Freescale Semiconductor, Inc.
Application Note

Document Number: AN5127
Rev. 1, 05/2015

How to Run the MQX™ RTOS on Various RAM Memories for i.MX 6SoloX

Contents

1 Introduction

This document describes how to customize software to enable Cortex®-M4 to run MQX™ RTOS on different RAM memories, such as OCRAM and DDR SDRAM.

By default, the i.MX 6SoloX Linux® BSP and MQX RTOS releases support Cortex-M4 to run on the QuadSPI NOR flash. In some cases, however, users need to run the MQX RTOS on DDR or OCRAM instead of the NOR flash.

This document is based on the following environments:

- i.MX 6SoloX SABRE-SDB
- Freescale MQX 4.1 i.MX 6SoloX GA
- Yocto 3.10.53 GA
- ARM® Development Studio 5 (DS-5™) version 5.18.0
- IAR 7.20

2 Scenario Overview

The i.MX 6SoloX processors are the newest multi-core architecture designed by using the ARM Cortex®-A9 and ARM Cortex-M4 cores. This dual-core architecture enables the device to run an open operating system (OS) like Linux on the Cortex-A9 core and an RTOS like MQX on the Cortex-M4 core.

This Asymmetric Multi-core Processor (AMP) system has the following advantages:

- Improves the power efficiency for a reduced carbon footprint. For example, one core enters low power mode while the other core is running.
- Balances different task requirements in single silicon chip, such as real-time vs. non real-time tasks and simple computing vs. complex computing tasks.
- Meets various application scenarios, such as automobile, industrial automation, consumer electrics.

Based on these conditions, Freescale provides the possibility to run the Linux OS on the Cortex-A9 core and run the MQX RTOS on the Cortex-M4 core at the same time. The reference board has the following memory features:

- 1 GB DDR
- 256 KB NOR Flash through QuadSPI
- 128 KB OCRAM

However, the release BSP only supports to run the MQX RTOS on the QuadSPI NOR flash and run the Linux OS on DDR by default. To determine whether to run the MQX RTOS on various RAM memories, consider the following cases:

- The custom board has no NOR flash because of the BOM cost or hardware space.
- Downloading to RAM is faster than to the flash and is more convenient for debugging.
- Running the program out of RAM memory might be faster and enhance the application performance.

This document describes how to customize software to make Cortex-M4 to run the MQX RTOS on various RAM memories. It includes:

- Running the MQX RTOS on DDR
- Running the MQX RTOS on OCRAM

To run both the MQX RTOS and the Linux kernel on DDR, consider the following points:

- How to define the memory layout
- How to change the Linux kernel start address
- How to change the MQX RTOS start address
- How to change the U-Boot start address

- How to run the demo

To run the MQX RTOS on OCRAM while running the Linux kernel on DDR, consider the following points:

- How to define the memory layout
- How to change the MQX RTOS start address
- How to run the demo

3 How to Define the Memory Layout

Table 1 shows the memory layout for the release BSP.

Table 1 Memory layout

Core	OS type	Start address	Size	Memory type
Cortex-A9	Linux	0x80000000	1024 MB	DDR
Cortex-M4	MQX	0x78000000	256 KB	QSPI NOR flash

3.1 Running the MQX RTOS on DDR

To run the MQX RTOS on DDR, you need to allocate some DDR memory to the MQX RTOS and this specific memory should not be accessed by the Linux OS. Therefore, it is a better solution to leave a section of memory at the beginning of DDR for MQX RTOS and allocate the rest of the DDR memory to the Linux kernel.

For example, if you have a system with 1 GB of DDR, you can partition the system as follows.

Table 2 System partition

Core	OS type	Start address	Size	Memory type
Cortex-A9	Linux	0x81000000	1008 MB	DDR
Cortex-M4	MQX	0x80000000	16 MB	DDR

In this case, pay attention to the following points:

- The Linux kernel requires the start address to be a multiple of 16 MB. This means that the addresses should be incremented by 0x1000000, so we can only change the start address to 0x81000000, 0x82000000, etc.
- The DDR size is scalable. The maximum is 2 GB and the minimum is 256 MB. The DDR memory size should be allocated according to the actual memory size.
- The DDR memory connected to the Cortex-M4 (> 0xa0000000) space should not be allocated to execute the Cortex-M4 code, because that region is predefined as the device type and XN. We can select the region for Cortex-M4 from (0x80000000 - 0x9FFFFFFF).

3.2 Running the MQX RTOS on OCRAM

Theoretically, the MQX RTOS can run on any address of OCRAM. In some cases, however, if your OCRAM has other purposes, it should define the memory layout carefully. In reference BSP, when U-Boot enters command line mode, most of the OCRAM space used by the boot ROM codes is freed, but the first 0x1000 bytes starting from 0x00900000 are used by DDR frequency change for implementing Linux low power mode. This address range cannot be used for the MQX RTOS and should be reserved for the Cortex-A9 Linux Software.

Table 3 New memory layout

Core	OS type	Start address	Size	Memory type
Cortex-A9	Linux	0x80000000	1024 MB	DDR
Cortex-M4	MQX RTOS	0x00901000	64 KB	OCRAM

In this case, pay attention to the following points:

- It is not necessary to rebuild the Linux kernel and U-Boot.
- The memory size should not exceed the threshold 128 KB of OCRAM.
- 0x00900000 - 0x00901000 is reserved for frequency scaling that is described by the device tree. If your Linux does not need low power mode support for DDR, you can change the start address to 0x00900000.

4 How to Change the MQX RTOS Start Address

The methods of changing the MQX RTOS start address for different development tools are similar. In this document, we use ARM Development Studio 5 (DS-5™) to build the MQX RTOS.

For more information about how to use DS-5, see the *Getting Started with ARM Development Studio 5 (DS-5™) with Freescale MQX™ RTOS* (MQXGSRTOS).

4.1 Configuring ARM Development Studio 5 (DS-5™)

The MQX RTOS build procedure has two targets:

- Flash target

  Default target. With the flash target, the code and application run in the flash memory. In the default BSP, this flash memory is the NOR flash through QSPI.

- RAM target

  The instructions are the same as the flash target except one additional step when building the application: Change the link file flash.xxx to ram.xxx. With the RAM target, the code and application run in the RAM memory, for example, DDR and OCRAM.

In common cases, the RAM target is used for debugging and the flash target is used for final application deployment.

To enable M4 to run the MQX RTOS on the DDR SDRAM memory, select the RAM target. This document takes the mutex project as an example.

1. When you import the mutex project, the default configuration is **Int Flash Release**. Rename it to **RAM Release**, as shown in Figure 1.

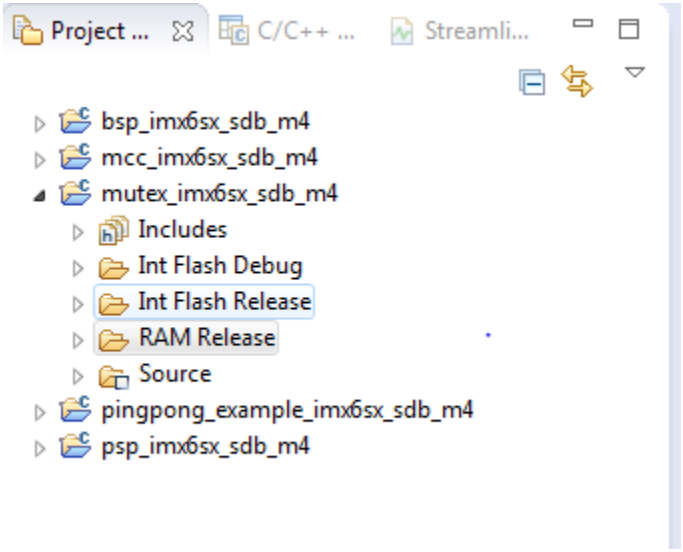

Figure 1 Default configuration

2. Change the link file flash.scf to ram.scf, as shown in Figure 2.

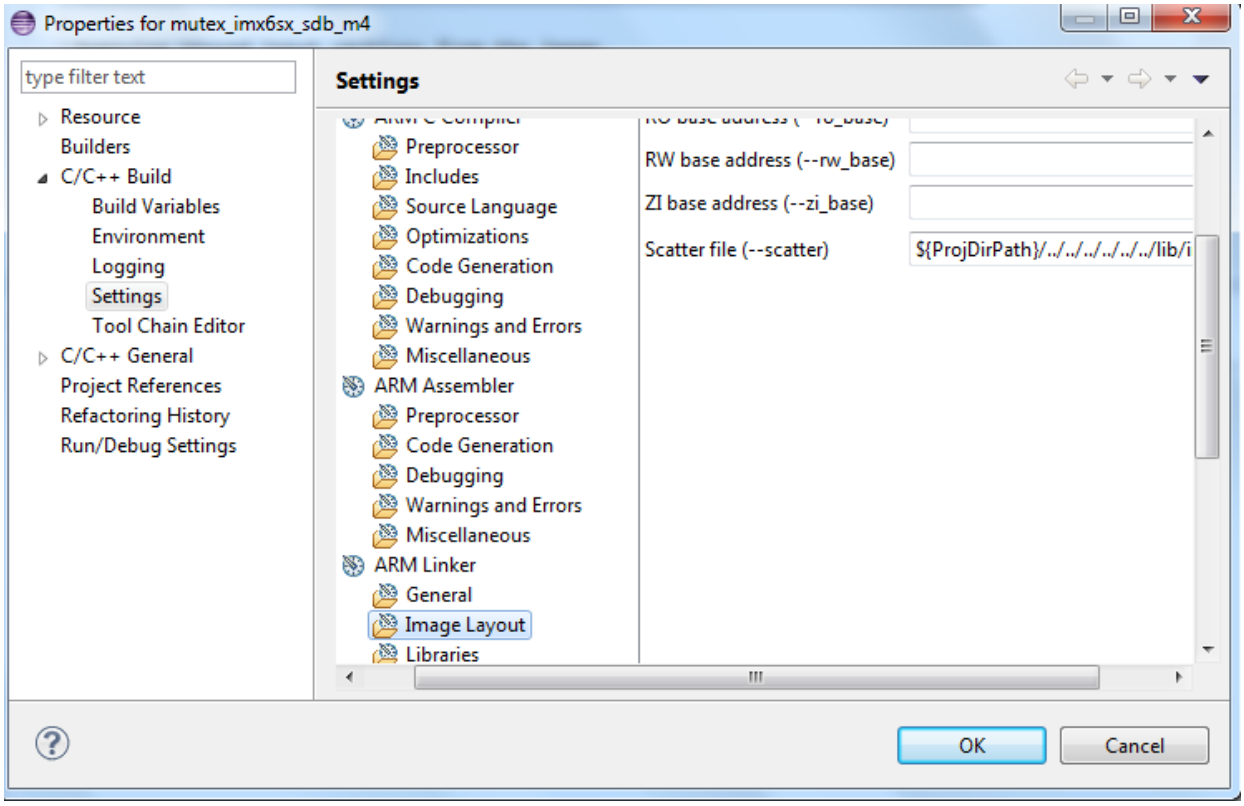

Figure 2 Changing the link file

For more information about how to select the RAM target, see the *Freescale MQX™ RTOS 4.1.0 i.MX 6SoloX Release Notes* (MQXIMX6SXRN).

4.2 Modifying link files and build

- For the **RAM Release** version, edit <install dir>/lib/imx6sx_sdb_m4.ds5/release/bsp/ram.scf.
- For the **RAM Debug** version, edit <install dir>/lib/imx6sx_sdb_m4.ds5/debug/bsp/ram.scf.

The modifications of codes are as follows:

- Original code:

```
#define CODE_BASE_ADDR_START    0x78000000
#define CODE_BASE_ADDR_END      0x7803FFF0
```

- New code for running the MQX RTOS on DDR:

```
#define CODE_BASE_ADDR_START    0x80000000
#define CODE_BASE_ADDR_END      0x80FFFFFF
```

- New code for running the MQX RTOS on OCRAM:

```
#define CODE_BASE_ADDR_START    0x00901000
```

```
#define CODE_BASE_ADDR_END      0x00921000
```

When the build process is completed, the mutex.axf will be at:

- <install dir>/mqx/examples/mutex/build/ds5/mutex_imx6sx_sdb_m4/RAM Release
- <install dir>/mqx/examples/mutex/build/ds5/mutex_imx6sx_sdb_m4/RAM Debug

Under the DS-5 command prompt, access the directories above and run:

```
fromelf --bin --output=mutex.bin mutex.axf
```

5 How to Change the Linux Kernel Start Address

In some other Linux kernel BSPs, they may support configuring the kernel start address from the configuration menu, but in the Linux 3.10.53 BSP release, this feature is not supported. You need to modify the kernel files to configure the start address. This section describes the modifications that are required in the kernel.

The following files need to be modified:

- arch/arm/Kconfig and /arch/arm/mach-imx/Kconfig
- arch/arm/mach-imx/Makefile.boot
- arch/arm/Makefile

5.1 Modifying the Kconfig files

ZRELADDR is the physical address where the decompressed kernel image is placed. If AUTO_ZRELADDR is selected, the address is determined at run-time by masking the current PC with 0xf8000000. This assumes that the zImage is placed in the first 128 MB of the memory.

```
#ifdef CONFIG_AUTO_ZRELADDR
             @ determine final kernel image address
             mov    r4, pc
             and    r4, r4, #0xf8000000
             add    r4, r4, #TEXT_OFFSET
#else
             ldr    r4, =zreladdr
```

In the default Kconfig files, because the kernel is for multiple platforms and is based on the MXC architecture, CONFIG_ARCH_MULTIPLATFORM and CONFIG _ARCH_MXC should be selected. For example, imx_v7_defconfig, CONFIG_ARCH_MULTIPLATFORM, and CONFIG _ARCH_MXC are all default, but this will cause CONFIG_AUTO_ZRELADDR to be selected. In this case, the zImage is placed in the first 128 MB of the DDR memory, which may overlap the DDR memory space used by the MQX RTOS. For example, the zImage is placed in 0x80001000 by default, which is not expected.

Therefore, you need to make the following modifications to set CONFIG_AUTO_ZRELADDR to default N:

- Delete “select AUTO_ZRELADDR” from the CONFIG_ARCH_MULTIPLATFORM menu. The file is /arch/arm/Kconfig.
- Delete “select AUTO_ZRELADDR if !ZBOOT_ROM” from the CONFIG _ARCH_MXC menu. The file is /arch/arm/mach-imx/Kconfig.

When the configuration for the kernel is complete, the CONFIG_AUTO_ZRELADDR is N, and you can manually modify the physical address where the decompressed kernel image is placed.

5.2 Adding arch/arm/mach-imx/Makefile.boot

When CONFIG_AUTO_ZRELADDR is not set, the physical address where the decompressed kernel image is placed is defined by zreladdr. Usually, the Linux kernel uses the Makefile.boot file to load custom-board boot parameters. In the Linux 3.10.53 BSP, the file is deleted and the kernel automatically computes this physical address according to the SOC type, for example, 0x80008000 for i.MX 6SoloX, 0x10008000 for i.MX 6Quad/DualLite and i.MX 6SoloLite.

In this case, we only consider i.MX 6SoloX SOC, and add Makefile.boot as follows:

```
zreladdr-y := 0x81008000
```

Pay attention to the following points:

- The DDR memory map is different between i.MX 6Quad/DualLite, i.MX 6SoloLite, and i.MX 6SoloX. ZRELADDR is different between the SOC types. In this case, only i.MX 6SoloX is considered.
- ZRELADDR is variable, but it should be compliant with the memory layout. This address should be less than 128 MB from the Linux kernel start address, or it fails to boot.

  ```
  Linux kernel start address:      0x80000000 0x81000000 0x82000000
  zreladdr:                        0x80001000 0x81008000 0x82008000
  ```

  This is because the kernel automatically fixes PHYS_OFFSET according to ZRELADDR.
- The 0x8000 offset is a common value. Do not change it. The first 16 KB is for the page table, and the second 16 KB is for boot parameters.

5.3 Modifying arch/arm/Makefile

In the Linux 3.10.53 BSP, Makefile.boot is not included in the build system, so you need to add it to the build system.

- Original code:

  ```
  ifeq ($(CONFIG_ARCH_MULTIPLATFORM),y)
  MACHINE  :=
  endif
  ```
- New code:

  ```
  ifeq ($(CONFIG_ARCH_MULTIPLATFORM),y)
  MACHINE  := arch/arm/mach-imx
  endif
  ```

5.4 Using the patch to apply the modifications

Use the patch to apply the modifications described above:

```
# cd build/tmp/work/imx6sxsabresd-poky-linux-gnueabi/linux-imx/3.10.53-r0/git
```

```
# git am 0001-Kernel-M4-Reserve-16M-DDR-memory-for-M4-core-use.patch
# . ./setup-environment build/
#  bitbake -v -f -c configure linux-imx
#  bitbake -v -f -c compile linux-imx
#  bitbake -v -f -c install  linux-imx
#  bitbake -v -f -c deploy linux-imx
```

6 How to Change the U-Boot Start Address

By default, U-Boot runs on DDR and it occupies some memory at the beginning of DDR. If you also want to run the MQX RTOS on DDR, change U-Boot start address first, or it will conflict with the MQX RTOS.

Generally, we change the U-Boot start address to be consisten with the Linux kernel start address. For example, if the Linux kernel start address is 0x81000000, we should change the U-Boot start address to 0x81000000.

It is easy to change the U-Boot start address. You only need to modify the physical memory map value in /include/configs/mx6sxsabresd.h as follows:

- Original code:

```
#define PHYS_SDRAM                MMDC0_ARB_BASE_ADDR
#define PHYS_SDRAM_SIZE                  SZ_1G
```

- New code:

```
#define PHYS_SDRAM                (MMDC0_ARB_BASE_ADDR + SZ_16M)
#define PHYS_SDRAM_SIZE                  (SZ_1G - SZ_16M)
```

In addition, you can also change the U-Boot default environment variables and it is very convenient for you to load the MQX RTOS image and Linux image in the boot command prompt. These changes are optional.
The following is an example to set these environment variables:

- Original code:

```
#define CONFIG_LOADADDR                  0x80800000
#define CONFIG_SYS_AUXCORE_BOOTDATA 0x78000000
#define CONFIG_SYS_MEMTEST_START  0x80000000
```

- New code:

```
#define CONFIG_LOADADDR                  0x81800000
#define CONFIG_SYS_AUXCORE_BOOTDATA 0x80000000
#define CONFIG_SYS_MEMTEST_START  (0x80000000 + SZ_16M)
```

Use the patch to apply the changes above:

```
# cd  build/tmp/work/imx6sxsabresd-poky-linux-gnueabi/u-boot-imx/2014.04-r0/git
# git am 0001-U-Boot-M4-Reserve-16M-DDR-memory-for-M4-core-use.patch
# . ./setup-environment build/
#  bitbake -v -f -c compile u-boot
#  bitbake -v -f -c install u-boot
#  bitbake -v -f -c deploy u-boot
```

7 How to Run the Demo

7.1 Prerequisites

No matter what memory the MQX RTOS runs on, this demo should meet the following requirements:

- It should have uboot.imx for i.MX 6SoloX. This U-Boot is used for loading dtb, zImage, and MQX application images to RAM and booting the MQX application and Linux kernel.
- It should have imx6sx-sdb-m4.dtb for i.MX 6SoloX: This dtb should prevent resource conflicts between Cortex-A9 and Cortex-M4.
- It should have mutex_imx6sx_sdb_m4.bin. This is an MQX application demo image.
- It should have zImage. This is a Linux kernel demo image.

For more information about how to boot the Linux OS and how to run the MQX application, see the *i.MX Linux User's Guide* (IMXLUG).

NOTE

The following steps are based on the U-Boot. This U-Boot image demo is for minimum change version and some optional environment variables are not modified in the source codes. Therefore, we should modify these environment variables in run-time. If you have modified the related environment variables in the source code and rebuilt it, you can skip these steps.

7.2 Running the MQX RTOS on DDR

Assume that you have an SD/MMC image for booting from the SD4 slot, and you have copied mutex.bin, zImage, and imx6sx-sdb-m4.dtb to the FAT file system on the SD card. Perform the following steps:

1. Load the MQX application binary to DDR.

   ```
   fatload mmc 2:1 0x80000000 mutex_imx6sx_sdb_m4.bin
   ```

2. Flush the U-Boot data cache.

   ```
   dcache flush
   ```

3. Run the MQX application on DDR.

   ```
   bootaux 0x80000000
   ```

4. Run the Linux kernel.

   ```
   run bootcmd
   ```

7.3 Running the MQX RTOS on OCRAM

Assume that you have an SD/MMC image for booting from the SD4 slot, and you have copied mutex.bin, zImage, and imx6sx-sdb-m4.dtb to the FAT file system on the SD card. Perform the following steps:

1. Load the MQX application binary to OCRAM.

   ```
   fatload mmc 2:1 0x00901000 mutex_imx6sx_sdb_m4.bin
   ```

2. Flush the U-Boot data cache.

   ```
   dcache flush
   ```

3. Run the MQX application on OCRAM.

   ```
   bootaux 0x00901000
   ```

4. Boot the Linux kernel and mount filesystem.

   ```
   run bootcmd
   ```

8 Low Power Consumption

In some cases, you may need to reduce the power consumption of the entire system. To meet these low power application requirements, Freescale implements some low-level power management drivers.

For i.MX 6SoloX, there are two cases:

- The Linux OS is suspended while Cortex-M4 is in low-power idle mode.

  In this case, the MQX RTOS is possibly not running, and then Cortex-M4 is in low-power idle mode. There are the following suspend maps for the Linux OS:

  - **Standby**: maps to STOP mode, which saves power significantly, because all the blocks in the system enter the low-power state, except for the ARM core, which is still powered on, and the memory is in self-refresh mode to retain its contents.
  - **Mem** (suspend to RAM): maps to DORMANT mode, which saves power significantly, because all the blocks in the system enter the low-power state, except for the memory, which is in the self-refresh mode to retain its contents.

- The Linux OS is suspended while Cortex-M4 is running.

  In this case, the MQX RTOS is running and Cortex-M4 is not in low-power idle mode. No matter Standby and Mem, they both map to WAIT mode, which ensures that Cortex-M4 CLK is alive.

For more information about the low power mode for the Linux OS, see the *i.MX 6 Linux Reference Manual* (IMX6LXRM).

For more information about the low power mode for i.MX 6SoloX, see the *i.MX 6SoloX Applications Processor Reference Manual* (IMX6SXRM).

9 Influence and Impact on the System

With this function support, Cortex-M4 can run the MQX RTOS on the RAM memory (OCRAM or DDR) instead of the QuadSPI NOR flash only. The influence and impact on the system are as follows:

- When the MQX application image is loaded to DDR or OCRAM, even though it is faster than to load to the NOR Flash, because these RAMs is not non-volatile memory, it needs to load the image every time on booting. This increases the booting time of the system.
- When the MQX RTOS runs on DDR or OCRAM, it can improve the performance on the MQX side, because the speed to access the RAM is faster than to access the flash.
- When the MQX RTOS runs on DDR, Cortex-M4 shares the DDR bandwidth with Cortex-A9. It is a risk to reduce the performance on the Linux side. It is not suitable for the DDR bandwidth hogging applications.
- When the MQX RTOS runs on DDR, because the Linux kernel requires the start address to be a multiple of 16 MB, the minimum reserved DDR size is 16 MB. The reserved space that is not occupied by the MQX image becomes invalid. This is a waste of the DDR memory.

10 Revision History

Table 1 Table 4 Revision history

Revision number	Date	Substantive changes
1	05/2015	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5127
Rev. 1
05/2015