Freescale Semiconductor
Application Note

Document Number: AN2831
Rev. 0, 01/2008

MPC5500 Boot Assist Module

by: Alistair Robertson
Powertrain Systems Engineer

1 Overview

Freescale Semiconductor's MPC5500 family of highly integrated microcontrollers offer a greater level of functionality than earlier generations of microcontrollers, such as the MPC500 family. A host of new features including the direct memory access (DMA) and the memory management unit (MMU) has resulted in an increase in system initialization requirements. This has subsequently led to the introduction of the boot assist module (BAM).

The BAM is a nonvolatile memory based software program. The BAM's primary function is to perform essential system initialization and to locate and execute the application code. The BAM also supports serial download of user code. The execution of the BAM is affected by the censorship status. BAM is executed when reset is negated.

The BAM supports four different modes of locating and executing user code:

Contents

- Boot from internal flash
- Boot from external memory
- Boot from external memory with arbitration for multi-master systems
- Serially download user code via the enhanced serial communications interface (eSCI) or FlexCAN.

When booting from the internal flash or the external memory, the BAM reads a reset configuration half word (RCHW) and configures the core watchdog and external bus interface (EBI) accordingly. The BAM also supports password protection when serially downloading boot-code. This serial download function allows the BAM to be used for censorship recovery, flash programming and even high level debug.

This application note details how to use the different boot modes of MPC5500 devices, providing example code where necessary. This acts as a supplement to the information provided in the reference manuals for the MPC5500 devices available at www.freescale.com.

This document describes the BAM implemented in the following devices:

Table 1. Devices supported by AN2831

Device	Core	VLE Supported	SRAM Size	Bus Width
MPC5533	e200z3	Yes	48K	No external bus interface (EBI)
MPC5534	e200z3	Yes	64K	16-bit EBI*
MPC5553	e200z6	No	64K	16 or 32-bit selectable*%
MPC5554	e200z6	No	64K	16 or 32-bit selectable*
MPC5561	e200z6	Yes	192K	16-bit*
MPC5565	e200z6	Yes	80K	16-bit*
MPC5566	e200z6	Yes	128K	16 or 32-bit selectable
MPC5567	e200z6	Yes	80K	16 or 32-bit selectable*

*= 16-bit only in 324 PBGA package

% = No EBI on 208 packages

Figure 1 shows the simplified program flow of the BAM.

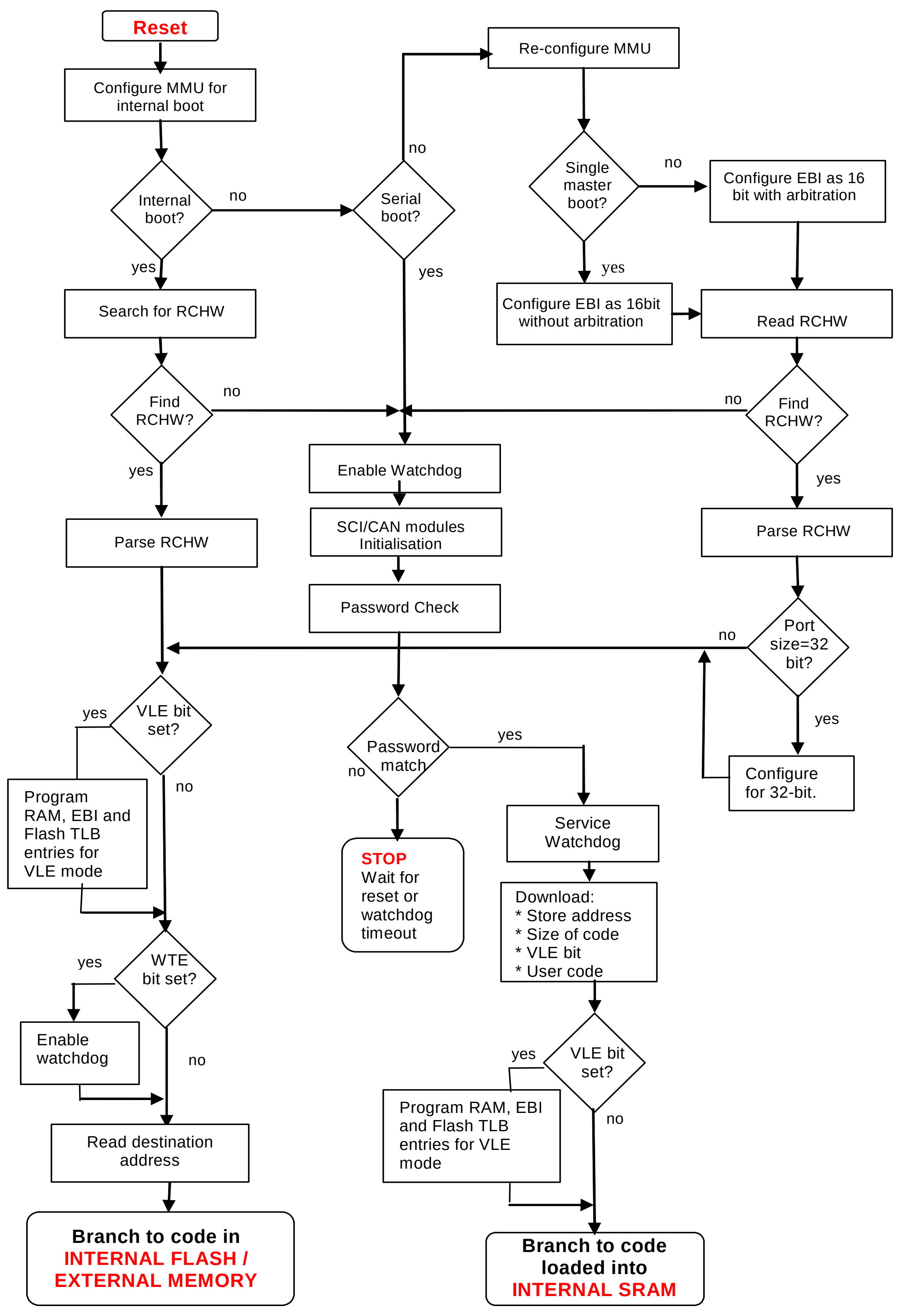
Reset
Configure MMU for internal boot
Internal boot?
no
yes
Search for RCHW
Find RCHW?
no
yes
Parse RCHW
VLE bit set?
yes
no
Program RAM, EBI and Flash TLB entries for VLE mode
WTE bit set?
yes
no
Enable watchdog
Read destination address
Branch to code in INTERNAL FLASH / EXTERNAL MEMORY
Serial boot?
no
yes
Re-configure MMU
Single master boot?
no
yes
Configure EBI as 16 bit with arbitration
Configure EBI as 16bit without arbitration
Read RCHW
Find RCHW?
no
yes
Parse RCHW
Port size=32 bit?
no
yes
Configure for 32-bit.
Enable Watchdog
SCI/CAN modules Initialisation
Password Check
Password match
yes
no
STOP Wait for reset or watchdog timeout
Service Watchdog
Download: * Store address * Size of code * VLE bit * User code
VLE bit set?
yes
no
Program RAM, EBI and Flash TLB entries for VLE mode
Branch to code loaded into INTERNAL SRAM

Figure 1. MPC5500 Boot Assist Module Flow Diagram

NOTE

Figure 1 shows the generic BAM program flow diagram for the MPC5500 family. Not all MPC5500 derivatives support the 16-/32-bit EBI features detailed.

This is a simplified program flow and is not representative of the actual code implementation. Liberties have been made to ease understanding of the flow.

2 Boot Modes

There are four different boot modes supported by the BAM. These, together with the required entry conditions are listed in Table 2. Refer to the MPC5554 Reference Manual Rev 3.1, Section 16.3.2, for a complete description.

Table 2. BAM Boot Modes

BOOT CFG [0:1]	Censorship Control 0x00FF_ FDE0	Serial Boot Control 0x00FF_ FDE2	Boot Mode Name	Internal Flash State	Nexus State	Serial Password
00	!0x55AA	Don't care	Internal – censored	Enabled	Disabled	Flash
	0x55AA		Internal – public	Enabled	Enabled	Public
01	Don't care	0x55AA	Serial – flash password	Enabled	Disabled	Flash
		!0x55AA	Serial – public password	Disabled	Enabled	Public
10	!0x55AA	Don't Care	External – no arbitration – censored	Disabled	Enabled	Public
	0x55AA		External – no arbitration – public	Enabled	Enabled	Public
11	!0x55AA	Don't care	External – external arbitration – censored	Disabled	Enabled	Public
	0x55AA		External – external arbitration – public	Enabled	Enabled	Public

NOTE

'!' = 'NOT', meaning any value other than the value specified. Values 0x0000 and 0xFFFF must not be used.

Internal boot mode is the default selection if the BOOTCFG pins are not sampled at reset (RSTCFG not asserted as RSTOUT negates). Serial download mode is the default selection if internal or external boot mode is selected but no valid RCHW is found.

3 Internal and External Boot Modes

3.1 RCHW

The reset configuration half word (RCHW) controls the program flow of the BAM. Boot ID = 0x5A signifies that the RCHW is valid. WTE = 1 enables / WTE = 0 disables the core watchdog timer. PSO selects an external bus port size of 16-bit if PSO = 1 and 32-bit if PSO = 0. VLE = 1 enables variable length encoding. Unused bits must be written to zero to ensure compatibility with future MPC5500 devices.

0 … 4	5	6	7	8 … 15
	WTE	PS0	VLE	BOOTID (0b01011010)

Figure 2. Reset Configuration Half Word

The RCHW can be located at various locations.

Table 3. RCHW Locations

Boot Mode	RCHW locations	Comment
Internal flash	0x00_0000 (LAS0) 0x00_4000 (LAS1) 0x01_0000 (LAS2) 0x01_c000 (LAS3) 0x02_0000 (LAS4) 0x03_0000 (LAS5)	The lowest address of any of the six Low Address Spaces (LAS) in internal flash memory. When booting from internal memory, the BAM can ignore ECC errors in the RCHW thus avoiding the possibility of being stuck in an endless loop. As a further safety measure, a valid RCHW could be placed at more than one LAS location.
External memory (with/without arbitration)	0x2000_0000	The lowest address of an external memory device, enabled by chip select CS0 using either 16 or 32-bit wide external data bus.

The BAM has no means of determining in advance whether the external memory port size is 16 or 32-bits wide until the RCHW has been read. Therefore the initial read of the external memory is configured as a 16-bit read. The BAM also has no knowledge of the partitioning of external memory, therefore only the first address of the external memory is valid.

For internal flash, after reading the 16-bit RCHW, the next 32-bit aligned location must contain the application code start vector. The same rules apply to external memory as shown in the tables below.

Table 4. Internal Flash

Logical Address	Code	Description
0x0001_C000	0x045A_XXXX	RCHW located at flash LAS L3, watchdog enabled.
0x0001_C004	0x0008_0000	Application code start resides in flash has H0 address 0x0008_0000

Table 5. 16 -Bit External Memory

Logical Address	Code	Description
0x2000_0000	0x025A	RCHW located at start of external memory, Watchdog disabled, 16-bit port size.
0x2000_0002	0xXXXX	Don't care
0x2000_0004	0x0008	Application code 'START' resides at Flash HAS H0 address 0x0008_0000
0x2000_0006	0x0000	

The address at the start of the application code is read by the BAM. This is data and is not a branch instruction therefore program breakpoints set at this address do not work. Data/read breakpoints must be used.

3.2 Generating Boot Code

There are several programming techniques for developing application code with the RCHW and the address or START positioned at a valid location.

NOTE

The following code examples were developed using Windriver's Diab C\C++ compiler.

3.2.1 Using #PRAGMA Directives to Create Absolute Sections

The simplest method is to create absolute sections using the # pragma section-directive. The advantage of creating an absolute section is that it requires no modification to the linker file. The compiler creates an .abs.xxxxxxxx section at compile time, that the linker automatically locates at address defined by xxxxxxxx.

```
#pragma section SCONST address=0x0001c000

const short int ResetConfigWord = 0x005A;

extern void _start();

#pragma section SDATA address=0x0001c004

int start_address = (int)_start;

#pragma section CODE

//application code here…
```

- SCONST & SDATA are predefined section class names. Default settings are far-absolute address-mode and read-only access mode.
- _start defined in init code such as crt0.s file or equivalent

The user must take care to ensure that the memory areas defined in the linker file do not overlap the 2 x 32-bit words used for the RCHW and the _start address. For the example code given, the linker file must not use address range from 0x1C000 to 0x1C008.

3.2.2 Define RCHW within Linker File

In this approach, memory and section commands within the linker command file can specify a memory area to be used for the RCHW. The advantage of this approach is that RCHW can be easily moved without having to modify application code. The following extract from a linker command file gives an example of how the memory and section commands can be updated to include the RCHW.

```
MEMORY
{
// 2M Internal FLASH
RCHW   :      org = 0x00000000,len = 0x00000008
int_flash:    org = 0x00000008,len = 0x001FFFF7
// 2M External Memory
ext_mem:org = 0x20000000,len = 0x00080000
// 64K Internal RAM
int_sram:     org = 0x40000000,len = 0x00010000
}

SECTIONS
{
// FLASH data
.rchw         :      { *(.rchw) } > RCHW
.init         :      {} > int_flash
.text         :      {} > int_flash
.flash_data:  {} > int_flash

// RAM data
.data         :      {} > int_sram
.sdata        :      {} > int_sram
.sbss         :      {} > int_sram
.sdata2       :      {} > int_sram
.sbss2        :      {} > int_sram
.bss          :      {} > int_sram
}
```

In the initialization assembly code (crt0.s or equivalent), the following section must be added:

```
.section               .rchw// As defined in SECTIONS command in linker file.
.LONG        0x005A0000 // Watchdog disabled, 32-bit port size.
.LONG        _start // Start vector, defined in user initialization code.
```

4 Serial Boot Mode

When operating in serial boot mode the BAM can download a program into internal RAM using either eSCI or FlexCAN. The protocol for serial download involves sending:

- A 64-bit password
- A 32-bit destination address for application code
- A 32-bit word that consists of:
  — The 31-bit size of the application code in bytes
  — The VLE bit
- The application code.

In the case of eSCI transmission, each byte received by the MPC5500 is echoed. In the case of FlexCAN, each data packet received is echoed. Therefore, even if not monitoring the echoed bytes or packets, a delay must be inserted between transmissions of each byte/packet to allow enough time for the previous echo to be transmitted. Refer to device reference manual for specific FlexCAN & eSCI settings and information on download protocol.

4.1 Generating Serial Download Code

When loading code to internal RAM (using the BAM serial download or otherwise) the user must be aware that error correction coding (ECC) is implemented for all SRAM (SRAM). It is essential that the ECC parity bits are initialized after power on. A 64-bit cache inhibited write to each location in SRAM must be used to initialize the SRAM array. Code downloaded to SRAM by the serial download mode of the BAM is loaded in 64-bit writes, initializing the SRAM as the code is downloaded. However care must be taken to ensure that SRAM areas allocated to variables, heap and stack are also initialized.

There are several ways to handle this:

- Assign all stack and variables to non-SRAM address space.
- Develop code in assembler, which does not rely on a stack or any variables.
- Initialize additional SRAM array space to accommodate variables/stack.

4.1.1 Assign Variables and Stack to Non-ECC RAM

There are several smaller RAM arrays within the peripheral modules, such as the eTPU, the FlexCAN and the eDMA, which do not have ECC implemented.

Table 6. Available RAM Arrays

MPC5554 RAM Type	Size	MPC5554 Location
eTPU parameter RAM	3K	0xC3FC_8000 to 0xC3FC_BFFF
FlexCAN_A message buffers 0 – 63[1]	1K	0xFFFC_0080 to 0xFFFC_047F
FlexCAN_B message buffers 0 – 63	1K	0xFFFC_4080 to 0xFFFC_447F
FlexCAN_C message buffers 0 – 63	1K	0xFFFC_8080 to 0xFFFC_847F
eDMA transfer control descriptors[2]	2K	0xFFF4_5000 to 0xFFF4_5800

[1] Message buffer 0 in FlexCAN_A is used for CAN serial download mode by the BAM and the remaining message buffers are used as scratch pad RAM. (The serial download section of the BAM was written in C and requires RAM allocation for variables). When control is passed to the application code, the BAM no longer uses FlexCAN_A message buffers and then they are available for use. If the downloaded user code makes use of the FlexCAN module, then the CANx_MCR[MAXMB[0..5] can be set to minimize the number of buffers used, and therefore maximize the free available memory.

[2] Although the eDMA transfer control descriptor (TCD) memory space is available for use, extreme care must be taken when using this to ensure that the TCD[START] bit is not written to, potentially triggering an unwanted eDMA transfer. It is not recommended to use this memory area unless absolutely necessary.

These RAM memory locations can not be used for the executable code. Whether downloading using FlexCAN or eSCI the BAM collects each 8 bytes of transmitted information and performs 64-bit writes to the internal RAM. Only the SRAM array supports the 64-bit writes implemented by the BAM. An advantage of using non-system RAM for stack and heap allocation is that it removes the time delay required to initialize the entire SRAM array

4.1.2 Develop Download Code in Assembler Code

By developing serial-download application in assembler code using internal GPRs only, the user can avoid using uninitialized SRAM locations.

4.1.3 Initialize the Entire SRAM Array

There are two methods for ensuring the entire SRAM array is initialized.

- Insert SRAM ECC initialization code:

  The serial download mode of the BAM performs 64-bit writes to the internal SRAM using the STMW instruction. Each 64-bit write performed by the BAM actively enables the ECC bits for that double word. By placing SRAM initialization code at the start of the download code (before stack or heap is established), the entire SRAM array can be initialized before executing any code that could make use of uninitialized SRAM.

  For example: consider the case where 32K of code is downloaded via eSCI. This could occupy address range 0x4000_0000 to 0x4000_7FFF. The remaining 32K of RAM from address 0x4000_8000 to 0x4000_FFFF can be initialized by adding the following code to the start of the download code.

```
Clear_GPRs
li r16, 0                  # Clear general purpose registers 16 - 32
li r17, 0                  # This is to ensure that the SRAM is initialized to zero.
li r18, 0                  # Although not always strictly necessary it is
…..                        # good practice to initialize memory to be used for
li r31, 0                  # variables to zero.

init_SRAM:
lis r11,0x4000             # Load address 0x4000_8000
ori r11,r11,0x8000
li r12, 0x200              # 32k/4 bytes/16 GPRs = 512 = 0x200.
mtctr r12                  # Move contents of r12 into counter

init_SRAM_loop:
stmw r16,0(r11)            # write all 16 GPRs to SRAM
addi r11,r11,0x40          # increment the ram pointer by 64 bytes
bdnz init_SRAM_loop        # loop for all remaining 32k of SRAM
blr
```

- Manipulate S-record/binary to initialize entire array:

  By simply appending or prefixing the downloadable user code with enough arbitrary values to create a 64K array the user can ensure that all of the SRAM is initialized. This can be performed manually or by software. For example, the SREC_CAT* utility provides a means to manipulate S-records, binary files and C arrays. To buffer an S-record so that it fills the entire 64K address range between 0x4000_0000 and 0x4001_0000 use the command line:

  – srec_cat *file*.s19 -fill 0x00 0x400000000 0x40003FFE -o *updatedfile*.s19

  To convert an S-record into a C-array for easily incorporating into serial download code use the command line:

  – srec_cat *file*.s19 -o *file*.c -C-array

* SREC_CAT is not a Freescale Semiconductor product. It is part of the GNU S-record tools suite available at http://srecord.sourceforge.net/

4.2 Serial Boot Password

To commence downloading code in serial boot mode a valid 64-bit password must first be downloaded. This can be either the user defined flash password or the pre-defined public password depending upon the censorship control and serial boot control bits, refer to Table 2.

- The predefined public password is 0xFEED_FACE_CAFE_BEEF.
- Within the flash password none of the 4 x 16-bit half words can be 0x0000 or 0xFFFF.

With knowledge of the flash password, the user can recover censored devices. This involves downloading a flash programmer into the internal SRAM that reprograms the censorship control bits, therefore unlocking the censored flash array.

4.2.1 Password Security Features

If an incorrect password is transmitted, the password is still echoed. There is no immediate indication if an incorrect password has been received by the BAM. Only if the next packet of information, that must define the address of _start is echoed does the user know if the password was accepted.

There are 2^{64} different password combinations. The only way to test these combinations is to try and download them. Each incorrect attempt requires either a watchdog timeout or a reset and the BAM to run again. Therefore, even with the maximum FlexCAN baud rate, it takes 100,000’s of years to exercise all 2^{64} passwords combinations.

Finally, the internal mechanism for comparing the downloaded password with the stored password is configured so that no trace of the stored password remains in any register that is not cleared by a reset, such as core GPRs, thus providing an extra level of security.

5 Core Watchdog Timer

In internal or external boot mode the BAM enables the watchdog, only if the RCHW[WTE] bit is set. The BAM programs the time base registers (TBU and TBL) to 0x0000_0000_0000_0000 and enables the core watchdog timer with a time-out period of 3 x 2^{17} system clock cycles. For example, an 8 MHz crystal generating a 12 MHz system clock results in a watchdog timeout of 32.7 ms.

When the BAM switches to serial boot mode the watchdog is always enabled. The watchdog is refreshed only after the correct flash/public password is accepted and subsequently after each write to the internal SRAM. This means, the user must either service the watchdog or disable it after serial download is complete. The watchdog can be serviced in several ways.

- Clear the TBU and TBL bits to refresh the timer. This is acceptable when the code download has just completed, but may be less desirable if the downloaded application code makes use of the decrementer or fixed-interval-timer that are dependent upon the timebase.
- Stop the timebase by writing HID0[TBEN] = 0
- Periodically clearing the TSR[WIS] bit

- Increase the timeout period

The simplest method is to increase the timeout period to ensure that there is sufficient time for the downloaded code to run. For example, the default settings for the watchdog in serial boot mode are: WP = 0b01, WPEXT = 0b1001, selecting a timeout in the order of 2^{17} system clocks. The watchdog can be effectively disabled by increasing the timeout by writing the TCR[WP, WPEXT] bits. For example, setting WP=0b00 & WPEXT = 0b0000 gives a timeout of in the region of 2^{64} system clocks.

For information referring to the watchdog refer to the e200z6 Core Reference Manual available at www.freescale.com. Further information, including example code, can be found in AN2817 *MPC5500 Watchdog Timer.*

6 BAM Execution Time

The boot time from the internal flash based on an 8 MHz clock source, is <100 µs for internal flash if the RCHW is on the first block of the flash.

For external memory, the boot time is <150 µs.

For serial boot it is approximately 200 µs before the BAM waits for the first data from the SCI or CAN.

7 Debug Mode

The BAM program is not executed when the MCU comes out of reset in debug mode. Consequently, steps must be taken to perform the relevant MMU initialization that would normally have been implemented by the BAM. The EBI and relevant pins must also be initialized if the user intends to use external memory.

Three options are:

- If an application code with a valid RCHW resides in the internal flash or external memory. The BAM can be allowed to run and stopped using either hardware breakpoints or manually by the debugger.
- If no application code with a valid RCHW is present in either internal flash or external memory, the BAM execution can be stopped by using breakpoints. This can halt the execution of the BAM after it has set up the MMU but before the internal boot mode is entered. This ensures that the watchdog remains disabled allowing the user to continue with debug.
- The MMU can be manually setup by writing directly to the MMU MAS registers and executing tlbwe instructions. This bypasses the BAM altogether. Example scripts for Lauterbach Trace32 and Metroworks Codewarrior for MPC5500 are included in Appendix A.

8 Summary

This application note has highlighted the key features of the BAM and provided examples of how to develop code for internal and external boot and serial download.

Although the BAM is implemented on the entire MPC5500 family, the information in this application note was based on the devices in Table 1.

Appendix A

A.1 Debug Mode MMU Initialization Scripts

The following example scripts setup the MMU in a similar manner the way the BAM does. Both the internal flash and external memory space are set up in this example.

Table A-1. Example MMU Settings

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
0	Peripheral bridge B	0xFFF0_0000	0xFFF0_0000	1 Mbyte	Cache inhibited Guarded Big endian Global PID
1	Internal flash	0x0000_0000	0x0000_0000	16 Mbytes	Cache enabled Not guarded Big endian Global PID
2	External bus	0x2000_0000	0x2000_0000	16 Mbytes	Cache enabled Not guarded Big endian Global PID
3	SRAM	0x4000_0000	0x4000_0000	256 kbytes	Cache inhibited Not guarded Big endian Global PID
4	Peripheral bridge A	0xC3F0_0000	0xC3F0_0000	1 Mbyte	Cache inhibited Guarded Big endian Global PID

In each case all MMU accesses must go through the MMU assist registers (MAS0-MAS3). The contents of the MAS registers are copied into the TLB with the tlbwe instruction.

A.2 Lauterbach Trace 32

The following script file is executed as a ‘*.cmm’ file executed after the devices come out of reset in debug mode, but before application code is loaded.

```
; Setup MMU for peripheral B modules, base address = 0xFFF0_0000

; TLB0, 1 MByte memory space, guarded, do not cache, all access

MMU.TLBSET 0 0xC0000500 0xFFF0000A 0xFFF0003F

; Set up MMU for internal flash, base address = 0x0000_0000

; TLB1, 16 MByte memory space, not guarded, cacheable, all access

MMU.TLBSET 1 0xC0000700 0x00000000 0x0000003F
```

```
; Set up MMU for external memory, base address = 0x2000_0000

; TLB2, 16 MByte memory space, not guarded, cacheable, all access

MMU.TLBSET 2 0xC0000700 0x20000000 0x2000003F

; Set up MMU for internal RAM, Base address = 0x4000_0000

; TLB3, 256 KByte memory space, not guarded, do not cache, all access

MMU.TLBSET 3 0xC0000400 0x40000008 0x4000003F

; Set up MMU for peripheral A modules, base address = 0xC3F0_0000

; TLB4, 1 MByte memory space, not guarded, do not cache, all access

MMU.TLBSET 4 0xC0000500 0xC3F00008 0xC3F0003F
```

How to Reach Us:

Home Page:
www.freescale.com

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

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

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

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

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2831
Rev. 0
01/2008