Boot mode handling for XMC1000

XMC1000

About this document

Scope and purpose

This application note provides an introduction to the boot modes available in the XMC1000 Microcontroller family. It also includes hints on its usage for users who wish to change the Boot Mode Index (BMI) value and boot up from Alternate Boot Mode (ABM)

Intended audience

This document is intended for engineers who are familiar with the XMC1000 Microcontroller family.

References

[1] The user's manual can be downloaded from http://www.infineon.com/XMC

[2] DAVE™ and its resources can be downloaded from http://www.infineon.com/DAVE

Table of contents

1 Boot Mode Index (BMI) for booting up

Normally, in microcontrollers, the states of boot pins are read after the power-on reset sequence to determine the device boot mode. However, if these pins are reserved for boot mode selection, this limit the usable pins available for this application.

The XMC1000 Microcontroller supports a boot-pin-less concept known as Boot Mode Index (BMI) to determine the boot mode after power-on reset. The entered boot mode depends on the BMI value stored in the flash configuration sector 0 (CS0). During boot up, the start-up software for the XMC1000 device determines the start-up mode to be entered from the BMI value. In XMC1400 series device, “Boot from BMI” and “Boot from pins” mode are both supported.

Table 1 lists the boot modes supported in the XMC1000 series devices.

Note: For the (XMC1100, XMC1200, XMC1300 and XMC1400) boot kits, the devices are pre-programmed to user mode with debug enabled (SWD0). In this configuration, the application program starts to run after power-up.

Note: Default boot up is from “Boot from BMI”. “Boot from pins” is only supported in XMC1400 series device.

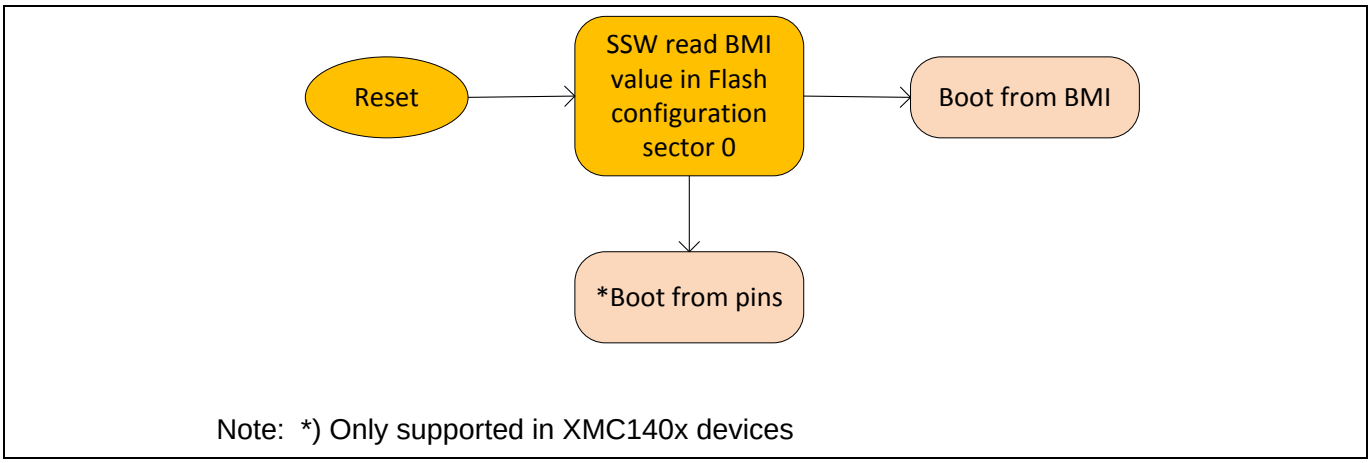
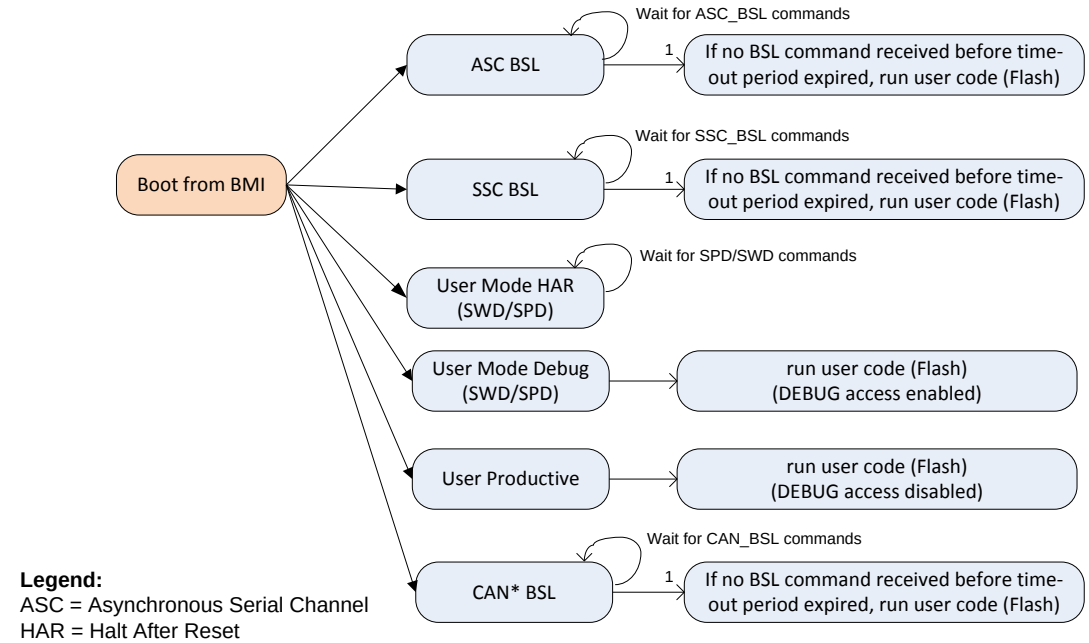


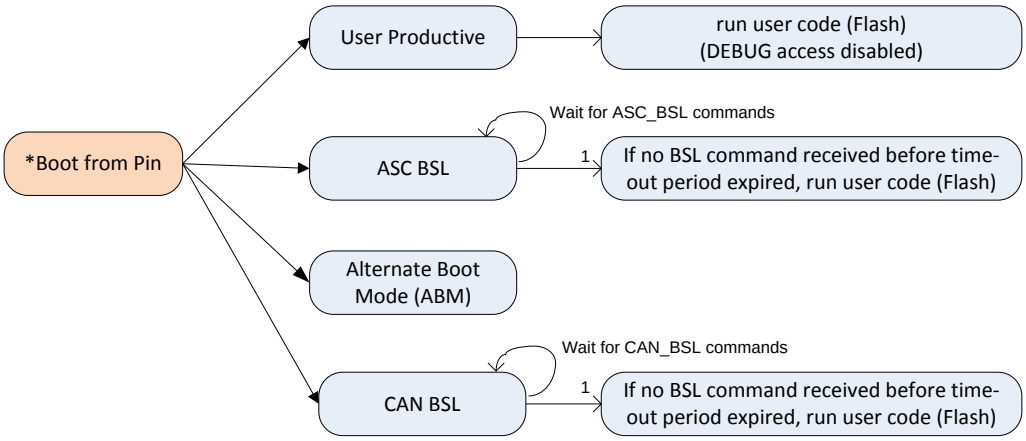
Figure 1 Boot ROM operating mode after power-on reset



Legend:
ASC = Asynchronous Serial Channel
HAR = Halt After Reset
BSL = Bootstrap Loader Mode
SSC = Synchronous Serial Communication
SPD = Single Pin Debug
SWD = Serial Wire Debug
CAN = Controller Area Network

Note: 1) for BSL with time-out mode
*) Only supported in XMC140x devices

Figure 2 Boot options from Boot Mode Index (BMI)



Legend:
ASC = Asynchronous Serial Channel
BSL = Bootstrap Loader Mode
CAN = Controller Area Network

Note: 1) for BSL with time-out mode
*) Only supported in XMC140x devices

Figure 3 Boot options from pins

1.1 Boot mode use cases

Table 1 Boot mode use cases

Mode	Use case	Boot type	
		BMI	Pins
ASC Bootstrap Loader mode, ASC_BSL	• Allow Flash programmer to download Flash erasing and programming routine using UART protocol to the XMC1000 device.	✓	✓
SSC Bootstrap Loader mode, SSC_BSL	• Allow code download from a SPI-compatible serial EEPROM into SRAM using SSC protocol to the XMC1000 device.	✓	-
User mode with debug enabled and HAR (SWD) or User mode HAR (SWD)	• Serial Wire Debug is ARM propriety debug protocol for ARM Cortex™ processor. • User code will not run after power-up. • Allows debugging at the beginning of user code.	✓	-
User mode with debug enabled and HAR (SPD) or User mode HAR (SPD)	• Single Pin DAP is Infineon propriety debug protocol. • Allows single pin debugging. • Pin-saving for XMC1000's 16 pin package. • User code will not run after power-up. • Allows debugging at the beginning of user code.	✓	-
User mode with debug enabled (SWD) or User mode debug (SWD)	• User code will run after power-up. • Supports debugging using serial wire debug protocol.	✓	-
User mode with debug enabled (SPD) or User Mode debug (SPD)	• User code will run after power-up and supports debugging using single pin debug protocol.	✓	-
User productive mode	• Flash protection scheme. • Debugging is not supported.	✓	✓
CAN Bootstrap Loader mode, CAN_BSL	• Allow SRAM download of user application using CAN protocol to the XMC1000 device.	✓	✓
Alternate Boot Mode (ABM)	• Allow startup from a user defined flash address	-	✓

Note: CAN BSL and Alternate Boot Mode (ABM) modes are only supported in XMC1400 series device.

1.2 Programming / debugging pin

Two sets of pins are available for you to choose from to use for programming or debugging the XMC1000 device. The selection of a port pin depends on the BMI value. In the subsequent sections, the mode selection processes are described.

Note: For ASC_BSL mode, both channel 0 and 1 are ready for UART communication after power-up, so the user does not need to choose which channel to use for ASC_BSL communication.

Note: For CAN_BSL, both node 0 and 1 are ready for MultiCAN communication after power-up, the selection of the nodes and the detection of the Txd and Rxd pins are done automatically. If the start-up software detects the initialization frame on P0.14, it would configure P0.15 for Txd functionality on Node 0.

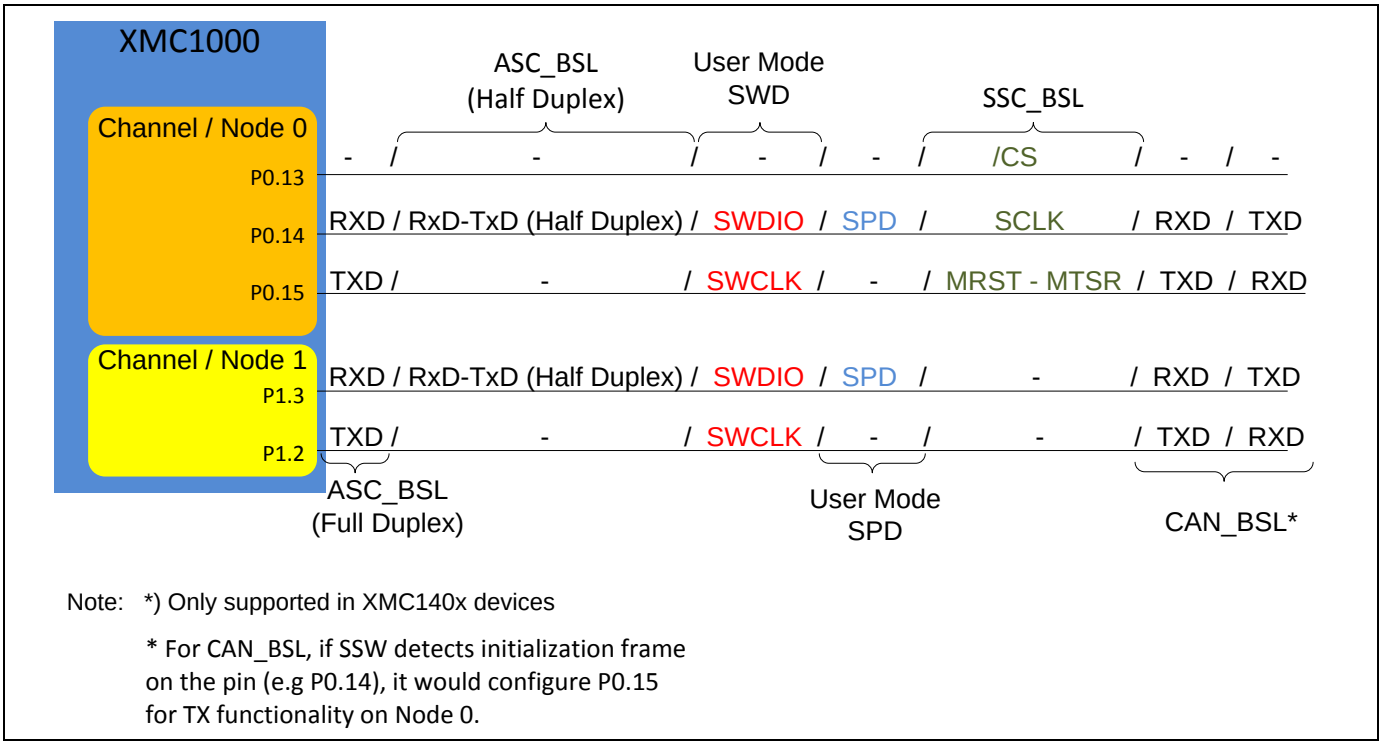
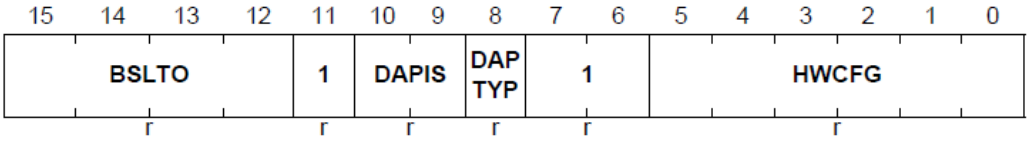


Figure 4 Pins used to communicate with the XMC1000 device in each boot mode

1.3 Boot Mode Index(BMI)

The XMC1000 device BMI value is read from memory location 0x10000E00. The factory default BMI value at delivery is 0xFFC0, meaning that the device is in ASC Bootstrap Loader mode.

BMI
Boot Mode Index **CS0 offset $0E00_H$** **Delivery state: $FFC0_H$**

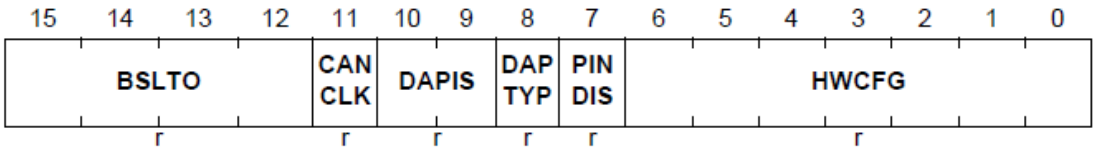


Field	Bits	Typ	Description
HWCFG	[5:0]	r	**Start-up mode selection:** 000000_B ASC Bootstrap Loader mode (ASC_BSL) 000001_B User productive Mode (UPM) 000011_B User mode with debug enabled (UMD) 000111_B User mode with debug enabled and HAR (UMHAR) 001000_B SSC Bootstrap Loader mode (SSC_BSL) 010000_B ASC BSL mode with time-out (ASC_BSLTO) 011000_B SSC BSL mode with time-out (SSC_BSLTO) 111010_B Secure Bootstrap Loader mode (SBSL) 1) else Not defined 2)
DAPTYP	8	r	**DAP Type Selection** 0_B Serial wire debug interface is selected 1_B Single pin debug interface is selected
DAPIS	[10:9]	r	**SWD/SPD Input/Output Selection** 00_B SWD/SPD_0 pin is selected 01_B SWD/SPD_1 pin is selected 10_B Reserved 11_B Reserved
BSLTO	[15:12]	r	**ASC BSL Time-out value** The time-out duration is BSLTO*2664000 MCLK cycles, the supported time-out range is 0.3-5s (333...4995ms)
1	[7:6], 11	r	**Reserved, must be programmed to 1**

1) Only in device versions supporting it

Figure 5 Boot Mode Index (BMI) bit field description for XMC1100, XMC1200 and XMC1300 device series

BMI
Boot Mode Index **CS0 offset $0E00_H$** **Delivery state: $FFC0_H$**



Field	Bits	Typ	Description
HWCFG	[6:0]	r	**Start-up Mode Selection:** 0000000_B CAN Bootstrap Loader mode (CAN_BSL) 0010000_B CAN BSL mode with time-out (CAN_BSLTO) 0100000_B Secure Bootstrap Loader mode over CANopen (SBSL CANopen) 1000000_B ASC Bootstrap Loader mode (ASC_BSL) 1000001_B User productive Mode (UPM) 1000011_B User mode with debug enabled (UMD) 1000111_B User mode with debug enabled and HAR (UMHAR) 1001000_B SSC Bootstrap Loader mode (SSC_BSL) 1010000_B ASC BSL mode with time-out (ASC_BSLTO) 1011000_B SSC BSL mode with time-out (SSC_BSLTO) 1111010_B Secure Bootstrap Loader mode over ASC (SBSL ASC) else Reserved [1]
PINDIS	7	r	**Boot Configuration Type Selection** 0_B Boot from pins is selected 1_B Boot from BMI is selected
DAPTYP	8	r	**DAP Type Selection** 0_B Serial wire debug interface is selected 1_B Single pin debug interface is selected
DAPDIS	[10:9]	r	**SWD/SPD Input/Output Selection** 00_B SWD/SPD_0 pin is selected 01_B SWD/SPD_1 pin is selected 10_B Reserved 11_B Reserved
CANCLK	11	r	**CAN Clock Source for CAN BSL Mode** 0_B Synchronous CAN clock via internal oscillator (DCO1) with enabled trimming via external reference is selected 1_B Synchronous CAN clock via external oscillator (OSC_HP) is selected
BSLTO	[15:12]	r	**ASC BSL Time-out value** The time-out duration is BSLTO*2664000 MCLK cycles, the supported time-out range is 0.3-5s (333...4995ms)

1) ATTENTION: Installing a reserved BMI value will cause device delivery state to be restored upon the next reset

Figure 6 Boot Mode Index (BMI) bit field description for XMC1400 device series

1.4 Booting up from BMI

The default boot configuration is “Boot from BMI” for XMC1000 devices. Each of the Boot modes are described in this section.

Note: In the XMC1400 series device, the option to “Boot from BMI” is selected if the BMI value is programmed with BMI.PINDIS = 1.

1.5 ASC_BSL - ASC Bootstrap loader mode

After power-up/master reset, the device waits for a falling edge transition at its RXD pin after exit from the startup software. This falling edge transition indicates the ASC_BSL handshaking has started.

In ASC_BSL mode, the user can download the code into SRAM starting at address 0x20000200. After the code is received and stored, the Start-up software will run the user’s code at SRAM address 0x20000200.

The ASC_BSL can communicate with the host PC in full duplex or half duplex mode. This depends on the header byte that the host PC sends to the XMC1000 device during the ASC_BSL handshaking.

ASC_BSL supports a time-out option. If the XMC1000 device does not receive the start and header bytes from the host within the duration defined in BMI.BLSTO, it will jump to the flash location whose address is stored at 0x10001004, and execute from there.

Table 2 BMI values and Port settings for ASC_BSL mode

Header byte	BMI value	UART communication	RXD Pin	TXD Pin
0x12	0xFFC0	Half duplex	P0.14	P0.14
			P1.3	P1.3
0x6C	0xFFC0	Full duplex	P0.14	P0.15
			P1.3	P1.2
0x12	0xFFD0 (time-out = 4995 ms @ MCLK = 8 MHz)	Half duplex	P0.14	P0.14
			P1.3	P1.3
0x6C	0xFFD0 (time-out = 4995 ms @ MCLK = 8 MHz)	Full duplex	P0.14	P0.15
			P1.3	P1.2

1.5.1 User mode with debug enabled and HAR (UMHAR)

After power-up/master reset, the device will wait for SWD/SPD connection after exit from the startup software. If it encounters a system reset, the device will behave in a user mode (debug) manner. The user can access the full debug functionality using SWD/SPD to establish a connection to the XMC1000 device.

This mode is recommended when the user wants to develop and debug their code, especially in motor applications. After power-up, the user’s application code will not run and the user is in full control on where the code execution should stop (e.g at which part of the application code), via the debugger.

Table 3 BMI value and port settings for user mode with debug enabled and HAR (UMHAR)

BMI value	SWD / SPI Signal	Pin used
0xF8C7	SWDIO_0	P0.14
	SWDCLK_0	P0.15
0xF9C7	SPD_0	P0.14

0xFAC7	SWDIO_1	P1.3
	SWDCLK_1	P1.2
0xFBC7	SPD_1	P1.3

1.5.2 User mode with debug enabled (UMD)

After power-up, the device will start to run from the flash location address stored at 0x10001004. The specified SPD/SWD pins are automatically configured to allow for debugger access.

This mode is recommended when you want to test the code in a real application environment. You can update your working code using the debugger for future upgrades. This 'hot-attached' mode is the default connection mode for the ARM® Cortex™ processor.

Table 4 BMI value and port settings for user mode with debug enabled (UMD)

BMI value	SWD / SPD Signal	Pin used
0xF8C3	SWDIO_0	P0.14
	SWDCLK_0	P0.15
0xF9C3	SPD_0	P0.14
0xFAC3	SWDIO_1	P1.3
	SWDCLK_1	P1.2
0xFBC3	SPD_1	P1.3

1.5.3 User productive mode (UPM)

Attention: ***This mode should only be used once the user has confirmed that their code is FULLY TESTED.***

After power-up, the device will start to run from the flash location address stored at 0x10001004.

This mode provides MEMORY PROTECTION by not allowing external tools such as the debugger, read/write access to the flash memory.

Table 5 BMI value and port settings for user productive mode (UPM)

BMI value	SWD / SPD signal	Pin used
0xF8C1	-	-

It is possible to change the BMI value in this mode. This is done by embedding the provided "Request BMI installation routine (new BMI)" in the user code. The "Request BMI installation" routine is provided inside the XMC1000 ROM at location 0x00000108. The application software can run this routine to change the BMI value under user-defined conditions. For example, the change of BMI value can be triggered from an external interrupt or from GPIO pin latch values.

For code protection purposes, if the current BMI value is User Productive Mode (UPM) and the newly requested BMI value is NOT User Productive Mode (UPM), the Start-up software will erase the full flash of the device, and re-install the default BMI (ASC_BSL). The user needs to call the 'Request BMI installation (new BMI)' routine again via ASC_BSL mode if that is not the desired BMI value.

An example using an external interrupt to trigger the “Request BMI installation” routine is provided in chapter 2.1.1.

1.5.4 CAN_BSL - CAN Bootstrap loader mode

After power-up/master reset, based on the settings of the BMI.CANCLK, either the internal or external oscillator is selected for the synchronous CAN clock source. There are three protocol phases (namely initialization, acknoweldgement and data transmission) for the user application to be downloaded. The size of the user-available SRAM determines the maximum size of the downloadable application code.

In the initialization phase, if the startup software detects an initialization frame on P0.14, it would configure P0.15 for TX functionality on Node 0, and vice versa. On the other hand, if the initialization frame is detected on P1.3, then P1.2 would be configured for TX functionality for Node 1.

In CAN_BSL mode the user can download their program into SRAM, starting at address 0x20000200. After the program is received and stored, the start-up software will run the user’s program at SRAM address 0x20000200.

CAN_BSL supports a time-out option. If the device does not receive the initialization frame from the host within the duration defined in BMI.BLSTO, it will jump to the flash location whose address is stored at 0x10001004, and execute from there.

Note: CAN BSL is only supported in XMC1400 series devices. When the external oscillator is selected, a stable external clock (OSC_HP) is mandatory. For transfer rates of 1 Mbps, the user has to ensure that the external clock is at least 10 MHz.

Table 6 BMI values and port settings for CAN_BSL mode

BMI value	Clock	RXD Pin	TXD Pin
0xFF80	External oscillator	P0.14	P0.15
		P0.15	P0.14
		P1.3	P1.2
		P1.2	P1.3
0xFF90 (time-out = 4995 ms @ MCLK = 8 MHz)	External oscillator	P0.14	P0.15
		P0.15	P0.14
		P1.3	P1.2
		P1.2	P1.3
0xF780	Internal oscillator	P0.14	P0.15
		P0.15	P0.14
		P1.3	P1.2
		P1.2	P1.3
0xF790 (time-out = 4995 ms @ MCLK = 8 MHz)	Internal oscillator	P0.14	P0.15
		P0.15	P0.14
		P1.3	P1.2
		P1.2	P1.3

1.6 Booting up from pins

The option to boot up from pins is selected if the BMI value is programmed with BMI.PINDIS = 0. Upon master reset, the values at boot pins P4.6 and P4.7 are latched. These values shall be evaluated and used to enter the selected boot modes.

Note: "Boot from pins" mode is only supported in the XMC1400 series controller. In order to boot up correctly, a valid BMI.HWCFG must be programmed in the BMI value.

Upon master reset, the values at the boot pins P4.6 (STSTAT.HWCON[0]) and P4.7(STSTAT.HWCON[1]) are latched in.

HWCON[1]	HWCON[0]	Boot Mode
0	0	User Productive Mode
0	1	ASC BSL
1	0	Alternate Boot Mode
1	1	CAN BSL[1)]

1) A programmed value of BMI.BSLTO results in a time-out duration defined by the bit field. If time-out duration is not intended, BMI.BSLTO shall be configured with 0.

Figure 7 Boot Pin Mode for XMC1400 series device

1.6.1 User Productive Mode (UPM)

Refer to Chapter 1.5.3.

1.6.2 ASC_BSL - ASC Bootstrap Loader mode

Refer to Chapter 1.5.

1.6.3 ABM – Alternate Boot Mode

After power-up/master reset, the values at boot pins P4.6 and P4.7 are latched to STSTAT.HWCON. If the latched value to HWCON = 10, it enters to ABM mode. On entry to this bootmode, the startup software branches to the start address of the ABM user code, only if a matching magic key and code length value greater than 0 is fulfilled. The ABM header is placed in the last 32 bytes of the user flash area.

Note: In this mode, a user defined application should first be downloaded to the user-defined flash address via the standard bootstrap loader or SWD. This must be followed by a change in the BMI to boot via pins along with the selection of ABM boot mode and followed by a master reset for the new boot mode to take effect.

Table 7 Alternate Boot Mode Header (ABMHD) structure

Offset Address	Size (Byte)	Field Name	Description
0x00	4	MAGICKEY	Magic key = 0xA5C3E10F
0x04	4	STADDABM	Start address of the ABM user code
0x08	4	CODELENGTH	Length of the user code of ABM (This value should be greater than 0)

Table 8 Start address of the ABMHD structure for different flash options

Flash options (KBytes)	Start address of the ABMHD structure
200	0x10032FE0
128	0x10020FE0
64	0x10010FE0
32	0x10008FE0

1.6.4 CAN_BSL - CAN Bootstrap Loader mode

Refer to chapter 1.5.4.

2 Programming the BMI value

When the XMC1000 device is delivered from the factory, the default boot mode is ASC_BSL. If you want to connect the device to the debugger, you need to change the BMI value to 'User mode with debug enabled', or to 'User mode with debug enabled and HAR'. The change is made via the "Request BMI installation routine (new BMI)", available in ROM address 0x00000108. Use this routine to change the current BMI value to the required value.

In the following sections, example code is used to illustrate how to change the BMI value using software on an external interrupt. An introduction to Memtool (v4.5 and later) for changing the BMI value is also provided in the following sections.

Note: A "Master reset" is to be executed in order for the BMI value to be updated. However, if the power supply is switched off before the "Master reset" occurs, the BMI value will revert to the default value (ASC_BSL), instead of the desired BMI value. It is therefore very important to keep the VDDP of the XMC1000 device operating voltage stable when programming the BMI value. If user is changing BMI from User Productive mode to ASC_BSL mode, please keep the power supply of the XMC1000 stable for at least 7 seconds for a 200kB flash size (MCLK@8MHz) after "Request BMI installation" routine is called. If the user is changing BMI to any other mode, then please keep the power supply of the XMC1000 stable for at least 10 milliseconds (MCLK@8MHz) after "Request BMI installation" routine is called.

2.1.1 Programming the BMI by calling a user routine upon external interrupt

In this example, an external interrupt is triggered based on a rising edge event detected on P2.0. In the interrupt handler, the routine to install a new BMI to ASC_BSL mode is called. The rising event is set up using the Event Request Unit (ERU) that triggered an interrupt on ERU0.SR0.

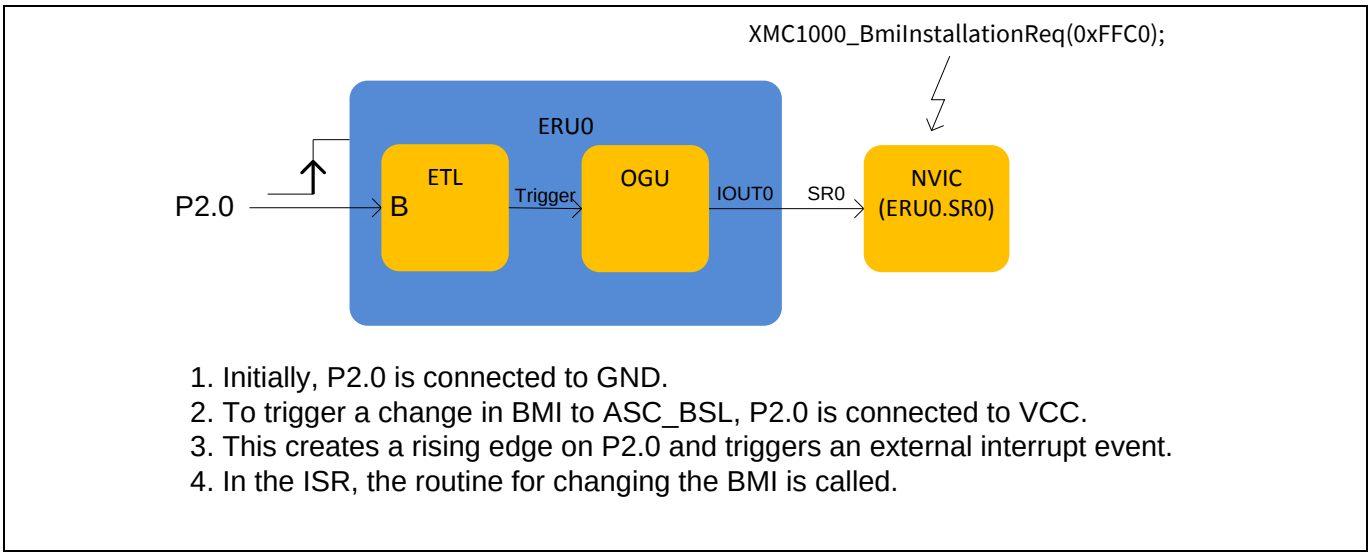


Figure 8 Changing the BMI from an external interrupt

2.1.2 Macro and variable settings

```
/* XMC lib project includes: */
#include <xmc_eru.h>
#include <xmc_gpio.h>
#include <xmc_scu.h>

/* Project definitions */
#define ROM_FUNCTION_TABLE_START        (0x00000100)
#define _BmiInstallationReq             (ROM_FUNCTION_TABLE_START + 0x08)

/* Pointer to Request BMI installation routine */
#define XMC1000_BmiInstallationReq      \
                (*((unsigned long (**) (unsigned short)) _BmiInstallationReq))
```

2.1.3 XMC lib peripheral configuration structure

```
/* XMC GPIO configuration */
XMC_GPIO_CONFIG_t input_config =
{
  .mode = XMC_GPIO_MODE_INPUT_TRISTATE,
  .input_hysteresis = XMC_GPIO_INPUT_HYSTERESIS_STANDARD
};



/* Event Trigger Logic Configuration - ERU0.0B0 (P2_0) selected */
XMC_ERU_ETL_CONFIG_t ERU0_ETL_Config =
{
  .input_a = (uint32_t)XMC_ERU_ETL_INPUT_A0, /* Event input selection for A(0-3) */
  .input_b = (uint32_t)XMC_ERU_ETL_INPUT_B0, /* Event input selection for B(0-3) */

  .enable_output_trigger = (uint32_t)1,
  .status_flag_mode = (XMC_ERU_ETL_STATUS_FLAG_MODE_t)XMC_ERU_ETL_STATUS_FLAG_MODE_HWCTRL,

  /* Select the edge/s to convert as event */
  .edge_detection = XMC_ERU_ETL_EDGE_DETECTION_RISING,

  /* Select the source for event */
  .output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL0,
  .source = XMC_ERU_ETL_SOURCE_B
};

/* Output Gating Unit Configuration - Gated Trigger Output */
XMC_ERU_OGU_CONFIG_t ERU0_OGU_Config =
{
  .peripheral_trigger         = 0U, /* OGU input peripheral trigger */
  .enable_pattern_detection   = false, /* Enables generation of pattern match event */
```

```
  /* Interrupt gating signal */
  .service_request            = XMC_ERU_OGU_SERVICE_REQUEST_ON_TRIGGER,
  .pattern_detection_input    = 0U
};
```

2.1.4 Interrupt service routine function implementation

```
/* Interrupt Handler for External Trigger Interrupt */
void ERU0_0_IRQHandler(void)
{
  // BMI_installation routine to set BMI = ASC_BSL
  XMC1000_BmiInstallationReq(0xFFC0);
}
```

Note: For an XMC1400 series device, due to the interrupt handler "void ERU0_0_IRQHandler(void)" should be replaced with "void IRQ3_Handler (void)"

2.1.5 Main function implementation

```
int main(void)
{
  /* Sets up the ERU- ETL and OGU for the external trigger event */
  XMC_ERU_ETL_Init(XMC_ERU0, 0, &ERU0_ETL_Config);
  XMC_ERU_OGU_Init(XMC_ERU0, 0, &ERU0_OGU_Config);

  /* Initializes the gpio input and output */
  XMC_GPIO_Init(P2_0, &input_config);

  /* Enable the interrupt - ERU0_SR0 */
  // XMC_SCU_SetInterruptControl(3, XMC_SCU_IRQCTRL_ERU0_SR0_IRQ3); //Only for XMC140x
  NVIC_EnableIRQ(3U);

  /* Placeholder for user application code. */
  while(1U)
  {
  }
}
```

Note: For an XMC1400 series device, uncomment the code line "XMC_SCU_SetInterruptControl(3, XMC_SCU_IRQCTRL_ERU0_SR0_IRQ3);"

2.2 Using Memtool to change the BMI value

Memtool is a free tool from Infineon Technologies that can be used to support the programming of XMC1000 devices.

Note: Other professional tools can also support changing the BMI value, such as Universal Access Device 2 from PLS for example, but those tools are not described in this document.

Table 9 Memtool (v4.5 and higher) connection configuration

Hardware	Driver	Connection methods
XMC1000 Boot Kit	Virtual COM port	ASC_BSL
XMC™ Link	Virtual COM port	ASC_BSL
DAP miniwiggler v3.x	DAS	ASC_BSL, SWD, SPD

2.2.1 Memtool connection to XMC1000 using virtual COM port via XMC1000 CPU card

This mode of connection can only be used when the XMC1000 device BMI value is ASC_BSL mode.

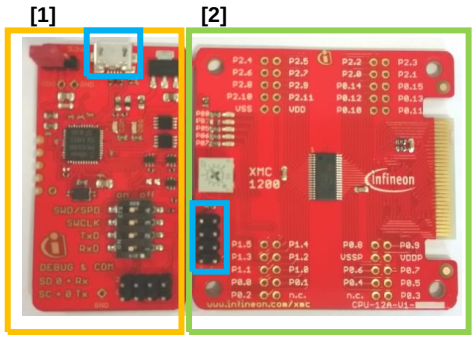
There are 2 cases where the connection to the device can be done through the virtual COM port. The first is using the XMC1000 boot kit, where the CPU card has an on-board COM and SEGGER J-Link debugger. The other case is using an XMC™ Link, an isolated debug probe for all XMC™ Microcontrollers, on a standalone MCU board. The debug probe is based on SEGGER J-Link debug firmware, which enables use with DAVE™ and all major third-party compiler/IDEs known from the wide ARM® ecosystem.

Once the appropriate connection is made, Memtool uses the COM port for ASC_BSL communication with the XMC1000 device. If you have any virtual COM port equipment, Memtool can be used to perform ASC_BSL communication with the XMC1000 device and change the BMI value.

For the pin connection please refer to **Figure 4**.

Note: The XMC1000 boot kit can be ordered online from: www.infineon.com/xmc1000 -> XMC1000 kits.

Note: XMC™ Link can be ordered online from: www.infineon.com/xmclink.



XMC1200 CPU Board

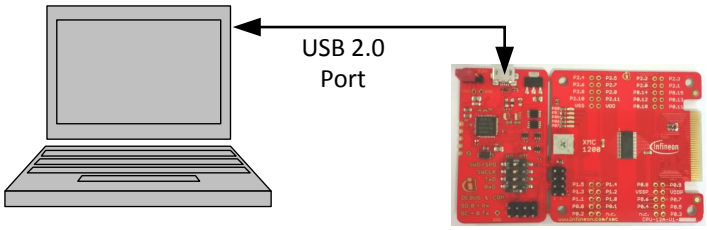
XMC1200 CPU board is built with:
[1]On-board COM and Segger J-Link Debugger,
[2] XMC1200 device

For memtool connection to the device through Virtual COM port, the device will need to be in ASC_BSL mode. There are 2 ways that this can be done.

Method 1: If the board has an on-board COM and Segger J-Link Debugger similar to the XMC1200 CPU board. You can connect it to the USB connection built on the board [1].

Method 2: If the board is a standalone board with only [2] MCU and a header debug connector, it can be connected to the XMC™ Link.

Method 1: Memtool connection using on-board COM



Method 2: Memtool connection using XMC™ Link

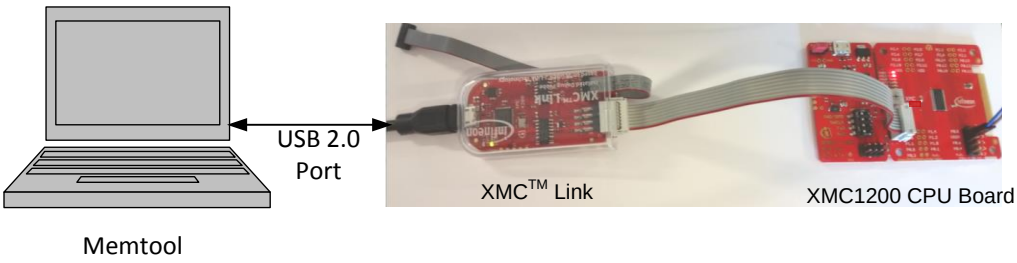


Figure 9 Method to connect via the virtual COM port using Memtool

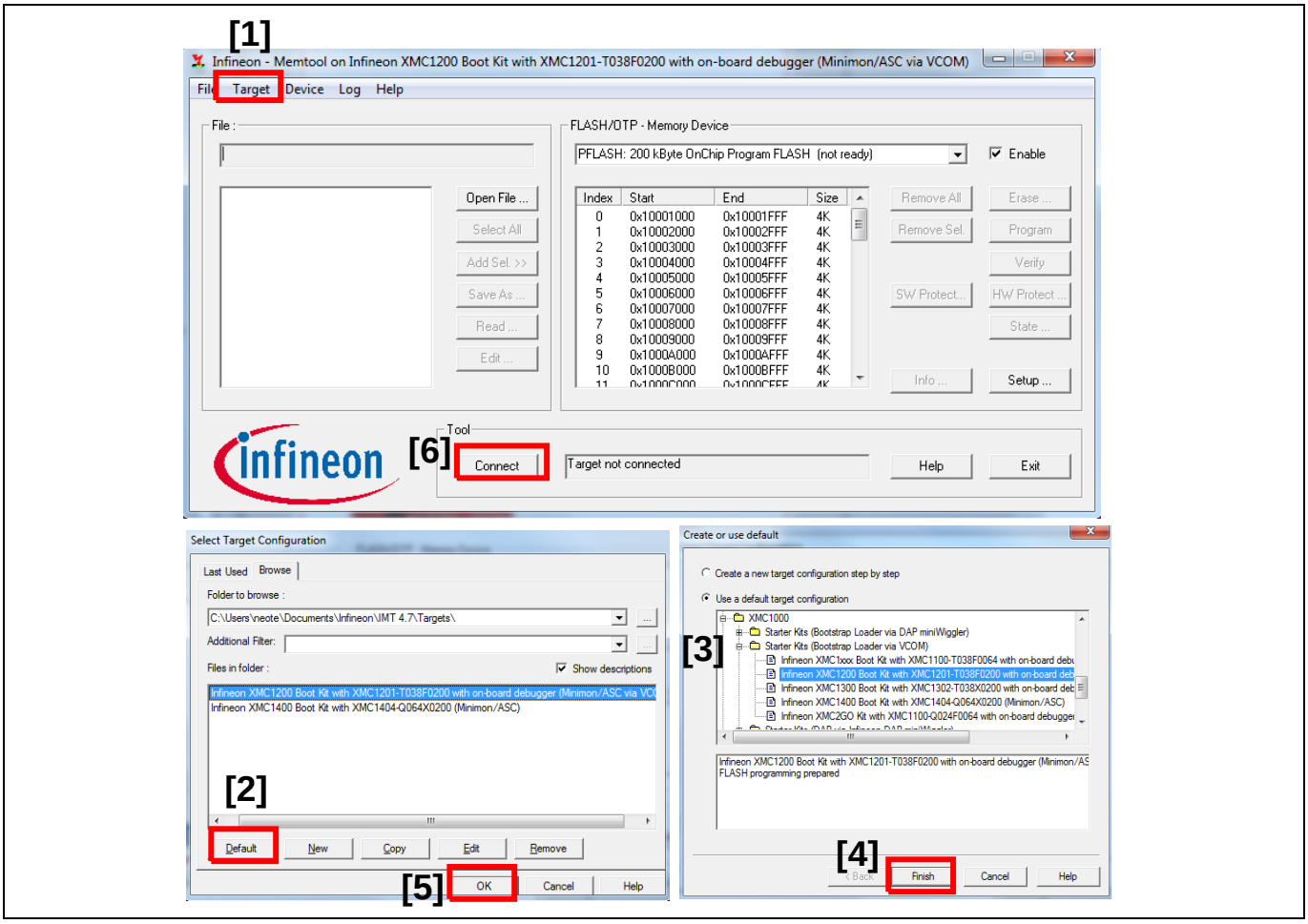


Figure 10 Setup Memtool to connect to XMC1000 using virtual COM port

Procedure

1. Open Memtool (v4.5 and higher) and select **[1]** “Target → Change …”.
2. New dialog box “Select target configuration” opens. Select **[2]** “Default” button.
3. New dialog box “Create or use default” opens. Select from the default target configuration available where a list of XMC1000 boards is available for selection.
   - If the BMI of the XMC1200 boot kit is in ASC_BSL mode, select **[3]**“Infineon XMC1200 boot kit with XMC1201-T038F0200 with on-board debugger (Minimon/ASC)” and click **[4]** “Finish”
4. Click **[5]** “OK”. Connect the USB cable to the XMC1200 boot kit board and click **[6]**”Connect” button.

2.2.2 Memtool connection to XMC1000 using DAS via DAP miniWiggler

This mode of connection can be used when the XMC1000 device BMI value is ASC_BSL, user mode debug (SWD/SPD) or user mode HAR (SWD/SPD).

The DAP miniWiggler can be ordered online from: www.infineon.com/das -> DAP miniWiggler.

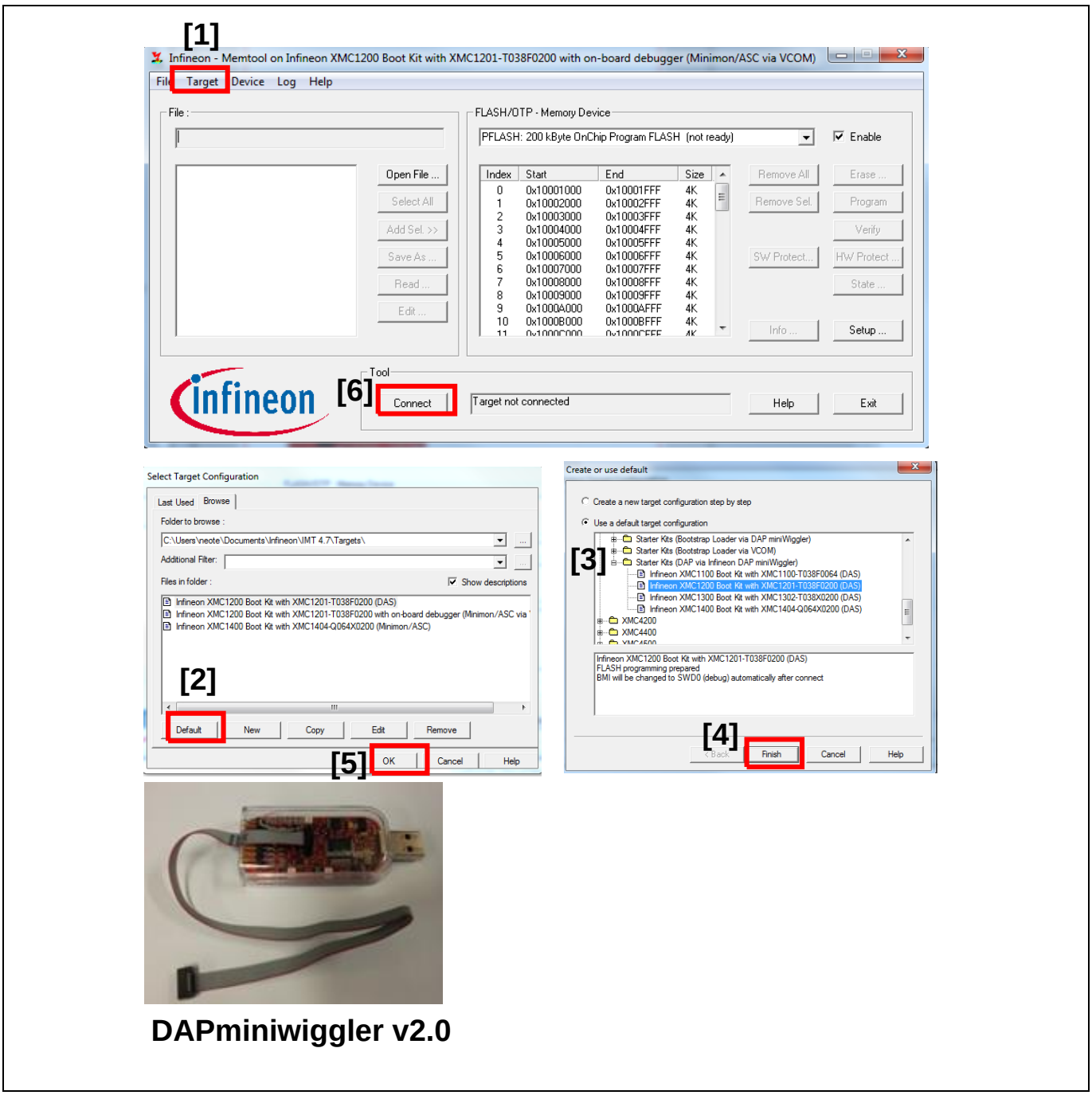


Figure 11 Setup Memtool to connect to the XMC1200 target board using DAS

Procedure

1. Open Memtool (v4.5 and higher) and select **[1]** “Target → Change …”.
2. New dialog box “Select target configuration” opens. Select **[2]** “Default” button.
3. New dialog box “Create or use default” opens. Select from the default target configuration available where a list of XMC1000 boards is available for selection.
   - If using the XMC1200 target board, select **[3]**“Infineon XMC1200 boot kit with XMC1201-T038F0200 (DAS)” and click **[4]** “Finish”
4. Click **[5]** “OK”. Plug the DAP miniWiggler to the USB port of the Host PC and connect the 10-pin connector of the DAP miniWiggler to the XMC1200 target board as show in Figure 12.
5. Connect the USB cable to the XMC1200 boot kit board and click **[6]**”Connect” button. Now, Memtool should be able to connect to the XMC1200 device.

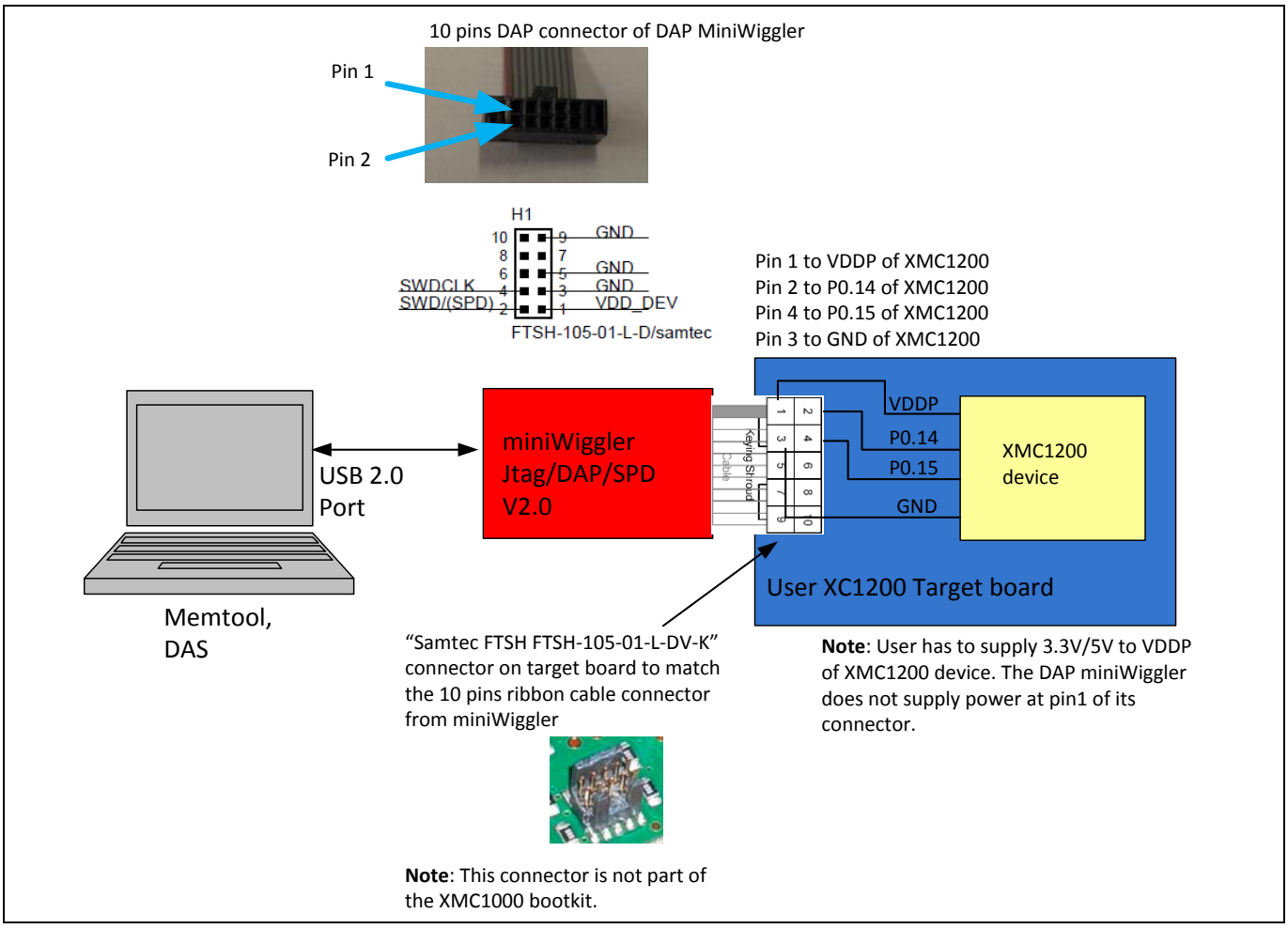


Figure 12 Connection diagram between DAP miniWiggler v2.0 and XMC1200 target board

2.2.3 Procedure for using Memtool to change the BMI value after connection

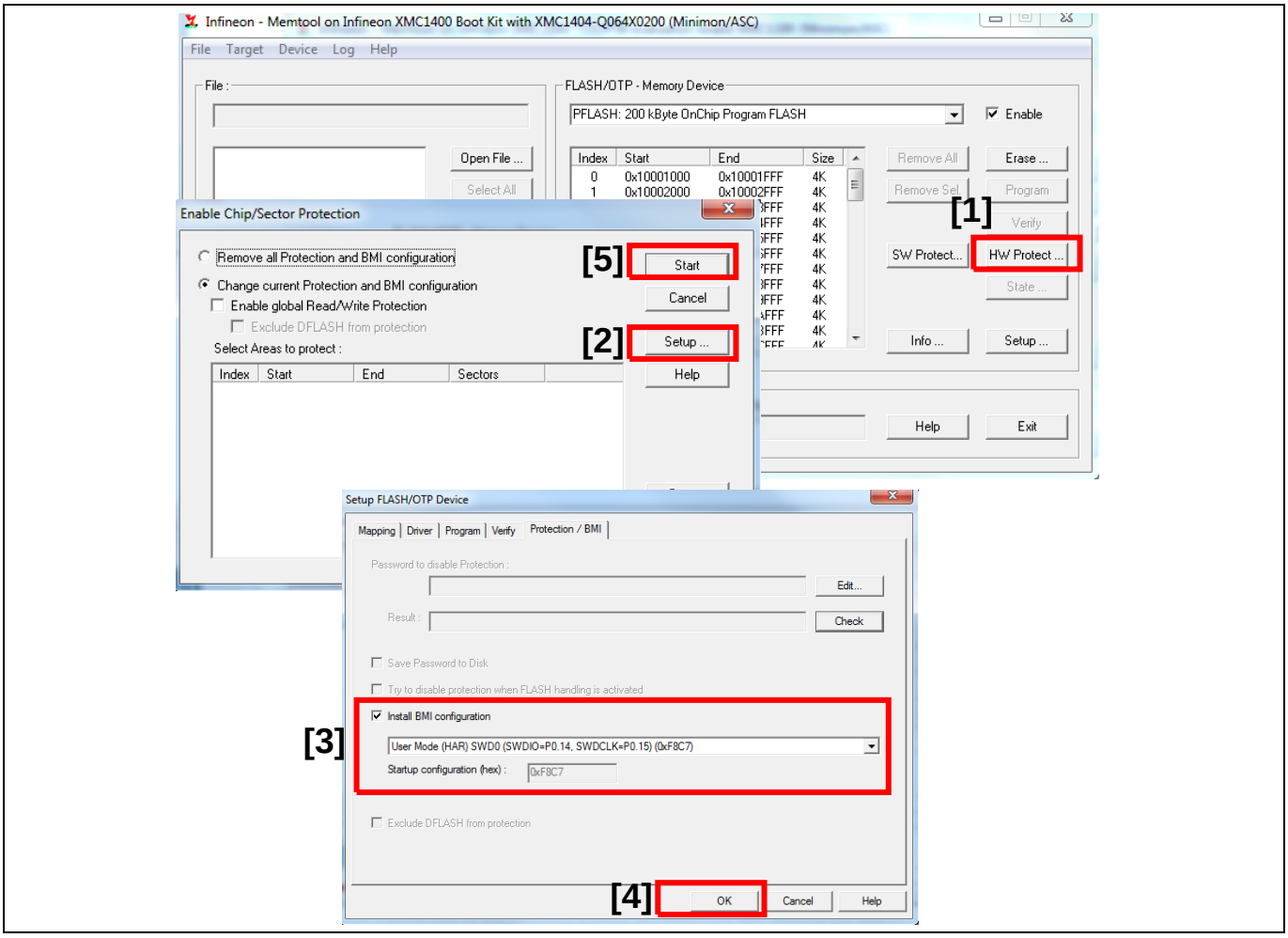


Figure 13 Using Memtool to change the BMI value

With the DAS driver, Memtool can connect to the XMC1000 device via ASC_BSL, SWD and SPD protocols. When using the virtual COM port, Memtool can only connect to the XMC1000 device via ASC_BSL. After connection to the XMC1000 device, to change the BMI value of the XMC1000 device:

1. Select **[1]** “HW protect…” button.
2. New dialog box “Enable chip/sector protection” opens. Select **[2]** “Setup…” button.
3. New dialog box “Setup flash/OTP device” opens. Select the new BMI with **[3]** Install BMI configuration. Once selected, click **[4]** “OK”.
4. In the dialog box “Enable chip/sector protection”, select **[5]** “Start” to change the BMI value of the XMC1000 device.

2.3 Using DAVE™ - 'BMI get set' feature to change BMI value

There is a BMI handling feature in DAVE™ (v3.1.10 and higher) release. This feature works with J-Link XMC4200 OBD (part of the XMC1000 boot kit), J-Link, J-Link EDU or XMC™ Link.

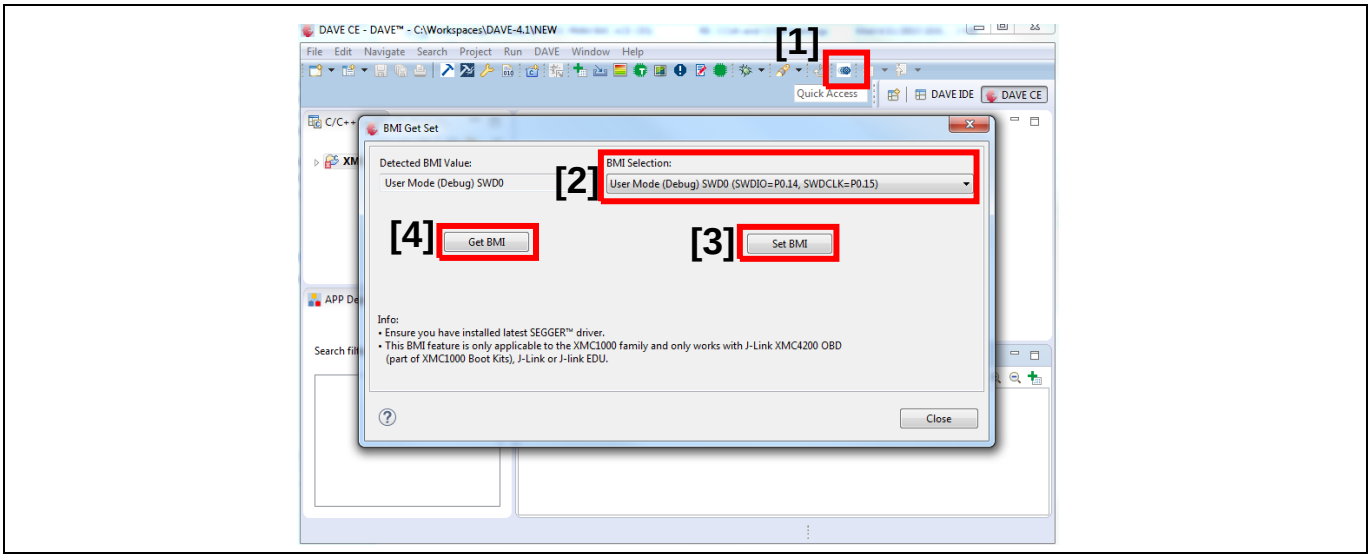


Figure 14 Illustration of 'BMI get set' feature in DAVE™ v3.1.10 and above

First, the J-Link debugger needs to connect to the XMC1000 device. Using the following steps in the DAVE™ IDE, the BMI can be updated:

1. Launch DAVE™ (v3.1.10 and higher) IDE and select **[1]** "BMI get set" button.
2. New dialog box "BMI get set" opens. Select **[2]** "BMI selection:", the desired NMI value to set to the XMC1000 device.
3. Select "Set BMI" **[3]** to install the new BMI value. The chosen BMI value will be updated after a system reset to the XMC1000 device.
4. To read the BMI value, click **[4]** "Get BMI". The BMI value will be read out from XMC1000 device and output at the 'Detected BMI value' field.

3 Booting up from alternate boot mode

In order for boot up from Alternate Boot Mode (ABM), there are two steps that need to be performed. In the first step, the user program for Alternate Boot Mode needs to be set up to be loaded into the designated code area as required by the application.

In the second step, the set up of the ABM header information and the changing of the Boot Mode Index (BMI) into “Boot from pins” is required. With the BMI value programmed with BMI.PINDIS = 0, the boot configuration type is selected to “Boot from pins”. On a master reset, based on the latched value into the STSTAT.HWCON, the selected bootmode is entered. If ABM is selected, the startup software will read the ABM header structure and will branch to the start address of the ABM user code, if a matching magic key is available.

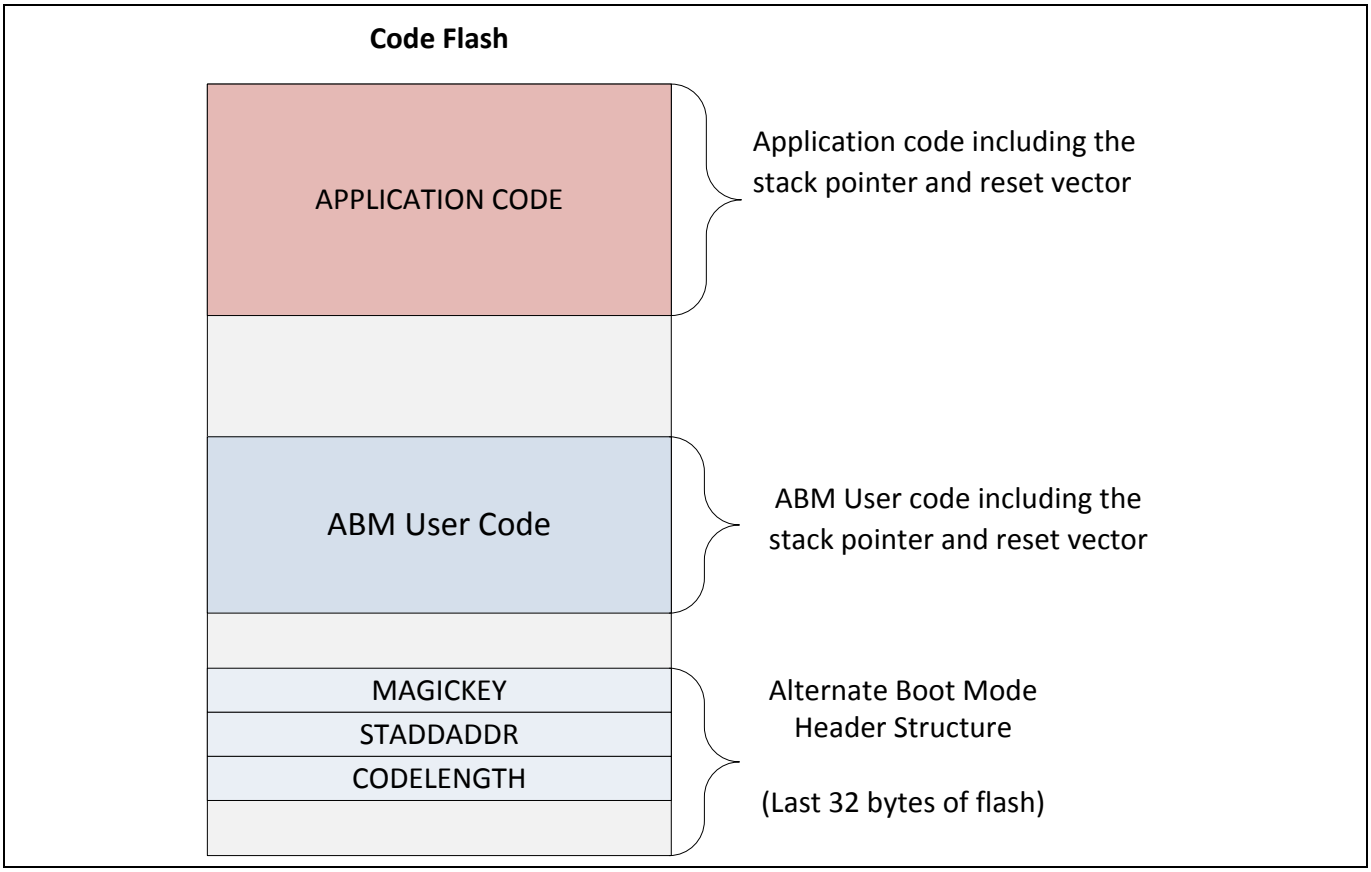


Figure 15 ABM concept for XMC1400 series device

In order to successfully boot into ABM, the ABM user code, as described in the first step, needs to be downloaded to the defined location prior to the change of the BMI value to ABM. The next step is to have the ABM header structure located at the last 32 bytes of the user flash area. Once all this is available, the final step is to change the BMI value to “Boot from pin”.

In the next sections, we will go through an example how to set up the user code into a defined address of the ABM user code, the setup of the ABM header as well as to change the BMI value to “Boot from pins”.

Note: “Boot from pins” is only supported in XMC1400 series devices.

3.1 Setting up the user program

In this example, a project is created on DAVE™ IDE for toggling a port pin at P4.1 at an interval of 10 Hz.

3.1.1 Linker script settings

The linker script is updated to locate the code at 0x10020000.

Change the start address of the code to 0x10020000

```
57 MEMORY
58 {
59     FLASH(RX) : ORIGIN = 0x10020000, LENGTH = 0x10000
60     SRAM(!RX) : ORIGIN = 0x20000000, LENGTH = 0x4000
61 }
62
```

Figure 16 Modify linker script to locate the user

3.1.2 Macro and variable settings

```
/* XMC lib Project includes */
#include <xmc_gpio.h>
```

3.1.3 XMC lib peripheral configuration structure

```
/* XMC GPIO Configuration */
XMC_GPIO_CONFIG_t output_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
};
```

3.1.4 Interrupt service routine function implementation

```
void SysTick_Handler(void)
{
        XMC_GPIO_ToggleOutput(P4_1);
}
```

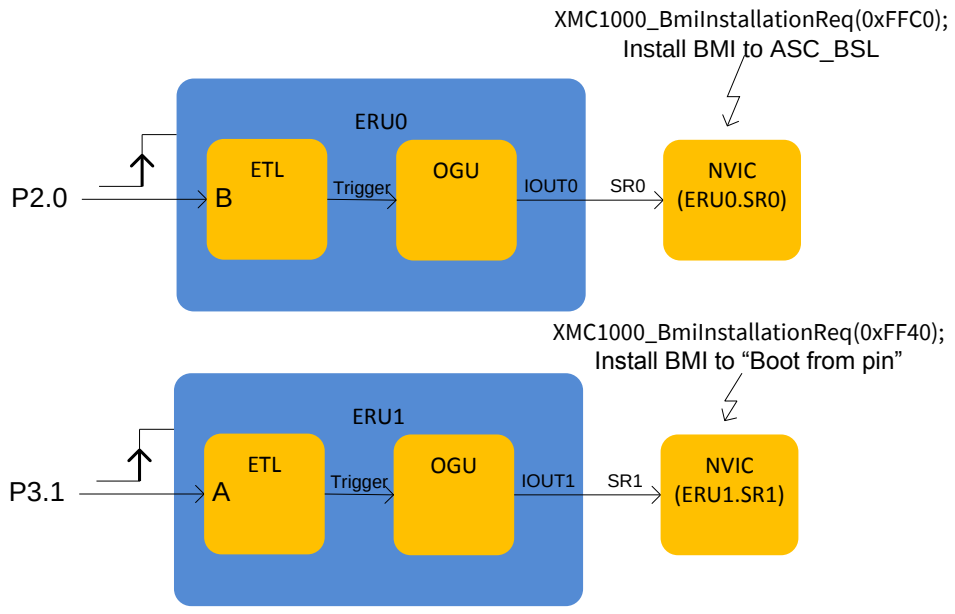
3.1.5 Main function implementation

```
int main(void)
{
  SysTick_Config(SystemCoreClock/10); /* Set up the systick timer at 10 Hz */

  XMC_GPIO_Init(P4_1, &output_config);

  /* Placeholder for user application code.
  while(1U)
  {
  }
}
```

3.2 Setting up the Alternate Boot Mode(ABM)

In this example, two sets of external interrupt events are set up on P2.0 and P3.1. On detection of a rising edge on the pins, an external interrupt is triggered.

- If the event is triggered by P2.0, the BMI value changes to “Boot from BMI: ASC_BSL”
- If the event is triggered by P3.1, the BMI value changes to “Boot from pins”
- Otherwise, P4.0 toggles at an interval of 10 Hz



1. Initially, P2.0 and P3.1 are connected to GND and Systick timer toggles P4.0 at 10Hz.

2. To trigger a change in BMI, an external interrupt is generated when the respective pins are connected to VCC. In the respective ISR, the BMI installation routine is called.
   - P2.0: Changed BMI mode to “Boot from BMI: ASC_BSL”
   - P3.1: Changed BMI mode to “Boot from pin”

3. If “Boot from pin” is selected, to enter to ABM, connect P4.7(VCC) and P4.6 (GND) before a PORST. After power-up/master reset, the ABM mode is entered and runs the ABM user defined.

4. If “Boot from pin” is selected, to enter to User Productive Mode, connect P4.7(GND) and P4.6 (GND) before a PORST. After power-up/master reset, code starts to run from Flash location address stored at 0x10001004.

Figure 17 Changing the BMI to ABM from an external interrupt event

3.2.1 Linker script settings

Define the ABM Header location at the last 32 bytes of the user flash area.

```
57 MEMORY
58 {
59    FLASH(RX)           : ORIGIN = 0x10001000, LENGTH = 0x30000
60    FLASH_ABM_HEADER    : ORIGIN = 0x10032FE0, LENGTH = 0x20
61    SRAM(!RX)           : ORIGIN = 0x20000000, LENGTH = 0x4000
62 }
63
```

Locate the ABM Header at the last 32 bytes of the user flash area.

```
108         . = ALIGN(4);
109         __extab_end = ABSOLUTE(.);
110     } > FLASH
111
112 /*****************************************************************************/
113     .const_abm :
114     {
115        KEEP(*(.abm_header) )
116     } > FLASH_ABM_HEADER
117
118 /*****************************************************************************/
119
```

Figure 18 Modify linker script to locate the ABM header

3.2.2 Macro and variable settings

```
/* XMC lib Project includes: */
#include <xmc_eru.h>
#include <xmc_gpio.h>
#include <xmc_scu.h>

/* Project definitions */
#define ROM_FUNCTION_TABLE_START        (0x00000100)
#define _BmiInstallationReq             (ROM_FUNCTION_TABLE_START + 0x08)

/* Pointer to Request BMI installation routine */
#define XMC1000_BmiInstallationReq      \
                (*((unsigned long (**) (unsigned short)) _BmiInstallationReq))
```

3.2.3 ABM header structure

```
typedef struct ABM_Header
{
  uint32_t MagicKey;            /**< Magic key. Always 0xA5C3E10F */
  uint32_t StartAddress;        /**< Start address of the program to load */
  uint32_t Length;              /**< Length of the program to load. */
} ABM_Header_t;
```

```
ABM_Header_t __attribute__((section(".abm_header")))
ABM_Header =
{
  .MagicKey     = 0xA5C3E10F,   /**< Magic key. Always 0xA5C3E10F */
  .StartAddress = 0x10020001,   /**< Start address of the program to load */
  .Length       = 0x55U,        /**< Length of the program to load. */
};
```

Note: The BX and BLX instructions result in a HardFault exception if bit[0] of Rm is 0.Therefore, the bit[0] of the StartAddress entered to the ABM header structure should always set to 1.

Note: Although the exact program length is not required, the length of program entered to the ABM header should be always be a non-zero value

3.2.4 XMC lib peripheral configuration structure

```
/* XMC GPIO configuration */
XMC_GPIO_CONFIG_t input_config =
{
  .mode = XMC_GPIO_MODE_INPUT_TRISTATE,
  .input_hysteresis = XMC_GPIO_INPUT_HYSTERESIS_STANDARD
};

XMC_GPIO_CONFIG_t output_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
};

/* Event trigger logic configuration - ERU0.0B0 (P2_0) selected */
XMC_ERU_ETL_CONFIG_t ERU0_ETL_Config =
{
  .input_a = (uint32_t)XMC_ERU_ETL_INPUT_A0, /* Event input selection for A(0-3) */
  .input_b = (uint32_t)XMC_ERU_ETL_INPUT_B0, /* Event input selection for B(0-3) */

  .enable_output_trigger = (uint32_t)1,
  .status_flag_mode = (XMC_ERU_ETL_STATUS_FLAG_MODE_t)XMC_ERU_ETL_STATUS_FLAG_MODE_HWCTRL,

  /* Select the edge/s to convert as event */
  .edge_detection = XMC_ERU_ETL_EDGE_DETECTION_RISING,

  /* Select the source for event */
  .output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL0,
  .source = XMC_ERU_ETL_SOURCE_B
};

/* Event Trigger Logic Configuration - ERU1.1A1 (P3_1) selected */
```

```
XMC_ERU_ETL_CONFIG_t ERU1_ETL_Config =
{
  .input_a = (uint32_t)XMC_ERU_ETL_INPUT_A1, /* Event input selection for A(0-3) */
  .input_b = (uint32_t)XMC_ERU_ETL_INPUT_B0, /* Event input selection for B(0-3) */

  .enable_output_trigger = (uint32_t)1,
  .status_flag_mode = (XMC_ERU_ETL_STATUS_FLAG_MODE_t)XMC_ERU_ETL_STATUS_FLAG_MODE_HWCTRL,

  /* Select the edge/s to convert as event */
  .edge_detection = XMC_ERU_ETL_EDGE_DETECTION_RISING,

  /* Select the source for event */
  .output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL1,
  .source = XMC_ERU_ETL_SOURCE_A
};


/* Output Gating Unit Configuration - Gated Trigger Output */
XMC_ERU_OGU_CONFIG_t ERU0_OGU_Config =
{
  .peripheral_trigger         = 0U, /* OGU input peripheral trigger */
  .enable_pattern_detection   = false, /* Enables generation of pattern match event */
  /* Interrupt gating signal */
  .service_request            = XMC_ERU_OGU_SERVICE_REQUEST_ON_TRIGGER,
  .pattern_detection_input    = 0U
};
```

3.2.5 Interrupt service routine function implementation

```
/* Interrupt handler for external trigger interrupt at P2.0 */
void IRQ_Hdlr_3(void)
{
  // BMI_installation routine to set BMI = ASC_BSL; Boot from BMI
  XMC1000_BmiInstallationReq(0xFFC0);
}

/* Interrupt handler for external trigger interrupt at P3.1 */
void IRQ_Hdlr_4(void)
{
  // BMI_installation routine to set BMI = Boot from pin
  XMC1000_BmiInstallationReq(0xFF40);
}
```

```
/* Systick timer to toggle LED on P4_0 at 10Hz interval */
void SysTick_Handler(void)
{
  XMC_GPIO_ToggleOutput(P4_0);
}
```

3.2.6 Main function implementation

```
int main(void)
{
  /* Set up the systick timer at 10Hz */
  SysTick_Config(SystemCoreClock/10);

  /* Sets up the ERU- ETL and OGU for the external trigger event */
  XMC_ERU_ETL_Init(XMC_ERU0, 0, &ERU0_ETL_Config);
  XMC_ERU_OGU_Init(XMC_ERU0, 0, &ERU0_OGU_Config);

  XMC_ERU_ETL_Init(XMC_ERU1, 1, &ERU1_ETL_Config);
  XMC_ERU_OGU_Init(XMC_ERU1, 1, &ERU0_OGU_Config);

  /* Initializes the gpio input and output */
  XMC_GPIO_Init(P2_0, &input_config);
  XMC_GPIO_Init(P3_1, &input_config);
  XMC_GPIO_Init(P4_0, &output_config);

  /* Enable the interrupt */
  XMC_SCU_SetInterruptControl(3, XMC_SCU_IRQCTRL_ERU0_SR0_IRQ3); //Only for XMC140x
  XMC_SCU_SetInterruptControl(4, XMC_SCU_IRQCTRL_ERU1_SR1_IRQ4); //Only for XMC140x

  NVIC_EnableIRQ(3U);
  NVIC_EnableIRQ(4U);

  /* Placeholder for user application code. */
  while(1U)
  {
  }
}
```

Revision history

Major changes since the last revision

Page or reference	Description of change
-	First release

Edition 2015-11-30

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN_201511_PL30_005

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.