Freescale Semiconductor, Inc.
应用笔记

Document Number: AN5127
Rev. 1, 05/2015

如何在 i.MX 6SoloX 的各种随机存取存储器 (RAM) 上运行 MQX™ 实时操作系统

目录

1 简介

本文档介绍如何根据需要修改软件，使 Cortex®-M4 可以在不同的 RAM（如内置 OCRAM 和外置 DDR SDRAM）上运行 MQX™ RTOS。

默认情况下，i.MX 6SoloX Linux®板级支持包 (BSP) 和 MQX 实时操作系统 (RTOS) 发行包支持 Cortex-M4 在 QuadSPI NOR Flash 上开始运行。然而在某些情况下，用户需要在 DDR 或 OCRAM（而非 NOR Flash）上运行 MQX RTOS。

本文档基于以下环境：

- i.MX 6SoloX SABRE-SDB
- Freescale MQX 4.1 i.MX 6SoloX GA
- Yocto 3.10.53 GA
- ARM® Development Studio 5 (DS-5™) v5.18.0
- IAR 7.20

2 应用情景概览

i.MX 6SoloX处理器是一种最新的基于ARM Cortex®-A9和ARM Cortex-M4内核设计的多核架构。该双核架构使得器件可以在Cortex-A9内核上运行一种开放的操作系统(OS)，如Linux，同时在Cortex-M4内核上运行实时操作系统(RTOS)，如MQX。

这种非对称多核处理器(AMP)系统具有以下优势：

- 可以改善能效，以减少碳排放。例如，在一个内核运行时，另一个内核进入低功耗模式。
- 能够在单个硅芯片中平衡不同的任务需求，如实时与非实时任务的需求，以及简单计算与复杂计算任务的需求。
- 能够满足各种应用场景（如汽车、工业自动化和消费电子）的需求。

基于此，飞思卡尔提供了可以同时在Cortex-A9内核上运行Linux OS和在Cortex-M4内核上运行MQX RTOS的可能。参考板上的存储器配置如下：

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

然而，默认情况下， BSP发行包仅支持在QuadSPI NOR Flash上运行MQX RTOS和在DDR上运行Linux OS。是否需要在不同的RAM上运行MQX RTOS，主要考虑如下因素：

- 由于BOM成本或硬件空间的限制，客户板上没有NOR Flash。
- 下载到RAM的速度比到Flash的更快，且更便于调试。
- 从RAM运行应用程序，其速度可能更快，从而提升性能。

本文档介绍如何根据需要修改软件，使Cortex-M4可以在不同的RAM上运行MQX RTOS。其中包括：

- 在DDR上运行MQX RTOS
- 在OCRAM上运行MQX RTOS

要在DDR上运行MQX RTOS和Linux内核，需考虑以下几点：

- 如何定义存储器布局
- 如何修改Linux内核的起始地址
- 如何修改MQX RTOS的起始地址
- 如何修改U-Boot的起始地址
- 如何运行演示例程

要在 DDR 上运行 Linux 内核的同时在 OCRAM 上运行 MQX RTOS，需考虑以下几点：

- 如何定义存储器布局
- 如何修改 MQX RTOS 的起始地址
- 如何运行演示例程

3 如何定义存储器布局

表 1 展示了 BSP 发行包定义的存储器布局。

表 1 存储器布局

内核	OS 类型	起始地址	大小	存储器类型
Cortex-A9	Linux	0x80000000	1024 MB	DDR
Cortex-M4	MQX	0x78000000	256 KB	QSPI NOR Flash

3.1 在 DDR 上运行 MQX RTOS

要在 DDR 上运行 MQX RTOS，需要将一些 DDR 存储器空间分配给 MQX RTOS，且 Linux OS 不能访问该指定存储器空间。因此，较好的解决方案是在 DDR 的开始处预留一个存储器扇区给 MQX RTOS 使用，而其余的 DDR 存储器空间分配给 Linux 内核。

例如，如果系统具有 1 GB 的 DDR，可对其进行如下分区。

表 2 系统分区

内核	OS 类型	起始地址	大小	存储器类型
Cortex-A9	Linux	0x81000000	1008 MB	DDR
Cortex-M4	MQX	0x80000000	16 MB	DDR

在此情况下，应注意以下几点：

- Linux 内核要求其起始地址为 16 MB 的整数倍。这意味着地址应以 0x1000000 递增，因此起始地址只能修改为 0x81000000、0x82000000 等。
- DDR 的容量大小可以调整。最大为 2 GB，最小为 256 MB。DDR 存储器大小应根据实际的存储器容量来分配。
- 连接到 Cortex-M4 的 DDR 存储器，地址大于 (0xa0000000) 的区域不应分配用于执行 Cortex-M4 代码，因为该区域预定义为 Cortex-M4 的外部器件地址空间和指令非执行区。我们可以从 (0x80000000 - 0x9FFFFFFF) 中选择用于 Cortex-M4 的区域。

3.2 在 OCRAM 上运行 MQX RTOS

理论上，MQX RTOS 可在 OCRAM 的任何地址上运行。然而，在某些情况下，如果 OCRAM 有其他用途，则应谨慎定义存储器布局。在参考 BSP 中，当 U-Boot 进入命令行模式时，ROM 启动代码使用的大部分 OCRAM 空间将被释放，但是从 0x00900000 开始的前 0x1000（4096）字节 OCRAM 将被作为执行 Linux 低功耗模式时 DDR 频率修改之用。该地址空间不能被 MQX RTOS 使用，而应预留给 Cortex-A9 内核运行的 Linux 软件。

表 3 新存储器布局

内核	OS 类型	起始地址	大小	存储器类型
Cortex-A9	Linux	0x80000000	1024 MB	DDR
Cortex-M4	MQX RTOS	0x00901000	64 KB	OCRAM

在此情况下，应注意以下几点：

- 无需重新编译 Linux 内核与 U-Boot。
- 存储器大小不应超出 OCRAM 的 128 KB 实际容量。
- 0x00900000 - 0x00901000 预留给 Linux 设备树描述的频率调整之用。如果 Linux 无需支持用于 DDR 的低功耗模式，则起始地址可以改为 0x00900000。

4 如何修改 MQX RTOS 的起始地址

针对使用不同开发工具修改 MQX RTOS 起始地址的方法均类似。在本文档中，我们使用 ARM Development Studio 5 (DS-5™) 来编译 MQX RTOS。

欲了解更多关于如何使用 DS-5 的信息，请参见 *Getting Started with ARM Development Studio 5 (DS-5™) with Freescale MQX™ RTOS* (MQXGSRTOS)。

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

MQX RTOS 编译流程生成两种目标文件：

- Flash 目标文件

  这是默认目标文件。通过 Flash 目标文件，在 Flash 存储器上运行应用程序代码。在默认 BSP 配置中，该 Flash 存储器为 QSPI 接口的 NOR Flash。

- RAM 目标文件

  RAM 目标文件编译流程，除了增加一个额外步骤外，其余指令与 Flash 目标文件编译流程一致：将链接文件从 flash.xxx 改为 ram.xxx。通过 RAM 目标文件，可以在 DDR 和 OCRAM 等随机存取存储器上运行应用程序代码。

通常情况下，RAM 目标文件用于调试，而 Flash 目标文件用于最终的应用实施。

要让 Cortex-M4 在 DDR SDRAM 上运行 MQX RTOS，应选择 RAM 目标文件。本文档以互斥工程为例。

1. 当导入互斥工程时，默认配置为 **Int Flash Release**。将其更改为 **RAM Release**，如图 1 所示。

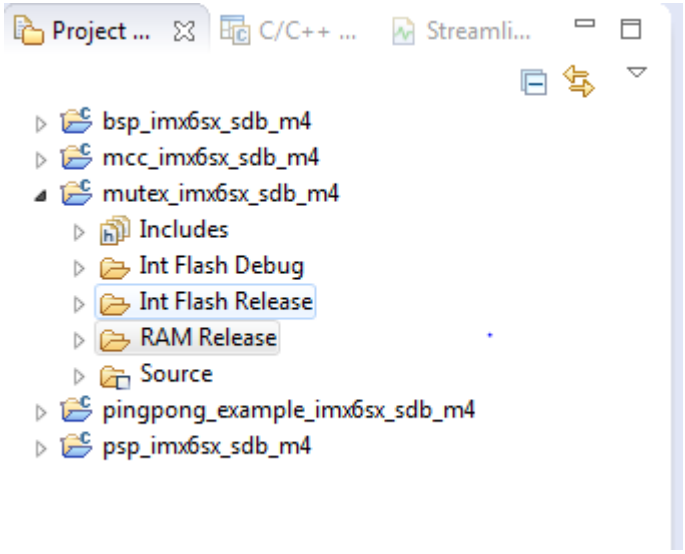

图 1 默认配置

2. 将链接文件从 flash.scf 改为 ram.scf，如图 2 所示。

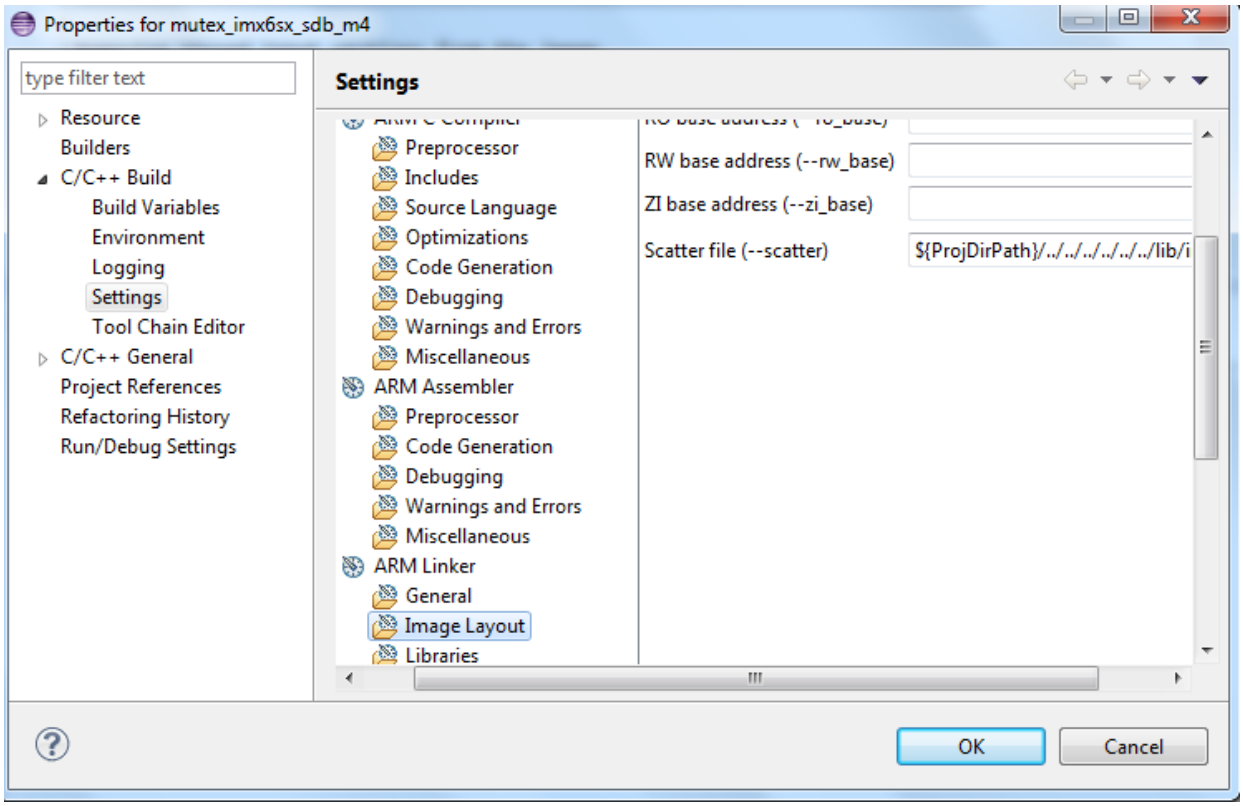

图 2 修改链接文件

欲了解更多关于如何选择 RAM 目标文件的信息，请参见 *Freescale MQX™ RTOS 4.1.0 i.MX 6SoloX Release Notes* (MQXIMX6SXRN)。

4.2 修改链接文件并编译

- 对于 **RAM 发行**版本，编辑<install dir>/lib/imx6sx_sdb_m4.ds5/release/bsp/ram.scf。
- 对于 **RAM 调试**版本，编辑<install dir>/lib/imx6sx_sdb_m4.ds5/debug/bsp/ram.scf。

代码修改如下：

- 原代码：

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

- 用于在 DDR 上运行 MQX RTOS 的新代码：

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

- 用于在 OCRAM 上运行 MQX RTOS 的新代码：

```
#define CODE_BASE_ADDR_START    0x00901000
#define CODE_BASE_ADDR_END      0x00921000
```

编译完成后，mutex.axf 将位于：

- <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

在 DS-5 的命令提示符下，从上述目录下运行：

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

5 如何修改 Linux 内核的起始地址

某些 Linux 内核版本的 BSP 可能支持从配置菜单设置内核起始地址，但使用 Linux 3.10.53 内核版本的 BSP 发行包不支持该功能。必须通过修改内核文件来配置其起始地址。本章节介绍在内核中所要进行的修改。

需要修改以下文件：

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

5.1 修改 Kconfig 文件

ZRELADDR 是放置解压缩内核镜像的物理地址。如果选择了 AUTO_ZRELADDR 配置选项，则实际地址实时地由 0xf8000000 屏蔽程序计数器当前值来确定。这里假设 zImage 镜像被存放于存储器的首个 128 MB 空间中。

```
#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
```

在默认的 Kconfig 文件中，由于内核用于支持基于 MXC 架构的多种平台，因而必须选择 CONFIG_ARCH_MULTIPLATFORM 和 CONFIG _ARCH_MXC 配置选项。例如，imx_v7_defconfig、CONFIG_ARCH_MULTIPLATFORM 和 CONFIG _ARCH_MXC 均为默认选项，这将导致 CONFIG_AUTO_ZRELADDR 项被选择。在此情况下，zImage 镜像将被置于存储器起初的 128 MB 空间中，这就会与 MQX RTOS 使用的 DDR 存储器空间重叠。例如，zImage 镜像被默认置于 0x80001000 起始的 DDR 存储器空间中，而我们并不希望发生这种情况。

因此，必须作以下修改使 CONFIG_AUTO_ZRELADDR 选项设为默认不选：

- 从 CONFIG_ARCH_MULTIPLATFORM 菜单中删除“select AUTO_ZRELADDR”。该文件位于/arch/arm/Kconfig。
- 从 CONFIG _ARCH_MXC 菜单中删除“select AUTO_ZRELADDR if !ZBOOT_ROM”。该文件位于/arch/arm/mach-imx/Kconfig。

以上内核配置完成后，CONFIG_AUTO_ZRELADDR 选项将被设为不选，从而可以手动修改用于放置解压缩的内核镜像的物理地址。

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

当 CONFIG_AUTO_ZRELADDR 选项未被选择时，放置解压缩的内核镜像的物理地址由 zreladdr 确定。通常，Linux 内核使用 Makefile.boot 文件来加载客户板的启动参数。在使用 Linux 3.10.53 内核版本的 BSP 中删除了该文件，内核将依据 SOC 类型自动设定该物理地址，例如，i.MX 6SoloX 设为 0x80008000，i.MX 6Quad/DualLite 和 i.MX 6SoloLite 设为 0x10008000。

在本例中，我们仅考虑 i.MX 6SoloX SOC，并添加 Makefile.boot 如下：

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

请注意以下几点：

- i.MX 6Quad/DualLite、i.MX 6SoloLite 和 i.MX 6SoloX 之间的 DDR 存储器映射各不相同，因而不同 SOC 对应的 ZRELADDR 也不一样。在本例中，只考虑 i.MX 6SoloX。
- ZRELADDR 是一个变量，但是应遵循存储器布局要求。该地址与 Linux 内核的起始地址之间的距离应小于 128 MB，否则将导致启动失败。

  ```
  Linux 内核的起始地址：  0x80000000 0x81000000 0x82000000
  zreladdr:             0x80001000 0x81008000 0x82008000
  ```

  这是因为内核会根据 ZRELADDR 自动确定 PHYS_OFFSET。
- 0x8000 偏移量是一个常规值。不要修改该值。首个 16 KB 用于存放页表，第二个 16 KB 用于存放启动参数。

5.3 修改 arch/arm/Makefile

在使用 Linux 3.10.53 内核版本的 BSP 中，Makefile.boot 没有被包含在编译系统，因此必须将其添加进来。

- 原代码：

  ```
  ifeq ($(CONFIG_ARCH_MULTIPLATFORM),y)
  MACHINE :=
  endif
  ```
- 新代码：

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

5.4 使用补丁来实施修改

使用补丁来实施上述修改：

```
# 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 如何修改 U-Boot 的起始地址

默认情况下，U-Boot 在 DDR 上运行，其占据 DDR 起始的部分存储器空间。如果想要在 DDR 上运行 MQX RTOS，必须先修改 U-Boot 的起始地址，否则其将与 MQX RTOS 产生冲突。

通常，我们将 U-Boot 的起始地址设置为与 Linux 内核起始地址一致。例如，如果 Linux 内核起始地址为 0x81000000，那么我们应将 U-Boot 起始地址修改为 0x81000000。

修改 U-Boot 的起始地址非常容易。只需在/include/configs/mx6sxsabresd.h 中修改物理存储器映射值，如下所示：

- 原代码：

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

- 新代码：

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

此外，您还可以修改 U-Boot 的默认环境变量，以便于从启动命令行中加载 MQX RTOS 镜像和 Linux 镜像。这些修改为可选项。
以下为设置这些环境变量的示例：

- 原代码：

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

- 新代码：

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

使用补丁来实施上述修改：

```
# 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 如何运行演示例程

7.1 前提条件

无论 MQX RTOS 从何种存储器上运行，演示例程都应满足以下要求：

- 应具有用于 i.MX 6SoloX 的 uboot.imx。该 U-Boot 用于加载 dtb、zImage 和 MQX 应用程序镜像到 RAM，并启动 MQX 应用程序和 Linux 内核。
- 应具有用于 i.MX 6SoloX 的 imx6sx-sdb-m4.dtb：该 dtb 应能防止 Cortex-A9 和 Cortex-M4 之间的资源冲突。
- 应具有 mutex_imx6sx_sdb_m4.bin。这是一个 MQX 应用程序演示镜像。
- 应具有 zImage。这是一个 Linux 内核演示镜像。

关于如何启动 Linux OS 以及如何运行 MQX 应用程序的更多信息，请参见 *i.MX Linux User's Guide* (IMXLUG)。

注

以下步骤基于 U-Boot。该 U-Boot 镜像演示为最少修改版本，未在源代码中修改某些可选的环境变量。因此，应在运行时修改这些环境变量。如果您已在源代码中修改了相关的环境变量并重新编译，则可以跳过这些步骤。

7.2 在 DDR 上运行 MQX RTOS

假设已准备好用于从 SD4 卡槽启动的 SD/MMC 镜像，并已将 mutex.bin、zImage 和 imx6sx-sdb-m4.dtb 复制到 SD 卡上的 FAT 文件系统中。执行以下步骤：

1. 将 MQX 应用程序二进制文件加载到 DDR。

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

2. 清除 U-Boot 数据缓存。

```
dcache flush
```

3. 在 DDR 上运行 MQX 应用程序。

```
bootaux 0x80000000
```

4. 运行 Linux 内核。

```
run bootcmd
```

7.3 在 OCRAM 上运行 MQX RTOS

假设已准备好用于从 SD4 卡槽启动的 SD/MMC 镜像，并已将 mutex.bin、zImage 和 imx6sx-sdb-m4.dtb 复制到 SD 卡上的 FAT 文件系统中。执行以下步骤：

1. 将 MQX 应用程序二进制文件加载到 OCRAM。

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

2. 清除 U-Boot 数据缓存。

   ```
   dcache flush
   ```

3. 在 OCRAM 上运行 MQX 应用程序。

   ```
   bootaux 0x00901000
   ```

4. 启动 Linux 内核并绑定文件系统。

   ```
   run bootcmd
   ```

8 低功耗

在某些情况下，您可能需要降低整个系统的功耗。为了满足这些低功耗应用的需求，飞思卡尔实现了一些底层电源管理驱动程序。

对于 i.MX 6SoloX，有两种情况：

- 在 Cortex-M4 处于低功耗空闲模式时，挂起 Linux OS。

  在此情况下，MQX RTOS 可能未运行，而 Cortex-M4 处于低功耗空闲模式。Linux OS 的挂起模式映射如下：

  - **Standby**：映射到 STOP 模式，可以大大节省功耗，因为除了 ARM 内核和处于自刷新模式以保留内容的存储器外，系统中的所有其他模块均进入低功耗状态。
  - **Mem** (suspend to RAM)：映射到 DORMANT 模式， 可以大大节省功耗，因为除了处于自刷新模式以保留内容的存储器外，系统中的所有其他模块均进入低功耗状态。

- 在 Cortex-M4 运行时，挂起 Linux OS。

  在此情况下，MQX RTOS 正在运行，Cortex-M4 不处于低功耗空闲模式。此时无论是 Standby 还是 Mem，它们均映射到 WAIT 模式，以确保 Cortex-M4 时钟处于活动状态。

欲了解关于 Linux OS 低功耗模式的更多信息，请参见 *i.MX 6 Linux Reference Manual* (IMX6LXRM)。

欲了解关于 i.MX 6SoloX 低功耗模式的更多信息，请参见 *i.MX 6SoloX Applications Processor Reference Manual* (IMX6SXRM)。

9 对系统的影响与作用

支持该功能使得 Cortex-M4 可以在 RAM 存储器（OCRAM 或 DDR）上运行 MQX RTOS，而不仅仅局限于 QuadSPI NOR Flash。其对系统的影响与作用如下：

- 当 MQX 应用程序镜像加载到 DDR 或 OCRAM 时，虽然其访问速度比加载到 NOR Flash 快，但是由于这些 RAM 不是非易失性存储器，在每次启动时都需要加载镜像。这增加了系统的启动时间。
- 当 MQX RTOS 在 DDR 或 OCRAM 上运行时，可以提升 MQX 的性能，因为访问 RAM 的速度比访问 Flash 快。
- 当 MQX RTOS 在 DDR 上运行时，Cortex-M4 与 Cortex-A9 共享 DDR 带宽。这就存在降低 Linux 性能的风险，因而不适用于 DDR 带宽占用型应用。
- 当 MQX RTOS 在 DDR 上运行时，由于 Linux 内核的起始地址必须为 16 MB 的整数倍，因而给 MQX RTOS 保留的最小 DDR 空间为 16 MB。这样 MQX 镜像未占用的保留空间变为不可用，从而造成了 DDR 存储器空间的浪费。

10 修订历史记录

表 4 修订历史记录

修订版本号	日期	重大变更
1	05/2015	初始版本

How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

Document Number: AN5127
Rev. 1, 05/2015