XMC1000

32-bit Microcontroller Series for Industrial Applications

Tooling Guide for XMC1000

AP32277

Application Note

Scope and purpose

This document is the latest tooling guide for XMC1000 ASC Boot Loader.

In chapter 1, Boot Mode Index (BMI) concept is described, which is different from using pin configuration to determine boot up mode.

In chapter 2, the ASC Bootstrap Loader protocol of XMC1000 is provided.

In chapter 3, an ASC Programmer with source code is provided to demonstrate how to perform Flash erase, programming, Flash readback and change BMI on the XMC1000 device.

Applicable Products

- XMC1000 Microcontrollers Family
- XMC1100/XMC1200/XMC1300 Boot Kit

References

Infineon: DAVE™, http://www.infineon.com/DAVE

Infineon: XMC Family, http://www.infineon.com/XMC

The example code supplied can be downloaded from Infineon XMC1000 Microcontroller Application Notes website.

Table of Contents

1 Boot Mode Index for Booting Up

A microcontroller would normally have a few boot mode selection pins that determine the Boot Mode after power-on reset. The XMC1000 family of products however are low pin count devices, so the use of multiple pins just for Boot up mode selection is not desirable.

Instead, the XMC1000 microcontroller depends on the Boot Mode Index (BMI) value to determine the boot mode after power-on reset. The BMI value is stored in the flash configuration sector 0 (CS0).

Figure 1 illustrates the various start-up modes which the XMC1000 device will enter after the Start-up Software has read the BMI value. The default start-up mode when the device is delivered from the factory is ASC BSL mode.

Note: For the XMC1100, XMC1200 and XMC1300 Boot Kits, the XMC1000 devices are programmed to User mode with debug enabled, so that the application program will start to run after power-up.

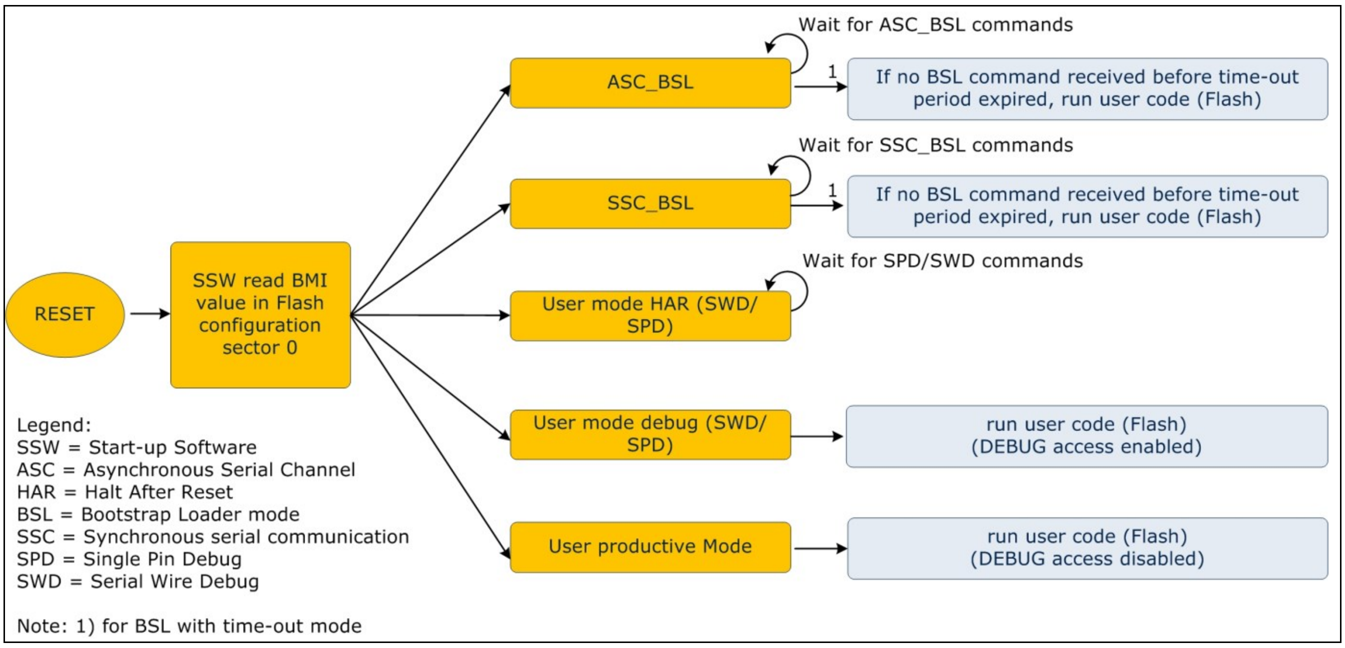

Wait for ASC_BSL commands
ASC_BSL
1
If no BSL command received before time-out period expired, run user code (Flash)
Wait for SSC_BSL commands
SSC_BSL
1
If no BSL command received before time-out period expired, run user code (Flash)
Wait for SPD/SWD commands
RESET
SSW read BMI value in Flash configuration sector 0
User mode HAR (SWD/ SPD)
User mode debug (SWD/ SPD)
run user code (Flash) (DEBUG access enabled)
User productive Mode
run user code (Flash) (DEBUG access disabled)
Legend:
SSW = Start-up Software
ASC = Asynchronous Serial Channel
HAR = Halt After Reset
BSL = Bootstrap Loader mode
SSC = Synchronous serial communication
SPD = Single Pin Debug
SWD = Serial Wire Debug
Note: 1) for BSL with time-out mode

Figure 1 Boot ROM operating mode after power-on reset

1.1 Boot Mode Use Cases

Table 1 Boot Mode Use Cases

Mode	**Use Case**
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 Flash Programmer to download Flash erasing and programming routine using SSC protocol to the XMC1000 device.

Mode	**Use Case**
User mode with debug enabled and HAR (SWD) or User mode HAR (SWD)	• Serial Wire Debug is ARM propriety debug protocol for ARM Cortex^{TM} 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) orUser mode debug (SWD)	• User code will run after power-up. • Supports debugging using serial wire debug protocol.
User mode with debug enabled (SPD) orUser 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.

1.2 Programming / Debugging Pin

Two sets of pins/channels are available for you to choose from to use for programming or debugging the XMC1000 device.

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.

The selection of a port pin depends on the BMI value.

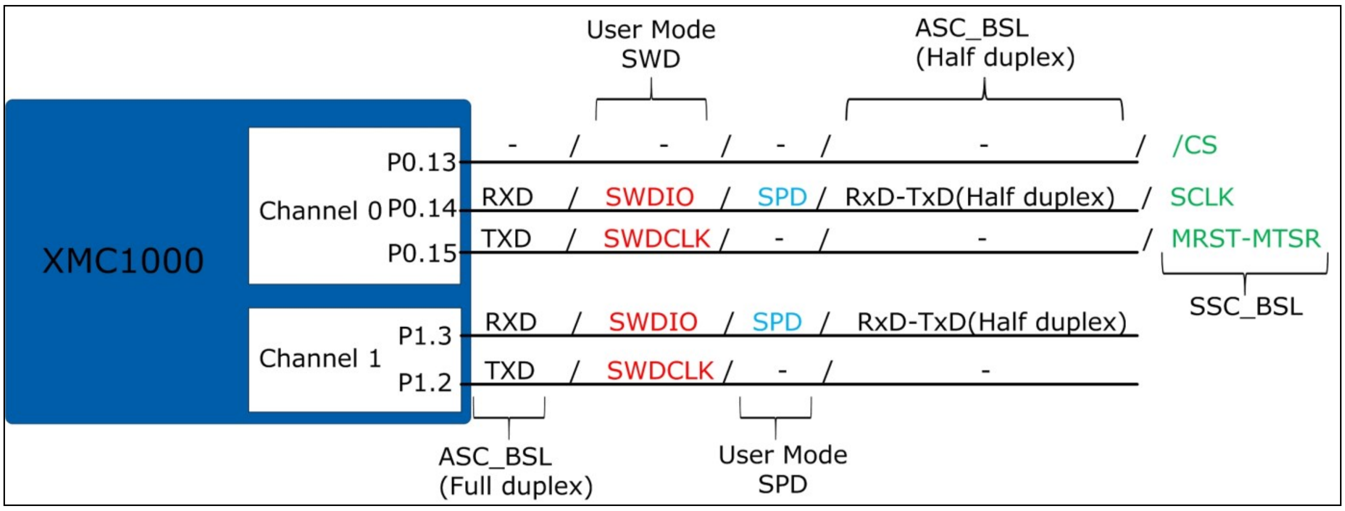
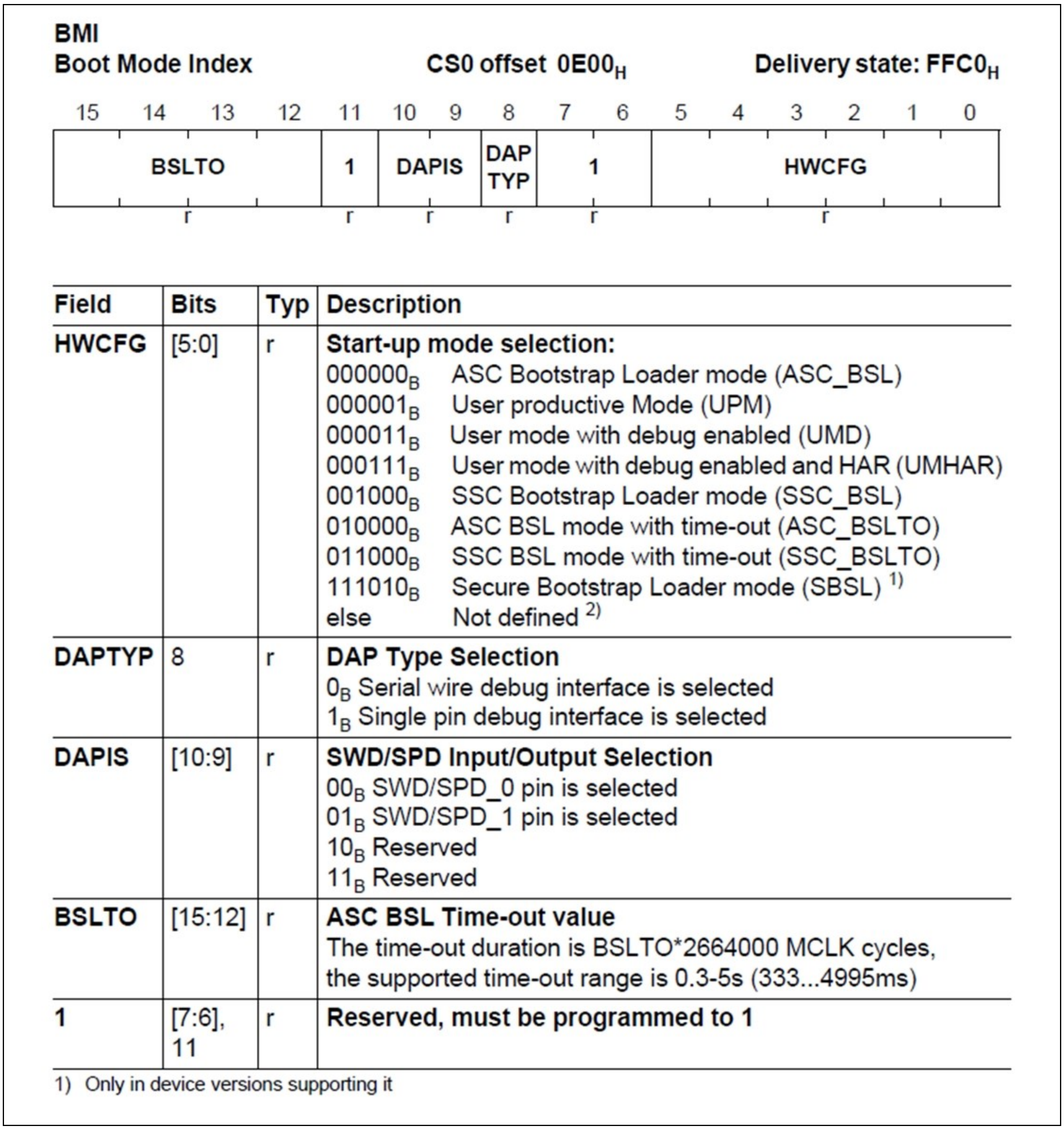


Figure 2 Pins used to communicate with the XMC1000 device in each Boot mode

Please refer to **Table 2**, **Table 3** and **Table 4** for more detail and further understanding of the selection process.

1.3 Boot Mode Index

The XMC1000 device BMI value is read from memory location 0x10000E00.



BMI
Boot Mode Index **CS0 offset 0E00$_H$** **Delivery state: FFC0$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSLTO				1	DAPIS		DAP TYP	1		HWCFG					
r				r	r		r	r		r					

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 3 Boot Mode Index bit field description

As shown in **Figure 3**, if the BMI value is 0xFFC0, it means that the device is in ASC Bootstrap Loader mode.

1.4 Description of the Boot Modes

The respective Boot modes are described in this section.

1.4.1 ASC_BSL - ASC Bootstrap Loader mode

After power-up/master reset, the device will wait 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 their own monitor program into XMC1000‟s SRAM, starting at address 0x20000200. After the monitor program is received and stored, the Start-up Software will run the user's monitor program at SRAM address 0x20000200, which can then be used to communicate with the Host PC for BMI installation, flash erasing, programming and verification of code/data download.

The ASC_BSL can communicate with the Host PC in full duplex or half duplex mode. This depends on the Header Byte which the Host PC sends to the XMC1000 device during the ASC_BSL handshaking.

ASC_BSL also supports a time-out option. As shown in **Figure 1**, 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
0x12	0xFFC0	Half Duplex	P1.3	P1.3
0x6C	0xFFC0	Full Duplex	P0.14	P0.15
0x6C	0xFFC0	Full Duplex	P1.3	P1.2
0x12	0xFFD0 (time-out = 4995ms @ MCLK = 8 MHz)	Half Duplex	P0.14	P0.14
0x12	0xFFD0 (time-out = 4995ms @ MCLK = 8 MHz)	Half Duplex	P1.3	P1.3
0x6C	0xFFD0 (time-out = 4995ms @ MCLK = 8 MHz)	Full Duplex	P0.14	P0.15
0x6C	0xFFD0 (time-out = 4995ms @ MCLK = 8 MHz)	Full Duplex	P1.3	P1.2

1.4.2 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 User Mode (debug) manner. The user can access the full debug functionality using SWD/SPD to establish 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 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

1.4.3 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

1.4.4 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, access (read/write) 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 still possible to change the BMI value in this mode, 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. Your application software can call this routine to change the BMI value under user-defined conditions, for example via an external interrupt or via GPIO pin latch values.

For code protection purposes, if the current BMI value is User productive Mode and the new requested BMI value is NOT User productive Mode, 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.

A code snippet on how to use an external interrupt to trigger the "Request BMI installation" routine is provided in section **1.5.1**.

1.5 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.

Note: A Master reset will be executed when the BMI value is updated. Note however, that 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 user is changing BMI to anyway 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.

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 using Memtool v4.5 to change the BMI value is also provided.

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

Here we provide example code which uses an external interrupt (rising edge trigger) to call the "Request BMI installation routine (new BMI)". The code uses P2.0 to trigger the Event Request Unit (ERU) which triggered Interrupt Node ERU0.SR0. Then, in the interrupt routine, it will call the XMC1000_BmiInstallationReq() routine with the desired BMI value as the parameter to pass.

```
/* Constants definitions */
// Start address of the ROM function table
#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))

void ERU0_Init(void)
{
// Enable IRQ3 = ERU0.SR0
      NVIC_EnableIRQ(3);
// Choose P2.0 as ERU0 input 0B0
      ERU0->EXISEL      &= 0xFFF3;

// Rising edge trigger
// Output to OGU0 of ERU0
      ERU0->EXICON[0] |= 0x0107;

// Enable event detection
      ERU0->EXOCON[0] |= 0x1024;
}

void ERU0_0_IRQHandler(void)
{
// Call the BMI_installation routine to set BMI = ASC_BSL
      XMC1000_BmiInstallationReq(0xFFC0);
}

int main(void)
{
      P1_0_set_mode(OUTPUT_PP_GP);
// set P2.0 as input pin, disable its ADC function
      PORT2->PDISC &= 0xFFFE;
      P2_0_set_mode(INPUT_PD);
// initialize ERU0 to set P2.0 as external interrupt
      ERU0_Init();

      while(1)
      {
            P1_0_toggle();    // toggle LED
            Delay100US(500);  // tune minimum and maximum flashing time
            ...
      }
      return 0;
}
```

1.5.2 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 6 Memtool v4.5 connection configuration

Hardware	Driver	Connection Methods
1) XMC1000 Boot Kit	Virtual COM Port	ASC_BSL
2) DAP miniwiggler v2.0	DAS	ASC_BSL, SWD, SPD

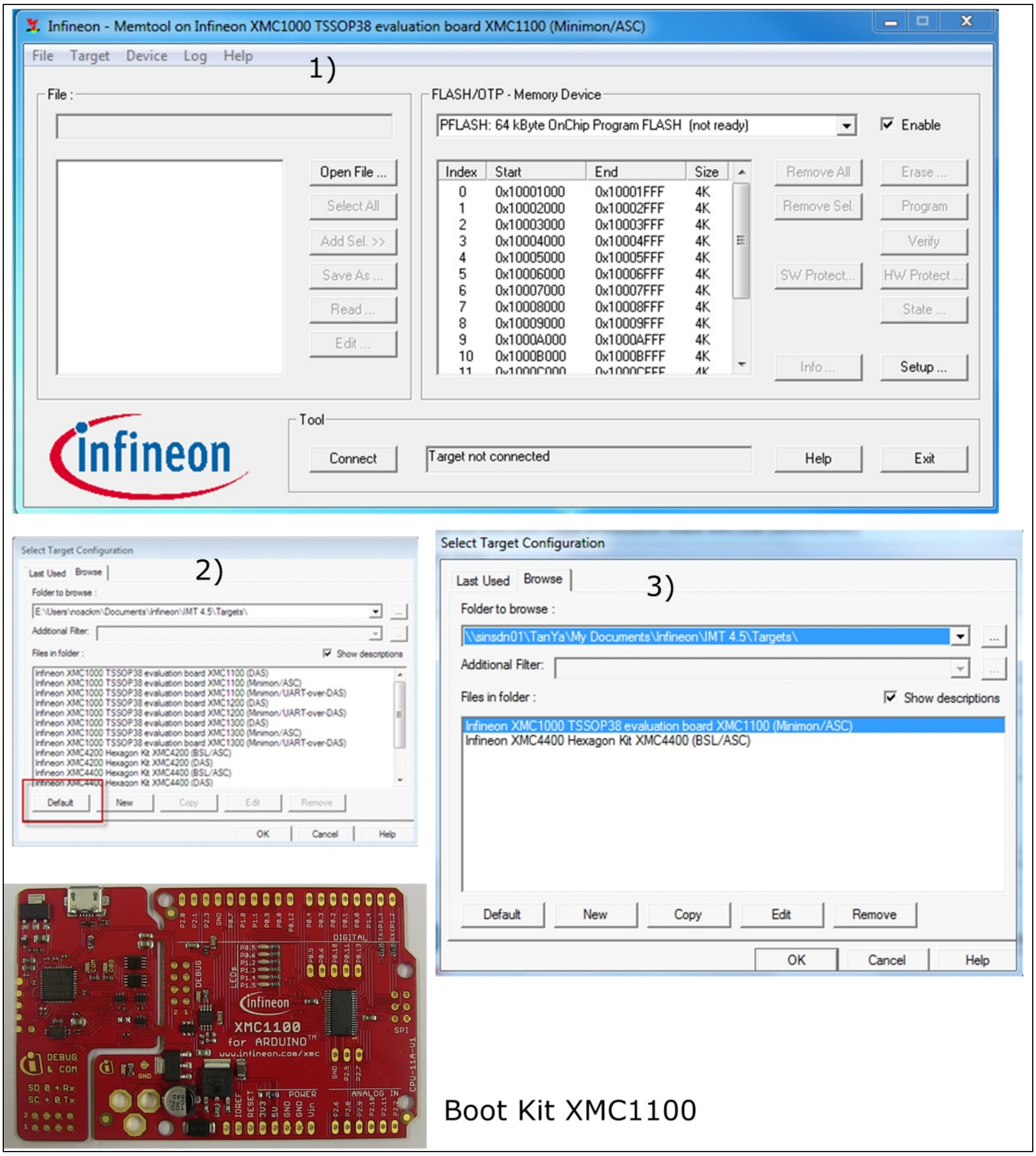
1.5.2.1 Memtool connection to XMC1000 using Virtual Com Port via XMC1000 CPU Card

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

The XMC1000 Boot Kit has a CPU Card with an on-board COM and SEGGER J-Link Debugger. Memtool uses the COM port for ASC_BSL communication with the XMC1000 device. So, 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 2**.

Note: The XMC1000 Boot Kit can be ordered online from: www.infineon.com/xmc1000 -> XMC1000 Kits.



Infineon - Memtool on Infineon XMC1000 TSSOP38 evaluation board XMC1100 (Minimon/ASC)

File Target Device Log Help

File :

FLASH/OTP - Memory Device

PFLASH: 64 kByte OnChip Program FLASH (not ready) ☑ Enable

Index	Start	End	Size
0	0x10001000	0x10001FFF	4K
1	0x10002000	0x10002FFF	4K
2	0x10003000	0x10003FFF	4K
3	0x10004000	0x10004FFF	4K
4	0x10005000	0x10005FFF	4K
5	0x10006000	0x10006FFF	4K
6	0x10007000	0x10007FFF	4K
7	0x10008000	0x10008FFF	4K
8	0x10009000	0x10009FFF	4K
9	0x1000A000	0x1000AFFF	4K
10	0x1000B000	0x1000BFFF	4K
11	0x1000C000	0x1000CFFF	4K

Tool

Target not connected

Select Target Configuration

Last Used Browse

Folder to browse : E:\Users\noackm\Documents\Infineon\IMT 4.5\Targets\

Additional Filter:

Files in folder : ☑ Show descriptions

```
Infineon XMC1000 TSSOP38 evaluation board XMC1100 (DAS)
Infineon XMC1000 TSSOP38 evaluation board XMC1100 (Minimon/ASC)
Infineon XMC1000 TSSOP38 evaluation board XMC1100 (Minimon/UART-over-DAS)
Infineon XMC1000 TSSOP38 evaluation board XMC1200 (DAS)
Infineon XMC1000 TSSOP38 evaluation board XMC1200 (Minimon/UART-over-DAS)
Infineon XMC1000 TSSOP38 evaluation board XMC1300 (DAS)
Infineon XMC1000 TSSOP38 evaluation board XMC1300 (Minimon/ASC)
Infineon XMC1000 TSSOP38 evaluation board XMC1300 (Minimon/UART-over-DAS)
Infineon XMC4200 Hexagon Kit XMC4200 (BSL/ASC)
Infineon XMC4200 Hexagon Kit XMC4200 (DAS)
Infineon XMC4400 Hexagon Kit XMC4400 (BSL/ASC)
Infineon XMC4400 Hexagon Kit XMC4400 (DAS)
```

Select Target Configuration

Last Used Browse

Folder to browse : \\sinsdn01\TanYa\My Documents\Infineon\IMT 4.5\Targets\

Additional Filter:

Files in folder : ☑ Show descriptions

```
Infineon XMC1000 TSSOP38 evaluation board XMC1100 (Minimon/ASC)
Infineon XMC4400 Hexagon Kit XMC4400 (BSL/ASC)
```

Figure 4 Setup Memtool to connect to XMC1000 Boot kit using virtual com port

Procedure

1. Open Memtool v4.5.
2. Click Target ->change of GUI 1).
3. Click default button of GUI 2), a list of XMC1000 boards is available for selection.

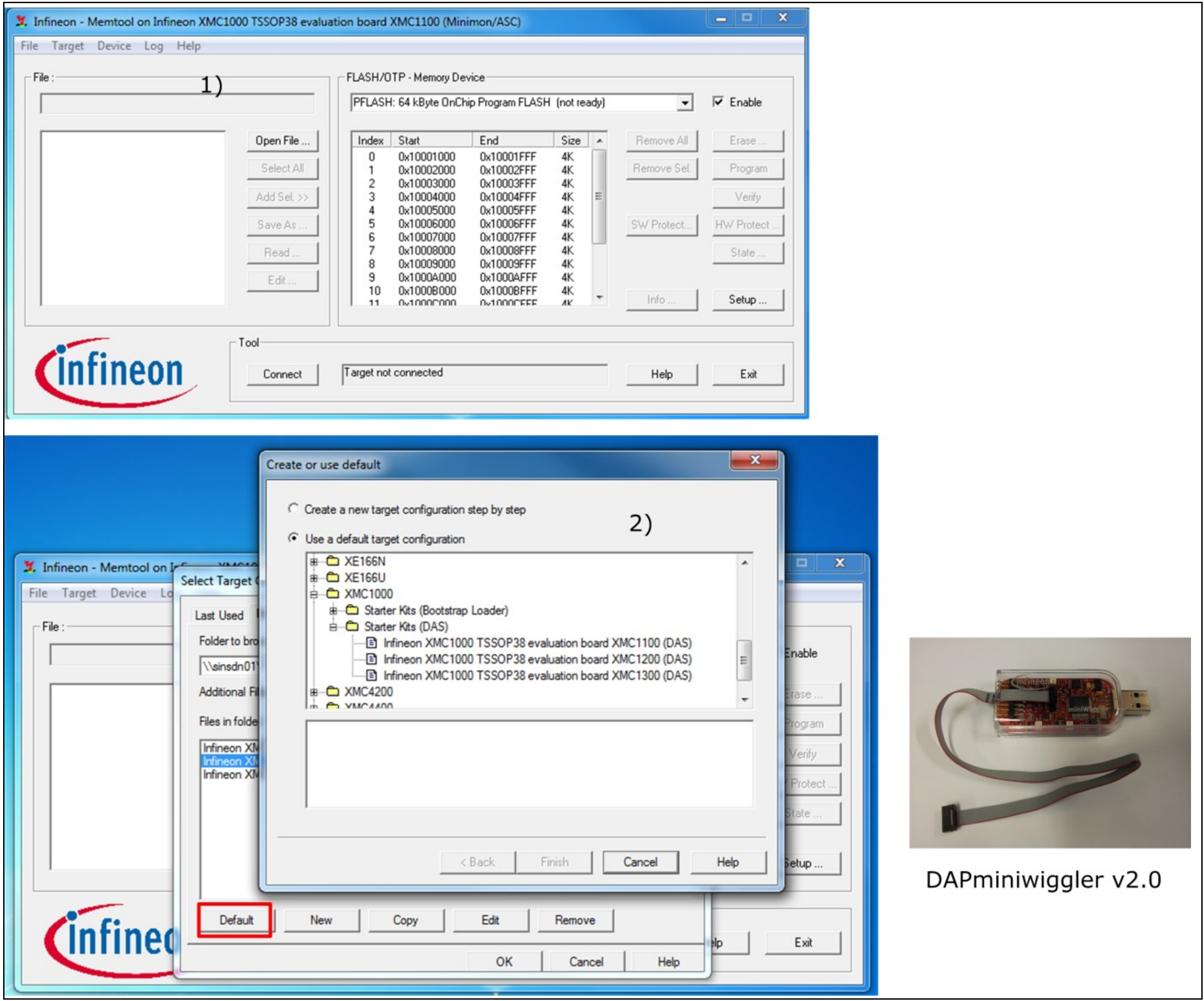
- If the BMI of the XMC1200 Boot Kit is ASC_BSL, then choose Infineon XMC1200 TSSOP38 evaluation board XMC1200 (Minimon/ASC).

4. Close GUI 3), connect the USB cable to the XMC1200 Boot Kit board and click Connect button of GUI 1).

1.5.2.2 Memtool connection to XMC1000 using DAS via DAP miniWiggler

This mode of connection can be used when 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.



PFLASH: 64 kByte OnChip Program FLASH (not ready)

Index	Start	End	Size
0	0x10001000	0x10001FFF	4K
1	0x10002000	0x10002FFF	4K
2	0x10003000	0x10003FFF	4K
3	0x10004000	0x10004FFF	4K
4	0x10005000	0x10005FFF	4K
5	0x10006000	0x10006FFF	4K
6	0x10007000	0x10007FFF	4K
7	0x10008000	0x10008FFF	4K
8	0x10009000	0x10009FFF	4K
9	0x1000A000	0x1000AFFF	4K
10	0x1000B000	0x1000BFFF	4K

Target not connected

( ) Create a new target configuration step by step
(•) Use a default target configuration

```
XE166N
XE166U
XMC1000
    Starter Kits (Bootstrap Loader)
    Starter Kits (DAS)
        Infineon XMC1000 TSSOP38 evaluation board XMC1100 (DAS)
        Infineon XMC1000 TSSOP38 evaluation board XMC1200 (DAS)
        Infineon XMC1000 TSSOP38 evaluation board XMC1300 (DAS)
XMC4200
```

Figure 5 Setup Memtool to connect to XMC1200 target board using DAS

Procedure

1. Open Memtool v4.5.
2. GUI 1) appears, click Target ->change.
3. Click default button in GUI 2), a list of XMC1000 boards is available for selection.

- If using the XMC1200 target board, then choose Infineon XMC1000 TSSOP38 evaluation board XMC1200 (DAS).

4. Close GUI 2), plug the DAP miniWiggler to the USB port of the Host PC and connect the 10pin connector of the DAP miniWiggler to the XMC1200 target board as shown in figure below.
5. Click Connect button of GUI 1). Now, Memtool should be connected to the XMC1200 device.

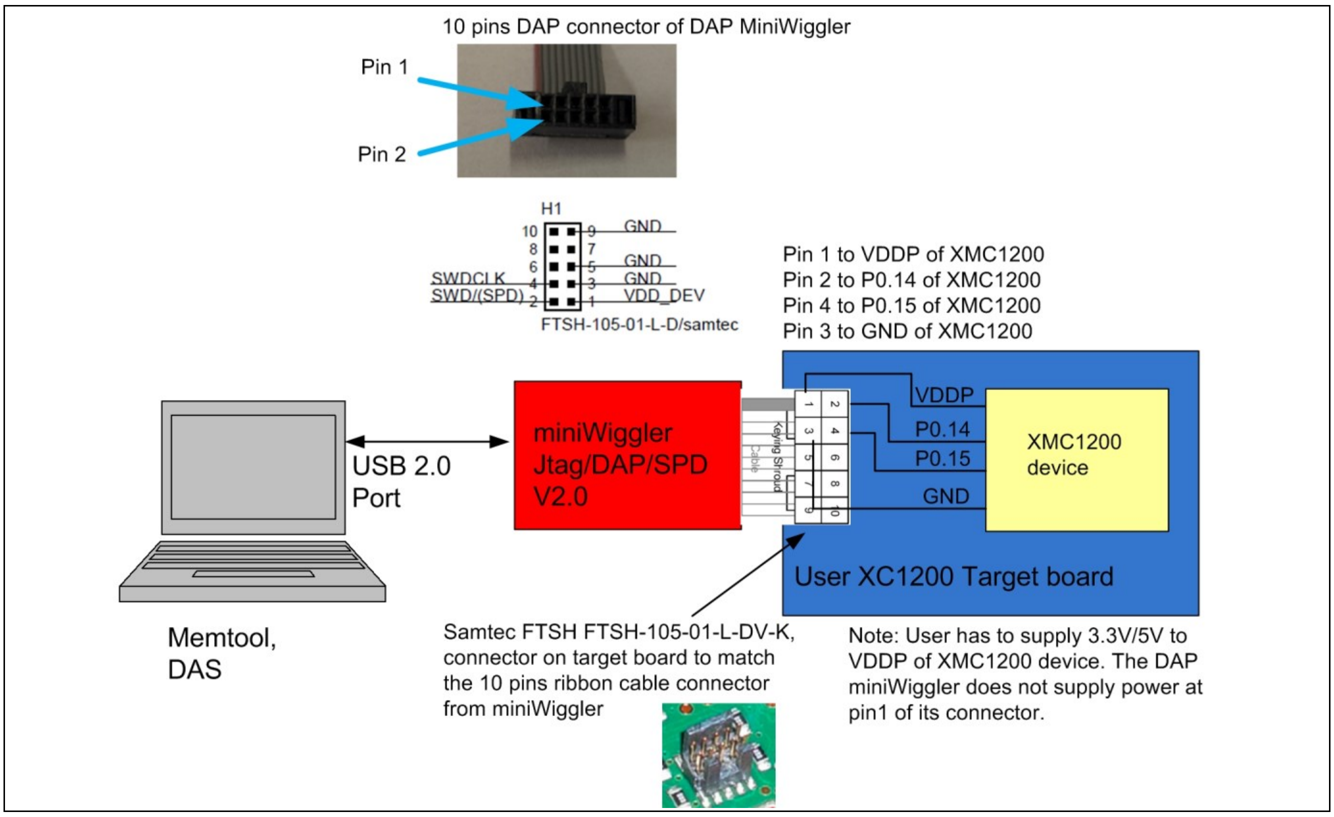
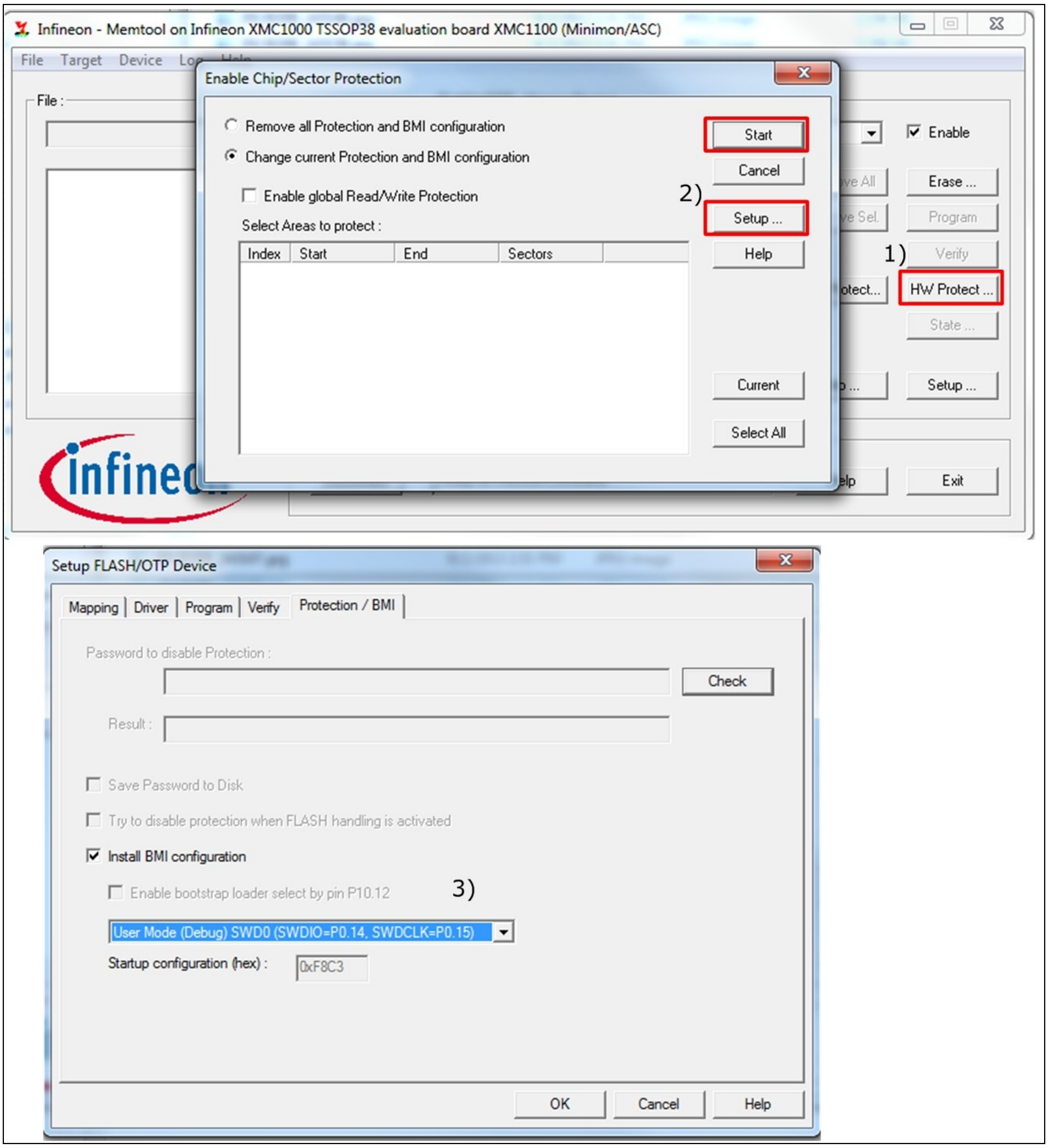

10 pins DAP connector of DAP MiniWiggler
Pin 1
Pin 2
H1
GND
GND
GND
VDD_DEV
SWDCLK
SWD/(SPD)
FTSH-105-01-L-D/samtec
Pin 1 to VDDP of XMC1200
Pin 2 to P0.14 of XMC1200
Pin 4 to P0.15 of XMC1200
Pin 3 to GND of XMC1200
USB 2.0 Port
miniWiggler Jtag/DAP/SPD V2.0
Keying Shroud
Cable
VDDP
P0.14
P0.15
GND
XMC1200 device
User XC1200 Target board
Memtool, DAS
Samtec FTSH FTSH-105-01-L-DV-K, connector on target board to match the 10 pins ribbon cable connector from miniWiggler
Note: User has to supply 3.3V/5V to VDDP of XMC1200 device. The DAP miniWiggler does not supply power at pin1 of its connector.

Figure 6 Connection diagram between DAP miniWiggler v2.0 and XMC1200 target board

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



Remove all Protection and BMI configuration
(●) Change current Protection and BMI configuration
☐ Enable global Read/Write Protection
Select Areas to protect :

Index	Start	End	Sectors

Password to disable Protection :
Result :
☐ Save Password to Disk
☐ Try to disable protection when FLASH handling is activated
☑ Install BMI configuration
☐ Enable bootstrap loader select by pin P10.12
User Mode (Debug) SWD0 (SWDIO=P0.14, SWDCLK=P0.15)
Startup configuration (hex) : 0xF8C3

Figure 7 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.

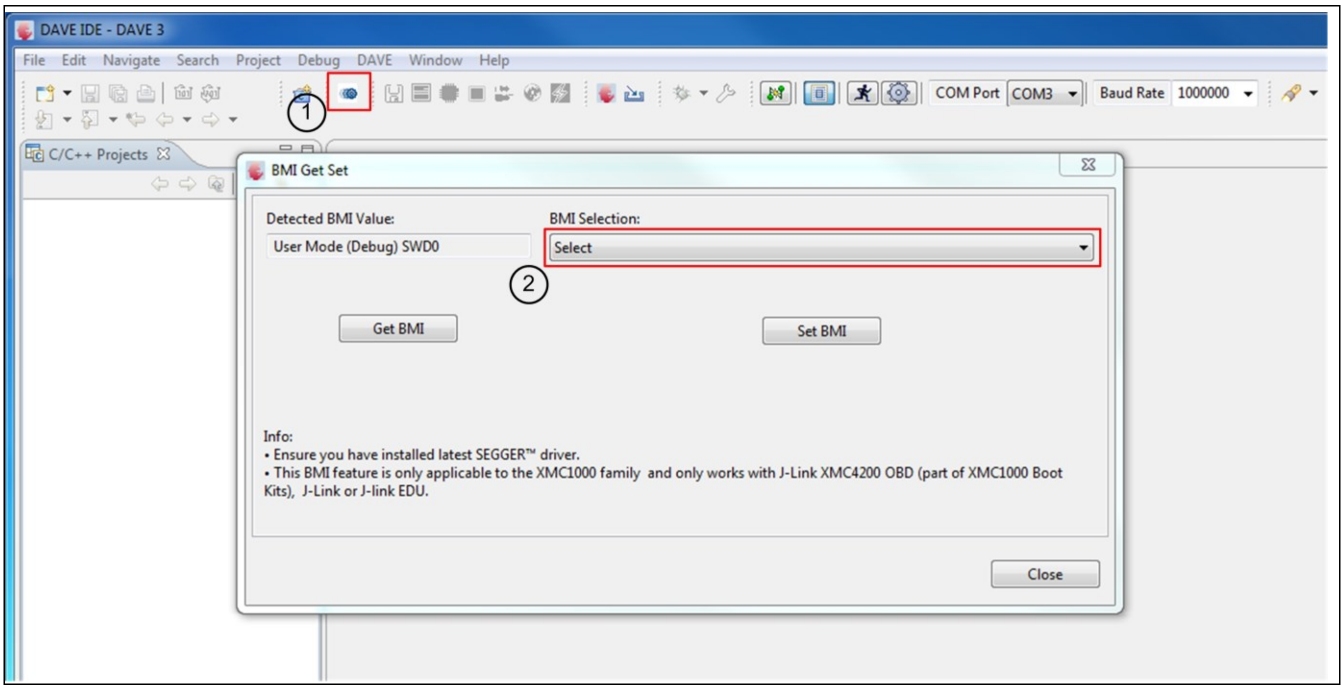
After connection to the XMC1000 device, to change the BMI value of the XMC1000 device:

1. Click the HW Protect… button of the main menu GUI 1).

2. Click the Setup… button of GUI 2) of **Figure 7**
3. From the pull-down menu of GUI 3) of **Figure 7**, user can choose the BMI value that he wanted.
4. Clicks the Start button of **Figure 7** GUI 2) and the BMI of the XMC1000 device is changed.

1.5.3 Using DAVE^{TM} v3.1.10 'BMI Get Set' feature to change BMI value

There is a BMI handling feature in DAVE^{TM} v3.1.10 release. This feature works with J-Link XMC4200 OBD (part of XMC1000 Boot Kit), J-Link or J-Link EDU.



Detected BMI Value: User Mode (Debug) SWD0

BMI Selection: Select

Info:
• Ensure you have installed latest SEGGER™ driver.
• This BMI feature is only applicable to the XMC1000 family and only works with J-Link XMC4200 OBD (part of XMC1000 Boot Kits), J-Link or J-link EDU.

Figure 8 Illustration of 'BMI Get Set' feature in DAVE^{TM} v3.1.8 and above

First, the J-Link debugger needs to connect to the XMC1000 device. Next, launch the DAVE^{TM} v3.1.10 IDE and clicks on the 'BMI Get Set' button indicated in (1). The 'BMI Get Set' dialog box will pop out. If user wants to write a new BMI value to the XMC1000 device, then choose the desire BMI value from the pull-down menu indicated in (2). Follow by clicking the 'Set BMI' button below. The chosen BMI value will write to the XMC1000 device, follow by a system reset to the XMC1000 device.

If user wants to read the BMI value of the XMC1000 device, then the 'Get BMI' button should be clicked. The BMI value will be read out from XMC1000 device and output at the 'Detected BMI Value' field.

2 ASC Loader

The XMC1000 family supports both Asynchronous Serial Interface (ASC) BSL and Synchronous Serial Channel (SSC) BSL. In this chapter we will demonstrate Bootstrap Loading using the ASC interface only.

Using the UART protocol, the Bootstrap Loader allows the Host PC to download user code to SRAM, starting at address 0x20000200. Once the last code byte is received and stored in SRAM, the device will run the user code.

Flash erasing, programming and a routine to change the Boot Mode Index (BMI) for example, could be embedded in this code. The user can communicate with the downloaded code via the UART protocol for programming, erasing and verifying the XMC1000 Flash.

If the XMC1000 device's BMI value is programmed to ASC Bootstrap Loader mode, then after power-up both USIC channel 0 and 1 are configured to ASC mode, 8 data bits, 1 stop bit, no parity and ready for UART communication. The received header byte will determine whether the UART communication is in full-duplex or half-duplex mode.

Note: The factory delivered XMC1000 device BMI value is set to ASC Bootstrap Loader mode by default.

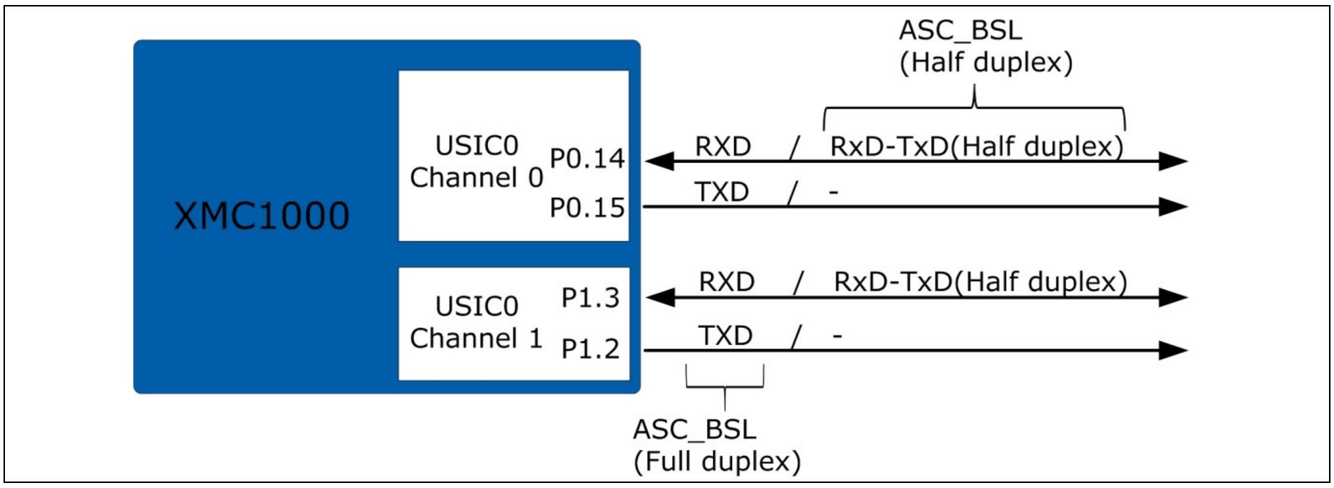


Figure 9 XMC1000 Pin usage for ASC Bootstrap Loader mode

There are two type of ASC Bootstrap Loader mode entry sequence:

- Standard ASC Bootstrap Loader mode
- Enhanced ASC Bootstrap Loader mode
  - The Enhanced ASC Bootstrap Loader mode supports a much higher baud rate than Standard mode, as shown in **Table 7**.

The flow in both modes consists of 2 stages.

5. Baud rate detection and mode selection sequence.
6. Download sequence.

Table 7 Supported Baud Rates

MCLK(MHz)	Standard baud rates supported with ASC Bootstrap Loader mode (kHz)		Maximum baud rate supported with Enhanced ASC Bootstrap Loader mode (MHz)
	Minimum	Maximum	
8	1.2	28.8	0.999
16	2.4	57.6	1.998
32	4.8	115.2	3.996

2.1 Stage 1: Baud rate detection and mode selection

The interaction between the XMC1000 ASC Bootstrap Loader and the Host is via a handshake protocol and the request/acknowledge/data bytes defined in below.

Table 8 Handshake protocol data definition

Name	Byte Length	Value	Description
BSL_ASC_F	1	0x6C	Header requesting full duplex ASC mode with the current baud rate.
BSL_ASC_H	1	0x12	Header requesting half duplex ASC mode with the current baud rate.
BSL_ENC_F	1	0x93	Header requesting full duplex ASC mode with a request to switch the baud rate.
BSL_ENC_H	1	0xED	Header requesting half duplex ASC mode with a request to switch the baud rate.
BSL_STEP	2	0xXXX	10-bit value (LSB aligned) to be programmed into selected USIC channel's FDR.STEP bit field. Most significant 6 bits should contain all 0.
BSL_BR_OK	1	0XF0	Final baud rate is established in enhanced ASC Bootstrap Loader mode.
BSL_ID	1	0x5D	Start and header bytes are received, baud rate is established.
BSL_ENC_ID	1	0xA2	Start and header bytes are received in enhanced ASC Bootstrap Loader mode, initial baud rate is established.
BSL_PDIV	2	0xXXX	10-bit value (LSB aligned) containing the selected USIC channel's BRG.PDIV bit field value. Most significant 6 bits should contain all 0.
BSL_BR_OK	1	0xF0	Final baud rate is established in enhanced ASC Bootstrap Loader mode.
BSL_OK	1	0x01	Data received is OK.
BSL_NOK	1	0x02	Failure encountered during data reception.

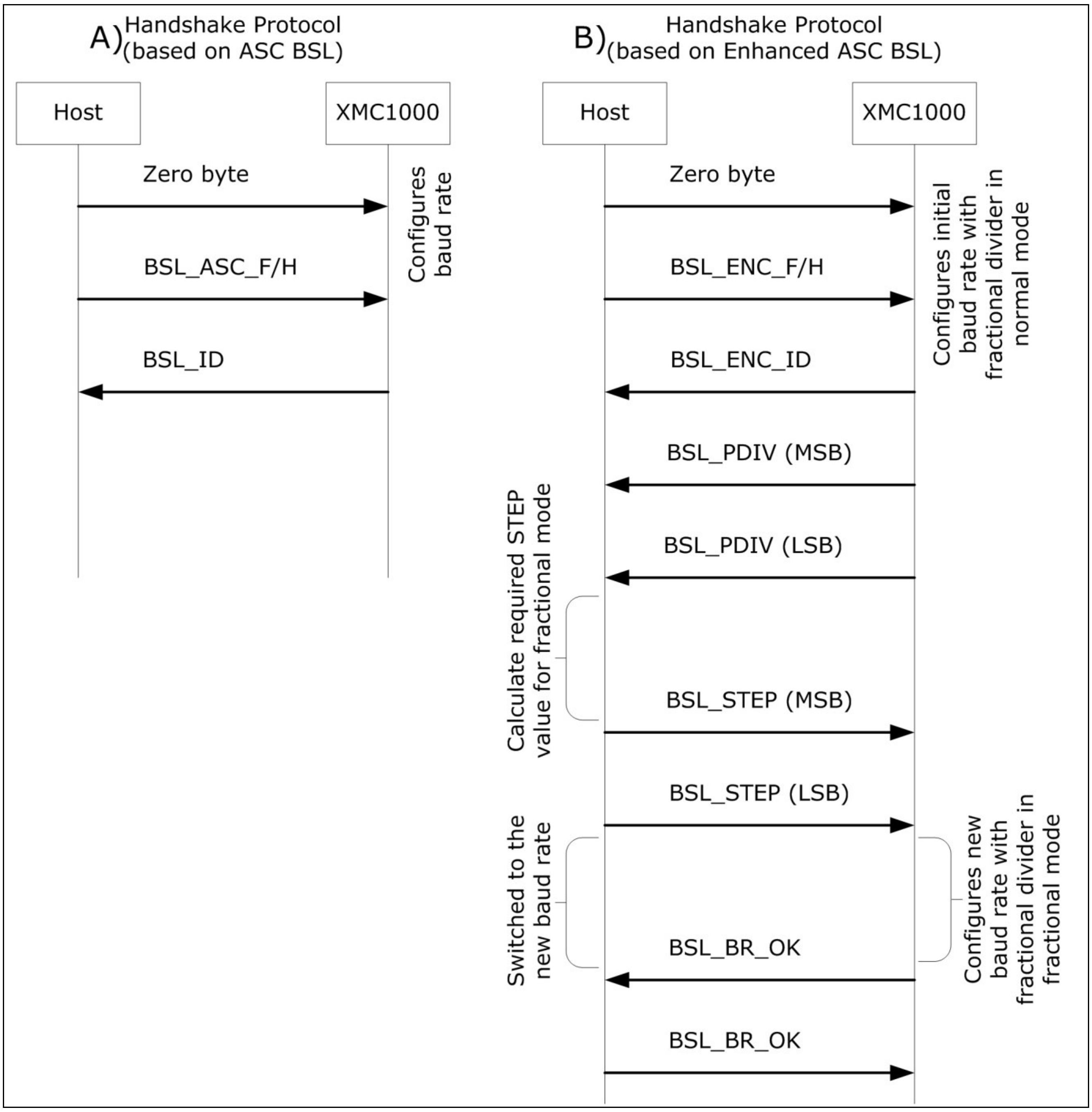

A) Handshake Protocol (based on ASC BSL)
Host
XMC1000
Zero byte
BSL_ASC_F/H
BSL_ID
Configures baud rate
B) Handshake Protocol (based on Enhanced ASC BSL)
Host
XMC1000
Zero byte
BSL_ENC_F/H
BSL_ENC_ID
Configures initial baud rate with fractional divider in normal mode
BSL_PDIV (MSB)
BSL_PDIV (LSB)
Calculate required STEP value for fractional mode
BSL_STEP (MSB)
BSL_STEP (LSB)
Switched to the new baud rate
Configures new baud rate with fractional divider in fractional mode
BSL_BR_OK
BSL_BR_OK

Figure 10 Handshake protocol for XMC1000 AA step device

For XMC1000 AB step device, there is only a small change in the Handshake protocol in enhanced ASC BSL mode as shown in Figure 11. The AA step device will reply BSL_BR_OK byte in final/new baudrate rate while AB step device will reply BSL_BR_OK byte in initial baudrate rate.

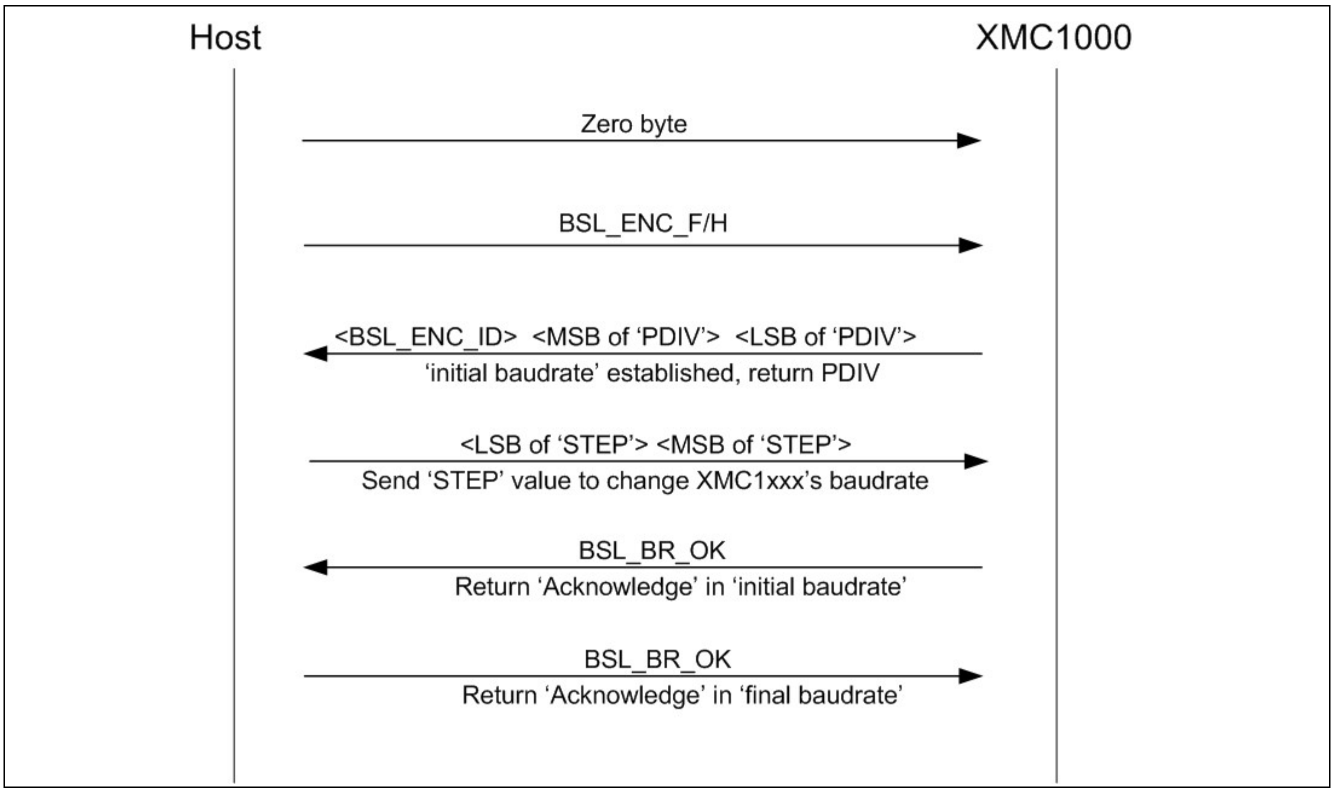

Host
XMC1000
Zero byte
BSL_ENC_F/H
<BSL_ENC_ID> <MSB of 'PDIV'> <LSB of 'PDIV'>
'initial baudrate' established, return PDIV
<LSB of 'STEP'> <MSB of 'STEP'>
Send 'STEP' value to change XMC1xxx's baudrate
BSL_BR_OK
Return 'Acknowledge' in 'initial baudrate'
BSL_BR_OK
Return 'Acknowledge' in 'final baudrate'

Figure 11 Handshake protocol for XMC1000 AB step device

Standard mode

In the standard ASC Bootstrap Loader mode, the XMC1000 device is able to calculate the baud rate based on the zero byte received from the Host. Whether to operate in full duplex or half duplex mode is determined by whether the header byte BSL_ASC_H or BSL_ASC_F is received from the Host.

Enhanced mode

When the Enhanced ASC Bootstrap Loader mode is selected, the XMC1000 device has to communicate in enhanced ASC Bootstrap Loader mode full duplex or half-duplex when it receives the BSL_ENC_F/H header byte. The XMC1000 device will transmit the 10-bit PDIV value to the Host.

The Host can calculate the MCLK that the XMC1000 device is running based on the formula:

- MCLK = Initial Baud Rate x (PDIV + 1) x 8

Example:

Initial Baud Rate = 19200 bps, PDIV = 0x00 0x33 = 0x0033 = 51.

MCLK = 19200 x 52 x 8 = 7.987200MHz ~ 8MHz

- Select a new baud rate based on the MCLK calculated with reference to Supported Baud Rates Table.
- Calculate the scaling factor; i.e. ratio of the new baud rate to the initial baud rate. Some examples are given in Scaling Factor Examples Table shown below. The scaling factor is rounded to the nearest integer in order to meet a frequency deviation of +/- 3%.

Table 9 Scaling Factor Examples

Initial standard baud rate (kHz)	Targeted higher baud rate (kHz)	Scaling factor rounded to the nearest integer
9.6	1500	156
19.2	1500	78
57.6	1500	26
115.2	1500	13

- Calculate the 10-bit value to be written into the STEP bit field of the USIC0_CHy_FDR register through the formula: STEP (in fractional divided mode) = (1024 x Scaling Factor) / (PDIV + 1)

Example:

Initial Baud Rate = 19200 bps, target baud rate = 256000 bps, PDIV = 51.

STEP = 1024 x (256000/19200)/52 = 262.5 ~ 263 = 0x0107 = 0x01 0x07

- The host send the 10-bit STEP value to the XMC1000 device
- Wait until the BSL_BR_OK is received from the XMC1000 device using the new baud rate.
- Echo the BSL_BR_OK back to the device using the new baud rate.

2.2 Stage 2: Download sequence

After the baud rate has been detected/configured and channel/mode (full/half duplex) selected, the ASC Bootstrap Loader waits for the 4 bytes describing the length of the application from the Host. The least significant byte is received first. This download sequence is the same for both standard and Enhanced ASC Bootstrap Loader mode.

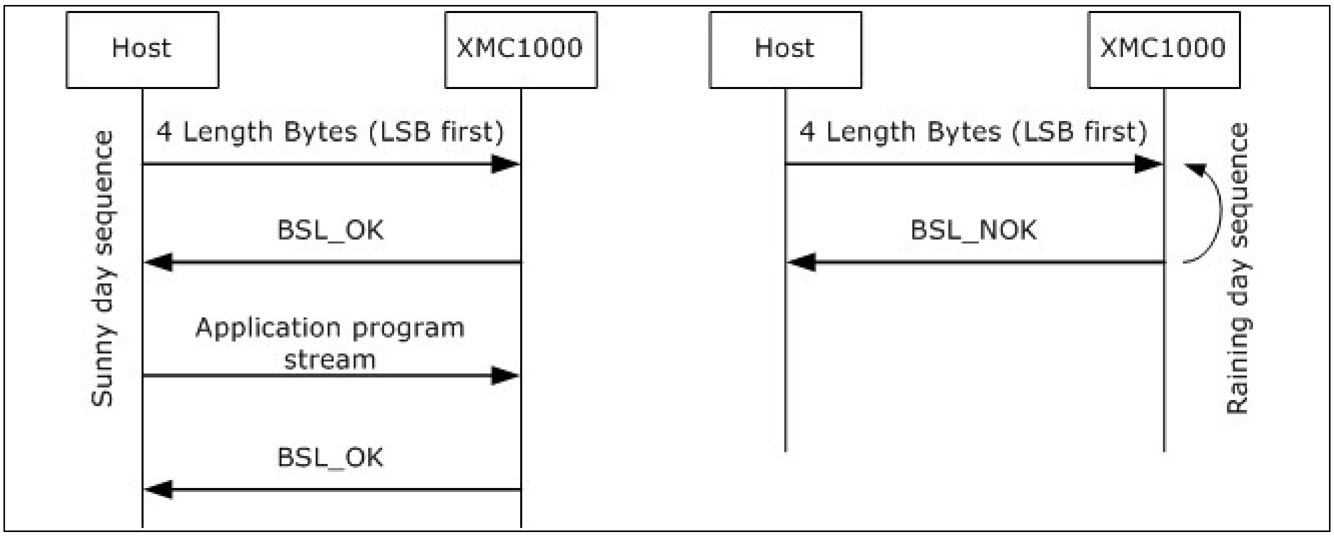


Figure 12 ASC Bootstrap Loader mode Application downloading sequence

If the application length is found to be ok, a BSL_OK byte is sent to the Host and then the Host sends the byte stream belonging to the application.

After the byte stream has been received, the XMC1000 microcontroller terminates the protocol by sending a final OK byte and then cedes control to the downloaded application. If the application length is found to be in error (i.e. application length is greater than the device SRAM size), a BSL_NOK byte is transmitted back to the Host and the XMC1000 microcontroller resumes waiting for the length of the application transmission.

3 ASC Programmer

The XMC1000 microcontroller family has a built-in Bootstrap Loading (BSL) mechanism that can be used for Flash programming. This mechanism is described in detail in the ASC Loader chapter of the XMC1000 Tooling Guide. The XMC1000 family of products provide hard coded Bootstrap Loader routines in the BootROM to carry out Flash programming; For example Flash writing, reading, erasing and verification. Furthermore, the BootROM also provides a routine to change Boot Mode Index (BMI) value of the XMC1000 device.

In this chapter we will demonstrate the implementation of ASC Bootstrap Loader Programmer via standard ASC Bootstrap Loader mode.

The target device is connected to a PC via the ASC interface. The Flash loader system demonstrated in this application note consists of two parts:

- Flash Loader Program
  - The Flash loader program is sent to the target device using the built-in Bootstrap Loading mechanism. Once the program is sent and executed, the Flash loader program establishes a communication protocol to receive commands from the HOST program that is running on the PC, and controls the Flash programming of the target device.
- HOST PC Program
  - The HOST program running on a PC uses the communication protocol defined by the Flash loader. It sends Flash programming commands and the code bytes to be programmed. The HOST program is application specific, so the HOST program in this application note is only an example.

Tool-chains

The Flash loader program for ASC is developed with the following tool-chains:

- DAVE4 development platform v4.0.0 (www.infineon.com/dave)

Example Flash program

An example Flash program, the project LED_Blinky that toggles an LED controlled by P0.5, is provided. The file Blinky.hex can be downloaded to Flash memory. The XMC1x_Load HOST PC program is developed with Microsoft Visual C++ 2010. The example source code is found in the following folders:

- .\DAVE4\ XMC1x_ASCLoader, contains the ASC BSL Loader developed using the GCC compiler.
- .\DAVE4\ XMC1300_Blinky, contains the Flash example program developed using the GCC compiler.
- .\XMC1x_Load\, holds the example HOST PC program that demonstrates the whole process of Flash programming. The project files can be compiled with Microsoft Visual C++2010.

In Chapter 3.5 -> Using the Demonstrator**,** describes in detail how to use the demonstrator to download your own program into Flash and run it.

3.1 ASC Bootstrap Loading

The communication between PC and the target device is established via the ASC interface. Figure below shows a hardware setup for this application. On the target device side, the channel 1 of USIC0 (U0C1) is used as ASC. Ports P1.3 and P1.2 are used as RxD and TxD, respectively.

- receive pin RxD at pin P1.3 (USIC0_CH1.DX0A)
- transmit pin TxD at pin P1.2 (USIC0_CH1.DOUT0)

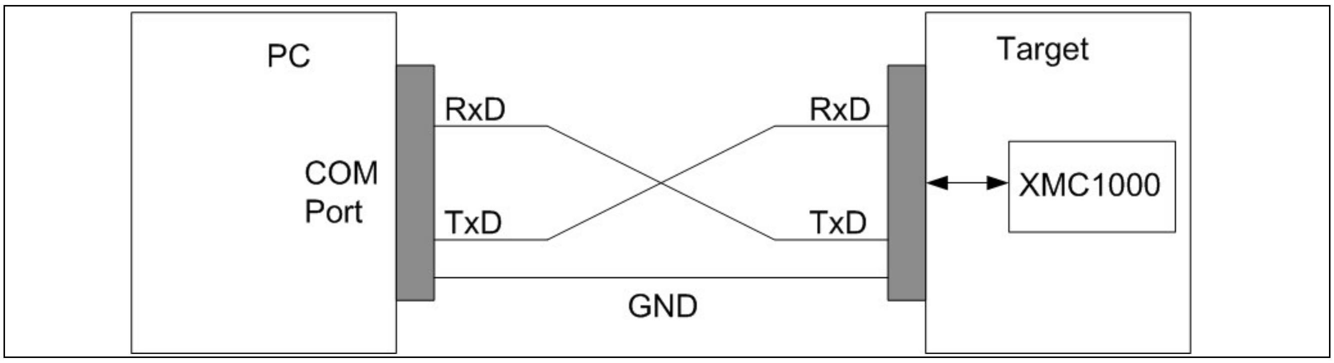


Figure 13 Connection between PC and target system for XMC1000 Bootstrap Loading

To run this program, the first step is to make the target device enter ASC BSL mode.

The boot mode of XMC1000 device depends on its BMI value which is preprogrammed when it is just out of factory. The BMI value is 0xFFC0 which is ASC Bootstrap Load Mode, so ASC Bootstrap Loader mode is entered upon a device reset.

The bootstrap loader procedure is as shown below.

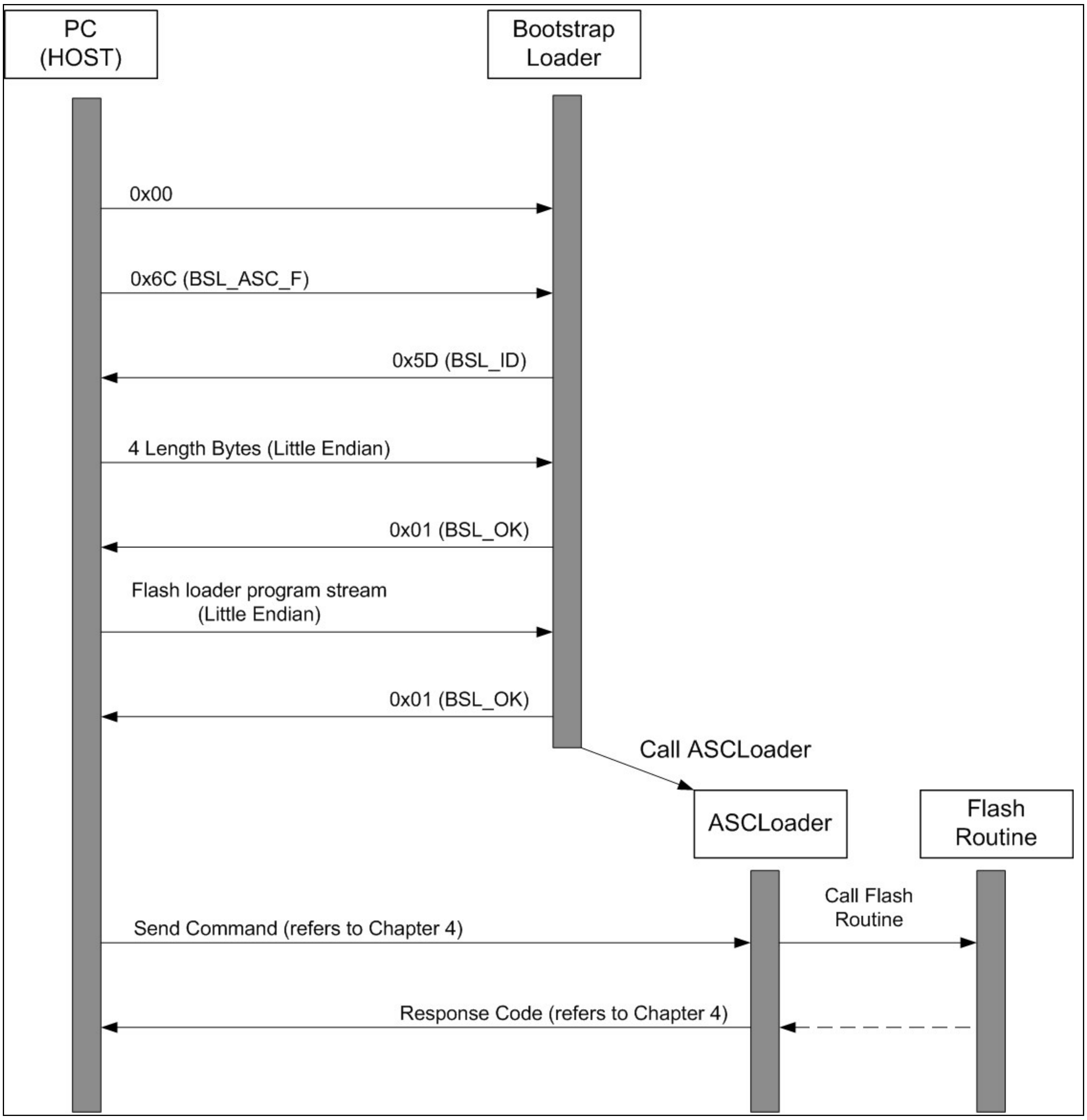

PC (HOST)
Bootstrap Loader
0x00
0x6C (BSL_ASC_F)
0x5D (BSL_ID)
4 Length Bytes (Little Endian)
0x01 (BSL_OK)
Flash loader program stream (Little Endian)
0x01 (BSL_OK)
Call ASCLoader
ASCLoader
Flash Routine
Call Flash Routine
Send Command (refers to Chapter 4)
Response Code (refers to Chapter 4)

Figure 14 ASC Bootstrap loader procedure for Flash programming

The HOST starts by transmitting a zero byte to help the device detect the baud rate. The XMC1000 device supports baud rates of up to 28800 bit/sec at MCLK = 8MHz. The ASC interface will be initialized for 8 data bits and 1 stop bit. Next, the HOST will send a header indicating whether to communicate in basic ASC BSL full duplex/half duplex mode or enhanced ASC BSL full duplex/half duplex mode. Using enhanced ASC BSL mode can achieve a 0.999MHz baud rate at MCLK of 8MHz. In this device guide, we are using Basic ASC BSL mode.

After the baud rate is detected by the device, the bootstrap loader transmits an BSL_ID byte 5DH back to the host. It then waits 4 bytes, describing the length of the Flash loading program from the HOST. The least significant byte is received first. If the application length is found to be acceptable by the BSL, a BSL_OK

(0x01H) byte is sent to the HOST, and the HOST sends the byte stream of the Flash loader. Once the byte stream is received, the BSL terminates the protocol by sending a final BSL_OK byte and then transfers control to the Flash loader program.

If there is an error in the application length (i.e. the application length is greater than device SRAM size), a BSL_NOK byte (0x02H) is transmitted back to the HOST and the BSL resumes its wait for the correct length of bytes.

The file XMC1x_ASCLoader.hex contains the Flash loading program. After XMC1x_ASCLoader is downloaded to SRAM and executed, it will first establish the communication between PC and the target device and then carry out Flash operations.

3.1.1 Flash Loader

The Flash Loader implements the Flash routines and establishes the communication between PC and the target device. The main part of XMC1x_ASCLoader (main.c) implements Flash routines providing the following features:

- Erase Flash sectors
- Erased, Program and verify the programmed Flash pages
- Change the Boot Mode Index (BMI) value of the XMC1000 device

3.1.2 DAVE4 Project Settings

The Flash loader DAVE^{TM} project is available in the .\DAVE4\XMC1x_ASCLoader folder. The project can be imported into the DAVE^{TM} IDE with the following steps:

- Open the DAVE^{TM} IDE
- Import the Infineon DAVE^{TM} project
- Select root directory as .\DAVE4\XMC1x_ASCLoader
- Finish the import

Note: The Flash Loader program must be located in the SRAM starting at 0x20000200, because the Flash Loader program can only run from SRAM. Therefore the default linker script file generated from DAVE4 cannot be used in the Flash loader project, because the default linker script file locates the codes in Flash starting at 0x10001000. The linker script file that locates the codes into SRAM is provided in the XMC1x_ASCLoader.ld folder. To change the linker script file go to project properties:

- Go to Settings->ARM-GCC C Linker->General->Script file (-T)
- Open “Browse…” to import the file XMC1x_ASCLoader.ld into the field

The Linker Script Language file XMC1x_ASCLoader.ld, defines the ROM memory for codes in SRAM starting from address 0x20000000.

The stack, heap and global variables are located in SRAM starting from address 0x20000000. The sector and page address must be specified to erase and program the Flash. An invalid address (an address that is not within the Flash boundaries) results in an address error. The XMC1000 memory organization is described in the Flash Memory Organization chapter.

Flash user codes can be executed starting from the Flash base address 0x10001000.

The Flash Loader defines a communication protocol to receive commands from the PC. Based on the command received, the corresponding Flash routine is executed. The communication structure is described in the Communication Protocol chapter.

3.1.2.1 Modification of DAVE^{TM} startup.s File

Attention: It is important to note that all clock setting functions in the startup_XMC1300.s file used in XMC1x_ASCLoader project, must be removed so that the clock settings made in the ASC bootstrap ROM code (firmware) can be kept without modification. For example, the following instructions in the DAVE^{TM} startup_XMC1300.S file must be removed:

- LDR R0, =SystemInit
- BLX R0

These instructions must be removed because the functions SystemInit() will change the clock settings, which will change the ASC baud rate and destroy the ASC communication between the Host PC and board after control handover from ROM code to the downloaded Flash loader program. If the baud rate is changed, the ASC communication between PC and board will be broken and the Flash programming will not more work.

The startup.s files provided in the XMC1x_ASCLoader project has been modified and the system init functions are removed.

3.2 Flash Memory Organization

The embedded Flash module in the XMC1000 family includes 200 KB (maximum) of Flash memory for code or constant data.

Flash memory is characterized by its sector architecture and page structure. Sectors are Flash memory partitions of different sizes. The offset address of each sector is relative to the base address of its bank which is given in Table 10. Derived devices (see the XMC1000 Data Sheet) can have less Flash memory. The FLASH bank shrinks by cutting-off higher numbered physical sectors.

Table 10 Flash Memory Map

Range Desciption	Size	Start Address
Program Flash	200 Kbytes	0x10001000

- Flash erasure is sector-wise.
- Sectors are subdivided into pages.
- Flash memory programming is page-wise.
- A Flash page contains 256 bytes.

The following table lists the logical sector structure in the XMC1000 family of products.

Table 11 Sector Structure of Flash

Sector	Address Range	Size
1	0x10001000 – 0x10001FFF	4 KB
2	0x10002000 – 0x10002FFF	4 KB

Sector	Address Range	Size
3	0x10003000 – 0x10003FFF	4 KB
4	0x10004000 – 0x10004FFF	4 KB
5	0x10005000 – 0x10005FFF	4 KB
6	0x10006000 – 0x10006FFF	4 KB
7	0x10007000 – 0x10007FFF	4 KB
8	0x10008000 – 0x10008FFF	4 KB
9	0x10009000 – 0x10009FFF	4 KB
10	0x1000A000 – 0x1000AFFF	4 KB
11	0x1000B000 – 0x1000BFFF	4 KB
12	0x1000C000 – 0x1000CFFF	4 KB
13	0x1000D000 – 0x1000DFFF	4 KB
14	0x1000E000 – 0x1000EFFF	4 KB
15	0x1000F000 – 0x1000FFFF	4 KB
16	0x10010000 – 0x10010FFF	4 KB
17	0x10011000 – 0x10011FFF	4 KB
18	0x10012000 – 0x10012FFF	4 KB
19	0x10013000 – 0x10013FFF	4 KB
20	0x10014000 – 0x10014FFF	4 KB
21	0x10015000 – 0x10015FFF	4 KB
22	0x10016000 – 0x10016FFF	4 KB
23	0x10017000 – 0x10017FFF	4 KB
24	0x10018000 – 0x10018FFF	4 KB
25	0x10019000 – 0x10019FFF	4 KB
26	0x1001A000 – 0x1001AFFF	4 KB
27	0x1001B000 – 0x1001BFFF	4 KB
28	0x1001C000 – 0x1001CFFF	4 KB
29	0x1001D000 – 0x1001DFFF	4 KB
30	0x1001E000 – 0x1001EFFF	4 KB
31	0x1001F000 – 0x1001FFFF	4 KB
32	0x10020000 – 0x10020FFF	4 KB
33	0x10021000 – 0x10021FFF	4 KB
34	0x10022000 – 0x10022FFF	4 KB
35	0x10023000 – 0x10023FFF	4 KB
36	0x10024000 – 0x10024FFF	4 KB
37	0x10025000 – 0x10025FFF	4 KB
38	0x10026000 – 0x10026FFF	4 KB
39	0x10027000 – 0x10027FFF	4 KB
40	0x10028000 – 0x10028FFF	4 KB

Sector	Address Range	Size
41	0x10029000 – 0x10029FFF	4 KB
42	0x1002A000 – 0x1002AFFF	4 KB
43	0x1002B000 – 0x1002BFFF	4 KB
44	0x1002C000 – 0x1002CFFF	4 KB
45	0x1002D000 – 0x1002DFFF	4 KB
46	0x1002E000 – 0x1002EFFF	4 KB
47	0x1002F000 – 0x1002FFFF	4 KB
48	0x10030000 – 0x10031FFF	4 KB
49	0x10032000 – 0x10032FFF	4 KB

3.3 Communication Protocol

The Flash loader program “XMC1x_ASCLoader” establishes a communication structure to receive commands from the HOST PC.

The HOST sends commands via transfer blocks. Three types of blocks are defined:

Header Block

Byte 0	Byte1	Bytes 2 … 14	Byte 15
Block Type (0x00)	Mode	Mode-specific content	Checksum

The header block has a length of 16 bytes.

Data Block

Byte 0	Byte1	Bytes 2 … 257	Bytes 258 … 262	Byte 263
Block Type (0x01)	Verification option	256 data bytes	Not used	Checksum

The data block has a length of 264 bytes.

EOT Block

Byte 0	Bytes 1 … 14	Byte 15
Block Type (0x02)	Not used	Checksum

The EOT block has a length of 16 bytes.

The action required by the HOST is indicated in the Mode byte of the header block.

The Flash loader program waits to receive a valid header block and performs the corresponding action. The correct reception of a block is judged by its checksum, which is calculated as the XOR sum of all block bytes excluding the block type byte and the checksum byte itself.

In ASC BSL mode, all block bytes are sent at once via the UART interface. The different modes specify the Flash routines that will be executed by the XMC1x_ASCLoader. The modes and their corresponding communication protocol are described in the following sections of this chapter.

3.3.1 Mode 0: Program Flash Page

Header Block

Byte 0	Byte1	Bytes 2 … 5	Byte 6 … 14	Byte 15
Block Type (0x00)	Mode (0x00)	Page Address	Not Used	Checksum

- Page Address (32bit)
  - Address of the Flash page to be programmed. The address must be 256-byte-aligned and in a valid range (see chapter 3), otherwise an address error will occur. Byte 2 indicates the highest byte, and byte 5 indicates the lowest byte.

After reception of the header block, the device sends either 0x55 as acknowledgement or an error code for an invalid block. The loader enters a loop waiting to receive the subsequent data blocks in the format shown below.

The loop is terminated by sending an EOT block to the target device.

Data Block

Byte 0	Byte1	Bytes 2 … 257	Bytes 258 … 262	Byte 263
Block Type (0x01)	Verification option	256 data bytes	Not used	Checksum

- Verification Option
  - Set this byte to 0x01 to request a verification of the programmed page bytes.
  - If set to 0x00, no verification is performed.
- Code bytes
  - Page content.

After each received data block, the device either sends 0x55 to the PC as acknowledgement, or it sends an error code.

EOT Block

Byte 0	Bytes 1 … 14	Byte 15
Block Type (0x02)	Not used	Checksum

After each received EOT block, the device sends either 0x55 to the PC as acknowledgement, or it sends an error code.

3.3.2 Mode 1: Execute 'Change BMI' routine

Header Block

Byte 0	Byte1	Bytes 2 … 5	Byte 6 … 14	Byte 15
Block Type (0x00)	Mode (0x00)	Page Address	Not Used	Checksum

The command causes a jump to the Change BMI routine located at address 0x00000108. The device will do a system reset and boot up according to the BMI value programmed.

3.3.3 Mode 3: Erase Flash Sector

Header Block

Byte 0	Byte1	Bytes 2 … 5	Byte 6 … 14	Bytes 10 … 14	Byte 15
Block Type (0x00)	Mode (0x03)	Sector Address	Sector Size	Not Used	Checksum

- Sector Address (32bit)
  - Address of the Flash sector to be erased. The address must be a valid sector address, otherwise an address error will occur.
  - Byte 2 indicates the highest address byte.
  - Byte 5 indicates the lowest address byte.
- Sector Size (32bit)
  - Size of the Flash sector to be erased. The size must be a valid sector size.
  - Byte 6 indicates the highest address byte.
  - Byte 9 indicates the lowest address byte.

The device sends either 0x55 to the PC as acknowledgement, or it sends an error code.

3.3.4 Mode 4: Read Flash Data (4 bytes)

Header Block

Byte 0	Byte1	Bytes 2 … 5	Byte 6 … 14	Bytes 10 … 14	Byte 15
Block Type (0x00)	Mode (0x04)	Sector Address	Not Used	Not Used	Checksum

- Sector Address (32bit)
  - Address of the Flash sector to be erased. The address must be a valid sector address, otherwise an address error will occur.
  - Byte 2 indicates the highest address byte.
  - Byte 5 indicates the lowest address byte.

The device sends either 0x55 to the PC as acknowledgement, or it sends an error code.

3.3.5 Response Code to the HOST

The Flash loader program will let the HOST know whether a block has been successfully received and whether the requested Flash routine has been successfully executed by sending out a response code.

Table 12 **Response Codes**

Response Code	Description
0x55	Acknowledgement, no error
0xFF	Invalid block type
0xFE	Invalid mode
0xFD	Checksum error
0xFC	Invalid address
0xFB	Error during Flash erasing
0xFA	Error during Flash programming
0xF9	Verification error
0xF8	Protection error

3.4 HOST PC Program Example

The XMC1000_Bootloader HOST program developed in C++ uses the communication structure described in Chapter -> Communication Protocol.

The file **XMC1x_load_API.cpp** contains the API for direct communication with the XMC1x_ASCLoader. The API includes the following functions:

Table 13 **API Functions**

API Function	Description
Init_uart	Initialize PC COM interface
Init_ASC_BSL	Initialize ASC BSL
Send_loader	Send the ASC Loader
bl_send_header	Send header block via ASC interface
bl_send_data	Send data block via ASC interface
bl_send_EOT	Send EOT block via ASC interface
bl_erase_flash	Erase Flash sectors
bl_download_flash	Download code to Flash
Make_flash_image	Create a Flash image from HEX file

The main program (XMC1x_Load.cpp) initializes ASC and sends XMC1x_ASCLoader to the target device.

The user must specify the HEX file to be downloaded. An example HEX file (XMC1300_Blinky_withApps.hex) is provided. The user code is first downloaded to Flash and the user can then execute the downloaded code if user changes the BMI value to User Mode (Debug) SWD0.

- The Flash erase procedure is implemented in the function bl_erase_flash().

- The Flash programming procedure is implemented in bl_download_flash().

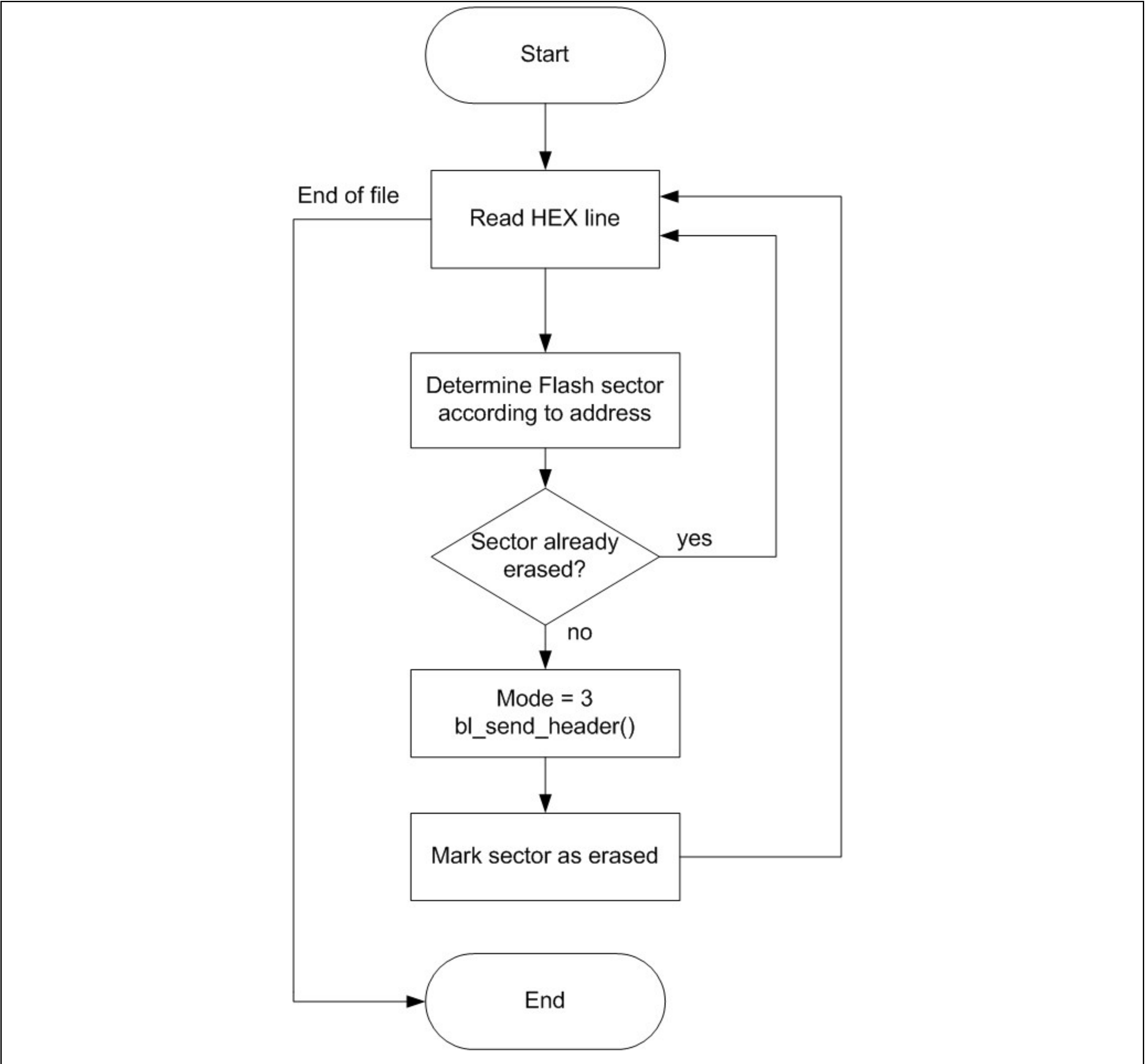


Figure 15 Flash erase procedure implemented in bl_erase_flash()

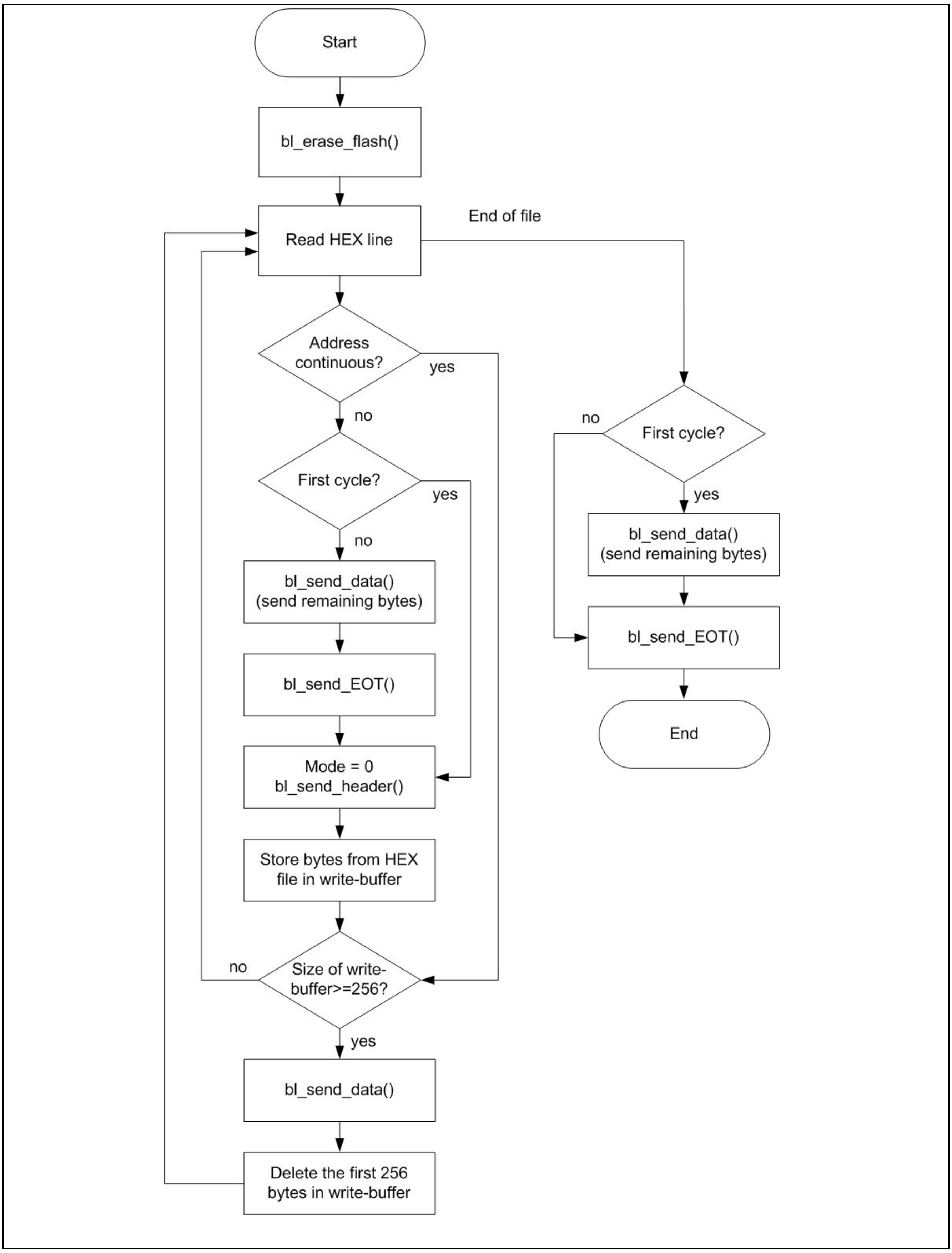

Start
bl_erase_flash()
Read HEX line
End of file
Address continuous?
yes
no
First cycle?
yes
no
bl_send_data() (send remaining bytes)
bl_send_EOT()
Mode = 0 bl_send_header()
Store bytes from HEX file in write-buffer
no
Size of write-buffer>=256?
yes
bl_send_data()
Delete the first 256 bytes in write-buffer
no
First cycle?
yes
bl_send_data() (send remaining bytes)
bl_send_EOT()
End

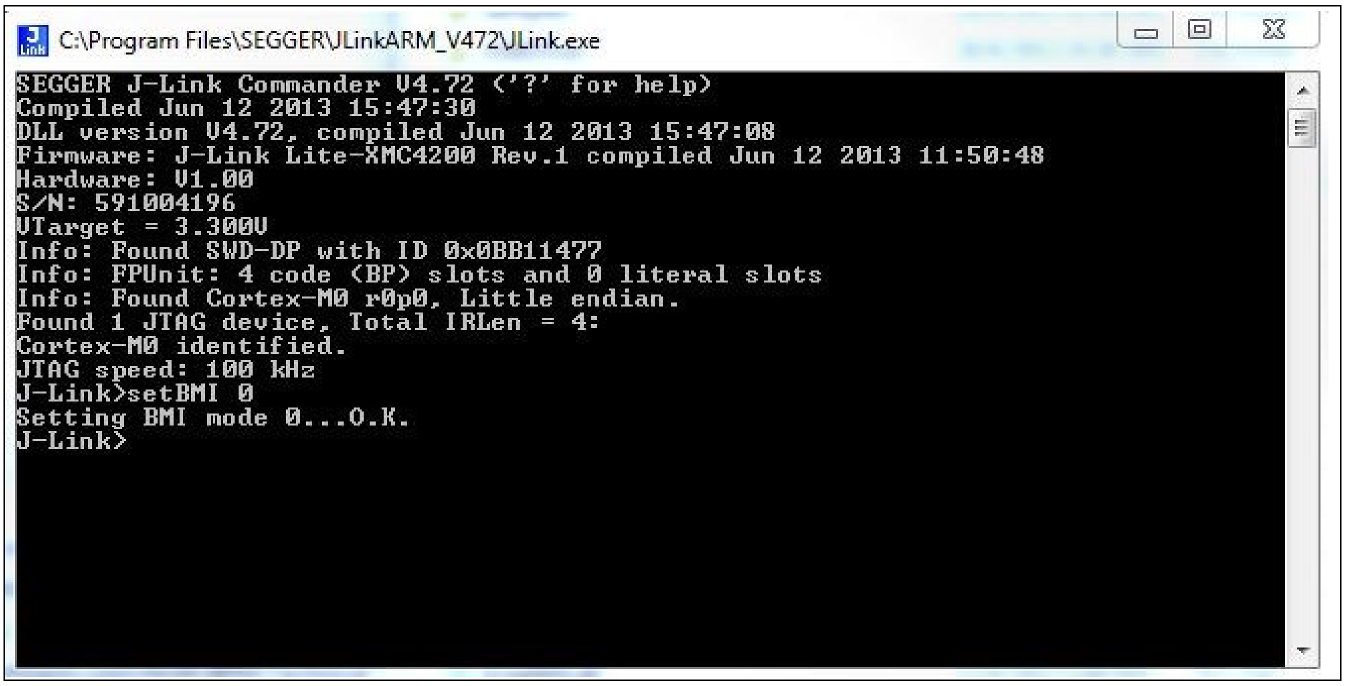
Figure 16 **Flash programming procedure implemented in bl_download_flash()**

3.5 Using the Demonstrator

The example programs have been tested on an Infineon XMC1300 CPU Card with XMC1302 AA step device. The user can use the example program to download user codes (hex file format) into Flash. Here we give a description how to do that.

3.5.1 Hardware Setup

The XMC1300 CPU Card are configured to User Mode (Debug) SWD_0, hence, we need to use J-Link commander to change the BMI value to ASC Bootstrap Load Mode. First, connect the XMC1300 CPU Card to the PC host via a USB cable. Then, double click on "JLink.exe" on SEGGER installation folder to run SEGGER J-Link Commander. Next, type setBMI 0 at the DOS prompt to change the BMI of the XMC1302 device to ASC_BSL mode.



```
SEGGER J-Link Commander V4.72 ('?' for help)
Compiled Jun 12 2013 15:47:30
DLL version V4.72, compiled Jun 12 2013 15:47:08
Firmware: J-Link Lite-XMC4200 Rev.1 compiled Jun 12 2013 11:50:48
Hardware: V1.00
S/N: 591004196
VTarget = 3.300V
Info: Found SWD-DP with ID 0x0BB11477
Info: FPUnit: 4 code (BP) slots and 0 literal slots
Info: Found Cortex-M0 r0p0, Little endian.
Found 1 JTAG device, Total IRLen = 4:
Cortex-M0 identified.
JTAG speed: 100 kHz
J-Link>setBMI 0
Setting BMI mode 0...O.K.
J-Link>
```

Figure 17 Using J-Link commander to change the BMI value of the XMC1302 device

3.5.2 Demonstrator File Structure

The following figure shows the file structure in the example programs.

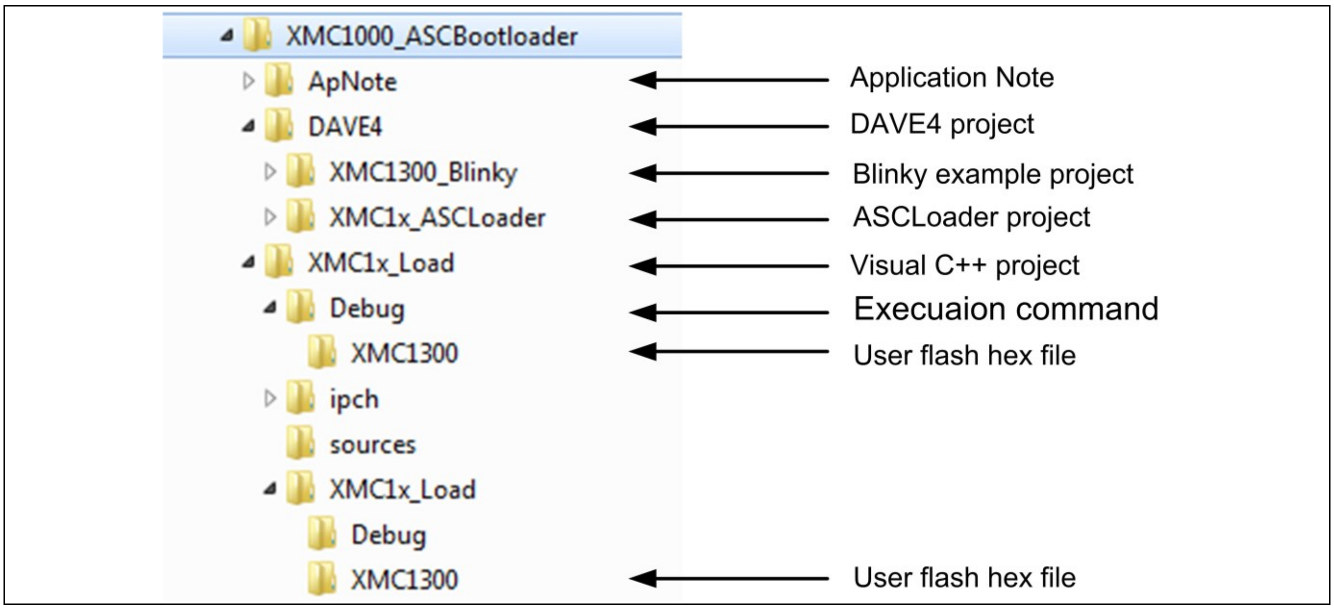


Figure 18 File structure of example programs

- This application note is contained in folder .\App
- The folders .\DAVE4 is the project generated using DAVE4's GNU compiler
- XMC1x_ASCLoader project contains the ASC Bootrstrap Loader program
- The XMC1300_Blinky project is the example project for LED blinking
- .\XMC1x_Load contains the Microsoft Visual C++ 2010 project for the Host PC
- The XMC1x_ASCLoader.hex and LED Blinky example XMC1300_Blinky.hex files are saved in .\XMC1_Load\Debug\XMC1300 and .\XMC1_Load\XMC1_Load\XMC1300, separately

3.5.3 Run the Demonstrator

Before starting the demonstrator, the hex file that needs to be downloaded into Flash and copied into the folders .\ XMC1x_Load \Debug\XMC1300 and .\ XMC1x_Load \ XMC1x_Load \XMC1300:

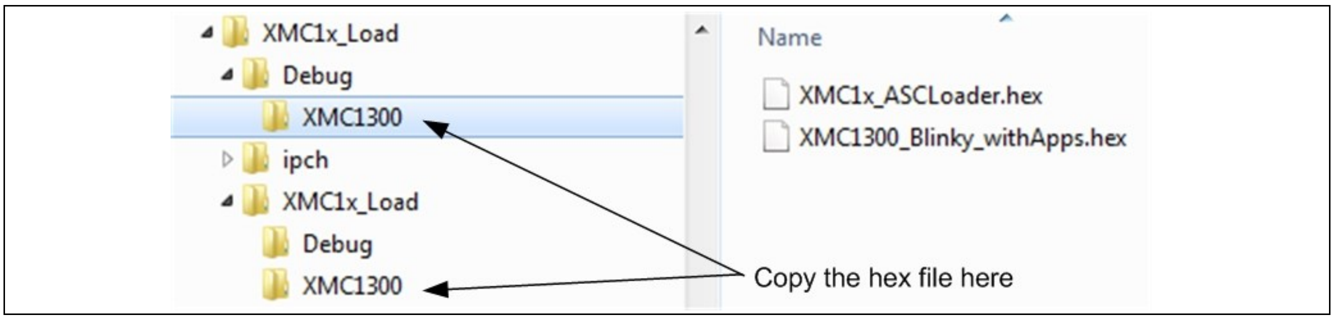


Figure 19 Location of object hex files to be flashed

There are two ways to start the demonstrator.

1. Double click the file XMCLoad.exe under .\ XMC1x_Load \Debug:

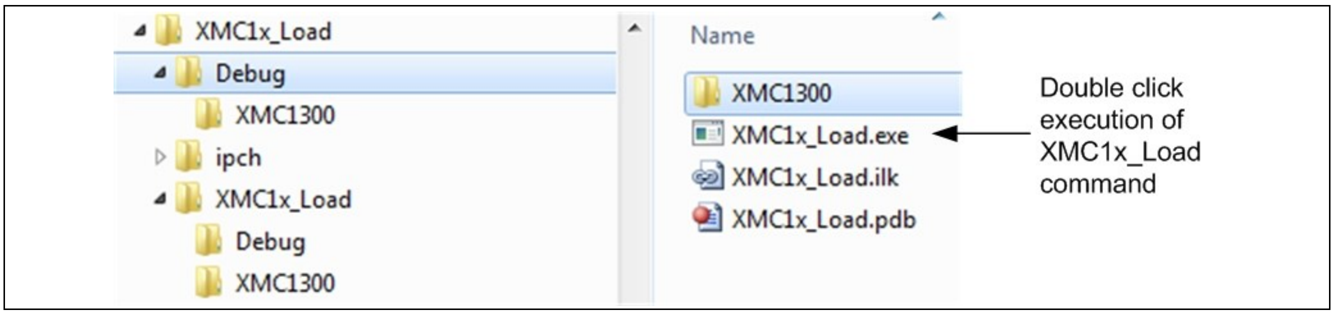


Figure 20 Direct start of demonstrator

Double click the file XMCLoad.sln file in the folder .\XMC1x_Load to open the Microsoft Visual C++ project. The project in this Device Guide is developed using Microsoft Visual C++ 2010.

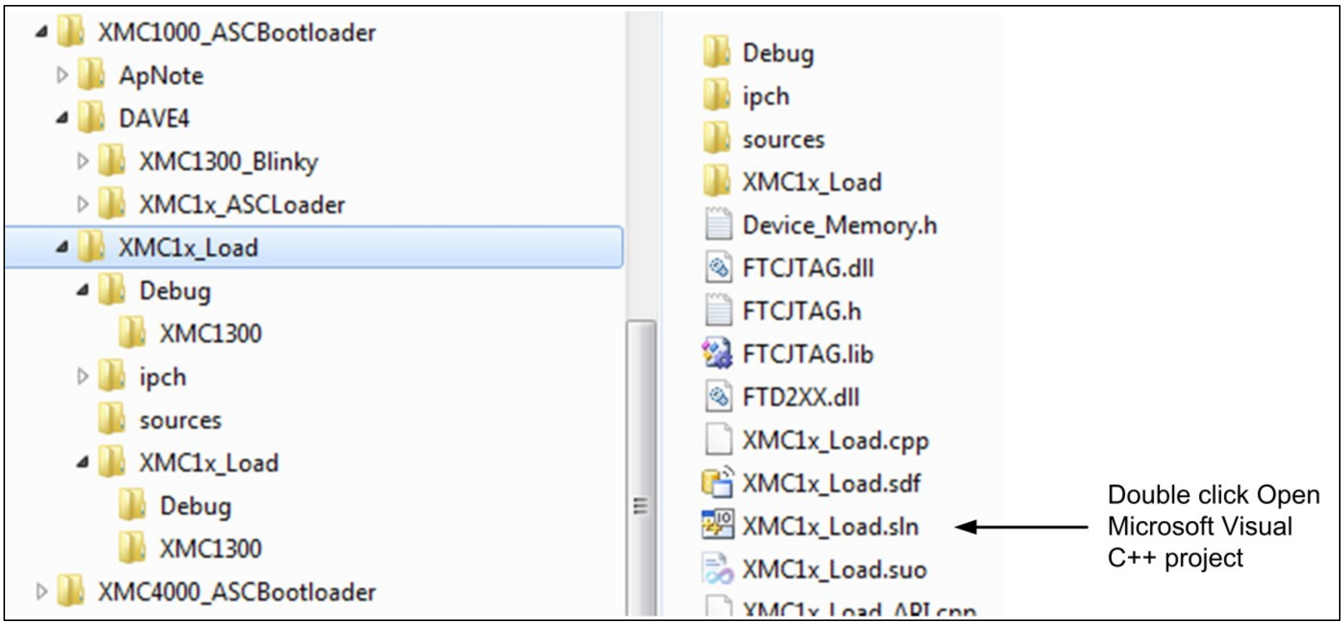
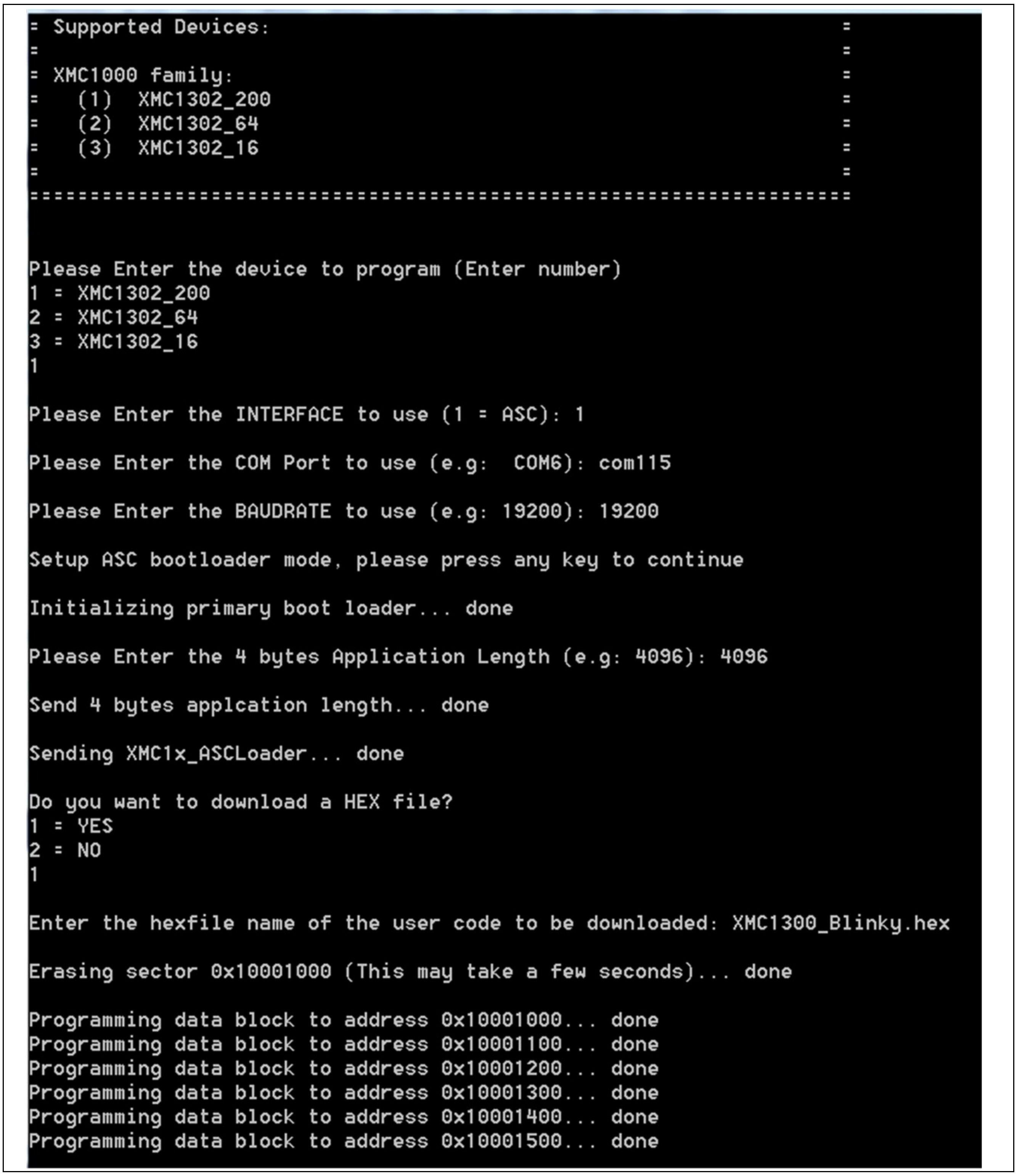


Figure 21 Start using Microsoft Visual project

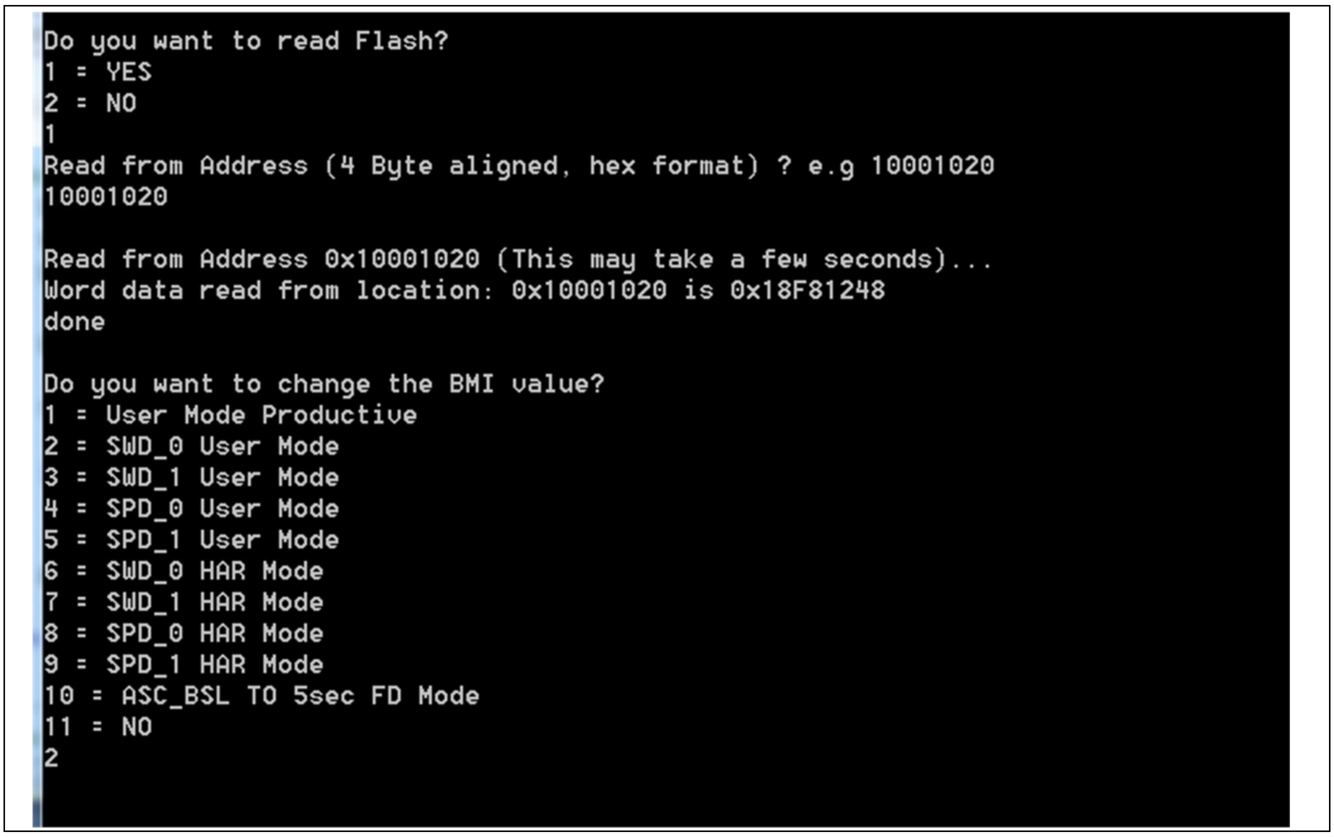
In Microsoft Visual project workbench the project can be started from the “F5” key.

On starting the demonstrator the following window is displayed:



```
=  Supported Devices:                                                   =
=                                                                       =
=  XMC1000 family:                                                      =
=    (1)  XMC1302_200                                                   =
=    (2)  XMC1302_64                                                    =
=    (3)  XMC1302_16                                                    =
=                                                                       =
=========================================================================


Please Enter the device to program (Enter number)
1 = XMC1302_200
2 = XMC1302_64
3 = XMC1302_16
1

Please Enter the INTERFACE to use (1 = ASC): 1

Please Enter the COM Port to use (e.g:  COM6): com115

Please Enter the BAUDRATE to use (e.g: 19200): 19200

Setup ASC bootloader mode, please press any key to continue

Initializing primary boot loader... done

Please Enter the 4 bytes Application Length (e.g: 4096): 4096

Send 4 bytes applcation length... done

Sending XMC1x_ASCLoader... done

Do you want to download a HEX file?
1 = YES
2 = NO
1

Enter the hexfile name of the user code to be downloaded: XMC1300_Blinky.hex

Erasing sector 0x10001000 (This may take a few seconds)... done

Programming data block to address 0x10001000... done
Programming data block to address 0x10001100... done
Programming data block to address 0x10001200... done
Programming data block to address 0x10001300... done
Programming data block to address 0x10001400... done
Programming data block to address 0x10001500... done
```

Figure 22 Start Window from Visual Project



```
Do you want to read Flash?
1 = YES
2 = NO
1
Read from Address (4 Byte aligned, hex format) ? e.g 10001020
10001020

Read from Address 0x10001020 (This may take a few seconds)...
Word data read from location: 0x10001020 is 0x18F81248
done

Do you want to change the BMI value?
1 = User Mode Productive
2 = SWD_0 User Mode
3 = SWD_1 User Mode
4 = SPD_0 User Mode
5 = SPD_1 User Mode
6 = SWD_0 HAR Mode
7 = SWD_1 HAR Mode
8 = SPD_0 HAR Mode
9 = SPD_1 HAR Mode
10 = ASC_BSL TO 5sec FD Mode
11 = NO
2
```

Figure 23 Window GUI illustrates the reading of Flash data and changing of BMI to SWD_0 User Mode

Follow the instructions in the window to finish the Flash programming.

Note: The hex file name that will be programmed into Flash must be given completely with the file extension; e.g. XMC1300_Blinky.hex. Otherwise, the program does not know the file name. The Flash loader program accepts only hex file format. Furthermore, the XMC1x_ASCLoader.hex is less than 4096 bytes, so the 4 bytes Application Length should be given with 4096.

After the hex file is programmed into Flash, user can program the BMI value to be User Mode (Debug) SWD_0, so that the program downloaded will be executed.

3.6 Reference Documents

Table 14 References

Document	Description	Location
XMC1300 User's Manual	User's Manual for XMC1300 device	http://www.infineon.com/xmc1000
Bootloader ASC	Tooling Guide for XMC4000	http://www.infineon.com/xmc4000

4 Revision History

Current Version is V1.2, 2015-05

Page or Reference	**Description of change**
V1.0, 2013-10	
	Initial Version
V1.2, 2015-05	
	1. Change the format 2. Adding workaround for Segger VCOM issue in example codes 3. Change DAVE3 example projects to DAVE4 4. Adding read flash command

www.infineon.com

Edition 2014-11

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
AP32277

Information
For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).