Document Number: AN4939
Rev 0, 04/2014

MPC5744P 软件启动和优化

作者: Jamaal Fraser

内容

1 简介 ... 1
2 概述 ... 2
3 启动代码 ... 2
4 MCU 优化 ... 14
5 结论 ... 16
A 代码 ... 16

1 简介

本应用笔记介绍 MPC5744P 32 位 Power Architecture®汽车微控制器的建议软件启动程序。包括启动 Power Architecture 内核、存储器保护单元(MPU)、时钟频率(PLL)、看门狗定时器、Flash 存储器控制器和内部静态 RAM。提供这些模块的建议配置设置旨在优化系统性能。

MPC5744P 是一款 32 位 Power Architecture 微控制器，可用于底盘和安全应用以及其它要求高汽车安全完整性等级(SIL)的应用。MPC5744P 的主机处理器内核是一个 e200 系列兼容 Power Architecture 内核的 CPU。z425n3 双内核具有极高的效率——高性能、极低功耗——最大工作频率为 200MHz。该处理器系列采用低成本的 PowerISA 2.06 架构。提供嵌入式浮点(EFPU2)辅助处理单元(APU)，通过通用寄存器(GPRs)支持实时单精度嵌入式数字运算。提供轻量级信号处理扩展(LSP) APU，通过 GPR 支持实时 SIMD 定点嵌入式数字运算。内核中执行的所有算术指令均针对 GPR 中的数据进行操作。z425n3 内核采用 VLE (可变长度编码) ISA，提供更佳的代码密度。有关 VLE ISA 的文档另请参考 PowerISA 2.06。注意，不直接支持基础版 PowerISA 2.06 固定长度 32 位指令集。

MPC5744P 具有两级存储体系结构、8 K 指令、4 K 数据高速缓存和 384 KB 片上 SRAM。提供 2.5 MB 内部 Flash 存储器。

© 2014 Freescale Semiconductor, Inc.

Review Draft

图 1. MPC5744P 结构框图

2 概述

讨论嵌入式软件应用程序结构时，有几个方面是需要考虑的。首先是它的执行方式。应用程序可以存储在内部 Flash 存储器中，或者从外部器件（比如调试器）下载，或者通过串行通信链路下载。这样会影响初始化过程中的某些步骤，这些是需要注意的。

3 启动代码

初始化程序的第一部分从复位向量或程序入口点开始，执行极少的配置，以准备随后的 C 代码运行环境。此阶段的另一个目标是优化启动程序的执行时间。这包括以特定顺序执行某些初始化步骤：

1. 复位配置
2. 初始化内核寄存器
3. 禁用看门狗定时器
4. 编程设置 PLL
5. 初始化 SRAM
6. 配置内核存储器保护单元
7. 使能指令和数据高速缓存

8. 初始化 C 语言运行时环境

3.1 复位配置

器件复位后，有多种方式开始执行软件。它们通过外部引脚和器件状态控制。采用下列顺序:

- 如果 FAB（强制备选引导模式）引脚设为在串行模式下引导，则可以强制器件进入备选引导加载程序模式。备选引导模式类型根据 ABS（备选引导选择器）引脚选择，并由引导访问模块(BAM)控制，参见表 1
- 如果未设置 FAB，那么系统状态和配置模块(SSCM)便搜索 Flash，并尝试识别带有效引导签名的 Flash 存储器扇区。
- 如果没有任何 Flash 存储器扇区含有有效引导签名，那么器件将进入静态模式。静态模式意味着器件进入低功耗模式 SAFE，并且处理器执行等待指令。

表 1. 硬件配置

FAB	ABS	待机-RAM 引导标志	引导 ID	引导模式
1	00	0	-	串行引导 SCI
1	01	0	-	串行引导 CAN
0	-	0	有效	单芯片
0	-	0	未找到	静态模式

通过 JTAG 或 Nexus 端口连接硬件调试器时，可以绕过标准引导程序。调试器可以通过调试接口将软件下载至 RAM 或 Flash，并指定执行的起始位置。这种情况下，可以使用配置脚本，通过调试器来完成大部分底层设备初始工作。

本应用笔记重点讨论内部 Flash 引导的情况，因为其在应用代码中包含了全部的初始化任务操作。在任何上电、外部或内部复位事件期间，除了软件复位，SSCM 开始在内部 Flash 存储器中，以表 2 中的预定义地址之一搜索有效复位配置半字(RCHW)。

表 2. 内部 Flash 中可能的 RCHW 位置

引导搜索顺序	地址	块大小
第 1 个	0x00F9_8000	16 K
第 2 个	0x00F9_C000	16 K
第 3 个	0x00FA_0000	64 K
第 4 个	0x00FB_0000	64 K
第 5 个	0x0100_0000	256 K
第 6 个	0x0104_0000	256 K
第 7 个	0x0108_0000	256 K
第 8 个	0x010C_0000	256 K

RCHW 是控制位集合，可指定复位后的最小 MCU 配置。如果未找到有效 RCHW，则 MCU 进入静态模式。RCHW 格式见下文表 3 :

表 3. 复位配置半字

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	VLE	引导识别符							

下一页继续介绍此表...

表 3. 复位配置半字 (继续)

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

RCHW 占据了位于引导位置的前 32 位内部存储器字的 16 个最重要位。随后 32 位含有引导向量地址。应用 RCHW 之后，SSCM 会转向该引导向量。软件初始化期间，链接器指令文件中的这些 32 位位置的保留空间如下：

```
MEMORY
{
    flash_rcw : org = FLASH_BASE_ADDR,    len = 0x8
    …
}

SECTIONS
{
    .rcw              : {} > flash_rcw
    …
}
```

在初始化代码文件中，这两个位置通过有效 RCHW 编码和代码入口点的起始地址符生成。

```
.section .rcw
    .LONG 0x015A0000     # RCHW
    .LONG _start         # Code entry point
```

调试时，若不执行 SSCM，则不使用 RCHW。

3.2 初始化内核寄存器

MPC5744P 的内核需要在使用寄存器之前对其进行初始化，否则两个内核将包含不同的随机数据。如果是这种情况，当数值存储到存储器中时（比如堆栈）就会导致锁步错误。

```
    #-------------------------------------------------------#
    # Initialize Core Registers                             #
    #-------------------------------------------------------#
    # GPR's 0-31
    e_li    r0, 0
    e_li    r1, 0
    e_li    r2, 0
    e_li    r3, 0
    e_li    r4, 0
    e_li    r5, 0
    e_li    r6, 0
    e_li    r7, 0
    e_li    r8, 0
    e_li    r9, 0
    e_li    r10, 0
    e_li    r11, 0
    e_li    r12, 0
    e_li    r13, 0
    e_li    r14, 0
    e_li    r15, 0
    e_li    r16, 0
    e_li    r17, 0
    e_li    r18, 0
    e_li    r19, 0
    e_li    r20, 0
    e_li    r21, 0
    e_li    r22, 0
    e_li    r23, 0
    e_li    r24, 0
```

```
e_li    r25, 0
e_li    r26, 0
e_li    r27, 0
e_li    r28, 0
e_li    r29, 0
e_li    r30, 0
e_li    r31, 0

# Init any other CPU register which might be stacked (before being used).

mtspr   1, r1                                  # XER
mtcrf   0xFF, r1
mtspr   CTR,  r1
mtspr   SPRG0, r1
mtspr   SPRG1, r1
mtspr   SPRG2, r1
mtspr   SPRG3, r1
mtspr   SRR0, r1
mtspr   SRR1, r1
mtspr   CSRR0, r1
mtspr   CSRR1, r1
mtspr   MCSRR0, r1
mtspr   MCSRR1, r1
mtspr   DEAR, r1
mtspr   IVPR, r1
mtspr   USPRG0, r1
mtspr   62, r1                                 # ESR
mtspr   8,r31                                  # LR
```

3.3 看门狗

本例中禁用软件看门狗定时器(SWT)，因此不会干扰应用调试会话。如果使能了 SWT，那么在初始化过程中可能会有某处需要看门狗服务，具体取决于看门狗的超时周期。

```
#*********************** Disable Software Watchdog (SWT) ****************
e_lis   r4, 0xFC05
e_or2i  r4, 0x0000

e_li    r3, 0xC520
e_stw   r3, 0x10(r4)

e_li    r3, 0xD928
e_stw   r3, 0x10(r4)

e_lis   r3, 0xFF00

e_or2i  r3, 0x010A
e_stw   r3, 0(r4)
```

3.4 编程设置 PLL

双 PLL 数字接口(PLLDIG)模块通过双 PLL 系统（由 PLL0 和 PLL1 模拟模块以及数字接口组成）提供用户接口和控制。两个模拟 PLL 模块级联，并将 PLL0 的 PHI1 输出馈入 PLL1 的时钟输入。双 PLL 架构的一个重要特性是能够从 PLL0 PHI 输出驱动外设，此输出未经调制，独立于内核时钟频率。内核和平台时钟由 PLL1 驱动。参见以下的图 2 。

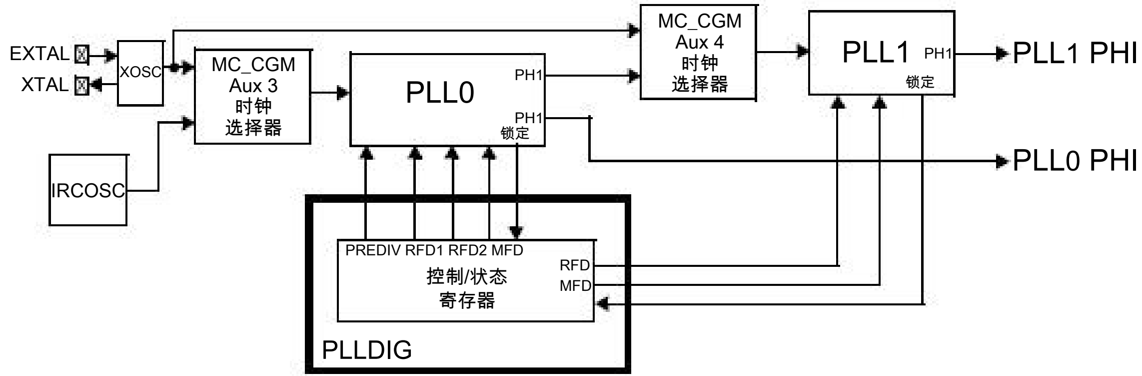

图 2. 双 PLL 数字接口结构框图

PLL0 是主 PLL，输出非频率调制时钟，通常用于 MPC5744P 模块，也可作为 PLL1 的参考频率。PLL1 是频率调制 PLL (FMPLL)，通常用于驱动系统时钟。

- PLL0_PHI - PLL0 主输出，用于驱动模块
- PLL0_PHI1 - PLL0 副输出，可用作 PLL1 时钟源
- PLL1_PHI - PLL1 输出，用作主系统时钟源。

PLLDIG 可采用外部振荡器(XOSC)、16 MHz 内部振荡器(IRCOSC)或直接通过 EXTAL 引脚提供参考频率。时钟生成模块(MC_CGM)控制 PLLDIG 输入和输出的全部多路复用。PLLDIG 的输出频率由 PLL0DV、PLL1DV 和 PLL1FD 寄存器控制，并根据下式计算：

$$f_{\mathrm{pll0_phi}} = f_{\mathrm{pll0_ref}} \times \frac{\mathrm{PLL0DV[MFD]}}{\mathrm{PLL0DV[PREDIV]} \times \mathrm{PLL0DV[RFDPHI]}}$$

$$f_{\mathrm{pll0_phi1}} = f_{\mathrm{pll0_ref}} \times \frac{\mathrm{PLL0DV[MFD]}}{\mathrm{PLL0DV[PREDIV]} \times \mathrm{PLL0DV[RFDPHI1]}}$$

$$f_{\mathrm{pll1_phi}} = f_{\mathrm{pll1_ref}} \times \left(\frac{\mathrm{PLL1DV[MFD]} + \frac{\mathrm{PLL1FD[FRCDIV]}}{2^{12}}}{2 \times \mathrm{PLL1DV[RFDPHI]}} \right)$$

有关各 PLL 的输入和输出参数，请参见表 4 和表 5 。

表 4. PLL0 基本参数

参数	最小值	最大值	单位
PLL0 输入时钟	8	40	MHz
PLL0 VCO 频率	600	1250	MHz
PLL0_PHI 频率	4.76	-625	MHz
PLL0_PHI1 频率	20	156	MHz
PLL0 锁定时间	-	100	μs

表 5. PLL1 基本参数

参数	最小值	最大值	单位
PLL1 输入时钟	38	78	MHz
PLL1 VCO 频率	600	1250	MHz
PLL1_PHI 频率	4.76	-625	MHz
PLL1 锁定时间	-	100	μs

如需获得最大性能，通常将输出设置如下:

- PLL0_PHI = 160 MHz
- PLL0_PHI1 = 40 MHz
- PLL1_PHI = 200 MHz

如需获得这些频率，请参见表 6 中关于 PLL0DV、PLL1DV 和 PLL1FD 寄存器不同位字段的示例值。

表 6. PLL 设置示例

参考	时钟	PREDIV	MFD	RFDPHI	FRCDIV
40 MHz XOSC	PLL0_PHI	1	8	2	na
40 MHz XOSC	PLL0_PHI1			8	na
40 MHz XOSC	PLL1_PHI	na	20	2	0

根据 DRUN 模式配置，退出复位的 PLL0 和 PLL1 是禁用的。下文给出初始化 PLLDIG 的步骤。

1. 配置 PLL0
   a. PLL0 禁用时，编程设置 PLL0 时钟源
   b. 向 PLL0DV 寄存器中编程设置适当值
   c. 开启 XOSC 和 PLL0
   d. 等待完成模式转换
2. 配置 PLL1
   a. PLL1 禁用时，编程设置 PLL1 时钟源
   b. 向 PLL1DV 寄存器中编程设置适当值
   c. 开启 PLL1
   d. 等待完成模式转换

下文示例设置 PLLDIG，生成上文所述的时钟。

```
#********************************* Program PLL *****************************

# Program PLL0 clock source
e_lis    r5, 0xFFFB                            # MC_CGM.AC3_SC address
e_or2i   r5, 0x0860

e_lis    r4, 0x0100                            # SELCTL=1, XOSC source of PLL0
e_or2i   r4, 0x0000

e_stw    r4, 0x0(r5)                           # Store MC_CGM.AC3_SC

# Program PLL1 clock course
e_lis    r4, 0x0100                            # SELCTL=1, XOSC source for PLL1
e_or2i   r4, 0x0000

e_stw    r4, 0x20(r5)                          # Store MC_CGM.AC4_SC

# Program PLL0 settings
e_lis    r3, 0xFFFB                            # PLLDIG base address
e_or2i   r3, 0x0100

e_lis    r4, 0x4002                            # RFDPHI1=8, PFDPHI=2
e_or2i   r4, 0x1008                            # PREDIV=1, MFD=8

e_stw    r4, 0x8(r3)                           # Store PLLDIG.PLL0DV

# Enable XOSC and PLL0
e_lis    r5, 0xFFFB                            # MC_ME base address
e_or2i   r5, 0x8000

e_lis    r4, 0x0013                            # Enable XOSC and PLL0 in DRUN mode and
```

```
    e_or2i   r4, 0x0072                          # select PLL0 as SYS_CLK

    e_stw    r4, 0x2C(r5)                        # Store MC_ME.DRUN_MC.R

    e_lis    r6, 0x3000                          # Load Mode & Key
    e_or2i   r6, 0x5AF0

    e_lis    r7, 0x3000                          # Load Mode & Key inverted
    e_or2i   r7, 0xA50F

    e_stw    r6, 4(r5)                           # Store MC_ME.MCTL.R
    e_stw    r7, 4(r5)                           # Store MC_ME.MCTL.R

    e_lis    r24, 0x0800                         # Load mask for MC_ME.GS.MTRANS
    e_lis    r25, 0x3000                         # Load mask for MC_ME.GS.S_CURRENT_MODE
mode_trans0:
    e_lwz    r4, 0(r5)                           # Load MC_ME.GS register
    se_and.  r24, r4                             # M_TRANS=0, transition complete
    e_bne    mode_trans0
    se_and.  r25, r4                             # Check that are in DRUN mode
    e_beq    mode_trans0

    # Program PLL1 settings
    e_lis    r4, 0x0002                          # RFDPHI=2
    e_or2i   r4, 0x0014                          # MFD=20

    e_stw    r4, 0x28(r3)                        # Store PLLDIG.PLL1DV

    # Enable PLL1 and select as SYS_CLK
    e_lis    r4, 0x0013                          # Enable PLL1 in DRUN mode and
    e_or2i   r4, 0x00F4                          # select PLL1 as SYS_CLK

    e_stw    r4, 0x2C(r5)                        # Store MC_ME.DRUN_MC.R

    e_lis    r6, 0x3000                          # Load Mode & Key
    e_or2i   r6, 0x5AF0

    e_lis    r7, 0x3000                          # Load Mode & Key inverted
    e_or2i   r7, 0xA50F

    e_stw    r6, 4(r5)                           # Store MC_ME.MCTL.R
    e_stw    r7, 4(r5)                           # Store MC_ME.MCTL.R

    e_lis    r24, 0x0800                         # Load mask for MC_ME.GS.MTRANS
    e_lis    r25, 0x3000                         # Load mask for MC_ME.GS.S_CURRENT_MODE
mode_trans1:
    e_lwz    r4, 0(r5)                           # Load MC_ME.GS register
    se_and.  r24, r4                             # M_TRANS=0. transition complete
    e_bne    mode_trans1
    se_and.  r25, r4                             # Check that are in DRUN mode
    e_beq    mode_trans1
```

3.5 存储器保护单元(MPU)

内核 MPU 具有下列特性:

- 24 个进入区域描述符表支持 6 个任意尺寸指令存储器区域、12 个任意尺寸数据存储器区域和 6 个额外的任意尺寸区域，可编程为指令区域或数据存储器区域
- 区域 0-5 指令、6-17 数据、18-23 共享。
- 可按区域设置访问权限和存储器属性
- 进程 ID 标识，提供位屏蔽 TID 值
- 在范围比较中可屏蔽高位地址位
- 针对选定的访问类型可绕过检查权限
- 具有一次写入逻辑以实现进入保护

- 硬件提供 Flash 无效支持和进入无效保护控制
- 能够有选择地利用区域描述符生成调试事件和观察点
- 通过 **mpure** 和 **mpuwe** 指令进行软件管理

默认禁用内核 MPU，用户可以选择保留禁用状态。记住，一旦使能内核 MPU，则进行任何访问都会检查区域表，除非在 MPU0CSR0 寄存器中绕过该访问类型的 MPU 保护。更多详情请参考 MPC5744P 参考手册。

有关软件如何设置内核存储器保护单元(MPU)，请参见表 7 。

表 7. MMU 配置

区域	说明	地址	大小	属性
0	Flash - 指令	0x0040_0000	28 MB	用户和超级用户读取、写入、执行。可高速缓存
1	SRAM - 指令	0x4000_0000	384 KB	用户和超级用户读取、写入、执行。可高速缓存
6	SRAM - 数据	0x4000_0000	384 KB	用户和超级用户读取、写入、执行。可高速缓存
7	Flash - 数据	0x0040_0000	28 MB	用户和超级用户读取、写入、执行。可高速缓存
8	P_BRIDGE 1/0 - 数据	0xF800_0000	128 MB	用户和超级用户读取、写入、执行。不可高速缓存，受保护

使能 MPU 后，区域设置可涵盖软件支持的任意访问类型。此代码提供了一个如何设置内核 MPU 的示例。

```
#**************************** Configure Core MPU *************************

    # Region 0 (Instruction): Internal Flash @ 0x0040_0000-0x01FF_FFFF
    e_lis    r3, 0xA100                # VALID=1, INST=1, SHD=0,
ESEL=0
    e_or2i   r3, 0x0F00                # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                  # TID=0, global
    e_lis    r3, 0x01FF
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                  # Upper Bound = 0x01FF_FFFF
    e_lis    r3, 0x0040
    e_or2i   r3, 0x0000                # Lower Bound = 0x0040_0000
    mtspr    mas3, r3
    msync                              # Synchronize since running from flash
    mpuwe
    se_isync                           # Synchronize since running from flash

    # Region 1 (Instruction): SRAM @ 0x4000_0000-0x4005_FFFF
    e_lis    r3, 0xA101                # VALID=1, INST=1, SHD=0,
ESEL=1
    e_or2i   r3, 0x0F00                # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                  # TID=0, global
    e_lis    r3, 0x4005
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                  # Upper Bound = 0x4005_FFFF
    e_lis    r3, 0x4000
    e_or2i   r3, 0x0000                # Lower Bound = 0x4000_0000
    mtspr    mas3, r3
    mpuwe
    mpusync

    # Region 6 (Data): SRAM @ 0x4000_0000-0x4005_FFFF
    e_lis    r3, 0xA000                # VALID=1, INST=0, SHD=0, ESEL=0
    e_or2i   r3, 0x0F00                # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
```

```
    e_or2i  r3, 0x0000
    mtspr   mas1, r3                # TID=0, global
    e_lis   r3, 0x4005
    e_or2i  r3, 0xFFFF
    mtspr   mas2, r3                # Upper Bound = 0x4005_FFFF
    e_lis   r3, 0x4000
    e_or2i  r3, 0x0000
    mtspr   mas3, r3                # Lower Bound = 0x4000_0000
    mpuwe
    mpusync

    # Region 7 (Data): Internal Flash @ 0x0040_0000-0x01FF_FFFF
    e_lis   r3, 0xA001              # VALID=1, INST=0, SHD=0,
ESEL=1
    e_or2i  r3, 0x0F00              # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr   mas0, r3
    e_lis   r3, 0x0000
    e_or2i  r3, 0x0000
    mtspr   mas1, r3                # TID=0, global
    e_lis   r3, 0x01FF
    e_or2i  r3, 0xFFFF
    mtspr   mas2, r3                # Upper Bound = 0x01FF_FFFF
    e_lis   r3, 0x0040
    e_or2i  r3, 0x0000              # Lower Bound = 0x0040_0000
    mtspr   mas3, r3
    mpuwe
    mpusync

    # Region 8 (Data): PBRIDGE1/0 @ 0xF800_0000-0xFFFF_FFFF
    e_lis   r3, 0xA002              # VALID=1, INST=0, SHD=0,
ESEL=2
    e_or2i  r3, 0x0F09              # UW=1, SW=1, UX/UR=1, SX/SR=1, non-Cacheable, guarded
    mtspr   mas0, r3
    e_lis   r3, 0x0000
    e_or2i  r3, 0x0000
    mtspr   mas1, r3                # TID=0, global
    e_lis   r3, 0xFFFF
    e_or2i  r3, 0xFFFF
    mtspr   mas2, r3                # Upper Bound = 0xFFFF_FFFF
    e_lis   r3, 0xF800
    e_or2i  r3, 0x0000              # Lower Bound = 0xF800_0000
    mtspr   mas3, r3
    mpuwe
    mpusync

    e_lis   r3, 0x0000
    e_or2i  r3, 0x0001
    mtspr   1014, r3                # Enable MPU
```

注意，在该示例中，对于区域 0 而言，mpuwe 前面是 msync，后面是 se_isync。由于执行的代码来自修改后的区域，因此需执行此同步步骤。

3.6 使能高速缓存

内核指令和数据高速缓存通过 L1 高速缓存控制和状态寄存器 0 与 1 (L1CSR0 和 L1CSR1) 使能。指令高速缓存通过设置 L1CSR1 中的 ICINV 和 ICE 位变为无效或使能。数据高速缓存通过设置 L1CSR0 中的 DCINV 和 DCE 使能。高速缓存无效操作需要一定时间，可中断或中止。由于此时启动程序中不涉及其它任务，因此不会将其中断或中止。用户可以进行置位操作，然后继续。

```
#*********************** Invalidate and enable caches ********************

# Instruction cache (I-CACHE)
e_li        r5, 0x3           # Start instruction cache invalidation and enable
mtspr       1011, r5          # Set L1CSR1.ICINV & ICE bits
```

```
    # Data cache (D-CACHE)
    e_li        r5, 0x3         # Start data cache invalidation and enable
    mtspr       1010, r5        # Set L1CSR0.DCINV & DCE bits
```

下列代码是一个更为鲁棒的高速缓存使能程序，如有需要可以使用。此代码检查并确保无效操作顺利完成，如果未完成便重试此操作，然后再使能高速缓存。此代码可在中断使能时使用，只要正确处理和清除这些中断即可。如果不中断就无法完成无效操作是由于系统中产生频繁的中断导致的，那么最好先禁用中断。

```
    #*********************** Invalidate and enable caches *********************
    # Instruction cache (I-CACHE)
__icache_cfg:
    e_li        r5, 0x2             # Start instruction cache invalidation
    mtspr       1011, r5            # Set L1CSR1.ICINV bit

    e_li        r7, 0x4
    e_li        r8, 0x2
    e_lis       r11, 0xFFFFFFFB@h
    e_or2i      r11, 0xFFFFFFFB@l

__icache_inv:
    mfspr       r9, 1011            # Read L1CSR1
    and.        r10, r7, r9         # L1CSR1.ICABT=1?, cache invalidation was aborted.
    e_beq       __icache_no_abort   # If 0, no abortion, jump to proceed.
    and.        r10, r11, r9
    mtspr       1011, r10           # Clear the L1CSR1.ICABT bit.
    e_b         __icache_cfg        # Branch back to retry.

__icache_no_abort:
    and.        r10, r8, r9         # L1CSR1.ICINV=0?, cache invalidation completed.
    e_bne       __icache_inv        # jump back to wait and re-check ICABT bit.

    mfspr       r5, 1011            # Read L1CSR0
    e_ori       r5, r5, 0x0001
    se_isync                        # wait for all previous instructions to complete
    msync                           # wait for data memory accesses coherency
    mtspr       1011, r5            # Set L1CSR1.ICE to enable data cache

    # Data cache (D-CACHE)
__dcache_cfg:
    e_li        r5, 0x2
    mtspr       1010, r5            # Set L1CSR0.DCINV bit. Start data cache invalidation

    e_li        r7, 0x4
    e_li        r8, 0x2
    e_lis       r11, 0xFFFFFFFB@h
    e_or2i      r11, 0xFFFFFFFB@l

__dcache_inv:
    mfspr       r9, 1010            # Read L1CSRO
    and.        r10, r7, r9         # L1CSR0.DCABT=1?, cache invalidation was aborted.
    e_beq       __dcache_no_abort   # If 0, no abortion, jump to proceed.
    and.        r10, r11, r9
    mtspr       1010, r10           # Clear the L1CSR0.DCABT bit.
    e_b         __dcache_cfg        # Branch back to retry invalidation of data cache.

__dcache_no_abort:
    and.       r10, r8, r9          # L1CSR0.DCINV=0?, cache invalidation completed.
    e_bne      __dcache_inv         # jump back to wait and re-check DCABT bit.

    mfspr       r5, 1010            # Read L1CSR0
    e_ori       r5, r5, 0x0001
    se_isync                        # wait for all previous instructions to complete
    msync                           # wait for data memory accesses coherency
    mtspr       1010, r5            # Set L1CSR0.DCE to enable data cache
```

3.7 SRAM 初始化

内部 SRAM 具有纠错码(ECC)功能。由于器件开启后这些 ECC 位可能含有随机数据，因此所有的 SRAM 位置都必须初始化后由应用码读取。针对整个 SRAM 块执行 64 位写操作便可完成初始化。此时，写入的数值并不重要；因此可以通过环路迭代并使用“存储多个字”指令来写入 32 个通用寄存器。MPC5744P 内核具有 384 KB 系统 SRAM 和 64 KB 本地 SRAM。

```
#*************************** Initialize SRAM *******************************
    # Initialize System SRAM
    # Store number of 128Byte (32GPRs) segments in Counter
    e_lis       r5, __SRAM_SIZE@h                 # Initialize r5 to size of SRAM (Bytes)
    e_or2i      r5, __SRAM_SIZE@l
    e_srwi      r5, r5, 0x7                       # Divide SRAM size by 128
    mtctr       r5                                # Move to counter for use with "bdnz"

    # Base Address of the internal SRAM
    e_lis       r5, __SRAM_BASE_ADDR@h
    e_or2i      r5, __SRAM_BASE_ADDR@l

    # Fill SRAM with writes of 32GPRs
sram_loop:
    e_stmw      r0,0(r5)                          # Write all 32 registers to SRAM
    e_addi      r5,r5,128                         # Increment the RAM pointer to next 128bytes
    e_bdnz      sram_loop                         # Loop for all of SRAM

    # Initialize Local SRAM
    # Store number of 128 Byte (32GPRs) segments in Counter
    e_lis       r5, __LOCAL_SRAM_SIZE@h           # Initialize r5 to size of SRAM (Bytes)
    e_or2i      r5, __LOCAL_SRAM_SIZE@l
    e_srwi      r5, r5, 0x7                       # Divide SRAM size by 128
    mtctr       r5                                # Move to counter for use with "bdnz"

    # Base Address of the Local SRAM
    e_lis       r5, __LOCAL_SRAM_BASE_ADDR@h
    e_or2i      r5, __LOCAL_SRAM_BASE_ADDR@l

    # Fill Local SRAM with writes of 32GPRs
lsram_loop:
    e_stmw      r0,0(r5)                          # Write all 32 registers to SRAM
    e_addi      r5,r5,128                         # Increment the RAM pointer to next 128bytes
    e_bdnz      lsram_loop                        # Loop for all of SRAM
```

3.8 C 语言运行时寄存器设置

Power Architecture 增强型应用程序二进制接口(EABI)指定某些通用寄存器具有 C 代码执行时的特殊含义。此时在初始化代码中设置堆栈指针、小数据和小数据 2 基址指针。符合 EABI 的 C 语言编译器将生成代码，并在后续使用这些指针。

```
    e_lis   r1, __SP_INIT@h     # Initialize stack pointer r1 to
    e_or2i  r1, __SP_INIT@l     # value in linker command file.

    e_lis   r13, _SDA_BASE_@h   # Initialize r13 to sdata base
    e_or2i  r13, _SDA_BASE_@l   # (provided by linker).

    e_lis   r2, _SDA2_BASE_@h   # Initialize r2 to sdata2 base
    e_or2i  r2, _SDA2_BASE_@l   # (provided by linker).
```

如上文注释中所述，这些数值在该项目的链接器命令文件中定义。

```
    __DATA_SRAM_ADDR  = ADDR(.data);
    __SDATA_SRAM_ADDR = ADDR(.sdata);

    __DATA_SIZE    = SIZEOF(.data);
```

```
__SDATA_SIZE  = SIZEOF(.sdata);

__DATA_ROM_ADDR  = ADDR(.ROM.data);
__SDATA_ROM_ADDR = ADDR(.ROM.sdata);
```

在内部 Flash 引导情况下，这些数值将用于从 Flash 中复制初始化数据至 SRAM，但 SRAM 必须首先初始化。

此运行时设置步骤可能有所不同，具体取决于编译器。请参考编译器文档。初始化 C 语言标准库还可能需要其它设置。

3.9 复制初始化数据

从 Flash 引导时，Flash 中存储的程序镜像将含有 C 语言编译器和链接器创建的各种数据段码。必须将初始化读-写数据从只读 Flash 复制到可读写 SRAM，然后再通过分支进入 C 语言主程序。

下文示例假定初始化数据数值以未经压缩的形式存储在 Flash 中。某些编译器会在 Flash 镜像中压缩这些数据，以节省空间。本应用笔记随附的示例代码调用编译器特定的_start 程序实现 C 语言运行时的设置和数据复制。本示例供参考。

```
    #************ Load Initialized Data Values from Flash into RAM ************
    # Initialized Data - ".data"
DATACOPY:
    e_lis       r9, __DATA_SIZE@ha      # Load upper SRAM load size
    e_or2i      r9, __DATA_SIZE@l       # Load lower SRAM load
size
    e_cmp16i    r9,0                    # Compare to see if equal to
0
    e_beq       SDATACOPY               # Exit cfg_ROMCPY if size is zero

    mtctr       r9                      # Store no. of bytes to be moved in counter

    e_lis       r10, __DATA_ROM_ADDR@h  # Load address of first SRAM load into R10
    e_or2i      r10, __DATA_ROM_ADDR@l  # Load lower address of SRAM load into R10
    e_subi      r10,r10, 1              # Decrement address to prepare for ROMCPYLOOP

    e_lis       r5, __DATA_SRAM_ADDR@h  # Load upper SRAM address
    e_or2i      r5, __DATA_SRAM_ADDR@l  # Load lower SRAM address
    e_subi      r5, r5, 1               # Decrement address to prepare for ROMCPYLOOP

DATACPYLOOP:
    e_lbzu      r4, 1(r10)              # Load data byte, incrementing ROM address
    e_stbu      r4, 1(r5)               # Store data byte into SRAM, update SRAM address
    e_bdnz      DATACPYLOOP             # Branch if more bytes to load from ROM

    # Small Initialized Data - ".sdata"
SDATACOPY:
    e_lis       r9, __SDATA_SIZE@ha     # Load upper SRAM load size
    e_or2i      r9, __SDATA_SIZE@l      # Load lower SRAM load size
    e_cmp16i    r9,0                    # Compare to see if equal to
0
    e_beq       ROMCPYEND               # Exit cfg_ROMCPY if size is zero

    mtctr       r9                      # Store no. of bytes to be moved in counter

    e_lis       r10, __SDATA_ROM_ADDR@h # Load address of first SRAM load into R10
    e_or2i      r10, __SDATA_ROM_ADDR@l # Load lower address of SRAM load into R10
    e_subi      r10,r10, 1              # Decrement address to prepare for ROMCPYLOOP

    e_lis       r5, __SDATA_SRAM_ADDR@h # Load upper SRAM address into R5 (from linker file)
    e_or2i      r5, __SDATA_SRAM_ADDR@l # Load lower SRAM address into R5 (from linker file)
    e_subi      r5, r5, 1               # Decrement address to prepare for ROMCPYLOOP

SDATACPYLOOP:
    e_lbzu      r4, 1(r10)              # Load data byte,incrementing ROM address
    e_stbu      r4, 1(r5)               # Store data byte into SRAM, update SRAM address
```

```
    e_bdnz      SDATACPYLOOP            # Branch if more bytes to load from ROM

ROMCPYEND:
```

3.10 其他

在实际操作前，MPC5744P SOC 另外还有一些重要的方面需要初始化。

- 模式控制
- 外设桥

第一个是器件的模式控制。此功能的详细说明和此模块的编程超出了本应用笔记的讨论范围，但本文会提供一些基本信息和设置。更多信息请参考 MPC5744P 参考手册。

MPC5744P 具有 10 种不同的模式。

- RESET
- DRUN
- SAFE
- TEST
- RUN0...3
- HALT0
- STOP0

系统模式包括：RESET、DRUN、SAFE 和 TEST。这些模式旨在为系统的配置和监控提供方便。用户模式是 RUN0...3、HALT0 和 STOP0 等模式，可配置为满足应用程序对于能源管理和可用处理能力的要求。默认情况下，复位后器件将在 DRUN 模式下运行，冻结所有外设。有必要执行一次寄存器写操作，以便开启所有外设执行。

```
/* Peripheral ON in every run mode */
MC_ME.RUN_PC[0].R = 0x000000FE;
```

其它操作包括外设桥(AIPS)。外设桥将交叉开关接口转换为一个能访问片上大多数从机外设的接口。复位后，仅内核使能，可读写访问与两个外设桥连接的模块。以下两个寄存器写操作可使能所有外设桥主机、内核、DMA、FlexRay、SIPI 和以太网。

```
/* Enable all PBridge Masters for Reads, Writes, and Master Privilege Mode. */
AIPS_0.MPRA.R = 0x70777700;
AIPS_1.MPRA.R = 0x70777700;
```

4 MCU 优化

本节将讨论以下各部分的优化可能性：

- Flash 控制器
- 分支目标缓冲器
- 交叉开关

4.1 Flash 优化

片上 Flash 阵列控制器退出复位状态具有故障安全设置。等待状态设为最大值，并禁用预取、读缓冲和流水线等操作。等待状态和完美的配置通常可根据工作频率并使用 MPC5744P 数据手册中的信息进行优化。预取和读缓冲一般可根据 Flash 外执行的代码类型而优化。可以修改下列代码，为 Flash 阵列的平台 Flash 配置寄存器 1(PFLASH_PFCR1)选择适当的值。

下文示例针对 200 MHz 工作频率设置，可实现下列优化：

- 使能行读取缓冲器
- 根据数据手册的建议，保留读取等待状态默认值(6)
- 根据数据手册的建议，在使能访问请求之间，使能 2 个保持周期的流水线

由于该示例通过 Flash 存储器执行，需要加载指令，以便执行 PFLASH_PFCR1 寄存器更新至 SRAM 的操作，然后暂时从 SRAM 中执行。

```
    #********************* Configure Flash Wait States *************************
    # Code is copied to RAM first, then executed, to avoid executing code from flash
    # while wait states are changing.

    # Base Address of the internal SRAM
    e_lis        r5, __SRAM_BASE_ADDR@h
    e_or2i       r5, __SRAM_BASE_ADDR@l

    e_b          copy_to_ram

    # Settings for SYS_CLK of 200 MHz
reduce_flash_ws:
    e_lis        r3, 0x0000     # APC=2 pipelined access 2 cycles before prev. data valid
    e_or2i       r3, 0x4601     # RWSC=6 additional wait states, P0_BFEN=1 line buffer enabled
    e_lis        r4, 0xFC03
    e_or2i       r4, 0x0000
    e_stw        r3, 0x0(r4)
    se_isync
    msync
    se_blr

copy_to_ram:
    e_lis        r3, reduce_flash_ws@h
    e_or2i       r3, reduce_flash_ws@l
    e_lis        r4, copy_to_ram@h
    e_or2i       r4, copy_to_ram@l
    subf         r4, r3, r4
    se_mtctr     r4
    se_mtlr      r5
copy:
    e_lbz        r6, 0(r3)
    e_stb        r6, 0(r5)
    e_addi       r3, r3, 1
    e_addi       r5, r5, 1
    e_bdnz       copy
    se_isync
    msync
    se_blrl
```

4.2 分支目标缓冲器

MPC5744P Power Architecture 内核具有可使能的分支预测优化特性，通过存储分支结果，使用这些结果预测相同位置未来分支的方向的方法来改进整体性能。如需初始化分支目标缓冲器，则需要对缓冲器执行 Flash 无效操作，并使能分支预测。向分支单元控制和状态寄存器(BUCSR)执行单次写操作即可实现。

```
    #***************************** enable BTB *********************************
    e_li    r3, 0x0201
```

```
mtspr   1013, r3
se_isync
```

注

如果应用程序在此初始化步骤之后修改存储器中的指令代码，则可能需要刷新并重新初始化分支目标缓冲器，因为它可能含有之前在修改位置处存在的代码分支预测。

4.3 交叉开关

大部分情况下，交叉设置可保留其复位默认值。在了解有关应用行为以及在交叉开关上不同主设备的使用情况下，可以自定义优先级并使用相应的算法，以便获得少许性能改进。例如，采用外设总线通信时，相比 CPU 负载/存储，DMA 传输可得益于较高的优先级设置。这样可以防止在 CPU 轮询某个外设中的状态寄存器时 DMA 传输停止。然而，这是一个特例，可能并不适用于所有应用。

5 结论

本应用笔记提出了一些初始化此器件并通过复位默认值优化部分设置的具体建议。本文给出的建议仅仅是一个切入点。其它需考虑的问题包括：编译优化和高效使用系统资源，比如 DMA 和高速缓存。更多信息请查阅 MPC5744P 参考手册。

附录 A 代码

A.1 init.s 文件

```
#***************************************************************************
#   LICENSE:
#   Copyright (c) 2014 Freescale Semiconductor
#
#   Permission is hereby granted, free of charge, to any person
#   obtaining a copy of this software and associated documentation
#   files (the "Software"), to deal in the Software without
#   restriction, including without limitation the rights to use,
#   copy, modify, merge, publish, distribute, sublicense, and/or
#   sell copies of the Software, and to permit persons to whom the
#   Software is furnished to do so, subject to the following
#   conditions:
#
#   The above copyright notice and this permission notice
#   shall be included in all copies or substantial portions
#   of the Software.
#
#   THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
#   EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
#   OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
#   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
#   HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
#   WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
#   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
#   DEALINGS IN THE SOFTWARE.
#
# Composed By: Fraser, Jamaal
# Dated       : March 31, 2014
# Compiler    : Green Hills Multi
#
```

```
#
# FILE NAME: crt0_core_flash.s
#
# DESCRIPTION: This is the crt0 file for the core of the MPC5744P MCU
#
# REV   AUTHOR          DATE        DESCRIPTION OF CHANGE
# ---   -----------     ----------  ---------------------
# 0.1   D.McMenamin     27/Jun/11   Initial Version
# 0.2   B.Johnson       19/Jun/12   Modified for Panther
#                                   Removed SWT2, SWT1 initialization
# 0.3   B.Johnson       02/Dec/12   Changed SWT_CR to enable MAP0
# 0.4   J.Fraser        31/Mar/14   Added MMU and PLL init code and
#                                   modified order for optimization in regards
#                                   to MPC5744P SW StartUp App. Note.
#
#******************************************************************************

.globl    _start_core

#************************** .rcw reset config section ************************
.section .rcw
.LONG 0x015A0000                                # RCHW
.LONG _start_core                               # Code starts at _start_core
#******************************************************************************

.section .init , axv
.vle
.align 4
_start_core:

    #**************************** Init Core Registers ************************
    # The e200z4 core needs its registers initialized before they are used
    # otherwise in Lock Step mode the two cores will contain different random data.
    # If this is stored to memory (e.g. stacked) it will cause a Lock Step error.

    # GPR's 0-31
    e_li    r0, 0
    e_li    r1, 0
    e_li    r2, 0
    e_li    r3, 0
    e_li    r4, 0
    e_li    r5, 0
    e_li    r6, 0
    e_li    r7, 0
    e_li    r8, 0
    e_li    r9, 0
    e_li    r10, 0
    e_li    r11, 0
    e_li    r12, 0
    e_li    r13, 0
    e_li    r14, 0
    e_li    r15, 0
    e_li    r16, 0
    e_li    r17, 0
    e_li    r18, 0
    e_li    r19, 0
    e_li    r20, 0
    e_li    r21, 0
    e_li    r22, 0
    e_li    r23, 0
    e_li    r24, 0
    e_li    r25, 0
    e_li    r26, 0
    e_li    r27, 0
    e_li    r28, 0
    e_li    r29, 0
    e_li    r30, 0
    e_li    r31, 0
```

```
# Init any other CPU register which might be stacked (before being used).

mtspr   1, r1                           # XER
mtcrf   0xFF, r1
mtspr   CTR,  r1
mtspr   SPRG0, r1
mtspr   SPRG1, r1
mtspr   SPRG2, r1
mtspr   SPRG3, r1
mtspr   SRR0, r1
mtspr   SRR1, r1
mtspr   CSRR0, r1
mtspr   CSRR1, r1
mtspr   MCSRR0, r1
mtspr   MCSRR1, r1
mtspr   DEAR, r1
mtspr  IVPR, r1
mtspr   USPRG0, r1
mtspr   62, r1                          # ESR
mtspr   8,r31                           # LR


#************************ Disable Software Watchdog (SWT) ****************
e_lis   r4, 0xFC05
e_or2i  r4, 0x0000

e_li    r3, 0xC520
e_stw   r3, 0x10(r4)

e_li    r3, 0xD928
e_stw   r3, 0x10(r4)

e_lis   r3, 0xFF00

e_or2i  r3, 0x010A
e_stw   r3, 0(r4)


#******************************** Program PLL ****************************
# Program PLL0 clock source
e_lis   r5, 0xFFFB                      # MC_CGM.AC3_SC address
e_or2i  r5, 0x0860

e_lis   r4, 0x0100                      # SELCTL=1, XOSC source of PLL0
e_or2i  r4, 0x0000

e_stw   r4, 0x0(r5)                     # Store MC_CGM.AC3_SC

# Program PLL1 clock course
e_lis   r4, 0x0100                      # SELCTL=1, XOSC source for PLL1
e_or2i  r4, 0x0000

e_stw   r4, 0x20(r5)                    # Store MC_CGM.AC4_SC

# Program PLL0 settings
e_lis   r3, 0xFFFB                      # PLLDIG base address
e_or2i  r3, 0x0100

e_lis   r4, 0x4002                      # RFDPHI1=8, PFDPHI=2
e_or2i  r4, 0x1008                      # PREDIV=1, MFD=8

e_stw   r4, 0x8(r3)                     # Store PLLDIG.PLL0DV

# Enable XOSC and PLL0
e_lis   r5, 0xFFFB                      # MC_ME base address
e_or2i  r5, 0x8000

e_lis   r4, 0x0013                      # Enable XOSC and PLL0 in DRUN mode and
e_or2i  r4, 0x0072                      # select PLL0 as SYS_CLK
```

```
    e_stw   r4, 0x2C(r5)                   # Store MC_ME.DRUN_MC.R

    e_lis   r6, 0x3000                     # Load Mode & Key
    e_or2i  r6, 0x5AF0

    e_lis   r7, 0x3000                     # Load Mode & Key inverted
    e_or2i  r7, 0xA50F

    e_stw   r6, 4(r5)                      # Store MC_ME.MCTL.R
    e_stw   r7, 4(r5)                      # Store MC_ME.MCTL.R

    e_lis   r24, 0x0800                    # Load mask for MC_ME.GS.MTRANS
    e_lis   r25, 0x3000                    # Load mask for MC_ME.GS.S_CURRENT_MODE
mode_trans0:
    e_lwz   r4, 0(r5)                      # Load MC_ME.GS register
    se_and. r24, r4                        # Check M_TRANS bit clear signaling
transition complete
    e_bne   mode_trans0
    se_and. r25, r4                        # Check that are in DRUN mode
    e_beq   mode_trans0

    # Program PLL1 settings
    e_lis   r4, 0x0002                     # RFDPHI=2
    e_or2i  r4, 0x0014                     # MFD=20

    e_stw   r4, 0x28(r3)                   # Store PLLDIG.PLL1DV

    # Enable PLL1 and select as SYS_CLK
    e_lis   r4, 0x0013                     # Enable PLL1 in DRUN mode and
    e_or2i  r4, 0x00F4                     # select PLL1 as SYS_CLK

    e_stw   r4, 0x2C(r5)                   # Store MC_ME.DRUN_MC.R

    e_lis   r6, 0x3000                     # Load Mode & Key
    e_or2i  r6, 0x5AF0

    e_lis   r7, 0x3000                     # Load Mode & Key inverted
    e_or2i  r7, 0xA50F

    e_stw   r6, 4(r5)                      # Store MC_ME.MCTL.R
    e_stw   r7, 4(r5)                      # Store MC_ME.MCTL.R

    e_lis   r24, 0x0800                    # Load mask for MC_ME.GS.MTRANS
    e_lis   r25, 0x3000                    # Load mask for MC_ME.GS.S_CURRENT_MODE
mode_trans1:
    e_lwz   r4, 0(r5)                      # Load MC_ME.GS register
    se_and. r24, r4                        # Check M_TRANS bit clear signaling
transition complete
    e_bne   mode_trans1
    se_and. r25, r4                        # Check that are in DRUN mode
    e_beq   mode_trans1


    #*************************** Initialize SRAM ******************************
    # Initialize System SRAM
    # Store number of 128 Byte (32GPRs) segments in Counter
    e_lis       r5, __SRAM_SIZE@h          # Initialize r5 to size of SRAM (Bytes)
    e_or2i      r5, __SRAM_SIZE@l
    e_srwi      r5, r5, 0x7                # Divide SRAM size by 128
    mtctr       r5                         # Move to counter for use with "bdnz"

    # Base Address of the internal SRAM
    e_lis       r5, __SRAM_BASE_ADDR@h
    e_or2i      r5, __SRAM_BASE_ADDR@l

    # Fill SRAM with writes of 32GPRs
sram_loop:
    e_stmw      r0,0(r5)                   # Write all 32 registers to SRAM
    e_addi      r5,r5,128                  # Increment the RAM pointer to next 128bytes
    e_bdnz      sram_loop                  # Loop for all of SRAM
```

```
    # Initialize Local SRAM
    # Store number of 128 Byte (32GPRs) segments in Counter
    e_lis       r5, __LOCAL_SRAM_SIZE@h          # Initialize r5 to size of SRAM (Bytes)
    e_or2i      r5, __LOCAL_SRAM_SIZE@l
    e_srwi      r5, r5, 0x7                      # Divide SRAM size by 128
    mtctr       r5                               # Move to counter for use with "bdnz"

    # Base Address of the Local SRAM
    e_lis       r5, __LOCAL_SRAM_BASE_ADDR@h
    e_or2i      r5, __LOCAL_SRAM_BASE_ADDR@l

    # Fill Local SRAM with writes of 32GPRs
lsram_loop:
    e_stmw      r0,0(r5)                         # Write all 32 registers to SRAM
    e_addi      r5,r5,128                        # Increment the RAM pointer to next 128bytes
    e_bdnz      lsram_loop                       # Loop for all of SRAM


    #********************* Configure Flash Wait States *************************
    # Code is copied to RAM first, then executed, to avoid executing code from flash
    # while wait states are changing.

    # Base Address of the internal SRAM
    e_lis       r5, __SRAM_BASE_ADDR@h
    e_or2i      r5, __SRAM_BASE_ADDR@l

    e_b         copy_to_ram

    # Settings for SYS_CLK of 200 MHz
reduce_flash_ws:
    e_lis       r3, 0x0000                       # APC=2 pipelined access can be initiated 2
cycles before previous data is valid
    e_or2i      r3, 0x4601                       # RWSC=6 additional wait states, , P0_BFEN=1
line buffer enabled
    e_lis       r4, 0xFC03
    e_or2i      r4, 0x0000
    e_stw       r3, 0x0(r4)
    se_isync
    msync
    se_blr

copy_to_ram:
    e_lis       r3, reduce_flash_ws@h
    e_or2i      r3, reduce_flash_ws@l
    e_lis       r4, copy_to_ram@h
    e_or2i      r4, copy_to_ram@l
    subf        r4, r3, r4
    se_mtctr    r4
    se_mtlr     r5
copy:
    e_lbz       r6, 0(r3)
    e_stb       r6, 0(r5)
    e_addi      r3, r3, 1
    e_addi      r5, r5, 1
    e_bdnz      copy
    se_isync
    msync
    se_blrl


    #**************************** Configure Core MPU **************************
    # Region 0 (Instruction): Internal Flash @ 0x0040_0000-0x01FF_FFFF
    e_lis   r3, 0xA100                           # VALID=1, INST=1, SHD=0,
ESEL=0
    e_or2i  r3, 0x0F00                           # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr   mas0, r3
    e_lis   r3, 0x0000
    e_or2i  r3, 0x0000
    mtspr   mas1, r3                             # TID=0, global
```

```
    e_lis    r3, 0x01FF
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                          # Upper Bound = 0x01FF_FFFF
    e_lis    r3, 0x0040
    e_or2i   r3, 0x0000                        # Lower Bound = 0x0040_0000
    mtspr    mas3, r3
    msync                                      # Synchronize since running from flash
    mpuwe
    se_isync                                   # Synchronize since running from flash

    # Region 1 (Instruction): SRAM @ 0x4000_0000-0x4005_FFFF
    e_lis    r3, 0xA101                        # VALID=1, INST=1, SHD=0,
ESEL=1
    e_or2i   r3, 0x0F00                        # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                          # TID=0, global
    e_lis    r3, 0x4005
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                          # Upper Bound = 0x4005_FFFF
    e_lis    r3, 0x4000
    e_or2i   r3, 0x0000                        # Lower Bound = 0x4000_0000
    mtspr    mas3, r3
    mpuwe
    mpusync

    # Region 6 (Data): SRAM @ 0x4000_0000-0x4005_FFFF
    e_lis    r3, 0xA000                        # VALID=1, INST=0, SHD=0, ESEL=0
    e_or2i   r3, 0x0F00                        # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                          # TID=0, global
    e_lis    r3, 0x4005
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                          # Upper Bound = 0x4005_FFFF
    e_lis    r3, 0x4000
    e_or2i   r3, 0x0000
    mtspr    mas3, r3                          # Lower Bound = 0x4000_0000
    mpuwe
    mpusync

    # Region 7 (Data): Internal Flash @ 0x0040_0000-0x01FF_FFFF
    e_lis    r3, 0xA001                        # VALID=1, INST=0, SHD=0,
ESEL=1
    e_or2i   r3, 0x0F00                        # UW=1, SW=1, UX/UR=1, SX/SR=1, Cacheable
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                          # TID=0, global
    e_lis    r3, 0x01FF
    e_or2i   r3, 0xFFFF
    mtspr    mas2, r3                          # Upper Bound = 0x01FF_FFFF
    e_lis    r3, 0x0040
    e_or2i   r3, 0x0000                        # Lower Bound = 0x0040_0000
    mtspr    mas3, r3
    mpuwe
    mpusync

    # Region 8 (Data): PBRIDGE1/0 @ 0xF800_0000-0xFFFF_FFFF
    e_lis    r3, 0xA002                        # VALID=1, INST=0, SHD=0,
ESEL=2
    e_or2i   r3, 0x0F09                        # UW=1, SW=1, UX/UR=1, SX/SR=1, non-
Cacheable, guarded
    mtspr    mas0, r3
    e_lis    r3, 0x0000
    e_or2i   r3, 0x0000
    mtspr    mas1, r3                          # TID=0, global
    e_lis    r3, 0xFFFF
```

```
    e_or2i  r3, 0xFFFF
    mtspr   mas2, r3                            # Upper Bound = 0xFFFF_FFFF
    e_lis   r3, 0xF800
    e_or2i  r3, 0x0000                          # Lower Bound = 0xF800_0000
    mtspr   mas3, r3
    mpuwe
    mpusync

    e_lis   r3, 0x0000
    e_or2i  r3, 0x0001
    mtspr   1014, r3                            # Enable MPU


    #*********************** Invalidate and enable caches ********************
    # Instruction cache (I-CACHE)
__icache_cfg:
    e_li        r5, 0x2
    mtspr       1011, r5                        # Set L1CSR1.ICINV bit. Start instruction
cache invalidation

    e_li        r7, 0x4
    e_li        r8, 0x2
    e_lis       r11, 0xFFFFFFFB@h
    e_or2i      r11, 0xFFFFFFFB@l

__icache_inv:
    mfspr       r9, 1011                        # Read L1CSR1
    and.        r10, r7, r9                     # Check if L1CSR1.ICABT is set indicating
cache invalidation was aborted.
    e_beq       __icache_no_abort               # If 0, no abortion, jump to proceed.
    and.        r10, r11, r9
    mtspr       1011, r10                       # Clear the L1CSR1.ICABT bit.
    e_b         __icache_cfg                    # Branch back to retry invalidation of
instruction cache.

__icache_no_abort:
    and.        r10, r8, r9                     # Check if L1CSR1.ICINV is clear indication
cache invalidation completed.
    e_bne       __icache_inv                    # If ICINV bit still set jump back to wait
and re-check ICABT bit.

    mfspr       r5, 1011                        # Read L1CSR0
    e_ori       r5, r5, 0x0001
    se_isync                                    # wait for all previous instructions to
complete
    msync                                       # wait for preceding data memory accesses to
reach the point of coherency
    mtspr       1011, r5                        # Set L1CSR1.ICE to enable data cache

    # Data cache (D-CACHE)
__dcache_cfg:
    e_li        r5, 0x2
    mtspr       1010, r5                        # Set L1CSR0.DCINV bit. Start data cache
invalidation

    e_li        r7, 0x4
    e_li        r8, 0x2
    e_lis       r11, 0xFFFFFFFB@h
    e_or2i      r11, 0xFFFFFFFB@l

__dcache_inv:
    mfspr       r9, 1010                        # Read L1CSRO
    and.        r10, r7, r9                     # Check if L1CSR0.DCABT is set indicating
cache invalidation was aborted.
    e_beq       __dcache_no_abort               # If 0, no abortion, jump to proceed.
    and.        r10, r11, r9
    mtspr       1010, r10                       # Clear the L1CSR0.DCABT bit.
    e_b         __dcache_cfg                    # Branch back to retry invalidation of data
cache.
```

```
__dcache_no_abort:
    and.        r10, r8, r9                 # Check if L1CSR0.DCINV is clear indication
cache invalidation completed.
    e_bne       __dcache_inv                # If DCINV bit still set jump back to wait
and re-check DCABT bit.

    mfspr       r5, 1010                    # Read L1CSR0
    e_ori       r5, r5, 0x0001
    se_isync                                # wait for all previous instructions to
complete
    msync                                   # wait for preceding data memory accesses to
reach the point of coherency
    mtspr       1010, r5                    # Set L1CSR0.DCE to enable data cache


    #****************************** enable BTB *********************************
    # Flush and enable BTB
    e_li    r3, 0x201
    mtspr   1013, r3
    se_isync


    #************ Load Initialized Data Values from Flash into RAM ************
    # Initialized Data - ".data"
DATACOPY:
    e_lis       r9, __DATA_SIZE@ha      # Load upper SRAM load size (# of bytes) into R9
    e_or2i      r9, __DATA_SIZE@l       # Load lower SRAM load size into
R9
    e_cmp16i    r9,0                    # Compare to see if equal to
0
    e_beq       SDATACOPY               # Exit cfg_ROMCPY if size is zero (no data to
initialize)

    mtctr       r9                      # Store no. of bytes to be moved in counter

    e_lis       r10, __DATA_ROM_ADDR@h  # Load address of first SRAM load into R10
    e_or2i      r10, __DATA_ROM_ADDR@l  # Load lower address of SRAM load into R10
    e_subi      r10,r10, 1              # Decrement address to prepare for ROMCPYLOOP

    e_lis       r5, __DATA_SRAM_ADDR@h  # Load upper SRAM address into R5 (from linker file)
    e_or2i      r5, __DATA_SRAM_ADDR@l  # Load lower SRAM address into R5 (from linker file)
    e_subi      r5, r5, 1               # Decrement address to prepare for ROMCPYLOOP

DATACPYLOOP:
    e_lbzu      r4, 1(r10)              # Load data byte at R10 into R4,incrementing
(update) ROM address
    e_stbu      r4, 1(r5)               # Store R4 data byte into SRAM at R5 and update SRAM
address
    e_bdnz      DATACPYLOOP             # Branch if more bytes to load from ROM

    # Small Initialized Data - ".sdata"
SDATACOPY:
    e_lis       r9, __SDATA_SIZE@ha     # Load upper SRAM load size (# of bytes) into R9
    e_or2i      r9, __SDATA_SIZE@l      # Load lower SRAM load size into
R9
    e_cmp16i    r9,0                    # Compare to see if equal to
0
    e_beq       ROMCPYEND               # Exit cfg_ROMCPY if size is zero (no data to
initialize)

    mtctr       r9                      # Store no. of bytes to be moved in counter

    e_lis       r10, __SDATA_ROM_ADDR@h # Load address of first SRAM load into R10
    e_or2i      r10, __SDATA_ROM_ADDR@l # Load lower address of SRAM load into R10
    e_subi      r10,r10, 1              # Decrement address to prepare for ROMCPYLOOP

    e_lis       r5, __SDATA_SRAM_ADDR@h # Load upper SRAM address into R5 (from linker file)
    e_or2i      r5, __SDATA_SRAM_ADDR@l # Load lower SRAM address into R5 (from linker file)
    e_subi      r5, r5, 1               # Decrement address to prepare for ROMCPYLOOP
```

```
SDATACPYLOOP:
    e_lbzu      r4, 1(r10)              # Load data byte at R10 into R4,incrementing
(update) ROM address
    e_stbu      r4, 1(r5)               # Store R4 data byte into SRAM at R5 and update SRAM
address
    e_bdnz      SDATACPYLOOP            # Branch if more bytes to load from ROM

ROMCPYEND:


    #*********************** Enable ME bit in MSR ******************************
    mfmsr   r6
    e_or2i  r6, 0x1000
    mtmsr   r6


    #******************** Configure Stack **************************************
    e_lis   r1, __SP_INIT@h                         # Initialize stack pointer r1 to
    e_or2i  r1, __SP_INIT@l                         # value in linker command file.

    e_lis   r13, _SDA_BASE_@h                       # Initialize r13 to sdata base
    e_or2i  r13, _SDA_BASE_@l                       # (provided by linker).

    e_lis   r2, _SDA2_BASE_@h                       # Initialize r2 to sdata2 base
    e_or2i  r2, _SDA2_BASE_@l                       # (provided by linker).

    e_stwu  r0,-64(r1)                              # Terminate stack.

    # Jump to Main
    e_bl main
```

A.2 main.c 文件

```
/*
*   LICENSE:
*   Copyright (c) 2014 Freescale Semiconductor
*
*   Permission is hereby granted, free of charge, to any person
*   obtaining a copy of this software and associated documentation
*   files (the "Software"), to deal in the Software without
*   restriction, including without limitation the rights to use,
*   copy, modify, merge, publish, distribute, sublicense, and/or
*   sell copies of the Software, and to permit persons to whom the
*   Software is furnished to do so, subject to the following
*   conditions:
*
*   The above copyright notice and this permission notice
*   shall be included in all copies or substantial portions
*   of the Software.
*
*   THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
*   EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
*   OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
*   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
*   HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
*   WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
*   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
*   DEALINGS IN THE SOFTWARE.
*
* Composed By: Fraser, Jamaal
* Dated       : March 31, 2014
* Compiler    : Green Hills Multi
*
*
```

```
* FILE NAME: main.c
*
* DESCRIPTION:
*
*/

#include "project.h"


/************************************ Main ***********************************/
int main(void)
{

    /* Peripheral ON in every run mode */
    MC_ME.RUN_PC[0].R = 0x000000FE;

    /* Enable all PBridge Masters for Reads, Writes, and Master Privilege Mode. */
    AIPS_0.MPRA.R = 0x70777700;
    AIPS_1.MPRA.R = 0x70777700;

    while (1);

}
```

A.3 链接器定义文件

```
/*
*  LICENSE:
*  Copyright (c) 2014 Freescale Semiconductor
*
*  Permission is hereby granted, free of charge, to any person
*  obtaining a copy of this software and associated documentation
*  files (the "Software"), to deal in the Software without
*  restriction, including without limitation the rights to use,
*  copy, modify, merge, publish, distribute, sublicense, and/or
*  sell copies of the Software, and to permit persons to whom the
*  Software is furnished to do so, subject to the following
*  conditions:
*
*  The above copyright notice and this permission notice
*  shall be included in all copies or substantial portions
*  of the Software.
*
*  THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
*  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
*  OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
*  NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
*  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
*  WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
*  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
*  DEALINGS IN THE SOFTWARE.
*
* Composed By: Fraser, Jamaal
* Dated       : March 31, 2014
* Compiler    : Green Hills Multi
*
*
* FILE NAME: flash_z4Core.ld
*
* DESCRIPTION:
*
*/

DEFAULTS {
```

```
    // Define Boot Header area Size
    BOOTFLASH_SIZE = 0x8

    // Define Boot Header area Base Address
    BOOTFLASH_BASE_ADDR = 0x01000000

    // Define Core Flash Allocation
    FLASH_SIZE = 2M - BOOTFLASH_SIZE

    // Define Core Flash Base Address
    FLASH_BASE_ADDR =  BOOTFLASH_BASE_ADDR + BOOTFLASH_SIZE

    // Define Core SRAM Allocation
    SRAM_SIZE =  384K

    // Define SRAM Base Address
    SRAM_BASE_ADDR = 0x40000000

    // Define Local SRAM Allocation
    LOCALSRAM_SIZE =  64K

    // Define SRAM Base Address
    LOCALSRAM_BASE_ADDR = 0x50800000

    // Define Stack Size - located at end of SRAM
    STACK_SIZE = 1K

}

/*---------- DO NOT MODIFY ANYTHING BELOW THIS POINT -------*/
MEMORY {

    flash_rcw : org = BOOTFLASH_BASE_ADDR, len = BOOTFLASH_SIZE
    int_flash : org = FLASH_BASE_ADDR,     len = FLASH_SIZE
    int_sram  : org = SRAM_BASE_ADDR,      len = SRAM_SIZE-STACK_SIZE-16
    stack_ram : org = LOCALSRAM_BASE_ADDR  len = STACK_SIZE
}

SECTIONS
{
    .rcw                            : {} >  flash_rcw

    .isrvectbl    ALIGN(0x1000)     : {} > int_flash                  /* ISR Vector
Table - must be 4K aligned */
    .xptn_vectors ALIGN(0x1000)     : {} > .                          /* Exception
Vector Table (IVPR) - align 4K boundary */

    .init                           : {} > .
    .text                           : {} > .                          /* BookE Code
*/
    .vletext                        : {} > .                          /* VLE Code
*/
    .fixaddr                        : {} > .                          /* Required
for         */
    .fixtype                        : {} > .                          /* compatibility
with   */
    .secinfo                        : {} > .                          /* GHS provided
startup */
    .syscall                        : {} > .                          /*
code                 */

    .IVOR4_HWvectors ALIGN(0x1000)  : {} > .                          /* IVOR4 HW
Vector Table (IVPR) - align 4K boundary */

    .rodata                         : {*(.rdata) *(.rodata)} > .      /* Read Only
Data */

    .ROM.data   ROM(.data)          : {} > .                          /* Store
Initialized RAM Variables */
    .ROM.sdata  ROM(.sdata)         : {} > .                          /* temporarily
```

```
in Flash             */

    .data                              : {} > int_sram                   /* Initialized
Data                 */
    .bss                               : {} > .                          /* Uninitialized
Data               */
    .sdabase ALIGN (2)                 : {} > .                          /* Base location
for SDA Area       */
    .sdata                             : {} > .                          /* Small
Initialized Data (Area1)  */
    .sbss                              : {} > .                          /* Small
Uninitialized Data (Area1)*/
    .sdata2                            : {} > .                          /* Small
Initialized Constant Data */
    .sbss2                             : {} > .                          /* Small
Uninitialized Data (Area2)*/

    .heap   ALIGN(16) PAD(1K)          : {} > int_sram                   /* Heap Area  */
    .stack  ALIGN(4)  PAD(STACK_SIZE)  : {} > stack_ram                  /* Stack Area */

/*--------------------------------------------------------------*/
/*  Example of allocating section at absolute address          */
/*                                                              */
/*  .my_section 0x40001000 :{} > int_flash                     */
/*                                                              */
/*  Linker uses "0x40001000" address, rather than "int_flash" */
/*--------------------------------------------------------------*/

/*-------- LABELS USED IN CODE -------------------------------*/

/* Stack Address Parameters */
__SP_INIT      = ADDR(stack_ram) + SIZEOF(stack_ram);

/* Interrupt Handler Parameters */
__IVPR = ADDR(.xptn_vectors);

/* Labels for Copying Initialized Data from Flash to RAM */
__DATA_SRAM_ADDR  = ADDR(.data);
__SDATA_SRAM_ADDR = ADDR(.sdata);

__DATA_SIZE   = SIZEOF(.data);
__SDATA_SIZE  = SIZEOF(.sdata);

__DATA_ROM_ADDR  = ADDR(.ROM.data);
__SDATA_ROM_ADDR = ADDR(.ROM.sdata);

/* Labels Used for Initializing SRAM ECC */
__SRAM_SIZE=SRAM_SIZE;
__SRAM_BASE_ADDR =SRAM_BASE_ADDR;

__LOCAL_SRAM_SIZE=LOCALSRAM_SIZE;
__LOCAL_SRAM_BASE_ADDR =LOCALSRAM_BASE_ADDR;

/* These special symbols mark the bounds of RAM and ROM memory. */
/* They are used by the MULTI debugger.                          */

    __ghs_ramstart  = MEMADDR(int_sram);
    __ghs_ramend    = MEMENDADDR(int_sram);
    __ghs_romstart  = MEMADDR(int_flash);
    __ghs_romend    = MEMENDADDR(int_flash);

    __ghs_rambootcodestart = 0;           /* zero for ROM image */
    __ghs_rambootcodeend = 0;             /* zero for ROM image */
    __ghs_rombootcodestart = MEMADDR(int_flash);
    __ghs_rombootcodeend = MEMENDADDR(int_flash);

}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的"典型"参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括"经典值"在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2014 Freescale Semiconductor, Inc.

© 2014 飞思卡尔半导体有限公司

Document Number AN4939
Revision 0, 04/2014

Review Draft