

Initializing the MPC5777M Clock Generation Module and Progressive Clock Switching Feature

by: Curt Hillier and Allan Dobbin

1 Introduction

The MPC5777M Power Architecture® MCU targets high-end industrial and automotive powertrain applications that meet next-generation advanced engine control, functional safety, and security requirements. The MPC5777M supports a wide range of applications including factory automation, automotive engine management, hybrid and electric motor controllers, transmission control units, and aerospace engine control. The MCU architecture supports the following key elements:

Multi-core Performance – Unprecedented performance and integration for powertrain control within a power envelope of previous-generation MCUs

Distributed Processing – On-chip I/O processor offloads primary cores and allows higher parallel throughput; GTM timers and sigma delta ADC converters allow customers to do advanced filtering using on-chip knock hardware

Quality, Security and Safety Protection – AEC-Q100 quality, offers on-chip security encryption protection using Hardware Security Module (HSM) for SHE and Tamper Detect Module (TDM) for tamper proofing and supports ASIL-D and SIL-1 functional safety (ISO26262/ IEC61508)

This document details the MPC5777M specific PLL and clock divider settings to achieve 300 MHz Core 0 / Core 1 and 200 MHz Fast Crossbar (FXBAR) / Core 2 operation. In addition, the document describes the Progressive Clock Switching

Contents

1	Introduction.....	1
2	Clock initialization.....	2
2.1	Clock tree	2
3	Introduction to Progressive Clock Switching (PCS)	4
3.1	PCS operation.....	5
3.2	PCS control of current.....	5
3.3	PCS registers.....	5
3.4	PCS calculation example.....	6
3.5	PCS calculator.....	8
3.6	PCS sample oscilloscope plots....	12
4	Targeting code using Green Hills Systems (GHS) tools.....	14
4.1	Example 1: Targeting SW to System RAM and Core Local Instruction RAM (I-MEM).....	14
4.2	Example 2: Targeting SW to Flash and copying functions to run from Core Local Instruction RAM (I-MEM).....	17
5	Appendix A.....	20
5.1	C code listing.....	20



Clock initialization

feature, which supports a smooth ramp-up and ramp-down of device system clocks, for a 300 MHz clock example. The calculations for this example can be used for a 200 MHz source clock application by simply changing the inputs to the formulas.

Clock Divider Programming Restrictions:

The MPC5777M revision 1 silicon has restrictions on System Clock divider programming. The restrictions are removed in revision 2 silicon. In revision 1, the software that programs the System Clock Divider n Configuration Registers (SCn_DC) must run from the e200z425 Core 2 local memory, per the following errata description.

ERR003881 Description: An issue the user must manage when initializing devices is related to the clock divider programming process. Errata e3881 (ERR003881) restricts clock divider programming to Core 2 only. The errata states: "The Peripheral I/O Processor (IOP/Core 2) is the only core that can successfully change the clock ratio between the cores and the Cross Bar (XBAR) interfaces (including the peripheral bridge frequencies)."

Software Placement to RAM and Flash:

This document details software placement techniques that can be used to locate clock configuration code in core 2 local RAM, as required to work around e3881. Examples contained in the document illustrate defining code sections and targeting code sections to System RAM, Core 2 Local RAM, Flash, and Flash copy to RAM.

2 Clock initialization

At Power on Reset the MCU is clocked from the on chip 16MHz Internal RC Oscillator (IRCOSC). This section describes how to configure the clock related modules to run the MCU from the configurable and higher speed PLL's. It will also cover the setup of the clock trees to distribute and divide the clock sources to the buses and peripherals on the MCU.

2.1 Clock tree

The following diagram shows the clock tree configuration for the MPC5777M. In the software example used in this application note, PLL0 is configured to output a 400 MHz clock and PLL1 is configured to output a 600 MHz clock. Software configures both PLL0 and PLL1 to use the external oscillator (40 MHz) as an input.

**Note: All dividers shown in the diagram are integer dividers with a range of 1, 2, 3,..., n.
FCD are fractional clock dividers.
All clock dividers are 50% duty cycle.**

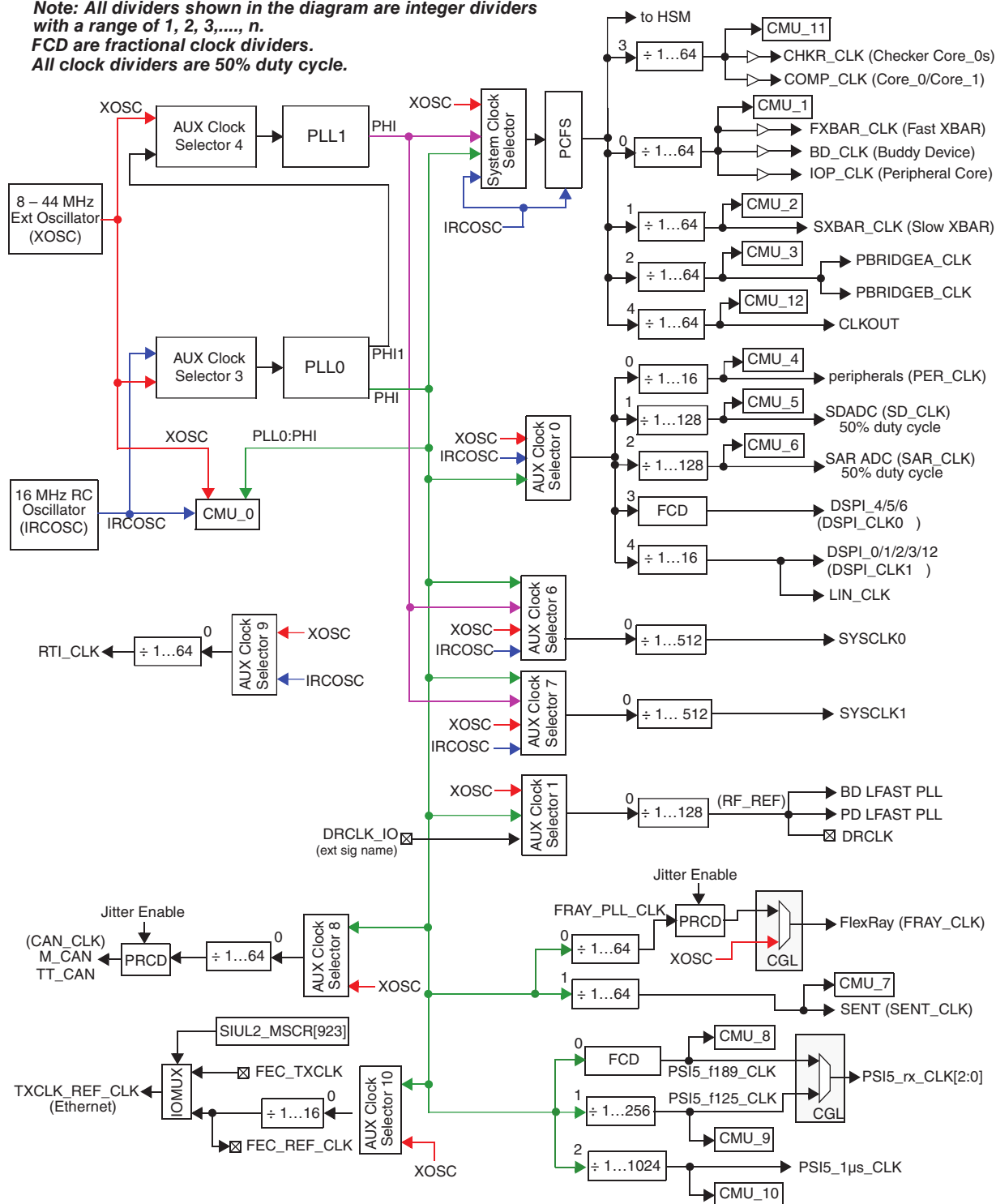


Figure 1. Block diagram of MPC5777M clock tree

Introduction to Progressive Clock Switching (PCS)

A summary of the clock tree settings that are configured in the example software is shown in the following table. Please note that Aux Clock Selector 1 for the Buddy Device (BD) is not configured in this example. The BD supports an additional overlay RAM (up to 2 MB in the MPC5777M) and a high-speed Nexus trace output port consisting of four lanes of Aurora (LVDS) operating up to 1.25 Gbit/s.

Table 1. Example of clock settings

Clock(s)	Aux Selector and Divider	Source Clock	Divide/Multiply Factor	Frequency
XOSC	N/A	N/A	N/A	40 MHz
PLL0	Aux 3	XOSC	10	400 MHz
PLL1	Aux 4	XOSC	15	600 MHz
Fast XBAR Buddy Device Peripheral Core 2	System 0	PLL1	3	200 MHz
Slow XBAR	System 1	PLL1	6	100 MHz
PBRIDGE_A PBRIDGE_B	System 2	PLL1	12	50 MHz
Checker Core 0 Core 0 / Core 1	System 3	PLL1	2	300 MHz
PER_CLK	Aux 0–0	PLL0	5	80 MHz
SDADC	Aux 0–1	PLL0	25	16 MHz
SAR_ADC	Aux 0–2	PLL0	25	16 MHz
DSPI_4/5/6	Aux 0–3	PLL0	4	100 MHz
DSPI_0/1/2/3/12, LIN_CLK	Aux 0–4	PLL0	4	100 MHz
FRAY_PLL_CLK	Aux 2–0	PLL0	10	40 MHz
SENT	Aux 2–1	PLL0	10	40 MHz
PSI5_f189_CLK	Aux 5–0	PLL0	Fractional	6.048 MHz
PSI5_f125_CLK	Aux 5–1	PLL0	10	40 MHz
PSI5_1uS_CLK	Aux 5–2	PLL0	10	40 MHz
SYSCLK0	Aux 6–0	PLL0	10	40 MHz
SYSCLK1	Aux 7–0	PLL1	10	60 MHz
CAN_CLK	Aux 8–0	XOSC	8	5 MHz
RTI_CLK	Aux 9–0	XOSC	4	10 MHz
FEC_REF_CLK	Aux 10–0	PLL0	16	25 MHz

3 Introduction to Progressive Clock Switching (PCS)

Changing device operating modes and/or clock frequency in the MPC5746M and MPC5777M typically results in instantaneous changes in current (IDD). These changes in current can cause undesired fluctuations in power supply voltage, causing the supply voltage at the device to operate at a higher or lower voltage than desired and causing possible triggering of a Low Voltage Detect (LVD) or High Voltage Detect (HVD). The PCS feature supports smooth transitions for frequency and mode changes, greatly reducing voltage overshoot, undershoot, and undesired LVDs / HVDs due to rapidly changing power supply load.

3.1 PCS operation

PCS is a feature of the clock generation module (MC_CGM) and is triggered by the mode entry module (MC_ME). To change the system clock frequency, a mode entry change must occur by writing the MC_ME mode control register (ME_MCTL). If the power level (PWRLVL field within the various mode configuration registers) is different between the current and target modes, the PCS mechanism is enabled causing the system clock frequency to ramp down and/or up in multiple steps. For example, consider two modes with two different PWRLVL settings in respective Mode Configuration Register: ME_RUN0_MC[PWRLVL] = 1, and ME_RUN1_MC[PWRLVL] = 2. If RUN0 is the current mode, and a mode change occurs to switch to mode RUN1, then PCS will be invoked since the PWRLVL settings are different between RUN0 and RUN1. PCS may also be activated by doing a mode entry change that remains in the same mode (e.g. DRUN -> DRUN) so long as the PWRLVL field is changed before the mode change.

In order to prevent sudden voltage drops and overshoots due to frequency and load changes, the MC_ME requests the MC_CGM to ramp the system clock frequency down and/or up based on the power level values of the current and target modes. During ramp-down, the rate of the frequency change is based on the PCS Switch Duration Register (CGM_PCS_SDUR), PCS Divider Change Rate (CGM_PCS_DIVCn), and PCS Divider End Value (CGM_PCS_DIVEN) registers, where n corresponds to the current system clock source selection. During ramp-up, the rate of the frequency change is based on the CGM_PCS_SDUR, CGM_PCS_DIVCn, and PCS Divider Start Value (CGM_PCS_DIVSn) registers, where n corresponds to the target system clock source selection.

3.2 PCS control of current

The following paragraphs describe the control of major components (CPU Cores, Cache, Peripherals). Components consuming larger amounts of power such as CPUs and cache, should be turned on at lower clock frequencies as detailed below.

Initializing Core 0/Core 1 e200z CPUs – This can either be done at the low frequency (before ramp up or after ramp down) or it can be done during the same mode entry change along with PCS. The Freescale example initialization software included in this application note contains CPU initialization as part of the same mode entry change where System Clock Dividers, Auxiliary Clock Dividers, PLL0, PLL1, and PCS are configured. The Mode Entry (MC_ME) technology manages all clock dividers, PCS, and CPU core initialization for the user. The ME module, turns on Core 0, Check Core, and Core 1 at the lower PCS frequency and then progressively switches clocks faster and faster, thereby minimizing any sharp changes in current consumption.

Enabling cache – Do at low frequencies before ramp up. Caches should be enabled as part of the start_core<n>.s code which executes during CPU core initialization.

Enabling Peripherals – Device peripherals consume a smaller fraction of overall device current. Peripherals can be initialized after the PCS completes when system clocks and peripheral clocks are running at their full application speed.

3.3 PCS registers

PCS is achieved by programming registers in the Clock Generation Module (MC_CGM). The user programs values for initial divider change, divider start (for ramp-up from slow to fast), divider end (for ramp-down from fast clock to slow clock), and the switch RATE. The following table lists the required configuration steps and associated registers for PCS.

Table 2. PCS registers

Step	Procedure description	Register : Comments
1	Configure Switch Duration	MC_CGM.PCS_SDUR: Defines the duration of one system clock switch step. Switch Duration = Number of 16 MHz clocks * SDUR * k ¹ steps
2	Configure DRUN power level	MC_ME.DRUN_MC[PWRLVL] : Power level must be set to different value than the power level setting for the mode currently in operation
3	Set the Divider Change Initial Value	MC_CGM.PCS_DIVCn.B.INIT : Defines the initial clock divider value.
4	Set the Divider Change Rate	MC_CGM.PCS_DIVCn.B.RATE : PCS Rate. See details in the PCS calculation example in this application note
5	Set the Divider Start Value (for clock ramp-up from slow to fast)	MC_CGM.PCS_DIVSn : Clock ramp-up start divider value. DIVSn is used for switching from slow clock to fast clock.
6	Set the Divider End Value (for clock ramp-down from fast to slow)	MC_CGM.PCS_DIVEn : Clock ramp-down end divider value. DIVEn is used for switching from fast clock to slow clock.

1. See PCS calculation example in this application note

3.4 PCS calculation example

The application designer can determine the register settings required for PCS, by first defining the following system design parameters:

- Normal Operating frequency (f_{src})
- Low Current Operating frequency, typically 16 MHz (f_{tgt})
- Maximum allowable IDD change in mA (assume 1 μ s PCS step duration)

To determine the maximum frequency change allowed, the change in current due to the change in device operating frequency must be known. For the MPC5777M, bench testing shows the device dynamic IDD change is 2.36 mA/MHz. Using this number, we can determine the maximum frequency change allowed, f_{chg} , using the following equation:

$$f_{chg} = \frac{\text{maxallowableIDDchange}}{\text{devicedynamicIDDchange}}$$

$$f_{chg} = \frac{50mA}{2.36mA/MHz}$$

$$f_{chg} = 21.2$$

Equation 1

where max allowable IDD change is determined by the user's power supply design.

The example below shows the steps required to calculate the needed register settings based on the Reference Manual formulas:

Given:

Normal operating frequency (f_{src}) = 300 MHz

Low current operating frequency (f_{tgt}) = 16 MHz

Maximum allowable IDD change in mA (assume 1 μ s PCS step duration) = 50 mA

Maximum frequency change (f_{chg}) = 21.2 MHz (calculated in steps above)

It is first necessary to determine the rate of frequency change. This is done by calculating:

$$a_{max} = \frac{f_{chg}}{f_{src}}$$

$$a_{max} = \frac{21.2MHz}{300MHz}$$

$$a_{max} = 0.07$$

Equation 2

$a_{max} = 0.05$ (rounded down to the nearest 0.05)

Next, based on $a_{max} = 0.05$, determine MC_CGM_PCS_DIVCn[RATE] by looking up a_{max} in the MC_CGM_PCS_DIVCn[RATE] values table in the Reference Manual. From this table, we find the RATE is 12 for $a_{max} = 0.05$. For the purposes of this application note, $d = \text{RATE} / 1000$.

RATE = 12

$d = 0.012$

To calculate k, number of steps, use the following equation:

$$k = 0.5 + \sqrt{0.25 - \frac{2 \times [1 - \frac{f_{src}}{f_{tgt}}]}{d}}$$

$$k = 0.5 + \sqrt{0.25 - \frac{2 \times [1 - \frac{300MHz}{16MHz}]}{0.012}}$$

$$k = 0.5 + 54.4$$

$$k = 54.9 \text{ steps}$$

Equation 3

Given the number of steps and the Switch Duration Register (SDUR) setting, the user can calculate the duration, t_{pcs} , of the PCS switching. In this formula, we shall round k up to the nearest whole number. 54.9 rounds up to the next nearest whole number of 55.

$$t_{pcs} = \frac{1}{16MHz} \times SDUR \times k$$

$$t_{pcs} = \frac{1}{16MHz} \times 16 \times 55$$

$$t_{pcs} = 55\mu s$$

Equation 4

At this point, we can calculate the initial divider increment change value, MC_CGM_PCS_DIVCn[INIT], by using the following equation:

$$INIT = d \times k \times 1000$$

$$INIT = 0.012 \times 55 \times 1000$$

$$INIT = 660$$

Equation 5

Next, let's calculate the divider start value (DIVSn) that determines the divide value required for the start of PCS switching. The following equation is used:

$$\begin{aligned}DIVSn &= \left[1 + \frac{d \times k \times [k+1]}{2}\right] \times 1000 - 1 \\DIVSn &= \left[1 + \frac{0.012 \times 55 \times [55+1]}{2}\right] \times 1000 - 1 \\DIVSn &= 19479\end{aligned}$$

Equation 6

Finally, to determine DIVE_n, the following equation applies:

$$\begin{aligned}DIVE_n &= 1000 \times \frac{f_{src}}{16MHz} - 1 \\DIVE_n &= 1000 \times \frac{300MHz}{16MHz} - 1 \\DIVE_n &= 18749\end{aligned}$$

Equation 7

As a result of these calculations, we now have the final list of values to be programmed into the PCS registers. To summarize:

MC_CGM.PCS_SDUR.R = 16;
MC_CGM.PCS_DIVCn.B.INIT = 660;
MC_CGM.PCS_DIVCn.B.RATE = 12;
MC_CGM.PCS_DIVSn.R = 19479;
MC_CGM.PCS_DIVE_n.R = 18749;

3.5 PCS calculator

To ease the process of determining PCS register settings, Freescale provides an Excel based calculator. The steps shown below detail how the user can enter their application parameters into the calculator and receive a set of register settings as a result. The following five diagrams illustrate each step in the process:

- Step 1: enter the source frequency (for example, 300 MHz)
- Step 2: enter IDD change per MHz for the MCU
- Step 3: enter max allowable IDD change for the application
- Step 4: round down the calculated a_{max} to the nearest 0.05
- Step 5: enter PCS step duration in microseconds

$f_{\text{target}} = 16 \text{ MHz}$, register settings are in **Bold text**

Parameter / Register	Example 1	Example 2	Example 3
source (MHz)	300	200	300
dynamic IDD change (mA/MHz)	2.36	2.36	
max allowable IDD change (mA/ μs)	50.0	50.0	
max Frequency step (MHz)	21.2	21.2	
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.071	0.106	
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.05	0.10	
step duration (μs)	1	1	
k steps	55.00	23.00	
duration (μs)	55	23	
PCS_SDUR[SDUR]	16	16	
PCS_DIVC1[RATE]	12	48	---
PCS_DIVCn[INIT]	660	1104	
PCS_DIVE_n[DIVE]	18749	12499	18749
PCS_DIVS_n[DIVS]	19479	14247	

Step 1: enter source frequency

Figure 2. Enter source frequency into the calculator

$f_{\text{target}} = 16 \text{ MHz}$, register settings are in **Bold text**

Parameter / Register	Example 1	Example 2	Example 3
source (MHz)	300	200	300
dynamic IDD change (mA/MHz)	2.36	2.36	2.36
max allowable IDD change (mA/ μs)	50.0	50.0	
max Frequency step (MHz)	21.2	21.2	0.0
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.071	0.106	
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.05	0.10	
step duration (μs)	1	1	
k steps	55.00	23.00	
duration (μs)	55	23	
PCS_SDUR[SDUR]	16	16	
PCS_DIVC1[RATE]	12	48	---
PCS_DIVCn[INIT]	660	1104	
PCS_DIVE_n[DIVE]	18749	12499	18749
PCS_DIVS_n[DIVS]	19479	14247	

Step 1: enter source frequency

Step 2: enter IDD change per MHz^[1]

Notes: [1] Device specific value for MPC5777M using bench measurement of IDD at 16 MHz and IDD at 300 MHz.

This is not a production guaranteed value, it is only an engineering estimate.

Figure 3. Enter IDD change per MHz for the MCU

Introduction to Progressive Clock Switching (PCS)

$f_{\text{target}} = 16 \text{ MHz}$, register settings are in **Bold text**

Parameter / Register	Example 1	Example 2	Example 3
source (MHz)	300	200	300
dynamic IDD change (mA/MHz)	2.36	2.36	2.36
max allowable IDD change (mA/ μs)	50.0	50.0	125.0
max Frequency step (MHz)	21.2	21.2	53.0
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.071	0.106	0.177
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.05	0.10	
step duration (μs)	1	1	
k steps	55.00	23.00	
duration (μs)	55	23	
PCS_SDUR[SDUR]	16	16	
PCS_DIVC1[RATE]	12	48	---
PCS_DIVCn[INIT]	660	1104	
PCS_DIVEEn[DIVE]	18749	12499	18749
PCS_DIVSn[DIVS]	19479	14247	

Step 1: enter source frequency

Step 2: enter IDD change per MHz^[1]

Step 3: enter max IDD change

Notes: [1] Device specific value for MPC5777M using bench measurement of IDD at 16 MHz and IDD at 300 MHz.

This is not a production guaranteed value, it is only an engineering estimate.

Figure 4. Enter maximum allowable IDD change

$f_{\text{target}} = 16 \text{ MHz}$, register settings are in **Bold text**

Parameter / Register	Example 1	Example 2	Example 3	
source (MHz)	300	200	300	Step 1: enter source frequency
dynamic IDD change (mA/MHz)	2.36	2.36	2.36	Step 2: enter IDD change per MHz ^[1]
max allowable IDD change (mA/ μs)	50.0	50.0	125.0	Step 3: enter max IDD change
max Frequency step (MHz)	21.2	21.2	53.0	
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.071	0.106	0.177	
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.05	0.10	0.15	Step 4: Round down to nearest 0.05 ^[2]
step duration (μs)	1	1		
k steps	55.00	23.00	19.00	
duration (μs)	55	23		
PCS_SDUR[SDUR]	16	16		
PCS_DIVC1[RATE]	12	48	112	
PCS_DIVCn[INIT]	660	1104	2128	
PCS_DIVEn[DIVE]	18749	12499	18749	
PCS_DIVSn[DIVS]	19479	14247	22279	

Notes: [1] Device specific value for MPC5777M using bench measurement of IDD at 16 MHz and IDD at 300 MHz.

This is not a production guaranteed value, it is only an engineering estimate.

[2] Round calculated a_{max} down to nearest 0.05.

Figure 5. Round down the result to nearest 0.05

$f_{\text{target}} = 16 \text{ MHz}$, register settings are in **Bold text**

Parameter / Register	Example 1	Example 2	Example 3	
source (MHz)	300	200	300	Step 1: enter source frequency
dynamic IDD change (mA/MHz)	2.36	2.36	2.36	Step 2: enter IDD change per MHz ^[1]
max allowable IDD change (mA/ μs)	50.0	50.0	125.0	Step 3: enter max IDD change
max Frequency step (MHz)	21.2	21.2	53.0	
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.071	0.106	0.177	
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.05	0.10	0.15	Step 4: Round down to nearest 0.05 ^[2]
step duration (μs)	1	1	2	Step 5: enter PCS step duration
k steps	55.00	23.00	19.00	
duration (μs)	55	23	38	
PCS_SDUR[SDUR]	16	16	32	} Register Values
PCS_DIVC1[RATE]	12	48	112	
PCS_DIVCn[INIT]	660	1104	2128	
PCS_DIVEn[DIVE]	18749	12499	18749	
PCS_DIVSn[DIVS]	19479	14247	22279	

Notes: [1] Device specific value for MPC5777M using bench measurement of IDD at 16 MHz and IDD at 300 MHz.

This is not a production guaranteed value, it is only an engineering estimate.

[2] Round calculated a_{max} down to nearest 0.05.

Figure 6. Enter PCS step duration (number of microseconds)

At the end of these steps, the PCS Calculator produces all register values.

3.6 PCS sample oscilloscope plots

The plots below show an MPC5777M device in a non-PCS operation and PCS enabled operation. IDD_LV current is initially around 50 mA with all the cores (three e200z7 and one e200z4) running from a divided 16 MHz RCOSC. When the Mode Entry switch occurs, the system clock is switched to run at full speed using PLL0 at 600 MHz (e200z7 cores with a divide-by-2 to 300 MHz and e200z4 with a divide-by-3 to 200 MHz) which brings the current up to approximately 700 mA.

In the following figure, PCS is not enabled. The current changes by approximately 700 mA in 6 microsecond and the mode switch is brief as indicated by a single toggle of the port pin. The low-bandwidth supply is not capable of providing this rate of current change which results in VDD_LV dropping by 150 mV for 10 microsecond. This glitch could cause the device to reset if VDD_LV goes below the Low Voltage Detect (LVD) level.

The oscilloscope channel definition is:

YELLOW ch1 – IDD_LV current at 200 mA/div.

BLUE ch3 – VDD_LV at 50 mV/div.

GREEN ch4 – software controlled toggling pin. Toggle rate is proportional to system clock frequency and pin toggles only during the mode transition.

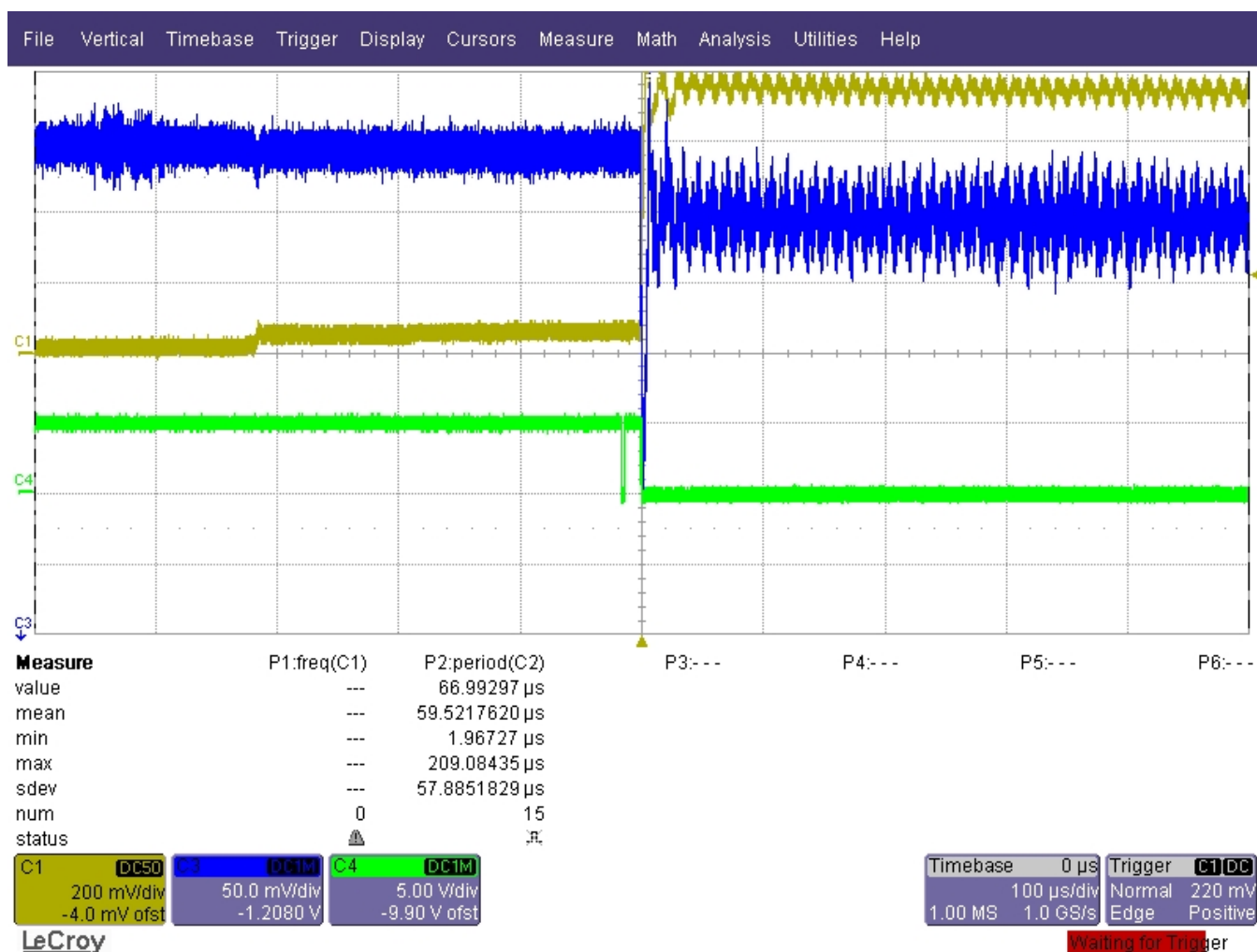


Figure 7. IDD_LV, VDD_LV, and I/O toggle rate with PCS disabled

In the next figure, PCS has been enabled with a rate equal to 0.05 as per the software example listed in this document. PCS causes the system clock frequency to ramp up over a longer time period as can be seen by the increasing toggling pin rate. Now the current ramp time is almost 500 μ s and there are no glitches visible on VDD_LV during the switch.

YELLOW ch1 – IDD_LV current at 200 mA/div.

BLUE ch3 – VDD_LV at 50 mV/div.

GREEN ch4 – software controlled toggling pin. Toggle rate is proportional to system clock frequency and pin toggles only during the mode transition.

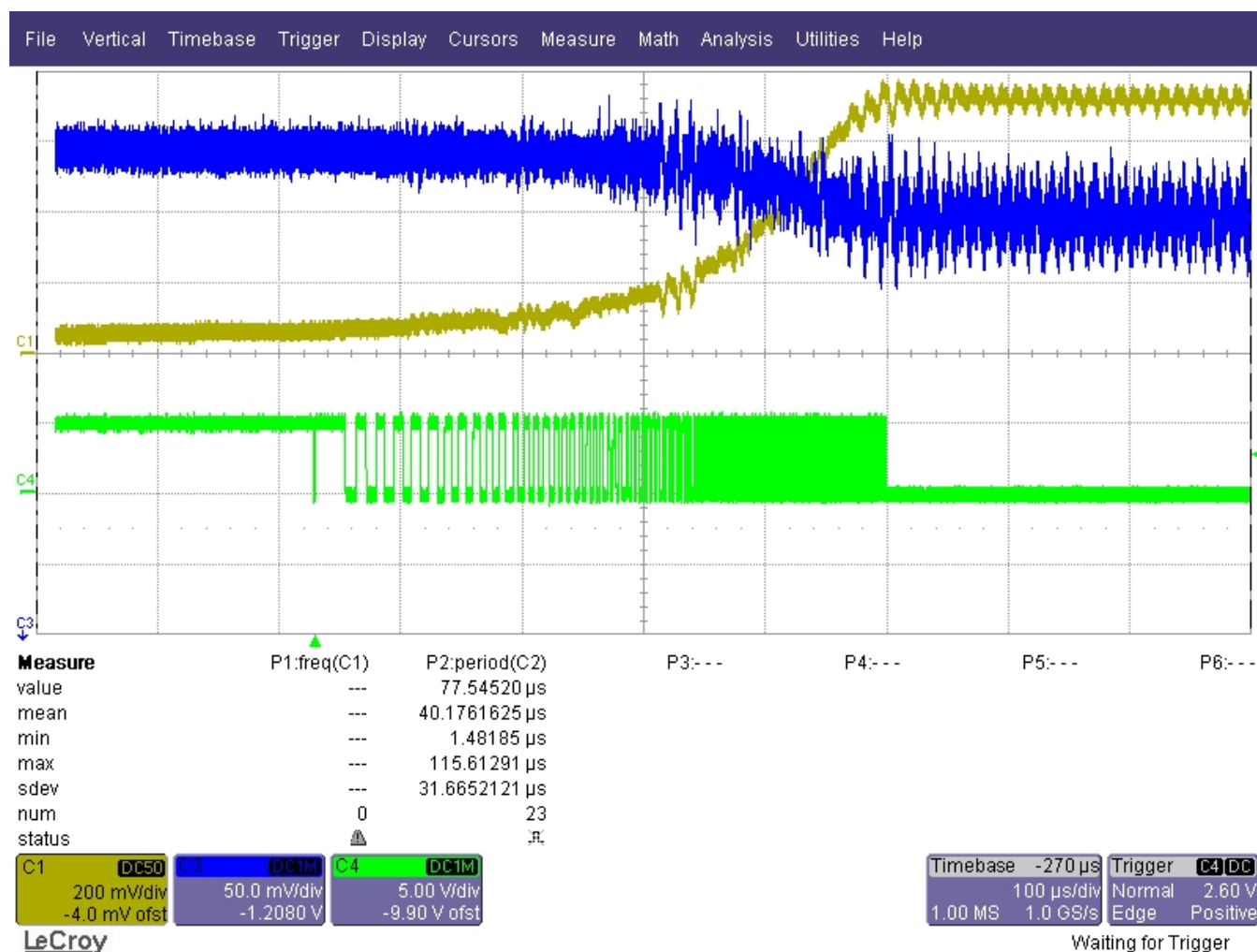


Figure 8. IDD_LV, VDD_LV, and I/O toggle rate with PCS enabled

4 Targeting code using Green Hills Systems (GHS) tools

When developing software for the MPC57xx family of devices, it may be beneficial to target application software to run from different memory locations: Flash, System RAM, and Core Local RAM targets can be used to accommodate development, test, and system performance needs.

The following two examples illustrate how GHS tools can support flexible software targeting in the MPC57xx family.

- Example 1 targets functions to run from System RAM and Core 2 Local RAM
- Example 2 targets software to run from Flash (ROM) and Core 2 Local RAM. In this example, specific code is copied from ROM to RAM for execution.

4.1 Example 1: Targeting SW to System RAM and Core Local Instruction RAM (I-MEM)

In this example, the system requires the main body of functions (e.g. `main()`, `myFunction()`, `ramInit()`, `i2cInit()`) to run from system RAM, base address of `0x4000_0000`. The system also requires the `MC_MODE_INIT()` function to run from Core 2 local memory, base address of `0x5200_0000`. The resulting memory map will contain the following sections:

- .init -> initialization code, located at 0x4000_0000 (System RAM)
- .vtext -> main(), myFunction(), ramInit(), i2cInit(), located at 0x4000_01C0 (System RAM)
- .mc_mode_init -> clock initialization function, located at 0x5200_0000 (I-MEM)

The following diagram illustrates the software locations:

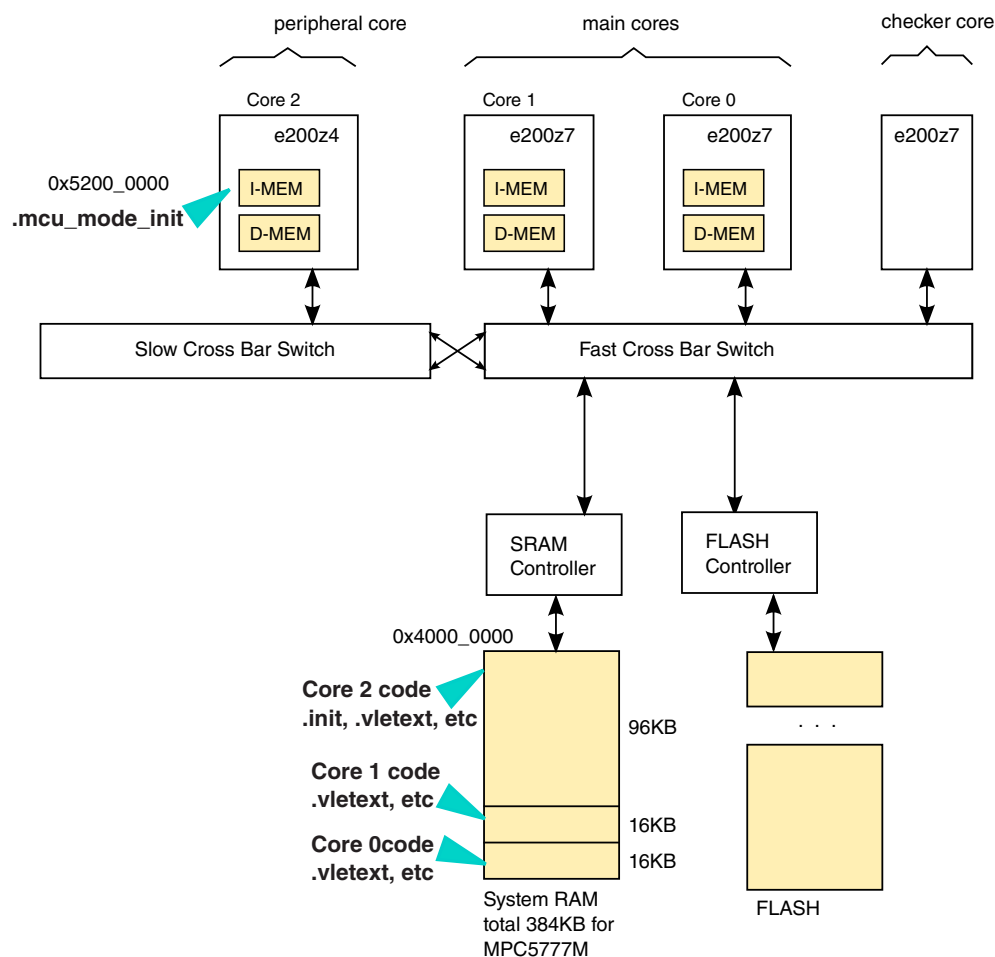


Figure 9. RAM locations (System RAM and Core 2 Local RAM)

There are two things required:

1. define the section by usage of a #pragma declaration in C source file
2. define the section map by using the SECTIONS directive in the linker file

The section definition is shown below - the first instance is a declaration of mc_mode_init in the mc_mode_init.c file, the second instance is a declaration for .init in the crt0_:

```

/*****
/* 1st instance - declaration in C file */
/* section name = mc_mode_init */
*****/

#include "../headers/project.h"

/***** GHS Section Information *****/
// define mc_mode_init as a section
// see also linker file

```

Targeting code using Green Hills Systems (GHS) tools

```
#pragma ghs section vletext=".mc_mode_init"

/***** Configure the Mode and Clock Tree *****/
void MC_MODE_INIT(void){
...

/*****/
/* 2nd instance - declaration in assembly file */
/* section name = .init */
/*****/

#***** GHS Section Information *****/
.section .init , axv
.vle

_start_core2:

#*****
```

The memory and section maps are defined in the linker file (e.g. tgt/flash_z4_core2.ld) and are shown below:

```
/*****/
MEMORY {
  int_sram : org = C2_SRAM_BASE_ADDR, len = C2_SRAM_SIZE-STACK_SIZE
  int_iram : org = C2_LOCALIMEM_BASE_ADDR, len = C2_LOCALIMEM_SIZE
  int_dram : org = C2_LOCALDMEM_BASE_ADDR, len = C2_LOCALDMEM_SIZE-STACK_SIZE
  stack_ram : org = (C2_LOCALDMEM_BASE_ADDR+C2_LOCALDMEM_SIZE-STACK_SIZE) len =
  STACK_SIZE
}

/*****/
GHS SECTIONS DEFINITION *****/
SECTIONS
{
  .init : {} > int_sram // system RAM
  .mc_mode_init : {mcu_init.o(.vletext)} > int_iram // Core2 local RAM
  .vletext : {} > int_sram // main & other VLE code

  // <include other definition here including stack, heap, data, interrupt vectors>
}

/*****/
```

The *.map file can be checked to confirm if the targeting compiled and linked successfully. Excerpts from the map file are shown below:

```
Link Date:Thu Nov 08 15:36:32 2012
Host OS:Windows XP Service Pack 3
Version:ELXR 2012.5.4 (c) Green Hills Software Build: Aug 27 2012
```

```
Load Map Thu Nov 08 15:36:32 2012
Image Summary
```

Section	Base	Size(hex)	Size(dec)	SecOffs
.init	40000000	000001b4	436	0000160
.mc_mode_init	52000000	00000440	1088	0000314
.text	400001b4	00000000	0	0000000
.vletext	400001c0	000005c2	1474	0000760
.fixaddr	40000782	00000000	0	0000000
.fixtype	40000782	00000000	0	0000000
.secinfo	40000782	00000000	0	0000000
.syscall	40000782	00000000	0	0000000
.xptn_vectors	40001000	00000106	262	0001000
.IVOR4_Handler	40002000	00000000	0	0000000
.isrvectbl	40002000	00000ff0	4080	0002000

In the above, `.init` and `.vtext` are targeted to SRAM, which is what we expect. Also, notice `.mcu_mode_init` located at Core 2 local memory starting at 52000000. To run a final confirmation on our project, you can load the code to the Matterhorn EVB and step through the code with a debugger to confirm `main()` executes from System RAM and `MC_MODE_INIT()` executes from Core 2 local memory.

References:

GHS MULTI: Building Applications for Embedded Power Architecture, available in <http://www.ghs.com>. See the following topics:

- Text and Data Placement
- Assigning Data to Custom Program Sections in C
- Defining a Section Map with the `SECTIONS` Directive

4.2 Example 2: Targeting SW to Flash and copying functions to run from Core Local Instruction RAM (I-MEM)

It is also useful to have application software targeted to run from Flash and smaller functions targeted to run from System RAM and/or Core local memory. The GHS compiler and linker tools support a Copy ROM feature which allows functions to be copied from Flash to System RAM / Core Local Memory. The following diagram illustrates software locations for this example:

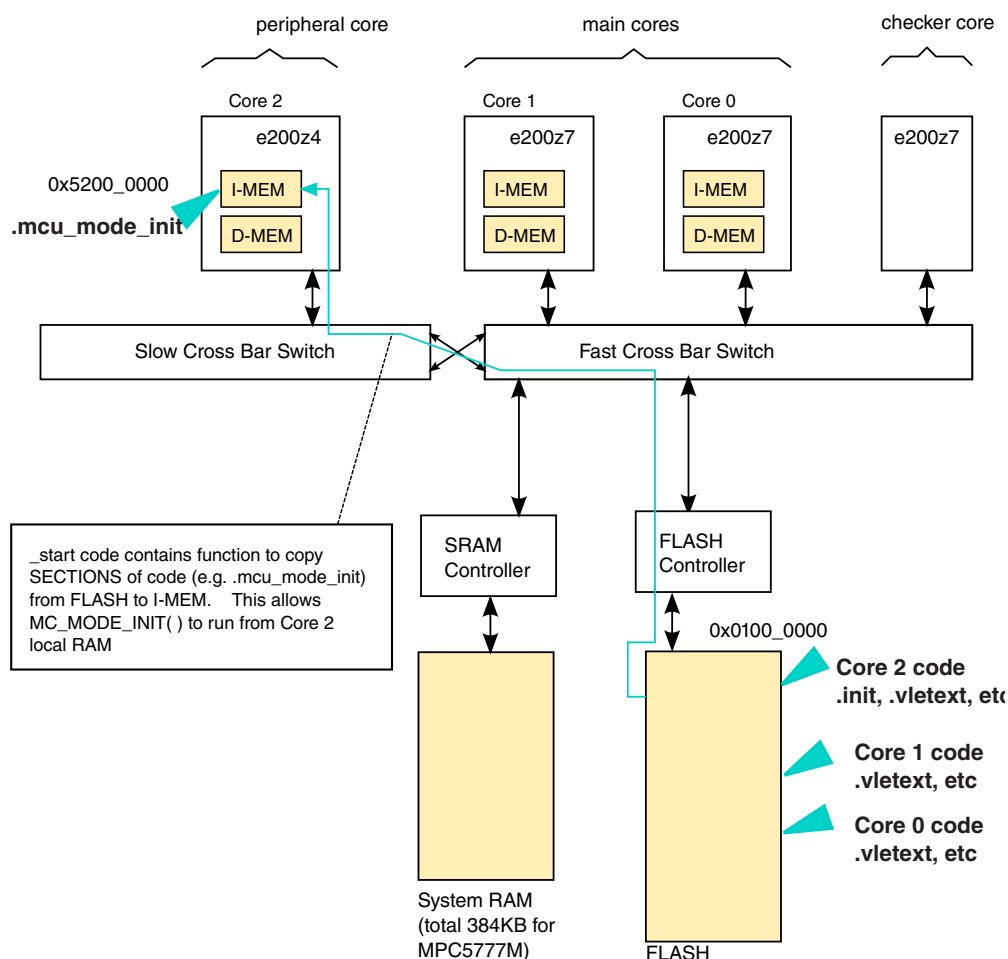


Figure 10. Memory locations (Flash with copy to Core 2 Local RAM)

Targeting code using Green Hills Systems (GHS) tools

There are two things the user must do to accomplish Flash based ROM copy to RAM:

1. define the section for copy by usage of a #pragma declaration in C source file
2. define the section map by using the SECTIONS directive and the copy ROM (ROM) directive in the linker file

NOTE

The programmer can use GHS start code to copy the code from Flash to RAM. In this example, the GHS start code copies the code in section "mc_mode_init" from Flash (int_flash) memory to Core 2 local RAM (int_iram).

The section definition is shown below:

```
/* *****
/* Example of GHS Section declaration in C file
/* section name = mc_mode_init
/* *****

#include "../headers/project.h"

/* ***** GHS Section Information *****
#pragma ghs section vletext=".mc_mode_init"

/* ***** Configure the Mode and Clock Tree *****
void MC_MODE_INIT(void){
...

/* *****
```

Once the section definition is in place, the user modifies the SECTIONS portion of the linker file to specify where to copy the section from ROM to RAM. This example illustrates the modifications to the crt0_core_flash.ld linker file:

```
/* *****
MEMORY {
    bootflash : org = BOOTFLASH_BASE_ADDR, len = BOOTFLASH_SIZE
    int_flash : org = FLASH_BASE_ADDR, len = FLASH_SIZE
    int_sram : org = C2_SRAM_BASE_ADDR, len = C2_SRAM_SIZE-STACK_SIZE
    int_iram : org = C2_LOCALIMEM_BASE_ADDR, len = C2_LOCALIMEM_SIZE
    int_dram : org = C2_LOCALDMEM_BASE_ADDR, len = C2_LOCALDMEM_SIZE-STACK_SIZE
    stack_ram : org = (C2_LOCALDMEM_BASE_ADDR+C2_LOCALDMEM_SIZE-STACK_SIZE) len = STACK_SIZE
}

/* *****
SECTIONS
{
    .bh : {} > bootflash

    .isrvectbl ALIGN(0x1000) : {} > int_flash // ISR Vector Table
    .xptn_vectors ALIGN(0x1000) : {} > . // Exception Vector Table (IVPR)

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

    .ROM.data ROM(.data) : {} > . // Store Initialised RAM Variables
    .ROM.sdata ROM(.sdata) : {} > . // temporarily in Flash

    .data : {} > int_sram // Initialised Data
    .bss : {} > . // Uninitialized Data
    .sdabase ALIGN (2): {} > . // Base location for SDA Area
    .sdata : {} > . // Small Initialized Data (Areal)
```

```
.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

// Modifications for Flash (ROM) to RAM copy
// target for mc_mode_init is Core 2 local memory
.mc_mode_init   : {mcu_init.o(.vletext)} > int_iram
.ROM.mc_mode_init ROM(.mc_mode_init) : {}           > int_flash

/*****/
```

The *.map file can be checked to confirm if the targeting compiled and linked successfully. Excerpts from the map file are shown below:

```
Link Date:Mon Nov 12 09:31:45 2012
Host OS:Windows XP Service Pack 3
Version:ELXR 2012.5.4 (c) Green Hills Software   Build: Aug 27 2012
```

```
Load Map Mon Nov 12 09:31:45 2012
Image Summary
```

Section	Base	Size(hex)	Size(dec)	SecOffs
.bh	01000000	00000020	32	0000200
.isrvectbl	01001000	00000ff0	4080	0001000
.xptn_vectors	01002000	00000106	262	0002000
.init	01002110	000002e0	736	0002110
.text	010023f0	00000000	0	0000000
.vletext	010023f0	00000d6a	3434	00023f0
.fixaddr	0100315a	00000000	0	0000000
.fixtype	0100315a	00000000	0	0000000
.secinfo	0100315c	00000048	72	000315c
.syscall	010031a4	00000006	6	00031a4
.IVOR4_HWvectors	01004000	00000000	0	0000000
.rodata	01004000	00000000	0	0000000
.ROM.data	01004000	0000007c	124	00031ac
.ROM.sdata	0100407c	00000000	0	0000000
.data	40000000	0000007c	124	00031ac
.bss	4000007c	00000204	516	0000000
.sdabase	40000280	00000000	0	0000000
.sdata	40000280	00000000	0	0000000
.sbss	40000280	00000000	0	0000000
.sdata2	40000280	00000000	0	0000000
.sbss2	40000280	00000000	0	0000000
.heap	40000280	00000400	1024	0000000
.stack	5280fc00	00000400	1024	0000000
.my_function	52000000	00000000	0	0000000
.ROM.my_function	0100407c	00000000	0	0000000
.mc_mode_init	52000000	0000057e	1406	0003228
.ROM.mc_mode_init	0100407c	0000057e	1406	0003228
.debug_info	00000000	0002fe1d	196125	00037a6
.debug_abbrev	00000000	0000056a	1386	00335c3
.debug_line	00000000	00000877	2167	0033b2d
.debug_macinfo	00000000	000007bf	1983	00343a4
.debug_frame	00000000	000003d8	984	0034b63
.PPC.EMB.apuinfo	00000000	00000020	32	0034f3b
.gstackfix	00000000	0000000c	12	0034f5c

To fully verify the ROM to RAM copy is functioning, the user can program the *.elf file to Flash and use a debugger to step through the code. The main() function should be executing from Flash (0x010023F0). The MC_MODE_INIT() function should be executing from core 2 local memory (0x5200_0000).

References:

GHS MULTI: Building Applications for Embedded Power Architecture, available in <http://www.ghs.com>. See the following topics:

- Text and Data Placement
- Assigning Data to Custom Program Sections in C
- Defining a Section Map with the SECTIONS Directive
- Copying a Section from ROM to RAM at Startup

5 Appendix A

This appendix contains sample software for Run Mode, Clocks, PCS, and System Memory Protection Unit (SMPU) initialization.

5.1 C code listing

For the purposes of this document, we shall consider a typical application consisting of a main function, a function call to MC_MODE_INIT(), and SMPU_Init().

The key premise behind the code locations is to have:

- main() and user application code all run from main System RAM and/or Flash
- MC_MODE_INIT runs from Core 2 local memory

Sample software is shown below for main(). The main() function shall be target to run either from SRAM (0x4000_0000) or Flash (see MPC5777M RM for possible Flash target addresses).

```
*=====*
```

```
* COPYRIGHT:                                     *
```

```
* Freescale Semiconductor, INC. All Rights Reserved. You are hereby          *
```

```
* granted a copyright license to use, modify, and distribute the              *
```

```
* SOFTWARE so long as this entire notice is retained without alteration      *
```

```
* in any modified and/or redistributed versions, and that such modified     *
```

```
* versions are clearly identified as such. No licenses are granted by        *
```

```
* implication, estoppel or otherwise under any patents or trademarks         *
```

```
* of Freescale Semiconductor, Inc. This software is provided on an           *
```

```
* "AS IS" basis and without warranty.                                         *
```

```
*                                                                              *
```

```
* To the maximum extent permitted by applicable law, Freescale                *
```

```
* Semiconductor DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED,         *
```

```
* INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A          *
```

```
* PARTICULAR PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD       *
```

```
* TO THE SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY         *
```

```
* ACCOMPANYING WRITTEN MATERIALS.                                           *
```

```
*                                                                              *
```

```
* To the maximum extent permitted by applicable law, IN NO EVENT             *
```

```
* SHALL Freescale Semiconductor BE LIABLE FOR ANY DAMAGES WHATSOEVER         *
```

```
* (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS,       *
```

```
* BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER            *
```

```
* PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.      *
```

```
*                                                                              *
```

```
* Freescale Semiconductor assumes no responsibility for the                  *
```

```
* maintenance and support of this software                                 *
```

```
*                                                                              *
```

```
*****/
```

```

/*****/
/* FILE NAME: main.c                                COPYRIGHT (c) Freescale 2013 */
/*                                                    All Rights Reserved */
/* DESCRIPTION: Main C program                      */
/*                                                    */

```

```

/*****
/* REV      AUTHOR      DATE      DESCRIPTION OF CHANGE      */
/* ---      - - - - -    - - - - -    - - - - -                */
/* 1.0 D.McMenamin    Sep 13, 2012  Initial Public Release      */
/* 1.1 C.Hillier      May 17, 2013  Adapted for Matterhorn      */
/*****
#include "../headers/project.h"
extern void MC_MODE_INIT(void);
extern void SMPU_Init(void);

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

    MC_MODE_INIT( );          // Setup the MCU clocks and modes
    SMPU_Init( );             // Configure the System Memory Protection Unit
    /* Insert User code here */

}

```

MC_MODE_INIT() and SMPU_Init() sample code is shown below. MC_MODE_INIT() shall be targeted to run from Core 2 local memory with a starting address of 0x5200_0000.

NOTE

This is required for MPC5777M Cut 1 devices. It is optional for MPC5777M Cut 2 devices.

```

*=====
* COPYRIGHT:
* Freescale Semiconductor, INC. All Rights Reserved. You are hereby
* granted a copyright license to use, modify, and distribute the
* SOFTWARE so long as this entire notice is retained without alteration
* in any modified and/or redistributed versions, and that such modified
* versions are clearly identified as such. No licenses are granted by
* implication, estoppel or otherwise under any patents or trademarks
* of Freescale Semiconductor, Inc. This software is provided on an
* "AS IS" basis and without warranty.
*
* To the maximum extent permitted by applicable law, Freescale
* Semiconductor DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED,
* INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
* PARTICULAR PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD
* TO THE SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY
* ACCOMPANYING WRITTEN MATERIALS.
*
* To the maximum extent permitted by applicable law, IN NO EVENT
* SHALL Freescale Semiconductor BE LIABLE FOR ANY DAMAGES WHATSOEVER
* (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS,
* BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER
* PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.
*
* Freescale Semiconductor assumes no responsibility for the
* maintenance and support of this software
*
*=====
/*****
/* FILE NAME: mcu_init.c                                COPYRIGHT (c) NXP 2016 */
/*                                                    All Rights Reserved */
/* DESCRIPTION: Provides functions to Initialize MPC5777M MCU */
/*                                                    */
/*!*****
/*! REV  AUTHOR      DATE      DESCRIPTION OF CHANGE      */
/*! ---  - - - - -    - - - - -    - - - - -                */
/*! 1.0 C.Hillier    May 17, 2013  Initial Public Release      */
/*! 2.0 C.Hillier    Dec 5, 2013   Updated for MPC5777M cut 2    */
/*! 2.1 C.Hillier    Dec 14, 2016  Updated PCS settings         */
/*!*****

```

Appendix A

```
#include "project.h"

// define mc_mode_init as a section
// see also linker file
#pragma ghs section vtext=".mc_mode_init"
#pragma ghs section data="mc_mode_init_data"
/*****
/*! MPC5777M Matterhorn PLL and Clock Configuration */
/*! Configure the Mode and Clock Tree */
*****/
void MC_MODE_INIT(void){
int i;
/*! 1 Clear any faults */
/*! Clear faults | MC_RGM.DES, MC_RGM.FES, and MC_ME.ME */
MC_RGM.DES.R = 0xFFFF;
MC_RGM.FES.R = 0xFFFF;
MC_ME.ME.R = 0x000005FF;

/*! 2 Set up peripheral run modes */
/*! Enable the modes Required | MC_ME.ME */
MC_ME.ME.R = 0x0000800F;
/*! Add MC_ME.PCTL[x].R initializations here */
/*! Set RUN Configuration Registers | MC_ME.RUN_PC[n] */
MC_ME.RUN_PC[0].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[1].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[2].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[3].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[4].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[5].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[6].R=0x000000FE; /* Peripheral ON in every mode */
MC_ME.RUN_PC[7].R=0x000000FE; /* Peripheral ON in every mode */
/*! 3 Configure System Clock Dividers */
/*! Configure System clock dividers */
/*! Full speed Core 0 / 1 = 300 MHz. PLL1 = 600 MHz. */
MC_CGM.SC_DIV_RC.R = 0x00000001; /* System clock divider ratios will */
/* change with next update. */
MC_CGM.DIV_UPD_TYPE.R = 0x80000000; /* System clock divider ratios updated */
/* on writing MC_CGM.DIV_UPD_TRIG. */
MC_CGM.SC_DC2.R=0x800B0000; /*! PBRIDGE Clock Divide by 12 (50 MHz) */
MC_CGM.SC_DC1.R=0x80050000; /*! SXBAR Clock Divide by 6 (100 MHz) */
MC_CGM.SC_DC0.R=0x80020000; /*! FXBAR Clock Divide by 3 (200 MHz) */
MC_CGM.SC_DC3.R=0x80010000; /*! Core0/1 Clock Divide by 2 (300 MHz) */
MC_CGM.SC_DC4.R=0x80020000; /*! System Clock Divide by 3 (200 MHz) */
MC_CGM.DIV_UPD_TRIG.R = 0xfeedface; /*! System clock divider ratio updates */
/* triggered. */
while (MC_CGM.DIV_UPD_STAT.B.SYS_UPD_STAT == 1); /*! Wait for System Clock */
/* Div Update Status == 0. */
/*! 4 Configure Auxiliary Clock Dividers */
/*! Enable and configure Aux clocks */
MC_CGM.AC0_DC0.R=0x00000000; /* before clock source update, */
/* disable Aux Clock 0 divider 0 */
/* since enabled by default after reset */
MC_CGM.AC0_SC.B.SELCTL=2; /* set PLL0 PHI for Aux Clock 0 */

MC_CGM.AC0_DC0.R=0x80040000; // program Aux Clock 0 divider 0
// peripheral clock -> Divide by = 4 + 1
// 400 MHz/5 = 80 MHz
MC_CGM.AC0_DC1.R=0x80180000; // program Aux Clock 0 divider 1
// SDADC clock -> Divide by 24 + 1.
// 400 MHz / 25 = 16 MHz
MC_CGM.AC0_DC2.R=0x801B0000; // program Aux Clock 0 divider 2
// SARADC clock -> Divide by 27 + 1
// 400 MHz / 28 = 14.6 MHz
MC_CGM.AC0_DC3.R=0x80030000; // program Aux Clock 0 divider 3
// DSPI_CLK0 -> Divide by 3 + 1
// 400 MHz / 4 = 100 MHz
MC_CGM.AC0_DC4.R=0x80030000; // program Aux Clock 0 divider 4
// DSPI_CLK1/LIN_CLK -> Divide by 3 + 1
// 400 MHz / 4 = 100 MHz
```

```

MC_CGM.AC2_DC0.R=0x80090000; // program Aux Clock 2 divider 0
// FlexRay -> Divide by 9 + 1
// 400 MHz / 10 = 40 MHz
MC_CGM.AC2_DC1.R=0x80090000; // program Aux Clock 2 divider 1
// SENT -> Divide by 9 + 1
// 400 MHz / 10 = 40 MHz
MC_CGM.AC5_DC0.R=0x80090000; // program Aux Clock 5 divider 0
// PSI5 -> Divide by 9 + 1
// 400 MHz / 10 = 40 MHz
MC_CGM.AC5_DC1.R=0x80090000; // program Aux Clock 5 divider 1
// PSI5 -> Divide by 9 + 1
// 400 MHz / 10 = 40 MHz
MC_CGM.AC5_DC2.R=0x80090000; // program Aux Clock 5 divider 2
// PSI5 -> Divide by 9 + 1
// 400 MHz / 10 = 40 MHz
/* CAN Clock Runs from XOSC by Default */
MC_CGM.AC8_DC0.R=0x80070000; // program Aux Clock 8 divider 0
// CAN Clock-> Divide by 8
MC_CGM.AC9_SC.B.SELCTL=1; // Select XOSC for Aux Clock 9
MC_CGM.AC9_DC0.R=0x80030000; // program Aux Clock 9 divider 0
// RTI/PIT-> Divide by 4
MC_CGM.AC10_DC0.R=0x800F0000; // program Aux Clock 10 divider 0
// ENET -> Divide by 15 + 1
// 400 MHz / 16 = 25 MHz
/*! Set the PRAMC Flow Through disable */
/*! SRAM requires additional wait state */
/*! Note: Do not change the FT_DIS bit while accessing System RAM.
Relocate code programming the FT_DIS bit to another memory area
(e.g. local Core memory). */
/*! Also, set the FT_DIS bit after programming clock dividers and before
setting PLLs and executing the Mode Entry change */
PRAMC.PRCR1.B.FT_DIS = 1; /*! Set Flow Through Disable. */

/*! Step 5 --- CONFIGURE XOSC PLL0 PLL1 --- */
/*! Route XOSC to the PLLs - IRC is default */

MC_CGM.AC3_SC.B.SELCTL=1; /*! Connect XOSC to PLL0 */
MC_CGM.AC4_SC.B.SELCTL=1; /*! Connect XOSC to PLL1 */

/*! Configure PLL0 Dividers - 400 MHz from 40 MHz XOSC */
PLLDIG.PLL0DV.B.RFDPHI = 1;
PLLDIG.PLL0DV.B.PREDIV = 4;
PLLDIG.PLL0DV.B.MFD = 40;
/*! fPLL0_VCO = (fpll0_ref * 2 * MFD) / PREDIV
/*! fPLL0_VCO = (40 MHz * 2 * 40) / 4
/*! fPLL0_VCO = 800 MHz
/*!
/*! fPLL0_PHI = (fpll0_ref * 2 * MFD) / (PREDIV * RFDPHI * 2)
/*! fPLL0_PHI = (40 MHz * 2 * 40) / (4 * 1 * 2)
/*! fPLL0_PHI = 400 MHz

/*! Put PLL0 into Normal mode */
PLLDIG.PLL0CR.B.CLKCFG = 3;
/*! Configure PLL1 Dividers - 600 MHz from 40 MHz XOSC */
PLLDIG.PLL1DV.B.RFDPHI = 1;
PLLDIG.PLL1DV.B.MFD = 30;
/*! fPLL1_VCO = (fpll1_ref * MFD)
/*! fPLL1_VCO = (40 MHz * 30)
/*! fPLL1_VCO = 1200 MHz
/*!
/*! fPLL1_PHI = (fpll0_ref * MFD) / (RFDPHI * 2)
/*! fPLL1_PHI = (40 MHz * 30) / (1 * 2)
/*! fPLL1_PHI = 600 MHz

/*! Put PLL1 into Normal mode */
PLLDIG.PLL1CR.B.CLKCFG = 3;
/*! 6 CONFIGURE PROGRESSIVE CLOCK SWITCHING (PCS), Configure Progressive Clock Switching
(PCS) to prevent glitches - 0.05 rate 70 steps. */
MC_CGM.PCS_SDUR.R = 16; /*! set Switch Duration */
MC_ME.DRUN_MC.B.PWRLVL=3; /*! Configure DRUN power level */

```

Appendix A

```
/*! Configure PLL1 PCS switch | See RM section "Progressive system clock switching" */
MC_CGM.PCS_DIVC4.B.INIT = 660; /*! Set the Divider Change Initial Value */
MC_CGM.PCS_DIVC4.B.RATE = 12; /*! Set the Divider Change Rate */
MC_CGM.PCS_DIVS4.R = 19479; /*! Set the Divider Start Value. */
MC_CGM.PCS_DIVE4.R = 18749; /*! Set the Divider End Value */

/*! 7 ----- Initialize e200z Cores ----- */
/* Enable cores if running from RAM and not using the BAF */
/*! Enable Cores - Will start on next mode transition */
/*! If core n is enabled, then */
/*! - Set MC_ME.CADDR[n] to the code start address (see linker file) */
/*! - Set MC_ME.CCTL[n] to enable core in all modes */
MC_ME.CCTL0.R = 0x00FE;
/* RAM addresses */
MC_ME.CADDR1.R = 0x40010001; /* Set Core 0 Start Address */
MC_ME.CCTL1.R = 0x00FE; /* Set modes in which Core 0 will run. */

MC_ME.CCTL2.R = 0x00FE; /* Set modes in which Checker Core will run.*/
MC_ME.CADDR3.R = 0x40012001; /* Set Core 1 Start Address */
MC_ME.CCTL3.R = 0x00FE; /* Set modes in which Core 1 will run. */

MC_ME.CADDR4.R = 0x40012001; /* Set HSM Start Address */
MC_ME.CCTL4.R = 0x00FE; /* Set modes in which HSM will run. */

/* 8 ----- Perform Mode Entry change ----- */
/* Set the System Clock. Enable XOSC and PLLs - PLL1 is sysclk, PWRLVL = 3. */
MC_ME.DRUN_MC.R = 0x301300F4;
/* Execute mode change: */
/* Re-enter the DRUN mode to start cores, clock tree dividers, PCS, and PLL1 */
MC_ME.MCTL.R = 0x30005AF0; /*! Write Mode and Key */
MC_ME.MCTL.R = 0x3000A50F; /*! Write Mode and Key inverted */
while(MC_ME.GS.B.S_MTRANS == 1); /*! Wait for mode entry complete */
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); /*! Check DRUN mode entered */
}

/*!*****
/*! MPC5777M Matterhorn System Memory Protection Unit (SMPU) configuration */
/*!*****

void SMPU_Init(void)
{
    // This code configures SMPU_0.
    // SMPU_0 supports Core 0, Core 1, and the checker core.

    /* Configure Flash region - cacheable */
    SMPU_0.RGD0_WORD0.R = 0x00400000; // Start Address
    SMPU_0.RGD0_WORD1.R = 0x013BFFFF; // End Address
    SMPU_0.RGD0_WORD2.R = 0xFFFFFFFF; // Permissions - set read / write for all masters
    SMPU_0.RGD0_WORD3.B.VLD = 1; // Valid bit

    /* Configure Flash region - cache inhibited */
    SMPU_0.RGD1_WORD0.R = 0x013C0000; // Start Address
    SMPU_0.RGD1_WORD1.R = 0x013FFFFFFF; // End Address
    SMPU_0.RGD1_WORD2.R = 0xFFFFFFFF; // Permissions - set read / write for all masters
    SMPU_0.RGD1_WORD3.B.CI = 1; // Cache-inhibited
    SMPU_0.RGD1_WORD3.B.VLD = 1; // Valid bit

    /* Configure Flash region - cache inhibited */
    SMPU_0.RGD2_WORD0.R = 0x01400000; // Start Address
    SMPU_0.RGD2_WORD1.R = 0x01FFFFFFF; // End Address
    SMPU_0.RGD2_WORD2.R = 0xFFFFFFFF; // Permissions - set read / write for all masters
    SMPU_0.RGD2_WORD3.B.CI = 1; // Cache-inhibited
    SMPU_0.RGD2_WORD3.B.VLD = 1; // Valid bit
}
```

```

/* Configure RAM region */
SMPU_0.RGD3_WORD0.R = 0x40000000;
SMPU_0.RGD3_WORD1.R = 0x60000000;
SMPU_0.RGD3_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD3_WORD3.B.VLD = 1;

/* Configure 16kB of overlay RAM region */
SMPU_0.RGD4_WORD0.R = 0x0D000000;
SMPU_0.RGD4_WORD1.R = 0x0D004000;
SMPU_0.RGD4_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD4_WORD3.B.VLD = 1;

/* Configure 1MB of Emulation Device overlay RAM region */
SMPU_0.RGD5_WORD0.R = 0x0C000000;
SMPU_0.RGD5_WORD1.R = 0x0C100000;
SMPU_0.RGD5_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD5_WORD3.B.VLD = 1;

/* Configure mirrored Flash region - overlaid; Cache-Inhibit = TRUE */
/* User Note: Configure this space to align with the amount of overlay RAM used */
SMPU_0.RGD6_WORD0.R = 0x08FC0000;
SMPU_0.RGD6_WORD1.R = 0x090FFFFFF;
SMPU_0.RGD6_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD6_WORD3.B.CI = 1;
SMPU_0.RGD6_WORD3.B.VLD = 1;

/* Configure mirrored Flash region - non-overlaid */
SMPU_0.RGD7_WORD0.R = 0x09100000;
SMPU_0.RGD7_WORD1.R = 0x093BFFFF; // McKinley
// SMPU_0.RGD6_WORD1.R = 0x0977FFFF; // Matterhorn
SMPU_0.RGD7_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD7_WORD3.B.VLD = 1;

/* Configure HSM (internal to HSM) */
SMPU_0.RGD8_WORD0.R = 0xA0000000;
SMPU_0.RGD8_WORD1.R = 0xA000FFFF;
SMPU_0.RGD8_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD8_WORD3.B.VLD = 1;

/* Configure Peripheral Space, cache-inhibit = TRUE */
SMPU_0.RGD9_WORD0.R = 0xF0000000;
SMPU_0.RGD9_WORD1.R = 0xFFFFFFFF;
SMPU_0.RGD9_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD9_WORD3.B.CI = 1;
SMPU_0.RGD9_WORD3.B.VLD = 1;

/* Matterhorn only */
/* Configure EBI region - non-overlaid */
SMPU_0.RGD10_WORD0.R = 0x10000000;
SMPU_0.RGD10_WORD1.R = 0x2FFFFFFF;
SMPU_0.RGD10_WORD2.R = 0xFFFFFFFF;
SMPU_0.RGD10_WORD3.B.VLD = 1;

/* Enable the module */
SMPU_0.CESR0.B.GVLD = 1;
}

```

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μ Vision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2017 NXP B.V.

Document Number AN4812
Revision 1, 01/2017