortex-M4 side	Size
TCM_L	0x007F_8000	0x007F_7FFF	0x1FFF_8000	0x1FFF_FFFF	32 KB
TCM_U	0x0080_0000	0x0080_7FFF	0x2000_0000	0x2000_7FFF	32 KB

NOTE

The Boot vector for the Cortex-M4 core is located at the start of the TCM_L whose address is 0x007F_8000 from the Cortex-A9. This is a different location than on the i.MX 7Dual/7Solo.

4.2. Detailed procedure

In order to reload the code on the Cortex-M4 core using the Cortex-A9 processor on i.MX 6SoloX, users must follow the steps listed below:

1. Issue a software platform reset by setting up **M4P_RST** (Bit 12) in the **SRC_SCR** (SRC_M4RCR[12]) register**.** Issuing a platform reset resets Cortex-M4 cores and associated memories**.** The address of **SRC_SCR** register is 0x020D_8000 for i.MX 6SoloX.
2. Load the code for Cortex-M4 processor into the TCM_L memory. For this application, we assume that the M4 code is compiled to execute from TCM_L memory. Users must program the binary file generated by FreeRTOS to the TCM_L address from the Cortex -A9 side which is 0x007F_8000 as listed in Table 3.
3. Once the file has been loaded the next step is to setup the Stack and PC pointer in the TCM_L memory. This is because after a reset the processor uses the TCM_L start address (0x007F_8000) as the first instruction. For this implementation, the stack value is the first four bytes found in the binary file generated for Cortex-M4 processor using the FreeRTOS source. The PC value is also four bytes long and is located at an offset of 0x4 in the binary file. This PC value is written to the OCRAM_S base address plus four which is (0x007F_8004). Table 4, further clarifies the Stack and PC addresses.

Table 4. Boot vectors location for Cortex-M4 core

Boot vectors	TCM_L location for boot vectors	Location of boot vectors in binary file
Stack pointer	0x007F_8000	First four bytes
Program counter	0x007F_8004	Four bytes after first four bytes

4. Once the startup address in the OCRAM_S has been modified according to the binary file and loaded into the memory , the next step is to ensure the **ENABLE_M4** (Bit 22) in **SRC_SCR** (SRC_SCR[22]) register is set to 1. Since the *bootaux* already booted a primary image, this bit should be 1. Performing a platform reset using **M4P_RST** (Bit 12) in the **SRC_SCR** (SRC_SCR[12]) register does not clear this bit. The last step is to set the **M4C_RST (**Bit 3) in the **SRC_SCR** (SRC_SCR[3]) register which will boot the new code on the Cortex-M4 processor.
5. Repeat 1-3 for reloading a new image. More details about the **SRC_SCR** register are provided in Figure 2.

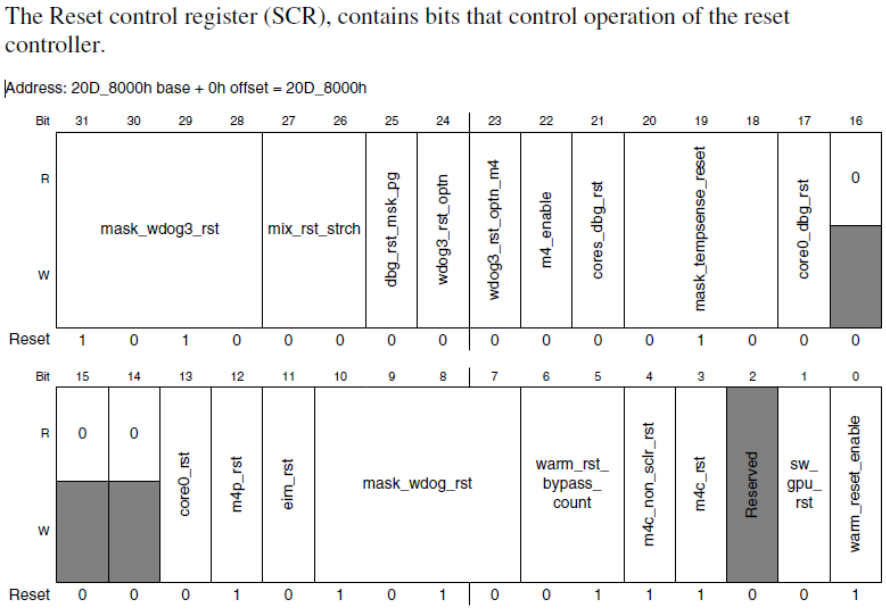

Figure 2. SRC Control Register (SRC_SCR)

4.3. Steps for reloading code on i.MX 6SoloX

A summary of the steps required to reload code on i.MX 6SoloX are as follows:

1. Issue a platform reset by setting the SRC_SCR[12] bit in the SRC_SCR register
2. Wait for the SRC_SCR[12] bit to be cleared by the hardware
3. Set the Stack pointer to the first four bytes of the binary file
4. Set the PC pointer to the next four bytes after the first four bytes of the binary file
5. Load the binary file starting at the address 0x007F_8000
6. Reset the Cortex-M4 by setting the SRC_SCR[3] bit in the SRC_SCR register.

5. References

1. i.MX 7Dual Applications Processor Reference Manual (document IMX7DRM)
2. i.MX 6SoloX Applications Processor Reference Manual (document IMX6SXRM)

6. Revision History

Table 5. Revision history

Revision number	Date	Substantive changes
0	08/2016	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN5317
Rev. 0
08/2016