Freescale Semiconductor
Application Note

Document Number: AN4027
Rev. 0, 07/2010

Initialization and Optimization Program for MPC563xM

by: Mong Sim
Applications Engineer
Microcontroller Solutions Group

1 Introduction

This initialization and optimization application note is written specifically for MPC563xM devices. The code for this application is executed in the internal flash memory. In the following sections, I will provide some details to activate those features and modules described in this application note. Please refer to the current version of Freescale document MPC563XMRM, *MPC563XM Microcontroller Reference Manual*, for a comprehensive explanation of the individual modules.

1.1 Objective

The objective of this application note is to show the reader how to initialize the MPC563xM and what impact different settings would have on system performance. For example, why might the user choose not to initialize the MMU? How does one initialize the flash controller, and how can the flash page buffers improve system performance with a program executing in flash? How

Contents

can the user initialize the internal static random access memory (SRAM) in different program execution modes? At the end of this application note, the reader will have gained the knowledge listed here:

- What the reset configuration half word and the different bit fields represent, and the purpose of having a jump address immediately after the configuration half word
- How the user can enable a branch target buffer, and what the system can gain from it
- How the user can enable the signal processing engine (SPE) feature in the e200z335
- Why the MMU may or may not need to be initialized
- How the user can initialize the flash controller, and how to configure the flash page buffers to improve system performance
- How the user can initialize the internal static random access memory in different modes of execution
- How the user can initialize the frequency-modulated phase-locked loop
- How the user can initialize the different feature sets to improve overall system performance

2 Initialization and optimization

This application note will provide its readers several systematic initialization procedures and the advantages and disadvantages of each; how different options would improve system performance and why this application note chooses a low-performance option to initialize certain modules in the system; and what role compiler optimization and software profiling plays in improving system performance. Finally, we will combine what we have learned into a complete initialization program for the MPC563xM with optimization based on the Dhrystone 2.0 benchmark program.

2.1 Program execution modes

Because the MPC563xM has limited SRAM and no external bus to support external SRAM, the user can only execute an application from flash memory (this is also referred to as ROMRUN mode). Therefore, this application note will focus on initialization and optimization programs that execute in flash memory.

2.2 Reset configuration half word and jump address

For those who are not familiar with the Automobile Power Architecture System on Chip (SoC), the MPC563xM has a built-in boot assist module (BAM). The BAM configures the MMU (please refer to the *MPC563XM Microcontroller Reference Manual* for full details) and searches the flash for the reset configuration half word (RCHW). After the BAM locates the RCHW, it will load the four-byte RCHW (bits 16 to 31 are clear) and apply the setting to the system. The RCHW is shown in Figure 1 and its attributes are described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W RESET[1]:	5’b0				SWT	WTE	PS0	VLE	BOOTID							

= Unimplemented or Reserved

1 The RCHW is in user programmed non-volatile memory. Therefore, it has no reset value.

Figure 1. Reset configuration half word (RCHW)

The SWT and WTE are the watchdog timer and the core watchdog timer respectively. PS0 is the port size and is set to zero (0 = 32-bit CS0 port size and 1 = 16-bit CS0 port size). Asserting the VLE bit will enable support for variable-length encoding.

Immediately after the RCHW is the jump address. The BAM will load this four-byte jump address after processing the RCHW. The BAM will execute a jump command and relinquish control to the application program.

To program the RCHW and jump address, the user must allocate eight bytes of flash space from address 0x000 to 0x007. Please see Table 1 for different boot addresses. Four bytes for the RCHW and four bytes for the jump address in the linker file are set like this:

Example 1. Linker file

```
MEMORY
{

     // Internal Flash
    flash_rsvd1  : ORIGIN = 0x00000000, LENGTH = 8
    …
}

SECTIONS
{
    …

     // ROM SECTIONS
    .resetvector                        NOCHECKSUM      :> flash_rsvd1
    …
}
```

Next, program this code in the initialization program file:

Example 2. RCHW and jump address

```
.section  ".resetvector","ax"

// Reset Configuration Halfword (RCHW) :BOOTID = 0x5a
.long   0x005a0000
.long   rombootcodestart
```

The RCHW is required only if the system is booted from flash. If the system is booted from a RAM image via the BAM, a start address is sufficient for a BAM application (please see Freescale documents AN3953,

"Serial Loader Application for BAM," AN2831, "MPC5500 Boot Assist Module," and AN3519, "Optimizing Performance for the MPC5500 Family," for more details).

Table 1 shows the different flash blocks available in the different MPC563xM devices. It also shows the flash blocks that the system can boot if a validate RCHW is detected in the first word of the allocated flash block.

Table 1. Boot addresses

Address	Use	Block	Size	MPC5632M (768K)	MPC5633M (1M)	MPC5634M (1.5M)	Bank
0x0000_0000	Low Address Space 256 KB	0[1]	16K	Available	Available	Available	Bank 0 Array 0
0x0000_4000		1a[1]	16K	Available	Available	Available	
0x0000_8000		1b	32K	Available	Available	Available	
0x0001_0000		2a[1]	32K	Available	Available	Available	
0x0001_8000		2b	16K	Available	Available	Available	
0x0001_C000		3[1]	16K	Available	Available	Available	
0x0002_0000		4[1]	64K	Available	Available	Available	
0x0003_0000		5[1]	64K	Available	Available	Available	
0x0004_0000	Mid Address Space 256 KB	6	128K	Not available	Available	Available	
0x0006_0000		7	128K	Not available	Available	Available	
0x0008_0000	High Address Space 1.0 MB	8	128K	Available	Available	Available	Bank 1 Array 1
0x000A_0000		9	128K	Available	Available	Available	
0x000C_0000		10	128K	Available	Available	Available	
0x000E_0000		11	128K	Available	Available	Available	
0x0010_0000		12	128K	Not available	Not available	Available	Bank 1 Array 2
0x0012_0000		13	128K	Not available	Not available	Available	
0x0014_0000		14	128K	Not available	Not available	Available	
0x0016_0000		15	128K	Not available	Not available	Available	
0x00FF_C000	Shadow Block 16 KB	S0	16K	Available	Available	Available	Bank 0 Array 0

[1] System can be booted from this block.

2.3 Core and system watchdog timers

The MPC563xM has two watchdog timers. The core watchdog timer is a sub-module of the e200z335 core, and the system watchdog timer is one of the modules embedded in the device. The user can disable these two watchdogs via the RCHW register in ROMRUN and ROMRAM modes. However, if running a RAM image via the BAM, these two watchdogs can be disabled, as shown in this code:

Example 3. Watchdog timer

```
//Disable  Core Watchdog
li       r12, 0x00
mtspr    340,r12


//Disable System Watchdog
lis      r12, 0xFFF3
ori      r12,r12,0x8000
lwz      r11,0(r12)
clrrwi   r11,r11,1
stw      r11,0(r12)
```

2.4 Memory management unit (MMU)

Initialization of the MMU via user software is optional — the BAM initializes the MMU after each power on reset (POR). Please see the *MPC563XM Microcontroller Reference Manual*, section 20.5.2, "BAM Program Operation." Here is some example code showing how to set up the MMU for the 256 KB space, starting at the address 0x4000_0000 for the internal SRAM:

Example 4. Memory management unit

```
// Set up MMU for Internal SRAM
lis      r10, 0x1003
mtspr    mas0, r10

lis      r10, 0xc000
ori      r10, r10, 0x0400
mtspr    mas1, r10

lis      r10, 0x4000
ori      r10, r10, 0x0008
mtspr    mas2, r10

lis      r10, 0x4000
ori      r10, r10, 0x003f
mtspr    mas3 ,r10

tlbwe
```

2.5 Branch target buffer (BTB)

The e200 core provides the BTB feature to perform branching prediction. The BTB must be flushed before use. Enabling the BTB will boost the system performance about 4 percent for an 80 MHz system clock. This assembly code shows how to flush and enable the BTB.

Example 5. Branch target buffer

```
//Flushes the BTB and Enable the BTB
li      r10 ,0x201
mtspr   1013,r10
```

2.6 Signal processing extension (SPE)

The SPE feature is computation orientated — please refer to the *MPC563XM Microcontroller Reference Manual* for more comprehensive details. The assembly code shown here will enable the SPE. Please refer to your compiler reference manual for how to turn on the SPE feature to generate SPE code.

Example 6. Signal processing extension

```
mfmsr    r10
oris     r10, r10, 0x0200    //Enable SPE
mtmsr    r10
```

2.7 Flash wait states and flash page buffers

Freescale recommends that users adhere to the values in Table 2 for internal flash wait state setting operations at different frequencies.

Table 2. Wait states setting vs. frequency of operation

Target max frequency (MHz)	APC	WWSC	RWSC
40	001	01	001
62	010	01	010
82	011	01	011
All	111	11	111

This assembly code sets the number of wait states for the system when operating at 80 MHz.

Example 7. Flash wait state

```
lis      r10,0x0001
ori      r10,r10, 0x6B15   // ≤82 MHz
lis      r11,0xC3F8        // PFlash Configuration Register 1
ori      r11,r11, 0x801C   //(PFCR1) address
stw      r10,0(r11)
```

The flash controller also offers four page buffers. Each buffer is 128 bits long and can hold one flash page. These four buffers are also associated with the prefetch controller that prefetches flash pages to the buffers. These page buffers support zero wait state fetches for page hits. At maximum operating frequency, three wait states are required for a fetch with a page miss. Please see the *MPC563XM Microcontroller Reference Manual* for comprehensive details on the flash page buffer.

This code shows how to configure the page buffers. All four buffers are available for any flash access — that is, there is no partitioning based on the access type. The flash buffers can be allocated for any flash access, or the buffers can be split between instruction fetches and data accesses. To set these buffers to a different configuration, please see the *MPC563XM Microcontroller Reference Manual*.

Example 8. Flash page buffers

```
lis      r10,0x0000 //PFCR2 globally defines the logical
stw      r10,8(r11) //partitioning of the four page buffers
```

2.8 Setting frequency of operation

The MPC563xM provides a frequency modulation phase-locked loop (FMPLL) to allow users to change the system frequency via a set of synthesizer registers. It is highly recommended to use the enhance synthesizer register to set the desired frequency of operation. The MPC563xM also supports a system frequency up to 80 MHz. In Example 9 the PLL is set to 80 MHz. If a different frequency of operation is preferred, use Equation 1 to set the FMPLL registers appropriately. Please refer to the *MPC563XM Microcontroller Reference Manual* for the device speed grade that you are using.

Example 9. FM phase-locked loop

```
//Program the FM Enhance PLL<<<no difference?>>>

//  MHz  :  80 70 60 50 40 30 20 10
//ESYNCR1:  40 35 60 50 40 60 40 1,40
//ESYNCR2:  1   1   2   2   2   3   3  3


//ESYNCR1

lis      r10,0xC3F8
lis      r11,0xF000        # EPREDIV -> 0-1 to 1110-15
ori      r11,r11,40        # EMFD    -> 32 to 96

//ESYNCR2

li       r12,0x0001        # ERFD    -> 0-2,4,8 and 11-16

//save registers with the shortest possible time

stw      r11,8(r10)        # ESYNCR1
stw      r12,12(r10)       # ESYNCR2

wait_for_lock:

lwz      r13,4(r10)         # load SYNSR
andi.    r13,r13,0x8
beq      wait_for_lock
```

F_{sys} is the desired system frequency and F_{ref} is the crystal frequency used in the system.

$$F_{sys} = F_{ref} \times \frac{EMFD}{(EPREDIV + 1) \times 2^{(ERFD+1)}}$$

Eqn. 1

2.9 Internal static random access memory initialization

The MPC5634 device includes 94 KB of general-purpose SRAM. Please see the *MPC563XM Microcontroller Reference Manual* for comprehensive details and Table 3 for the MPC563xM internal SRAM map. The SRAM block also provides 7-bit error checking and correction (ECC) with single-bit correction and 2-bit error detection for every 32-bit word. It is mandatory to initialize the SRAM after power on reset (POR). The user should be aware that the SRAM does not have to be initialized after all resets, only after POR resets. However, this application note does not discuss determining whether or not a reset was a POR reset. Attempting to read an uninitialized SRAM would generate a system exception.

The SRAM is initialized via writing one or more 32-bit words to it. A less than 32-bit write to the SRAM will generate a read/modify/write operation that will check the ECC value upon read. The SRAM initialization method is:

Example 10. Initialize SRAM ECC

```
//initialize 94k SRAM

li       r5,752
mtctr    r5
lis      r5,0x4000

sram_ecc:

stmw     r0,0(r5)
addi     r5,r5,128
bdnz     sram_ecc
```

Initializing the SRAM ECC is straightforward. However, initializing SRAM in serial boot mode is also not difficult. When initializing the SRAM ECC in RAM mode, users must know where the last 32-bit word is stored on the SRAM and start initialization from there.

To find out where the last word is stored, please refer to your linker file. Typically, a RAM mode linker file will have a section that consists of symbols dot bss followed by dot heap and dot stack. Using an ENDADDR command supplied by your linker macro, one can safely initialize the SRAM. (Please see AN3953, “Serial Loader Application for BAM,” for more details.)

Table 3. MPC563xM SRAM maps

Start Address	End Address	Description	MPC5632M	MPC5633M	MPC5634M
		Standby Size	24K	24K	32K
		Total SRAM Size	48K	64K	94K
0x4000_0000	0x4000_5FFF	24K SRAM	Standby SRAM	Standby SRAM	Standby SRAM

Table 3. MPC563xM SRAM maps (continued)

Start Address	End Address	Description	MPC5632M	MPC5633M	MPC5634M
0x4000_6000	0x40007FFF	8K SRAM	SRAM	SRAM	Standby SRAM
0x4000_8000	0x4000_BFFF	16K SRAM	SRAM	SRAM	SRAM
0x4000_C000	0x4000_FFFF	16K SRAM	Not available	SRAM	SRAM
0x4001_0000	0x4001_77FF	30K SRAM	Not available	Not available	SRAM

2.10 Crossbar initialization

The crossbar (XBAR) can support up to eight master ports and eight slave ports. It will allow for concurrent transactions from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time because of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will stall until the higher priority master completes its transactions.

The code below sets access priority for the eDMA over the core, to prevent an eDMA memory access timeout.

Example 11. Initialize crossbar

```
//initialize Crossbar

lis     r12,0xFFF0
ori     r12,r12,0x4000
lis     r11,0x0001
ori     r11,r11,0x0302
stw     r11,0(r12)
```

3 Initialization optimization dependency

Initialization is not a one-time action, but a progressive effort to fine-tune the system. The most important factor to consider when determining which features to initialize and how they should be initialized is the user's final application program. Nothing is more important than this. However, there are certain features that can be identified due to the application specification and the way the system is designed.

The initialization routine in Appendix A, "Initialization programs," is optimized using the Dhrystone 2.0 benchmark application to calibrate the different sets of features that will allow optimal performance of the MPC563xM EVB. Please see the performance bar chart in Figure 2. Readers will notice that the same application operating at the same frequency can have severe performance degradation if initialization parameters are not optimized.

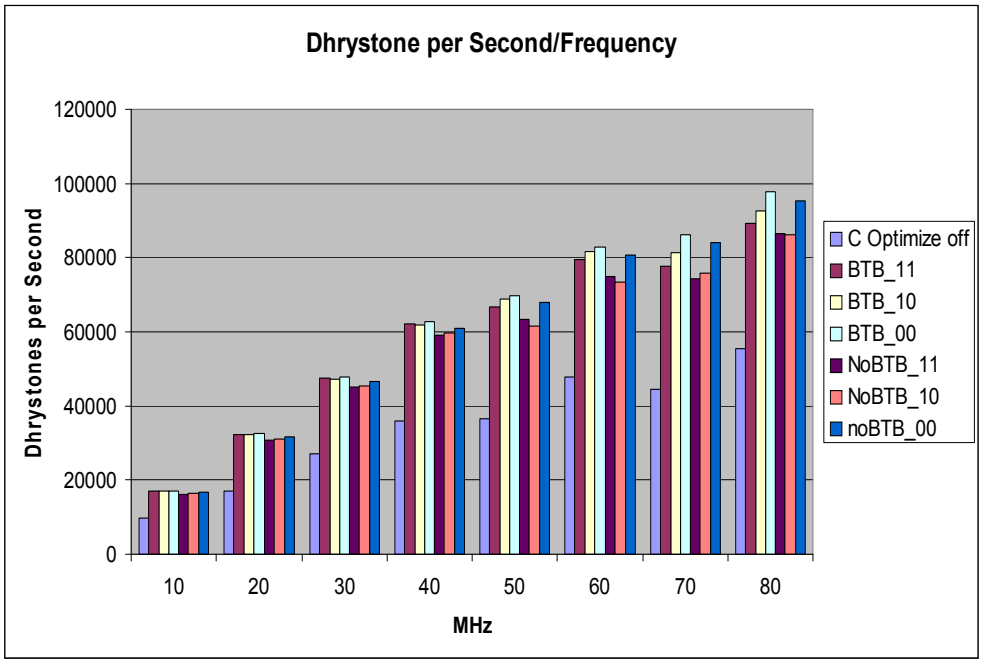

Figure 2. Dhrystone 2.0 benchmark performance chart

Legend for Figure 2:

- C Optimize off: BTB and SPE enable, flash page = 00 and no C optimization
- BTB_11: BTB and SPE enable, flash page = 11 with loop optimization
- BTB_10: BTB and SPE enable, flash page = 10 with loop optimization
- BTB_00: BTB and SPE enable, flash page = 00 with loop optimization
- NoBTB_11: BTB disable, SPE enable, flash page = 11 with loop optimization
- NoBTB_10: BTB disable, SPE enable, flash page = 10 with loop optimization
- noBTB_00: BTB disable, SPE enable, flash page = 00 with loop optimization

Flash page setting

- 00: No accesses may be performed by the processor core
- 01: Only read accesses may be performed by the processor core
- 10: Only write accesses may be performed by the processor core
- 11: Both read and write accesses may be performed by the processor core

Let us look at the bar chart at 80 MHz. Compare the bar labeled “C Optimize off” and the bar labeled “BTB_00” (C optimization is on): the performance disparity is huge. From this chart, one can see that software optimization plays a very large role in improving system performance. With C optimization, performance can improve as much as 43 percent — hardware optimization can improve approximately 12 percent.

4 Summary

The MPC563xM has very limited internal SRAM and no external bus to support external memory. Therefore all user application code is expected to run from the internal flash. For that reason, the initialization code and optimization code are targeted to programs that will be run from the system flash memory.

Please see Appendix A, "Initialization programs," for the MPC563xM initialization programs. Users are reminded that the initialization program is optimized using the Dhrystone 2.0 benchmark and may not reflect user application runtime behavior. Users are advised to consult the reference manual and change the initialization parameters appropriately. However, the order of initialization in this initialization program is recommended.

Appendix A Initialization programs

Example A-1. Initialization program ROMRUN, ROMRAM, and mixed mode

```
/*
 *  Mong Sim
 *  Freescale
 *  BoardInit for MPC563xM for MULTI
 *  9/30/2009
 */

        .weak  __ghs_rambootcodestart
        .weak  __ghs_rambootcodeend
        .weak  __ghs_rombootcodestart
        .weak  __ghs_rombootcodeend
        .globl __ghs_board_memory_init

__ghsautoimport_ghs_board_memory_init::
__ghs_board_memory_init:

        ; This routine must not use r3 through r6

        ; Set the MSR[SPE] bit to enable ev* instructions
        ; Disable floating point, external interrupts, and machine
        ; check interrupts.


//-----------------------------------------------------------------------------
// SPE Enable
//-----------------------------------------------------------------------------

        mfmsr   r10
        oris    r10, r10, 0x0200        #SPE Enable
        mtmsr   r10

//-----------------------------------------------------------------------------
// System Watchdog
//-----------------------------------------------------------------------------

        lis     r12, OxFFF3
        ori     r12, r12, 0x8000
        lwz     r11, 0(r12)
```

```
        clrrwi  r11, r11, 1
        stw     r11, 0(r12)

//-----------------------------------------------------------------------------
// Core Watchdog
//-----------------------------------------------------------------------------
        li      r12,0x00
        mtspr   340,r12


//-----------------------------------------------------------------------------
// Branch Target Buffer
//-----------------------------------------------------------------------------

        //BTB Enable
        li      r10, 0x201              #Disable 0x200
        mtspr   1013, r10

//-----------------------------------------------------------------------------
// Check operation
//-----------------------------------------------------------------------------

        ; If running from RAM, return.
        mflr    r12
        lis     r11, %hiadj(__ghs_rombootcodeend)
        addi    r11, r11, %lo(__ghs_rombootcodeend)
        cmplw   r12, r11
        bgelr
        lis     r11, %hiadj(__ghs_rombootcodestart)
        addi    r11, r11, %lo(__ghs_rombootcodestart)
        cmplw   r12, r11
        bltlr


//-----------------------------------------------------------------------------
// MMU
// user can choose not the initial the MMU explicitly
//-----------------------------------------------------------------------------

// Setup MMU for for Periph B Modules
        lis     r10, 0x1000
        mtspr   mas0, r10
        lis     r10, 0xc000
        ori     r10, r10, 0x0500
        mtspr   mas1, r10
        lis     r10, 0xfff0
        ori     r10, r10, 0x000a
        mtspr   mas2, r10
        lis     r10, 0xfff0
        ori     r10, r10, 0x003f
        mtspr   mas3 ,r10
        tlbwe

// Set up MMU for Internal SRAM
        lis     r10, 0x1003
        mtspr   mas0, r10
        lis     r10, 0xc000
```

```
        ori     r10, r10, 0x0400
        mtspr   mas1, r10
        lis     r10, 0x4000
        ori     r10, r10, 0x0008
        mtspr   mas2, r10
        lis     r10, 0x4000
        ori     r10, r10, 0x003f
        mtspr   mas3 ,r10
        tlbwe

// Setup MMU for Periph A Modules
        lis r10, 0x1004
        mtspr mas0, r10
        lis r10, 0xc000
        ori r10, r10, 0x0500
        mtspr mas1, r10
        lis r10, 0xC3F0
        ori r10, r10, 0x000A
        mtspr mas2, r10
        lis r10, 0xC3F0
        ori r10, r10, 0x003f
        mtspr mas3 ,r10
        tlbwe

// Setup MMU for External Memory
        lis r10, 0x1002
        mtspr mas0, r10
        lis r10, 0xc000
        ori r10, r10, 0x0700
        mtspr mas1, r10
        lis r10, 0x2000
        ori r10, r10, 0x0000
        mtspr mas2, r10
        lis r10, 0x2000
        ori r10, r10, 0x003f
        mtspr mas3 ,r10
        tlbwe

// Setup MMU for Internal Flash
        lis r10, 0x1001
        mtspr mas0, r10
        lis r10, 0xc000
        ori r10, r10, 0x0700
        mtspr mas1, r10
        lis r10, 0x0000
        ori r10, r10, 0x0000
        mtspr mas2, r10
        lis r10, 0x0000
        ori r10, r10, 0x003f
        mtspr mas3 ,r10
        tlbwe


//-----------------------------------------------------------------------------
// Program the FM Enhance PLL
//-----------------------------------------------------------------------------
```

```
        ; MHz    :  80 70 60 50 40 30 20 10
        ; ESYNCR1:  40 35 60 50 40 60 40 1,40
        ; ESYNCR2:   1  1  2  2  2  3  3 3


;ESYNCR1
        lis r10, 0xC3F8
        lis r11, 0xF000                 # EPREDIV -> 0-1  to 1110-15
        ori r11, r11,40                 # EMFD    -> 32 to 96
;ESYNCR2
        li  r12, 0x0001                 # ERFD    -> 0-2,4,8 and 11-16

        ;save registers with the shortest possible time

        stw r11, 8(r10)                 # ESYNCR1
        stw r12, 12(r10)                # ESYNCR2

wait_for_lock:

        lwz r13, 4(r10)                 # load SYNSR
        andi. r13, r13, 0x8
        beq wait_for_lock


//-------------------------------------------------------------------------
// Internal SRAM ECC Initialization
//-------------------------------------------------------------------------

//li        r5, 384                # 48 KB of SRAM
//li        r5, 512                # 64 KB of SRAM
 li         r5, 752                # 94 KB of SRAM
 mtctr   r5                        # 752*32*4
 lis     r5,0x4000

sram_ecc:

 stmw   r0,0(r5)
 addi   r5,r5,128
 bdnz   sram_ecc


//-------------------------------------------------------------------------
// Reduce FLASH wait-states
//-------------------------------------------------------------------------

        lis r10, 0x0001
        //ori   r10, r10, 0x0015        # 8MHz
        //ori   r10, r10, 0x2915        # 40MHz
        //ori   r10, r10, 0x4A15        # 62MHz

        ori    r10, r10, 0x6B15# 82MHz

        lis     r11, 0xC3F8             # PFlash Configuration Register 1
        ori     r11, r11, 0x801C        # (PFCR1) address
        stw     r10, 0(r11)

        lis     r10, 0x0000             # PFCR2 globally defines the logical
```

```
        stw     r10, 8(r11)             # partitioning of the four page buffers

//-----------------------------------------------------------------------------
// initialize Crossbar
//-----------------------------------------------------------------------------

lis      r12, 0xFFF0
ori      r12, r12, 0x4000
lis      r11, 0x0001                # Set DMA prior higher than Core
ori      r11, r11, 0x0302

//-----------------------------------------------------------------------------
// End Initialization
//-----------------------------------------------------------------------------

        blr

        .type   __ghs_board_memory_init, @function
        .size   __ghs_board_memory_init_memory, $-__ghs_board_memory_init

        .section  ".resetvector","ax"

__ghs_board_devices_resetvector:

        /* Reset Configuration Halfword (RCHW) : BOOTID = 0x5a  */
        .long   0x005a0000
        .long   __ghs_rombootcodestart

        .type __ghs_board_devices_resetvector, @function
        .size __ghs_board_devices_resetvector,$-__ghs_board_devices_resetvector
```

Example A-2. Initialization program RAM image

```
/*
*
*  Mong Sim
*  Freescale
*  BoardInit for MPC563xM
*  8/20/2009
*  Do not reserve space at 0-7 in RAM linker file
*/

//-----------------------------------------------------------------------------
// System Watchdog Disable
//-----------------------------------------------------------------------------

        lis     r12, SWT_CR@h
        ori     r12, r12, SWT_CR@l
        lwz     r11, 0(r12)
        clrrwi  r11, r11, 1
        stw     r11, 0(r12)

//-----------------------------------------------------------------------------
// Core Watchdog Disable
//-----------------------------------------------------------------------------
```

```
        li      r6,0x00
        mtspr   340,r6


//-----------------------------------------------------------------------------
// Enable SPE
//-----------------------------------------------------------------------------

        mfmsr r6
        oris  r6, r6, 0x0200
        mtmsr r6

//-----------------------------------------------------------------------------
// Enable BTB
//-----------------------------------------------------------------------------

        li      r0, 0x201
        mtspr   1013, r0

//-----------------------------------------------------------------------------
// Initial SRAM ECC
//-----------------------------------------------------------------------------

        lis     r30,0x0000
        lis     r31,0x0000
        lis     r11,0x4000
        ori     r11,r11,%lo(__ram_image_end)  #see linker file
sram_init:
        stmw    r30,0(r11)
        addi    r11,r11,8
        andi.   r12,r11,0xFFF0
        bne     sram_init
```

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4027
Rev. 0
07/2010