NXP Semiconductors
Application Note

Document Number: AN4670
Rev. 0, 12/2016

Initializing the MPC5746R

by: Bill Terry

1 Introduction

MPC5746R MCU is part of a family of devices that contain many new features coupled with high performance 55 nm CMOS technology to provide a substantial reduction of cost per feature and significant performance improvement. Initially intended for automotive powertrain applications, the MPC5746R is a 32-bit microcontroller that implements multiple e200z4 cores built on Power Architecture® technology and that can run at up to 200 MHz.

This application note describes the procedure that occurs during startup on the MPC5746R and describes the software requirements for initializing the device and starting code execution on multiple cores.

Example code is provided along with this application note. This code is intended to be an example of how a typical application can be configured to boot from flash memory.

Contents

1.1 Objective

After reading this application note the user should understand the following:

- The startup procedure of the MCU and the transition to execution of user software
- How to use DCF records to control initial device configurations
- How the boot header operates and how to create a boot header for a multicore application
- How to initialize SRAM memories
- How to configure the flash memory and to enable and use advanced performance features
- How to configure the PLLs and the clock tree
- How to use the Mode Entry module and boot the cores after a power on reset (POR).

2 Example Memory Partitioning

Multicore applications are usually configured to partition the device memory (both flash and SRAM) into sections. Typically, the flash memory is broken up into smaller sections based on the system requirements such as flash blocks that are dedicated program storage for a particular core. Other dedicated areas of the flash may be blocks that contain shared program code segments, shared data blocks (such as constant definitions), private data areas, and blocks that are used for EEPROM emulation. Additionally the various on-chip SRAM memories may be allocated by core.

With multiple cores, there are trade-offs for having either separate flash and SRAM sections, or a common memory section for flash and SRAM that is shared by all cores. A common memory pool allows easier management of memory boundaries and shared code and data. Separate memory areas, however, offer optimization options that are not available with shared memory pools.

This application note uses the separate memories approach. The flash, SRAM, and local memories are allocated as shown in Table 1. For convenience, a single 256 KB flash block is dedicated to each core for this simple example. The remainder of the flash memory is not defined for any particular use.

Table 1. Code memory allocation

		Flash		SRAM		Local memory			
						IMEM		DMEM	
		Base	Size	Base	Size	Base	Size	Base	Size
User DCF[1] Records	OTP[2]	0x0040_0200	28	—	—	—	—	—	—
Boot header Flash	RW	0x0100_0000	20	—	—	—	—	—	—
Core 0	RW	0x0104_0000	256 KB	0x4000_8000	112 KB	0x5100_0000	16K	0x5180_0000	32 KB
Core 1	RW	0x0100_0000	256 KB	0x4002_4000	112 KB	0x5000_0000	16K	0x5080_0000	32 KB

[1] Device Configuration Format records are discussed in Section 4, “DCF Records and Clients,” on page 6.

[2] OTP = One Time Programmable Flash memory

2.1 Flash memory

The device is provided with 4 MB of flash memory. This memory includes flash for the application code as well as blocks for TEST and UTEST data. Additionally, sections of the flash memory are provided for EEPROM emulation.

The TEST and UTEST locations of flash memory contain DCF records that may be modified or added by the user, as well as data that is programmed during factory testing.

2.2 Local memory

Each of the e200z425 cores provide 16 KB of tightly coupled instruction memory (I-MEM) and 32 KB of tightly coupled data memory (D-MEM). These low latency memory resources allow fast core access to instructions and data.

Within the scope of the application, I-MEM and D-MEM may be treated as normal SRAM. They are memory mapped, and using the linker file, the application executable can be built to locate specific code and data into these memory spaces to improve system performance. In the example included with this application note, the linker files are used to locate the stack for each core in the local D-MEM to improve system performance during context switching.

Optionally, the external interrupt vector handler code can be placed in I-MEM. This may be useful improving performance in any application that requires heavy use of the peripheral and system interrupts.

Both I-MEM and D-MEM are provided with ECC error correction/detection and must be initialized before use.

3 Power-on Sequence

This section explains the sequence of events from power on until the device begins execution of the application code.

Powering up and booting the device involves a set modules that work together to ensure the correct functionality:

- The Power Management Controller (PMC) ensures that all voltage levels are within specification
- The Reset Generation Module (MC_RGM) sequences the device through the steps of the reset sequence and interacts with the System Status and Configuration Module (SSCM)
- The System Status and Configuration Module (SSCM) initializes various other modules and configures the device to its default state
- The Boot Assist Flash (BAF) boots the device correctly by locating the boot header
- The Mode Entry Module (MC_ME) controls initial configurations for various modules

3.1 Power Management Controller

When power is applied to the microcontroller, the PMC controls and monitors the various voltage levels around the device. Specifically, the PMC monitors its own supply voltage, the supply voltages to all the high and low-voltage detect circuits, the trip points for all the high- and low-voltage detect circuits, the power supplies and reference voltages to the analog-to-digital converters as well as the major power supplies to the MPC5746R. The PMC holds the device in the POWERUP phase of the reset state machine until all required power domains reach their specified voltage levels. Power sequencing is not required.

3.2 Reset Generation Module

When the device is powered up correctly and exits the POWERUP phase, the MC_RGM takes over the control of the device and manages the reset sequence. The MC_RGM provides a register interface and various registers are available to monitor and control the chip reset sequence. The reset sequencer is a state machine that controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and controls the reset signals generated in the system. Figure 1 shows the reset sequence state machine.

Note in Figure 1 that there are different types of reset that may be generated when the device has reached an operational state. These different levels of reset (destructive, functional, and short) allow some system resources to be maintained in their last state during the reset event.

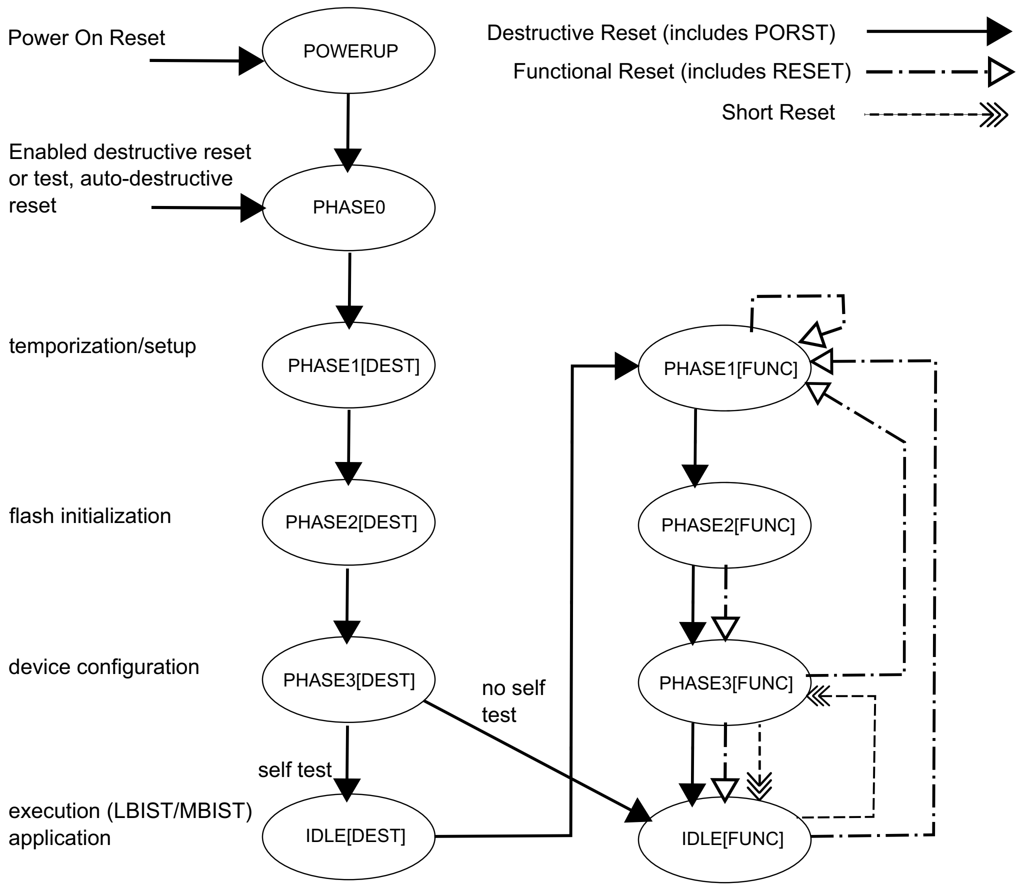

Figure 1. Reset state machine diagram

After a power-on reset (PORST) or a destructive reset, the reset sequence starts at PHASE0. The following phases are used for temporization and setup, flash initialization, and device configuration. The reset state machine then reaches the IDLE phase where the Built-In Self-Test (BIST) is conducted. The BIST is configured during PHASE3 controlled by DCF records. At the end of the BIST, a functional reset is triggered. After a functional reset, the device proceeds at PHASE1[FUNC]. It is also possible to disable the BIST. In this case, the device directly proceeds to phase IDLE[FUNC].

Three active-low reset signals are associated to the internal reset circuitry:

- PORST: released when the device leaves the POWERUP phase. The signal has a strong pull-down when the device is in POWERUP state, and a weak pull-down when it is not in this phase; that is, it has to pulled up externally to bring the device out of reset.
- RESET: released in phase IDLE[FUNC]. Forcing either of the two reset signals low keeps the device in reset.
- RSTOUT: allows a software generated reset output to other devices, independent of internal resets. This function is shared on a GPIO pin. See the MPC5746R Reference Manual for details.

NOTE

It is recommended to use 4.7 kΩ resistors as pull-ups for each of the PORST and RESET signals.

3.3 System Status and Configuration Module

The primary purposes of the SSCM are:

- To provide a mechanism for configuring and initializing the device during system boot
- To provide information about the current state of the system that is useful for configuring application software and for debugging the system

During the reset sequence, the RGM enables the SSCM. The SSCM reads the DCF records in both the TEST and UTEST flash memory areas. The DCF records contain device setup information that the SSCM then uses to write data to various registers in the device. The DCF records are primarily concerned with setting up the memory, configuring the Self-Test Control Unit (STCU), and providing initial device configuration values.

One of the DCF records that the SSCM reads from the TEST flash area contains the vector that points to the starting address of the BAF. This vector is sent to Core 1. Core 1 then executes the BAF to locate the boot header that contains boot information and reset vectors for all cores.

3.4 Boot Assist Flash

The BAF is executable software that is programmed in a 16 KB block of flash memory, mapped adjacent to the UTEST flash memory block (see Section 4.2, “UTEST flash memory,” on page 9). The BAF block base address is 0040_4000h. It is one time programmable (OTP) and is programmed during factory test. In the final stage of the reset sequence, phase IDLE[FUNC], the SSCM sets the Core 1 program counter to the starting address of the BAF, which then executes the BAF code. The main purpose of the BAF is to locate the boot header and copy the relevant information from the boot header into registers of the Mode Entry Module. In the case where the BAF cannot find a valid header, a serial boot load process is started.

3.5 Mode Entry Module

The MC_ME is responsible for delivering initial values to many registers in the device. The MC_ME also supplies reset vectors for all cores.

During the power-up sequence, the SSCM initializes the BAF to search for a valid boot header at predefined locations in the flash memory. The BAF derives the necessary boot information and reset vectors from the boot header. As the boot-up sequence progresses, the SSCM instructs the MC_ME to transfer the reset vectors to the respective processor cores. All CPU cores always receive their reset vectors from the MC_ME even though these reset vectors are in the boot header located in flash memory.

4 DCF Records and Clients

The Device Configuration Format (DCF) record is a mechanism to configure specific registers during system boot and to set up an initial configuration for the device after reset or start up. The DCF records are located in the UTEST section of flash memory. An on-chip register that can be configured by a DCF record is a DCF client.

4.1 DCF records

A DCF record is a 64-bit wide data field that contains 32-bit data that is written to DCF clients, along with address information and check bits as illustrated in Figure 2. DCF records are stored in both TEST and UTEST flash, however only UTEST flash can be written by the user.

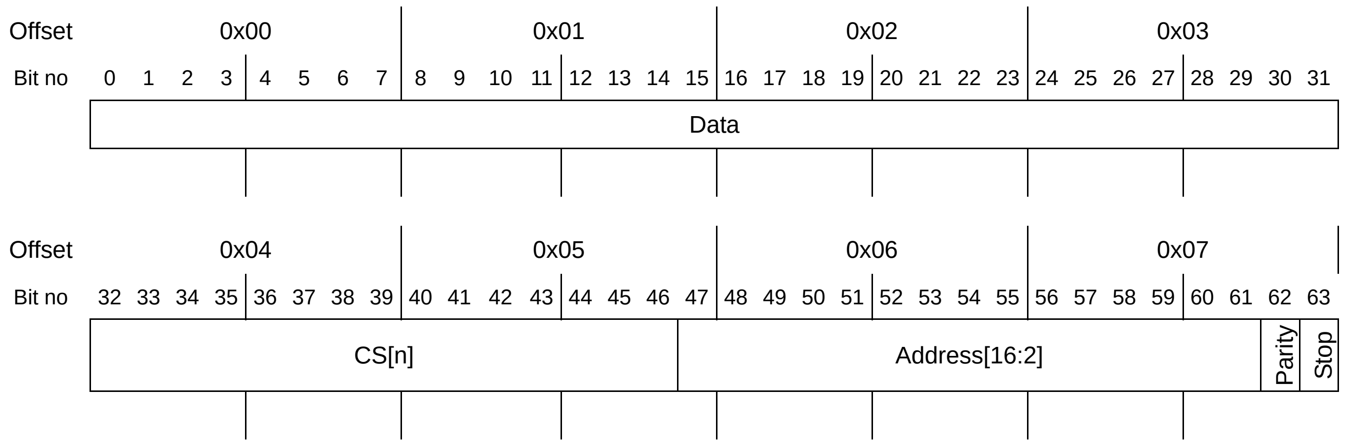

Figure 2. DCF record format

The DCF record bit fields are described below.

Field	Name	Description
0–31	Data[0:31]	32 bits of data that is to be written to the DCF client
32–46	CS[n]	Chip Select *n.* One Chip select is asserted (0b1) per DCF record to select the target module for the DCF client. All other Chip Selects should be negated (0b0).

Field	Name	Description
47–61	Address	Address of the DCF client within the selected module. **Note:** Address decoding for DCF clients may not match the standard software address map decoding. Details of DCF client addresses are defined in each module chapter of the device reference manual.
62	Parity	Parity Bit for the DCF Record. If the DCF record strategy (see the MPC5746R Reference Manual) uses parity, this bit must be set to 0b1.
63	Stop	Indicates the end of DCF records. 0b0 NOT the end of the list 0b1 End of the list **Note:** The erased state of flash is 0xFFFF_FFFF_FFFF_FFFF. Therefore, the list ends with the first unprogrammed double word. This location can be programmed with a new record to extend the list.

The DCF records provide a method to configure many registers in the device, allowing these registers (DCF clients) to be loaded with specific values during system boot.

There are two types of DCF records:

- TEST DCF records:
  - Developed by the factory, used mainly to program registers involved in trimming trip points for voltage comparators, adjusting analog to digital voltage supplies, trimming oscillator frequencies, and enabling RAM repair. The TEST DCF records are programmed into TEST flash during production and cannot be modified. TEST flash is not visible to the user.
- UTEST DCF records:
  - Programmed into UTEST flash, some UTEST DCF records are written by the factory and programmed during production testing. Others are written by the end user and programmed at the same time application code is programmed into the main flash memory. User-supplied UTEST DCF records are located at a specific address in UTEST memory (see Figure 5).

The relationship of TEST flash, UTEST flash, and factory test programming is shown in Figure 3 below.

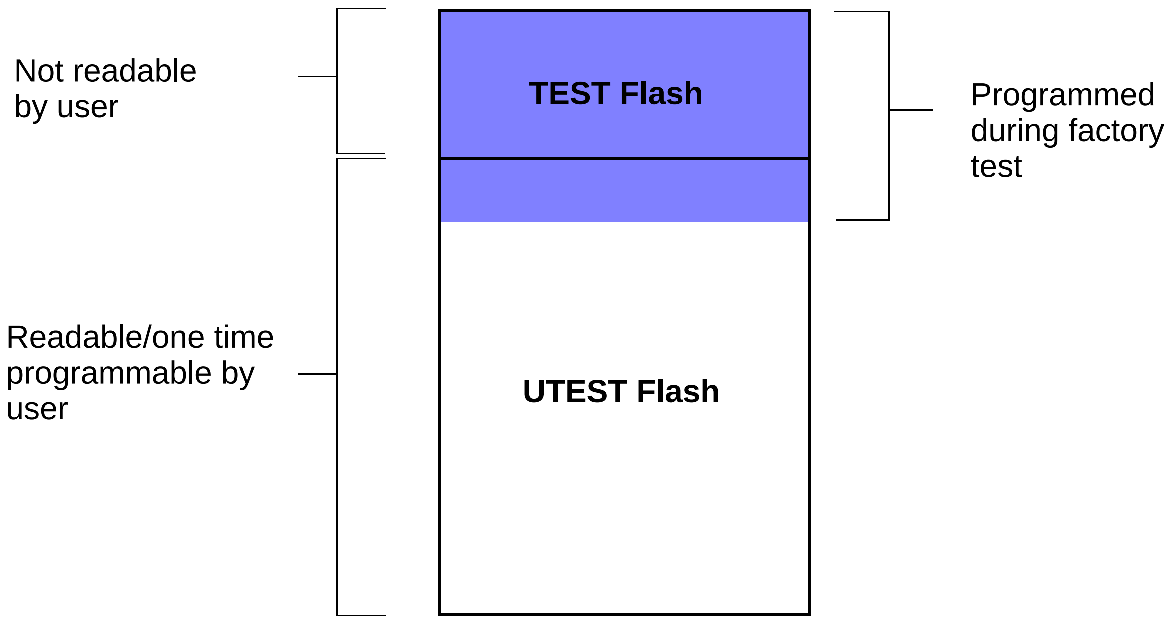
Not readable by user
Readable/one time programmable by user
TEST Flash
UTEST Flash
Programmed during factory test

Figure 3. TEST / UTEST flash memory

DCF clients are 32-bit wide hardware registers inside a module that receive and store the data from a DCF record. This allows boot time initialization of registers and configuration. DCF clients have a default value before any DCF records are written; additionally, they may have special writing constraints, such as Write Once or only allow bits to be written from 0b1 to 0b0 or vice versa (see Section 4.3.1). DCF clients do not need to implement all 32 bits.

Refer to the reference manual for a list of DCF clients and detailed descriptions of the various attributes of the DCF clients.

In the UTEST flash memory the following structure for the DCF records must be present:

1. The first record must be a start record (this record is written to UTEST flash during factory test):

0x00 0–31	0x04 32–63
0x05AA55AF	0x00000000

2. DCF records containing configuration data must immediately follow the start record with no blank records between.
3. The end of the configuration records are indicated by the presence of a stop record (which is simply an unprogrammed record).

There must never be an unprogrammed record in the DCF data structure, as it is interpreted as a stop record and subsequent records are ignored. This allows the user to program the records in several sessions, each time appending new records at the end of the list, as shown in Figure 4.

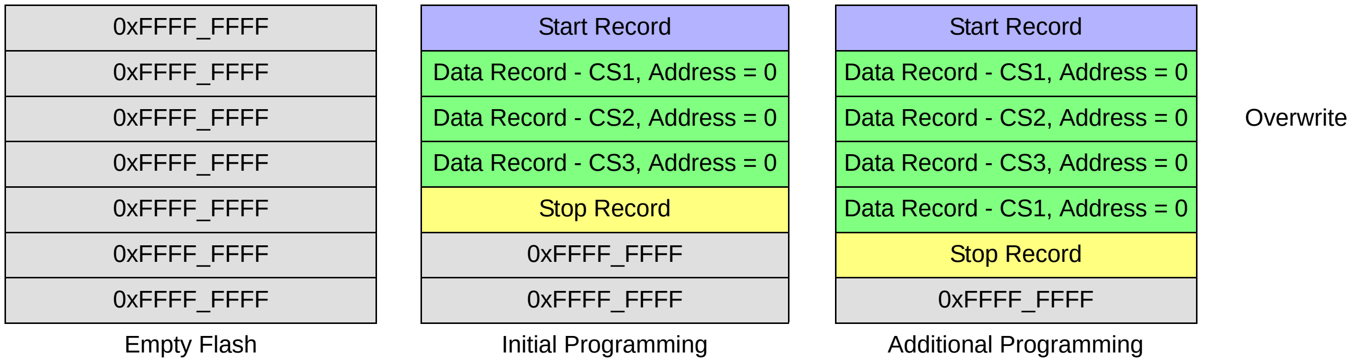

Figure 4. Programming DCF records

It is possible to have more than one DCF record that writes to the same DCF client. In this case, the later record usually overrides a DCF client value set by a previous record. However, not all DCF clients allow overwrites, this depends on the DCF client implementation. Please refer to MPC5746R reference manual DCF Record chapter for details.

4.2 UTEST flash memory

The MPC5746R UTEST memory allocation is shown in Figure 5 below. UTEST memory for the MPC5746R device is one time programmable (OTP) memory. There are two basic types of regions in UTEST flash memory:

- Areas specifically written during factory test that contain a combination of calibration and default data
- Areas allocated for DCF records (as discussed in the previous section) and other user programmed data

Because UTEST is OTP memory, program operations to this area are simply “over programmed”; there is no erase allowed. Note in Figure 4 that a new DCR record is added by simply overwriting (over programming) the existing UTEST DCF record area. Additional details about programming DCF records are covered in Section 4.4.

Address Range	32-bits		Size (bytes)	Notes
0x0040_0000–0x0040_0003	Temp Sensor 1 Calibration Constant K1	Temp Sensor 1 Calibration Constant K2	8	4 x 16-bit constants programmed during factory test
0x0040_0004–0x0040_0007	Temp Sensor 1 Calibration Constant K3	Temp Sensor 1 Calibration Constant K4		
0x0040_0008–0x0040_000B	Reserved		4	
0x0040_000C–0x0040_000F	Test Mode Disable Seal		4	User programmable
0x0040_0010–0x0040_001F	Test Mode Disable Block Select A		16	
0x0040_0020–0x0040_002F	Factory erase diary		16	
0x0040_0030–0x0040_003F	Test Mode Disable Block Select B		16	
0x0040_0040–0x0040_005F	Customer Single Bit Correction Area		32	Programmed during factory test
0x0040_0060–0x0040_007F	Customer Double Bit Correction Area		32	
0x0040_0080–0x0040_009F	Customer EDC after ECC Area		32	
0x0040_00A0–0x0040_00BF	Unique ID		32	
0x0040_00C0–0x0040_00C3	Soft DCF Record Start Address		4	User programmable
0x0040_00C4–0x0040_00C7	Temp Sensor 2 Calibration Constant K1	Temp Sensor 2 Calibration Constant K2	8	4 x 16-bit constants programmed during factory test
0x0040_00C8–0x0040_00CB	Temp Sensor 2 Calibration Constant K3	Temp Sensor 2 Calibration Constant K4		
0x0040_00CC–0x0040_00FF	Reserved		52	
0x0040_0100–0x0040_0103	Test Mode Override Passcode		4	Programmed by customer. Protected from Read Access once life cycle is advanced to OEM_PROD or later.
0x0040_0104–0x0040_011F	Reserved		28	
0x0040_0120–0x0040_013F	JTAG Password		32	
0x0040_0140–0x0040_015F	PASS Password Group 0		32	
0x0040_0160–0x0040_017F	PASS Password Group 1		32	
0x0040_0180–0x0040_019F	PASS Password Group 2		32	
0x0040_01A0–0x0040_01BF	PASS Password Group 3		32	
0x0040_01C0–0x0040_01FF	Reserved		64	
0x0040_0200–0x0040_020F	Life Cycle Slot 0 - PROD		16	Programmed by FSL during test.
0x0040_0210–0x0040_021F	Life Cycle Slot 0 - CUST_DEL		16	
0x0040_0220–0x0040_022F	Life Cycle Slot 0 - OEM_PROD		16	Programmed by customer.
0x0040_0230–0x0040_023F	Life Cycle Slot 0 - IN_FIELD		16	
0x0040_0240–0x0040_024F	Life Cycle Slot 4 - FA		16	
0x0040_0250–0x0040_02FF	Reserved		176	
0x0040_0300–0x0040_0307	DCF Start Record		8	
0x0040_0308–0x0040_0FFF	DCF Records		3320	
0x0040_1000–0x0040_3FFF	Customer OTP Data		12288	
0x0040_4000–0x0040_7FFF	BAF code		16384	

Figure 5. UTEST memory map

4.3 DCF clients

The MPC5746R incorporates various DCF clients that may be configured by DCF records as described in Section 4.1. The clients are generally grouped into four categories:

- Self Test Control Unit (STCU)
- Password and Device Security Module (PASS)
- Tamper Detection Module (TDM)
- Miscellaneous Clients (MISC)

Each category and the corresponding chip select bits are listed in Figure 6. The DCF record data field (bits 0–31 of the 64-bit DCF record) is not shown. See the DCF chapter of the MPC5746R Reference Manual for the complete list of available client addresses for each DCF chip select.

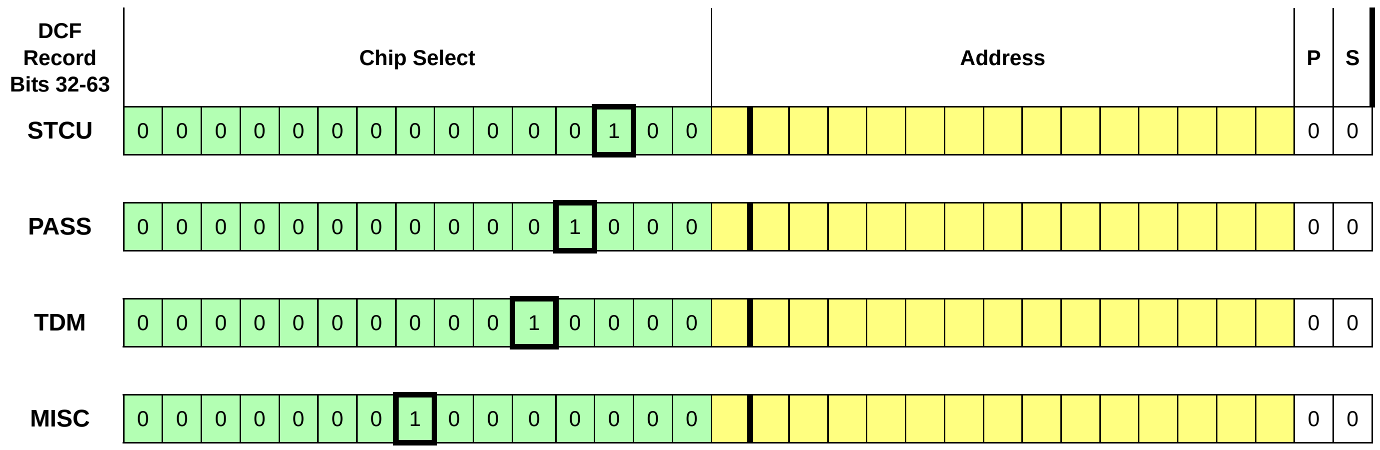

Figure 6. DCF record chip selects

In the following example, this DCF record modifies the DCF_RES_CTRL client, corresponding to the PMC_RES register in the PMC module.[1] The bits in the data field of this register determine whether a specific PMC detected event on the LVD/HVD circuit will cause a “destructive” reset or a “functional” reset (see Section 3.2). A full description of destructive and functional resets is found in the MPC5746R Reference Manual.

A value of 0b1 in the corresponding LVD/HVD reset configuration bit enables a functional reset on the LVD/HVD event detect, while a value of 0b0 (default) causes the detected event to trigger a destructive reset.

In this example, the following bits are set so that the corresponding event generates a functional reset:

- LVD_JTAG
- LVD_FEC
- LVD_MSC5
- LVD_MSC3
- LVD_SAR_ADC

1.Not all DCF clients have a corresponding memory mapped register.

- LVD_SD_ADC
- HVD_SAR_ADC
- HVD_SD_ADC

All remaining bits are left at the default value of 0b0, so the corresponding event generates a destructive reset.

DCF bit	0–15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Data	Reserved	LVD_JTAG	Reserved	LVD_FEC	LVD_MSC5	LVD_MSC3	LVD_SAR_ADC	LVD_SD_ADC	HVD_SAR_ADC	HVD_SD_ADC	Reserved	HVD_FLASH	Reserved	HVD_PMC	HVD_CORE	Reserved	LVD_CORE_COLD
	0000 0000 0000 0000	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0

DCF bit	32–46	47–61	62	63
CS/Addr	Chip Select - **MISC**	Address - **DCF_PMC_RES_CTRL**	**P**	**S**
	0000 0001 0000 000	0 0000 0000 0110 00	0	0

Figure 7. Example DCF record for the DCF_RES_CTL client

As shown in this example, the DCF record has a data field value of 0x0000_BF80, and a chip select/address value of 0x0100_0060.

4.3.1 DCF client special strategies

Though all DCF records have the same format, they can be used in different ways, and some have different methods, or write strategies, for writing the actual DCF record data:

- **None**: No special DCF strategy is used.
- **Parity**: Parity is enabled. (The corresponding DCF record parity bit must be set.)
- **Write Once**: A register using the Write Once strategy can only be written once. The DCF client ignores subsequent writes.
- **Triple Voting**: These clients have three copies of the register. The SSCM will write to all three registers in a single write cycle. The outputs of the three registers are majority-voted to determine the correct data value. Triple voting allows for a bit flip error to occur without changing the DCF client output data.
- **Triple Voting with Second Write**: DCF clients that use the Triple Voted with Second Write strategy have three copies of the register. The SSCM will write to all three registers in a single write cycle and the outputs of the three registers are majority-voted to determine the correct data value. During the second execution of Phase 3 of the reset sequence (see Section 3.2), the SSCM attempts to write the DCF client again. At this time, the DCF client checks to see that the register contains the same data that is being written again.

- **Write 0 only**: A bit in a DCF client can only be written from a logic 1 to a logic 0. An attempt to write a bit with this attribute to a logic 1 is ignored.
- **Write 1 only**: A bit in a DCF client can only be written from a logic 0 to a logic 1. An attempt to write a bit with this attribute to a logic 0 is ignored.
- **Spread Address (SPRD)**: There are three instances addressable separately on three different addresses (that is, spread out). In addition to spreading the addresses to three separate locations, SPRD_ADR also uses data transformation. This means that the data in the three sub-instances is not the same, but is stored in a transformed format. If the normal payload data is X, then the three sub-instances will store the data as a_0 = X, a_1= ~X and a_2 = shift_rotate_right(X). (For example, if X = 0b1011, then a1 = 0b0100, and a2 = 0b1101.)

There are additional parameters that are not the same for each DCF client, such as if the client is readable by software and if there is any specific order in which the client(s) must be written. The DCF Record chapter in the MPC5746R Reference Manual contains a complete list of the characteristics of each client on the device.

4.4 Programming DCF records

During a typical application development, the way DCF records are used may be changed once or more before the final software release.

The developer must maintain a history of what DCF records are already programmed when adding additional DCF records. The flash programming tool used in the development environment must be configured to only program the DCF record memory for new records, as opposed to a standard erase/program sequence. Depending on the flash programmer used, attempting to erase or overprogram OTP flash may result in a program failure, or other unpredictable behavior, and could leave the UTEST flash in an indeterminate state. In some extreme cases, this can even leave the device inaccessible by the debugger.

If you have any questions, consult with your tool vendor before programming DCF records to ensure that the DCF records are programmed as expected and without errors.

5 Boot Header

The boot header allows cores to automatically be started by the Boot Assist Flash (BAF), each core starting execution at user-defined addresses.

The boot header can be located in one of several blocks of the internal flash of the device. The boot header must be the first information in the block (starting at the lowest address of the block). The following table shows the base address of each block of the flash that can hold the boot header.

Table 2. Boot Header Search Options

Search Order	Flash Block	Address
1	16 KB Code flash memory block 1	0x00F9_C000
2	16 KB Code flash memory block 2	0x00FA_0000
3	16 KB Code flash memory block 3	0x00FA_4000
4	16 KB Code flash memory block 4	0x00FA_8000
5	256 KB Code flash memory block 1	0x0100_0000
6	256 KB Code flash memory block 2	0x0104_0000
7	256 KB Code flash memory block 3	0x0108_0000
8	256 KB Code flash memory block 4	0x010C_0000

The first header found that contains the value 0x005A in the first half-word is valid for booting. The whole header structure is shown in the following table.

Table 3. Boot Header Structure

Address Offset	Size (bits)	Contents
0x00	16	Boot header ID (0x005A)
0x02	16	Boot_CPU
0x04	32	Reserved for future use
0x08	32	Configuration bits (reserved for future use)
0x0C	32	Configuration bits (reserved for future use)
0x10	32	Core Reset Vector (Core 0/CPU0)
0x14	32	Core Reset Vector (Core 1/CPU1)
0x18	32	Core Reset Vector (Lock Step core/CPU_LS)

The Boot Header is broken into two parts, the Boot Identifier (ID) and the CPU core enable bits.

Table 4. Boot Header

Offset 0x0	Bit no	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Binary	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0
	Hex	0				0				5				A			
	Bit no	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	Binary													CPU1	CPUC	CPU0	
	Hex	0				0				0							0

Table 5. Boot Header Bit Field Descriptions

Bits	Field	Description
0–15	Boot Header ID[15:4]	Boot header ID (0x005A)
16–27	—	Reserved
28	CPU1	CPU1 is enabled (will start execution at CPU1 reset vector)
29	CPUC	CPU_SC (safety core) is enabled (will start execution at CPU_SC reset vector)
30	CPU0	CPU0 is enabled (will start execution at CPU0 reset vector)
31	—	Reserved

Table 6. Boot Header at 0x0100_0000

Field	Address	Value		Comment
Boot Header	0x0100_0000	0x005A	0x0008	Enable only the boot core (CPU1)
Reserved	0x0100_0004	—		Future use
Configuration Bits	0x0100_0008	0x0000	0x0000	Set to zeros, unused
Configuration Bits	0x0100_000C	0x0000	0x0000	Set to zeros, unused
Core 0 Reset Vector	0x0100_0010	0x01040000		
Core 1 Reset Vector	0x0100_0014	0x01000000		

6 Initialization Code

This section describes and provides examples of the typical initialization code that would be executed by a core from flash memory at startup. The boot header would direct the selected boot core to the start of this software.

6.1 Assembly language initialization sequence

The initialization sequence implemented in the example software is split into two sections: initialization code that is written in assembler, and initialization code that is written in C. The assembler code executes first, and the initialization flow is shown in Figure 8.

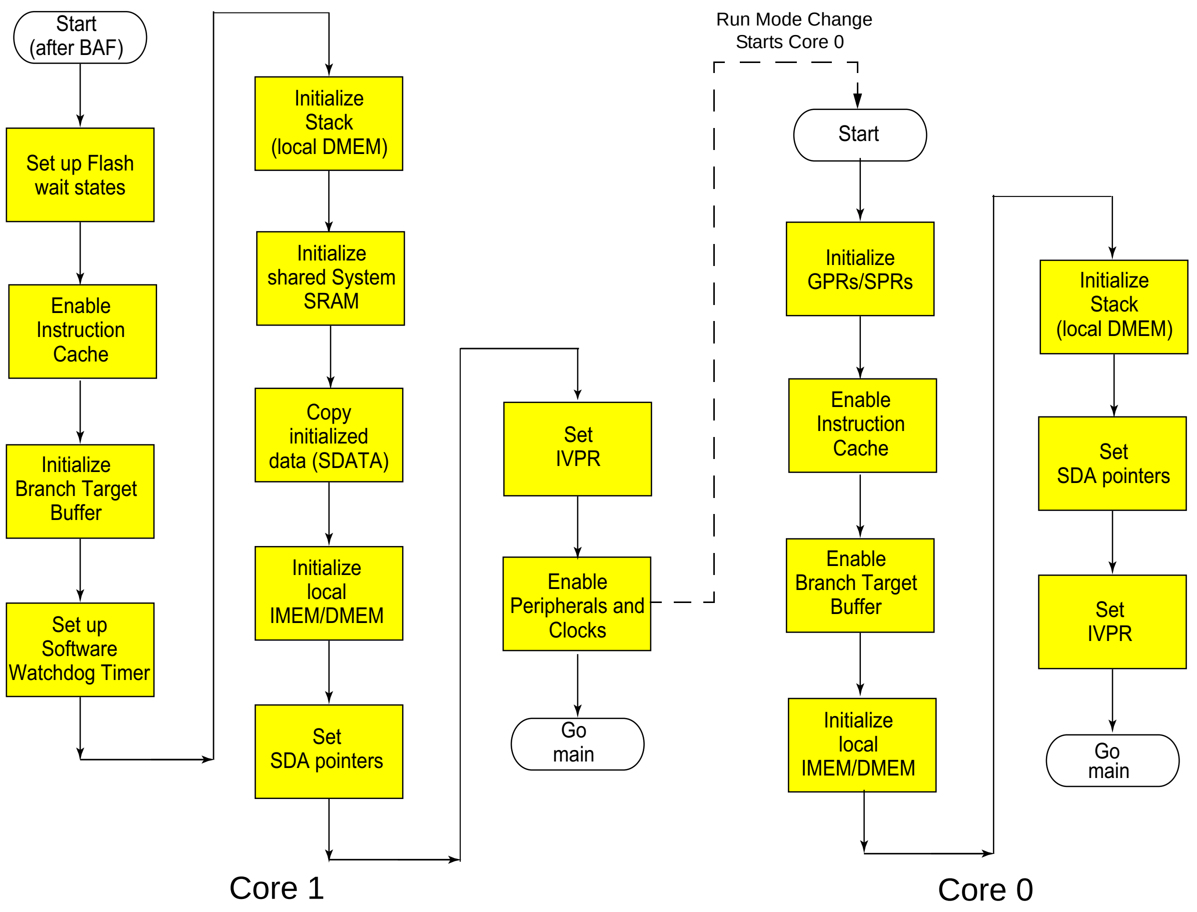
Start (after BAF)
Set up Flash wait states
Enable Instruction Cache
Initialize Branch Target Buffer
Set up Software Watchdog Timer
Initialize Stack (local DMEM)
Initialize shared System SRAM
Copy initialized data (SDATA)
Initialize local IMEM/DMEM
Set SDA pointers
Set IVPR
Enable Peripherals and Clocks
Go main
Core 1
Run Mode Change Starts Core 0
Start
Initialize GPRs/SPRs
Enable Instruction Cache
Enable Branch Target Buffer
Initialize local IMEM/DMEM
Initialize Stack (local DMEM)
Set SDA pointers
Set IVPR
Go main
Core 0

Figure 8. Assembly initialization flow—Core 1 (boot) and Core 0

6.2 Flash wait states and pipeline control

At power on reset, flash read wait states are set to add 6 additional clock cycles for a flash read operation and subsequent pipeline accesses to the flash are disabled. To maximize performance, the wait states should be reduced to match the target operating frequency of the MCU and the pipeline control enabled. At power on reset, the MCU executes at 16 MHz from its IRC. In the example software provided with this applications note, the MCU cores are configured to run at 200 MHz, so the wait states are initially configured for the eventual target operating frequency of 200 MHz. Table 7 shows the wait states and pipeline delay required for each operating frequency range.

Table 7. Flash Read Wait State and Address Pipeline Control guidelines

Frequency (fsys)	RWSC	APC
0 MHz < fsys < 33.3 MHz	0	0
33.3 MHz < fsys < 50.0 MHz	1	1
50.0 MHz < fsys < 66.7 MHz	1	0
66.7 MHz < fsys < 100 MHz	2	1

Table 7. Flash Read Wait State and Address Pipeline Control guidelines (continued)

Frequency (fsys)	RWSC	APC
100MHz < fsys < 133MHz	3	1
133MHz < fsys < 150MHz	4	2
150MHz < fsys < 167MHz	4	1
167MHz < fsys < 200MHz	5	2

The wait states and pipeline controls are configured in the flash memory controller's Platform Flash Configuration Register 1 (PFCR1) via the Address Pipeline Control (APC) and Read Wait State Control (RWSC) bits.

The flash access timings should not be updated while executing software from flash memory. Therefore, the code to write to this register must be copied into a RAM location and executed there. The following example code copies the flash wait state modification code into IMEM for execution. Note that the RAM (IMEM) is written in 32-bit words before being read to avoid an ECC error. This is discussed later in more detail in Section 6.6, "SRAM and ECC."

```
#*****************************************************
#     Setup Flash wait states                         *
#*****************************************************
# branch to the copy routine
   e_bl  _Flash_WS_config

# This is the code copied to I-MEM
reduce_flash_ws:
   #RWSC=5, APC=2 for 200MHz
   #Enable Data Prefetch, Instruction Prefetch but disable Line Read Buffers
   e_lis   r3, 0x0000
   e_or2i  r3, 0x4554

   #Enable Data Prefetch, Instruction Prefetch but disable Line Read Buffers
   e_lis   r4, 0x0000
   e_or2i  r4, 0x0054

   #PFLASH_PFCR1 address
   e_lis   r5, 0xFC03
   e_or2i  r5, 0x0000

   #PFLASH_PFCR2 address
   e_lis   r6, 0xFC03
   e_or2i  r6, 0x0004

   #Configure flash
   e_stw   r3, 0(r5)
   e_stw   r4, 0(r6)

   #RWSC=5, APC=2  for 200MHz
   #Enable Line Read Buffers to flush buffers
   e_lis   r3, 0x0000
   e_or2i  r3, 0x4555

   #Enable Line Read Buffers to flush buffers
   e_lis   r4, 0x0000
   e_or2i  r4, 0x0055

   #Configure flash again to flush&enable buffers
   e_stw   r3, 0(r5)
   e_stw   r4, 0(r6)
   se_blr

# Configuration code
_Flash_WS_config:

# load I-MEM start address
   e_lis   r5, 0x5100
   e_or2i  r5, 0x0000

# Initialize I-MEM
   e_stmw  r0,0(r5)

# backup current link register to r9
   mfspr   r9, 8

# Set link register to I-MEM start address
   mtlr    r5

# Prepare addresses to copy instructions to IMEM
   e_lis   r3, reduce_flash_ws@h
   e_or2i  r3, reduce_flash_ws@l
   e_lis   r4, _Flash_WS_config@h
   e_or2i  r4, _Flash_WS_config@l
   subf    r4, r3, r4
   mtctr   r4

# Copy to IMEM
copy:
   e_lbz   r6, 0(r3)
   e_stb   r6, 0(r5)
   e_addi  r3, r3, 1
   e_addi  r5, r5, 1
   e_bdnz  copy

# Jump to I-MEM start address
   se_blrl

# Restore link register from r9 to LR and return
   mtspr   8, r9
```

6.3 Branch Target Buffer

To resolve branch instructions and improve the accuracy of branch predictions, the e200z4 cores on the MPC5746R implement a dynamic branch prediction mechanism using a branch target buffer (BTB), a fully associative address cache of branch target addresses. Its purpose is to accelerate the execution of software loops with some potential change of flow within the loop body.

By default, this feature is disabled following negation of reset and execution of the BAF. It is controlled by the Branch Unit Control and Status Register (BUCSR). The BTB's contents are flushed and invalidated by writing BUSCR[BBFI] = 1 and it is enabled by writing BUSCR[BPEN] = 1.

```
#**************************************************
#     Enable Branch Target Buffer                 *
#**************************************************
# Flush and enable BTB - Set BBFI bit and BPEN bit
    e_li r3, 0x201
    mtspr 1013, r3
    se_isync
```

6.4 Disabling the SWT (in software)

The MPC5746R has a Software Watchdog Timer (SWT) for each of the cores. If the SWT is not required, it can be disabled. In order to disable the SWT, a sequence of operations must be performed in the proper order. Table 8 lists the addresses of the three SWTs in the MPC5746R devices. In addition, it is possible for tools to disable or service SWT1 through the Debug and Calibration Interface (DCI). By default, out of reset, only SWT1 is enabled.

6.4.1 SWT overview

The MPC5746R incorporates three software timer modules. The following table shows the association of each of the SWT modules to the two cores that are available on the MPC5746R. Note that SWT1 is enabled by default, so it must either be serviced or disabled during initialization. In this example SWT1, is disabled in the crt0_core1_flash.s code.

Table 8. Software Watchdog Timer base addresses

Module	Abbreviation	Base Address	Typical Association
Software Watchdog Timer 0	SWT0	0xFC05_0000	CPU0
Software Watchdog Timer 1	SWT1	0xFC05_4000	CPU1
Software Watchdog Timer 3	SWT3	0xFC05_C000	CPU0 and CPU1

Each of the SWT modules contains several registers. Only two of the registers are required to be written to disable the SWT, the SWT Control Register (SWT_CR) and the SWT Service Register (SWT_SR).

Table 9. SWT registers

Address	Register	Abbreviation	Access
Base Address + 0x0	SWT Control Register	SWT_CR	R/W
Base Address + 0x4	SWT Interrupt Register	SWT_IR	R/W
Base Address + 0x8	SWT Time-out Register	SWT_TO	R/W
Base Address + 0xc	SWT Window Register	SWT_WN	R/W
Base Address + 0x10	SWT Service Register	SWT_SR	W
Base Address + 0x14	SWT Counter Output Register	SWT_CO	R
Base Address + 0x18	SWT Service Key Register	SWT_SK	R/W

6.4.2 SWT disable sequence

Multiple steps are required to actually disable each of the SWT modules. The first step is to clear the soft lock bit in the control register (SWT_CR) using the clear soft lock service code.[1] See the following table for an example of disabling of the SWT. This example uses the SWT1, which is associated with CPU 1.

Table 10. Steps to Disable the SWT

Step	Operation	Description	Pseudo Code
1	Clear the soft lock bit in the Control register by setting the SWT service code	• Write the first value of the soft lock clear to the Watchdog Service Code, WSC = 0xC520	SWT_SR = 0x0000_C520
2		• Write the second value of the soft lock clear to the Watchdog Service Code, WSC = 0xD928	SWT_SR = 0x0000_D928
3	Disable the watchdog[1]	• Allow all masters to access the SWT, MAP=0xFF • Select the oscillator as the clock source, CSL=0b1 • Stop the SWT when in debug mode, FRZ=0b01 • Disable SWT, WEN=0	SWT_CR = 0xFF00_000A

[1] This is one example set of settings. There are other options that can be selected. Set WEN=0b1 to enable the watchdog.

6.4.3 SWT disable code

The following is an assembly code listing of example code for disabling the SWT. By default, only SWT1 is enabled out of reset. In the code that accompanies this Application Note, the SWTs are disabled in the assembly code initialization, however this function may also be performed in C code after control is passed to main( ).

1. The SWT Server Register supports two separate functions, soft lock clearing and servicing the watchdog. Clearing the soft lock is covered in this section. To service the watchdog (when the SWT is enabled), two values must be written to the SWT_SR[WSC], 0xA602 and 0xB480.

```
#***************************************************
#     Disable watchdog timers *
#***************************************************
# Disable SWT0 -
# R4 holds base address of SWT0 (0xFC050000)
    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)

# Disable SWT1
# R4 holds base address of SWT1 (0xFC054000)
    e_lis   r4, 0xFC05
    e_or2i  r4, 0x4000
    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)

# Disable SWT3
# R4 holds base address of SWT3 (0xFC05C000)
    e_lis   r4, 0xFC05
    e_or2i  r4, 0xC000
    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)
```

6.5 Initializing the core registers

Core 0 is coupled in lockstep with a checker core. Consequently, the core registers for both Core 0 and the checker core must be “synchronized” at start up.

At power on reset the majority of core registers have random contents; therefore, the same registers on Core 0 and its lockstep partner generally contain different values. To ensure that identical results are returned when the registers from each core are read, such as during a context save to the stack, the registers of each core must first be initialized. In other words, if a register is read from Core 0 while lock step is enabled and the content of this register is different from the value in the same register of the lock step core, then a lock error will be signaled to the Fault Collection and Control Unit (FCCU).

It is not necessary to initialize the registers of Core 1 although users may choose to do this. Generally, it is good practice to initialize the registers to the value of 0x00000000. A complete list of the core registers that should be initialized in Core 0 when the lock step core is enabled is provided in the crt0_core0_flash.s example file provided.

6.6 SRAM and ECC

The MPC5746R has 256 KB of general-purpose SRAM as well as local data and instruction SRAM within each core. Refer to the MPC5746R Microcontroller Reference Manual for comprehensive details and the MPC5746R SRAM map. All of the SRAM memories have 8-bit end-to-end error checking and correction (e2eECC) with single-bit correction and 2-bit error detection for every 32-bit word.

SRAM must be initialized after POR by executing 64-bit writes to all memory space. This 64-bit write causes the ECC syndrome bits to be calculated. Note that the SRAM does not have to be initialized after all resets, only after POR resets. Refer to the MPC5746R Reference Manual to determine which resets constitute a POR. Attempting to read any uninitialized SRAM normally generates a system exception.

To allow fast initialization, the store multiple word (stmw) instruction is used. This causes up to 32 GPRs to be stored to memory starting at a given base address utilizing 64-writes. Initialization code is shown below. The bounds of RAM to be initialized are provided by the compiler linker file. In this example, Core 1 is used to initialize the whole SRAM array.

Optionally, if multiple cores are started at POR and each core has a dedicated RAM space defined, then each core can initialize only the SRAM it will use. This parallel SRAM initialization can reduce the time required for all cores to finish initialization and begin execution of the application.

```
#***************** Initialize SRAM ECC ******************/
# 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, 0x8 # Divide SRAM size by 256
   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
```

Local SRAM should also be initialized using the same method. It is recommended that each core initialize its own local SRAM. This is shown in the included example software.

6.7 Cache

Each of the e200z425 cores provide 8 KB of instruction cache. The cache can improve system performance by providing fast core access to instructions recently fetched from flash.

There are several stages to enabling the cache. Not only does the cache itself have to be invalidated then enabled, but memory regions upon which it can operate must be configured in the System Memory Protection Unit (SMPU). See the MPC5746R Reference Manual for details of the SMPU module.

The control bits to configure, invalidate, and enable the instruction cache are contained within the core L1 Cache Control and Status Registers 0 (SPR1010) and 1 (SPR1011). First, the cache is invalidated, and once the invalidation completes the cache is enabled.

```
# invalidate and enable the instruction cache
__icache_cfg:
   e_li r5, 0x2
   mtspr 1011,r5
   e_li r7, 0x4
   e_li r8, 0x2
   e_lwi r11, 0xFFFFFFFB
__icache_inv:
   mfspr r9, 1011
   and. r10, r7, r9
   e_beq __icache_no_abort
   and. r10, r11, r9
   mtspr 1011, r10
   e_b __icache_cfg
__icache_no_abort:
   and. r10, r8, r9
   e_bne __icache_inv
   mfspr r5, 1011
   e_ori r5, r5, 0x0001
   se_isync
   msync
   mtspr 1011, r5
```

7 C Language Initialization Sequence

The C initialization sequence is shown in Figure 9. This could be performed in assembly if desired, but is shown in C in this example for easier readability. This sequence is performed by the `Clock_and_Mode_Init()` function in the `mcu_init.c` source code file.

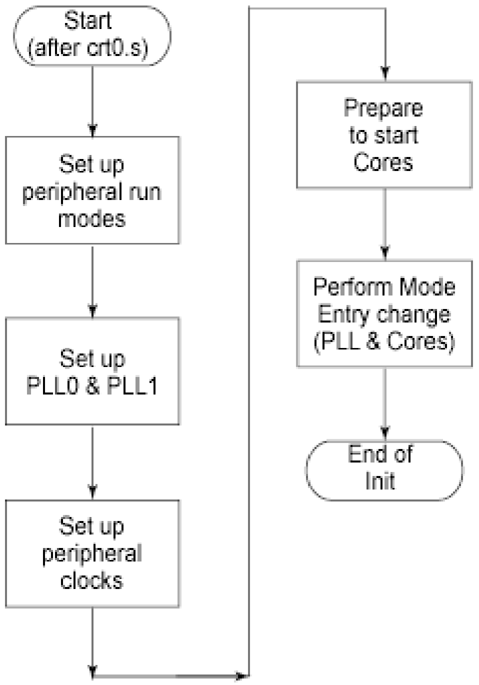

Figure 9. C Initialization sequence

7.1 Mode Entry core control summary

The MC_ME is used to change the operating modes of the device. Changes to the clocks, enabled peripherals, and cores can be changed with two writes to a register in the MC_ME module. The mode change requires two writes with a key and an inverted key to prevent accidental mode changes.

Table 11. MPC5746R cores

Core name	Core reference	Processor Version Register (PVR) value
Computational Core	Core 0	0x815E_0000
Checker Core (Lock Step Core)	Core 0 Checker (CORE_LS)	
Computational Core	Core 1	

Either of the cores can be enabled independently for any of the operating modes. The Mode Entry Control Register is used to configure in which modes each of the cores is enabled. The possible modes are:

- TEST
- SAFE
- DRUN

- RUN0
- RUN1
- RUN2
- RUN3
- HALT0
- STOP0

NOTE

The MC_ME module uses the terms Core 1 and Core 2, but the actual assignments of these terms to each of the cores in regards to the MC_ME operation is different than the naming convention in the rest of the device reference manual.

Table 12. Core Control Registers (ME_CCTLn)

Core	Device Reference	ME Core Definition[1]	ME_CCTRLn Address	Register	Full Address
Computational Core	Core 1	S_CORE0	Base address + 0x1C4	ME_CCTL0	0XFFFB_81C4
Computational Core	Core 0	S_CORE1	Base address + 0x1C6	ME_CCTL1	0XFFFB_81C6
Checker Core	Core 0 checker	S_CORE2	Base address + 0x1C8	ME_CCTL2	0XFFFB_81C8

[1] This is the name of the status bits in the MC_ME Core Status Register. The 0–2 numerals appended to *S_CORE* also correspond to the number of the ME_CCTL and ME_CADDR0 registers.

The start address for the each of the cores is stored in the corresponding Core Address Register. This can be done automatically with the Boot Header via user software or, when running with a debugger, by the debugger. When the BAF code is available, these can also be automatically loaded during the boot process.

Table 13. Core Control Registers (ME_CADDRn)

Core	Core Reference	ME_CADDRn Address	Register[1]	Full Address
Computational Core	Core 1	Base address + 0x1E0	ME_CADDR0	0XFFFB_81E0
Computational Core	Core 0	Base address + 0x1E4	ME_CADDR1	0XFFFB_81E4
Checker Core	Core 0 checker	Base address + 0x1E8	ME_CADDR2	0XFFFB_81E8

[1] Bit 31 (the least significant bit) must be set initially to reset the core and allow the core out of reset for cores 0 and 1.

To enable and start additional cores, a mode transition must be performed. The mode change requires an interlock write to the Mode Entry Mode Control Register (ME_MCTL). The interlock key values are the 16-bit values, 0x5AF0 followed by 0xA50F.

Table 14. Mode change example

Register	Address	Write Value	Description
ME_MCTL	Base address + 0x4	0x40005AF0	Enable RUN0 mode for all enabled in RUN0 mode.
		0x4000A50F	

The Mode Entry Core Status Register (ME_CS address 0xFFFB_801C0) can be read to verify which cores have been enabled.

7.2 Clock initialization

At POR, the MCU is clocked from the on-chip 16 MHz IRC oscillator. This section explains how to configure the modules to use the clock from the higher speed PLLs. It also covers the setup of the clock trees to distribute and divide the clock sources to the buses and peripherals on the MCU.

7.2.1 Clock tree

Figure 10 shows the clock tree configuration for the MPC5746R. In the example software used here, PLL0 is configured to run at 160 MHz and PLL1 at 200 MHz, both from the external oscillator.

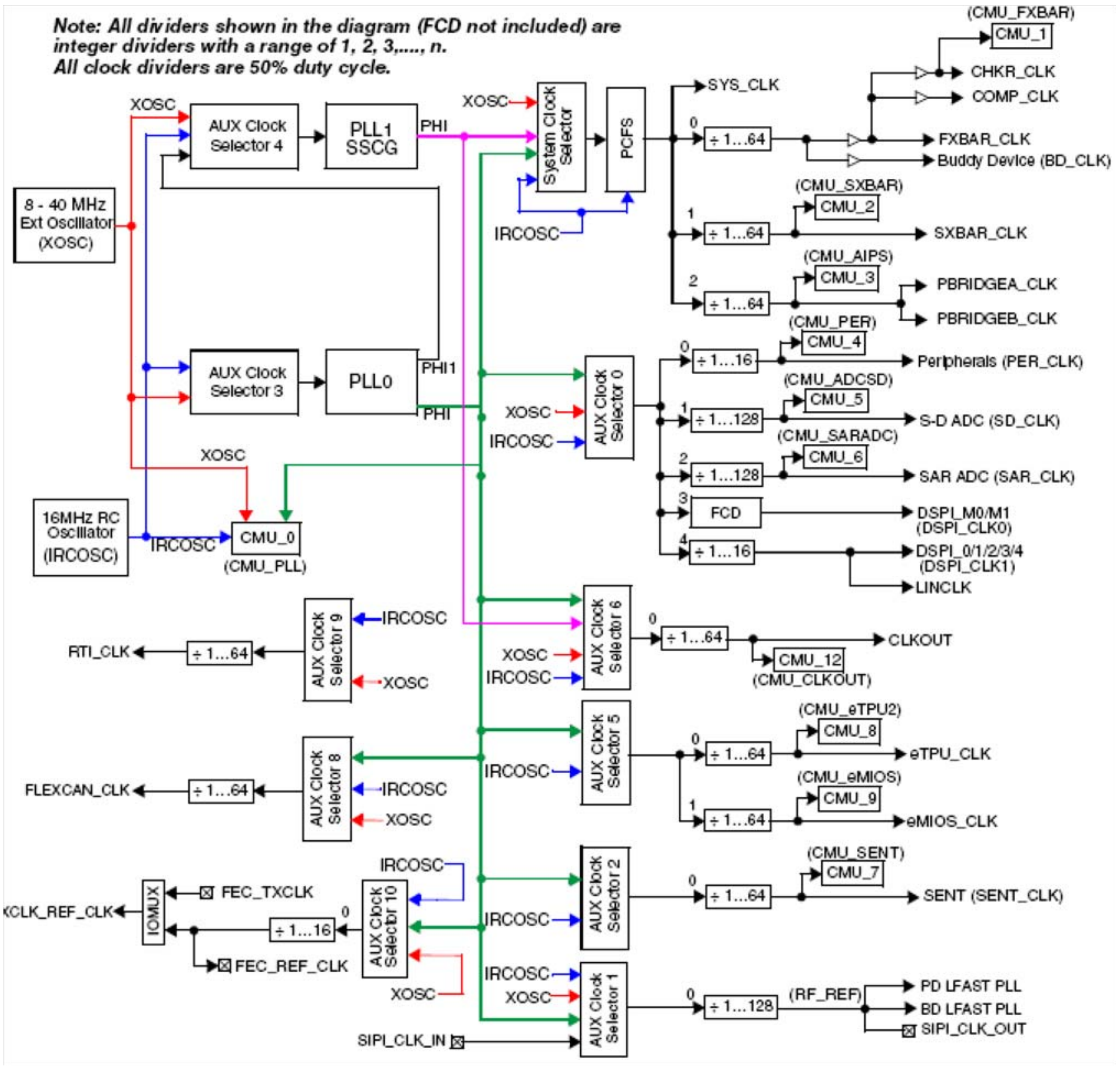

Figure 10. MPC5746R clock tree

A summary of the clock tree settings that are configured in the example software is shown in Table 15. Most module clocks are disabled in this configuration; however, an example configuration for SD_ADC and the PER_CLK are included and are typical of the configuration for other modules.

Table 15. Example clock settings

Clocks	Aux selector and divider	Source clock	Div / multiply factor	Frequency (MHz)
Slow XBAR	System Clock	PLL1	2	100
Fast XBAR	System Clock	PLL1	1	200

Table 15. Example clock settings (continued)

Clocks	Aux selector and divider	Source clock	Div / multiply factor	Frequency (MHz)
PBRIDGE_A PBRIDGE_B	System Clock	PLL1	4	40
PER_CLK	AUX0–0	PLL0	4	50
SD_ADC	AUX0–1	PLL0	10	16
SAR_ADC	AUX0–2	Disabled	—	—
DSPI_M0/M1	AUX0–3	Disabled	—	—
DSPI_0/1/2/3/4	AUX0–4	Disabled	—	—
SENT	AUX2–0	Disabled	—	—
CLKOUT	AUX6–0	PLL1	10	20
CAN_CLK	AUX8–0	Disabled	—	—
RTI_CLK	AUX9–0	Disabled	—	—
FEC_REF_CLK	AUX10–0	Disabled	—	—

7.3 Mode Entry peripheral clock gating summary

During all chip modes, each peripheral can be associated with a particular clock gating policy determined by two groups of peripheral configuration registers, ME_RUN_PC0..7 and ME_LP_PC0..7.

The run mode configuration registers are chosen during run modes RESET, TEST, SAFE, DRUN, and RUN3..0. The low-power peripheral configuration registers ME_LP_PC0..7 are chosen only during the low-power modes HALT0 and STOP0. The ME_LP_PC0..7 registers are not configured in this example.

All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the LP_CFG bit field of the ME_PCTLn registers.

NOTE

Any modifications to the ME_RUN_PC0..7, ME_LP_PC0..7, and ME_PCTLn registers do not affect the clock gating behavior until a new mode transition request is generated.

The following registers and bitfields are used to control peripheral clock gating for run modes (DRUN, TEST, SAFE, RUN0..3)

Table 16. Detail of ME_RUN_PCn Registers

Register	Bitfields	Action
ME_RUN_PC0	RUN3..0, DRUN, SAFE, TEST, RESET	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active

Table 16. Detail of ME_RUN_PCn Registers

Register	Bitfields	Action
ME_RUN_PC1	RUN3..0, DRUN, SAFE, TEST, RESET	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active
ME_RUN_PC2	RUN3..0, DRUN, SAFE, TEST, RESET	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active
. . .	. . .	. . .
ME_RUN_PC7	RUN3..0, DRUN, SAFE, TEST, RESET	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active

The following registers and bit fields are used to control peripheral clock gating for low-power modes (HALT0, STOP0):

Table 17. Detail of ME_LP_PCn Registers

Register	Bitfields	Action
ME_RUN_PC0	STOP0, HALT0	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active
ME_RUN_PC1	STOP0, HALT0	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active
ME_RUN_PC2	STOP0, HALT0	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active
. . .	. . .	. . .
ME_RUN_PC7	STOP0, HALT0	Set each bit field to 0 or 1. 0 = Peripheral is frozen with clock gated 1 = Peripheral is active

Once the user configures ME_RUN_PC0..7 and ME_LP_PC0..7, it may be desirable to configure Peripheral Control Registers (ME_PCTLn). The MPC57xx eTPU-based devices contain a unique instance of ME_PCTLn for every peripheral in the device implementation. Each ME_PCTLn register contains the following bit fields:

- DBG_F: controls peripheral state after entering debug mode (either frozen or operation specified by RUN_CFG/LP_CFG bits and chip mode)
- LP_CFG: controls which ME_LP_PC0..7 definition to use for the peripheral

- RUN_CFG: controls which ME_RUN_PC0..7 definition to use for the peripheral. By using the above registers and fields, the user can create a variety of peripheral clock gating configurations based on the operating modes of the device.

The ME_PCTL registers are not programmed in this example. See the MC_ME chapter of the reference manual for additional details.

7.4 Mode Entry example

The Mode Entry example illustrates how a programmer configures peripheral clock gating, low-power modes of operation, run modes of operation, clock dividers, PLLs, and CPU core enable / disable and start addresses. The Mode Entry example finishes with programming of the run mode (DRUN in this case) and the normal key / inverted key process for initiating a mode transition.

7.4.1 Mode Entry example design

The Mode Entry example includes the following major steps:

1. Clear RGM Functional Event Status and Destructive Event Status registers. Enable all ME modes (RESET_DEST, STOP0, HALT0, RUN3..0, DRUN, SAFE, TEST, and RESET_FUNC). The user can disable any modes not required by writing a 0b0 to the appropriate register bit.
2. Enable external crystal oscillator as clock by programming the DRUN_MC Register.
3. Configure PLL clock sources and setup the PLL dividers.
4. Set up the system clock.
5. Program the peripheral clock dividers.
6. Program the Run Peripheral Configuration Registers (ME_RUN_PC0..7) to define peripheral behavior during run modes (RESET, TEST, SAFE, DRUN, RUN0..3) and Low-Power Peripheral Configuration Registers (ME_LP_PC0..7) for operation during STOP0 and HALT0 modes.
7. Program per peripheral control registers (ME_PCTLn) to define peripheral behavior for low power modes (defined by ME_LP_PC0..7) and run modes (defined by ME_RUN_PC0..7).
8. Enable CPU cores by programming the ME_CCTLn registers and set the start (boot) address (in ME_CADDRn registers).
9. Trigger a mode transition to DRUN mode by programming the mode transition keys, waiting for the setting of the mode entry change complete bit (MC_ME_GS[S_MTRANS]) and confirming the DRUN mode has been entered (check the MC_ME_GS[S_CURRENT_MODE]).

The following table shows each step, the associated description, and pseudo code.

Table 18. Design of Mode Entry Example

Step	Operation	Description	Psuedo Code
1	Clear RGM events, enable ME modes	Clear RGM Functional Event and Destructive Event fields by writing 1 to clear.	Set MC_RGM.DES and MC_RGM.FES to 0xFFFF.
		Enable RESET_DEST, STOP0, HALT0, RUN3..0, DRUN, SAFE, TEST, and RESET_FUNC modes in the ME_ME	SET MC_ME_ME to 0x000005E2
2	Enable XOSC clock	Set the XOSC enable bit in the DRUN Mode Configuration register (ME_DRUN_MC)	Program ME_DRUN_MC = 0x001F0020
3	Configure PLL0 and PLL1 clock source.	• With PLL0 and PLL1 disabled, select the clock source for each.	Set CGM_AC3_SC[SELCTL] for XTAL sourcing PLL0 Set CGM_AC4[SELCTL] for XTAL sourcing to PLL1
4	Set PLL dividers	• Program desired PLL0 dividers • Program desired PLL1 dividers	Set PLL0DIV[PREDIV, RFDPHI1, MFD, RFDPHI] for 160MHz from 20MHz XTAL Set PLL1DIV[PREDIV, RFDPHI1, MFD, RFDPHI] for 200MHz from 20MHz XTAL
5	Configure System clock dividers (Fast Crossbar, Slow crossbar, PBRIDGE)	• Program fast XBAR—divide by 1 and enable • Program slow XBAR—divide by 2 and enable • Program PBRIDGE—divide by 4 and enable	• CGM.SC_DC[0] = 0x80000000 • CGM.SC_DC[1] = 0x80010000 • CGM.SC_DC[2] = 0x80030000
6	Enable and configure Aux clocks supplied to peripherals	Program Auxiliary Clock (AC) n and Divider Configuration (DC) m Registers. These assign divider values to peripheral clocks.[1]	See sample code.
7	Configure peripheral run modes.	Enable all run modes for all Run Peripheral Configuration Registers (ME_RUN_PC0..7)	Set ME_RUN_PC0..7 to 0xFE
8	Configure Low Power and Run Modes in Peripheral Control Registers.	Program Peripheral Control Registers (ME_PCTLn) to define per peripheral operation in RUN and Low Power configs. ME_PCTLn[LP_CFG] and ME_PCTLn[RUN_CFG] bit fields select which of the LP_PC0..7 and RUN_PC0..7 definitions to use for the given peripheral, n.	No configuration required for this example, reset values for ME_PCTLn[LP_CFG] = ME_PCTLn[RUN_CFG] = 0b000, which selects ME_LP_PC0 and ME_RUN_PC0 respectively.
9	enable CPU cores by programming the ME_CCTLn registers and set the start (boot) address and enable bit (in ME_CADDRn registers	Enable cores 0 and 2[2] (lockstep core). Set start address of cores 0 and 2 to 0x01040000. Start address of core 1 is in defined in RCHW (See Section 5, “Boot Header,” on page 13).	MC_ME.CADDR[1] = 0x01040001 MC_ME_CADDR[2] = 0x01040001 MC_ME.CCTL[1] = 0x00FE MC_ME.CCTL[2] = 0x00FE

Table 18. Design of Mode Entry Example (continued)

Step	Operation	Description	Psuedo Code

10	Perform mode entry change	Program mode and key, mode and inverted key, wait for mode entry to complete, check for DRUN mode entered.	MC_ME.MCTL = 0x30005AF0
			MC_ME.MCTL = 0x3000A50F
			wait for (MC_ME.GS.S_MTRANS == 0)
			check for:
			MC_ME.GS.S_CURRENT_MODE = 0x3

11	Setup clock selectors to allow CLKOUT to be viewed on external pin	• select PLL0 for CLKOUT • enable AC6 divider • divide by 10	• CGM.AC6_SC[SELCTL]=2 • CGM.AC6_DC0[DE]=1 • CGM.AC6_DC0[DIV]=9

[1] This step configures clock divider settings for PER_CLK, SD_CLK, SAR_CLK, DSPI_CLK0, DSPI_CLK1, LIN_CKLK, SENT_CLK, etc.

[2] Core 0 = index “1,” LS (checker) core = index “2,” Core 1 = index “0”

8 Revision history

Table 19. Document revision history

Revision number	Revision date	Description of changes
0	12/2016	Initial public release.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN4670
Rev. 0
12/2016