ASC Bootstrap Loader for XMC1000

XMC1000

About this document

Scope and purpose

This Application Note describes how to use the ASC BSL to download the program into flash for XMC1000 microcontroller family. The example codes are provided with Application Note to demonstrate how to perform Flash erases, programming, Flash readback on the XMC1000 device. The applicable products are XMC1000 Microcontrollers family. The example codes are tested on XMC1100/XMC1200/XMC1300/XMC1400 Boot Kit.

Intended audience

This application note is intended to developer for flash programming of XMC1000 family and customers who want to use ASC BSL to download the program in flash.

Table of Contents

About this document **1**
Table of Contents **2**
1 Introduction **3**
1.1 Tool-chains 3
1.1.1 Example Flash program 3
2 Principle of ASC Bootstrap Loading **4**
2.1 ASC Bootstrap Loader mode 4
2.2 ASC Loader 5
2.2.1 Stage 1: Baud rate detection and mode selection 6
2.2.2 Stage 2: Download sequence 9
3 ASC Programmer **11**
3.1 ASC Bootstrap Loading 11
3.2 Flash Loader 13
3.3 DAVE4 Project Settings 13
3.3.1 Modification of DAVETH startup.s File 14
3.4 Flash Memory Organization 14
3.5 Communication Protocol 16
3.5.1 Mode 0: Program Flash Page 17
3.5.2 Mode 1: Execute ‘Change BMI’ routine 18
3.5.3 Mode 3: Erase Flash Sector 18
3.5.4 Mode 4: Read Flash Data (4 bytes) 19
3.5.5 Response Code to the HOST 19
3.6 HOST PC Program Example 19
3.7 Using the Demonstrator 23
3.7.1 Hardware Setup 23
3.7.2 Demonstrator File Structure 24
3.7.3 Run the Demonstrator 24
3.8 Reference Documents 28
Revision History **29**

1 Introduction

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 BootROM chapter of the XMC1000 User manual. However the XMC1000 family of products does not provide any hard coded Bootstrap Loader routines in the BootROM to carry out Flash programming, for example Flash writing, reading, erasing and verification. Therefore a Flash loader program providing Flash routines must be implemented by the user.

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

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.

1.1 Tool-chains

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

- DAV4 development platform v4.1.4

1.1.1 Example Flash program

An example Flash program, the project XMC1x00_Blinky is provided for test purpose. The file Blinky.hex can be downloaded to Flash memory. The XMCLoad HOST PC program is developed with Microsoft Visual C++ 2010. The example source code is found in the following folders:

- .\DAV4\XMC1x_ASCLoader\, contains the ASC BSL Loader developed using the GCC compiler.
- .\DAV4\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.

Chapter 3 describes in detail how to use the demonstrator to download your own program into Flash and run it.

2 Principle of ASC Bootstrap Loading

2.1 ASC Bootstrap Loader mode

In ASC_BSL mode the user can download 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.

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

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 (see Table 1).

Table 1 Table 1 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

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.

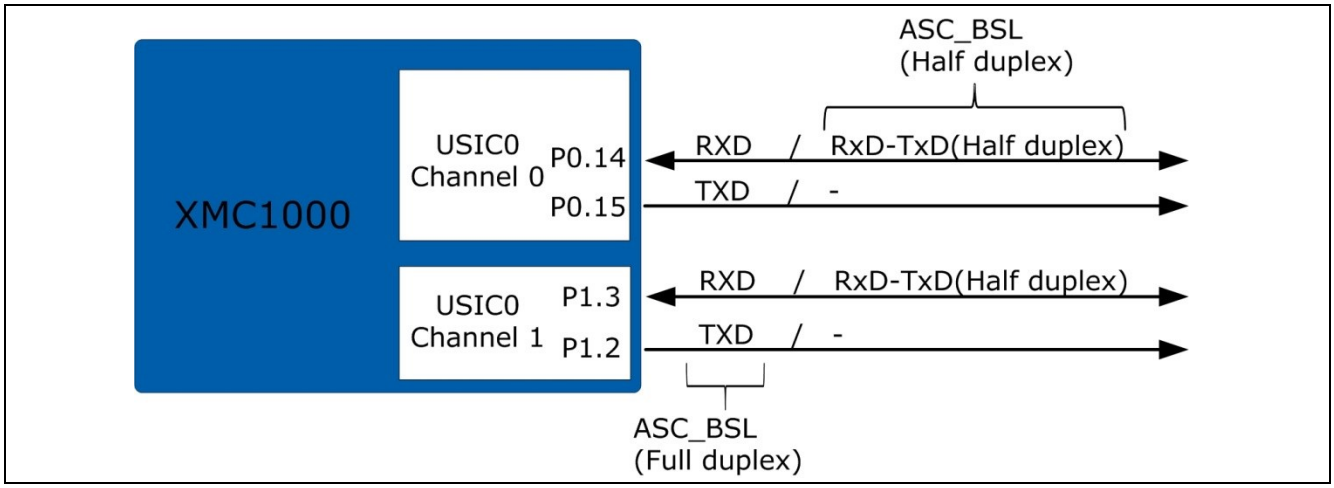

Figure 1 XMC1000 Pin usage for ASC Bootstrap Loader mode

2.2 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.

There are two types 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 2.

The flow in both modes consists of 2 stages.

1. Baud rate detection and mode selection sequence.
2. Download sequence.

Table 2 Table 2 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
48	9.6	153.6	5.994

2.2.1 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 Table 3 and Figure 2 below.

Table 3 Table 3 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.

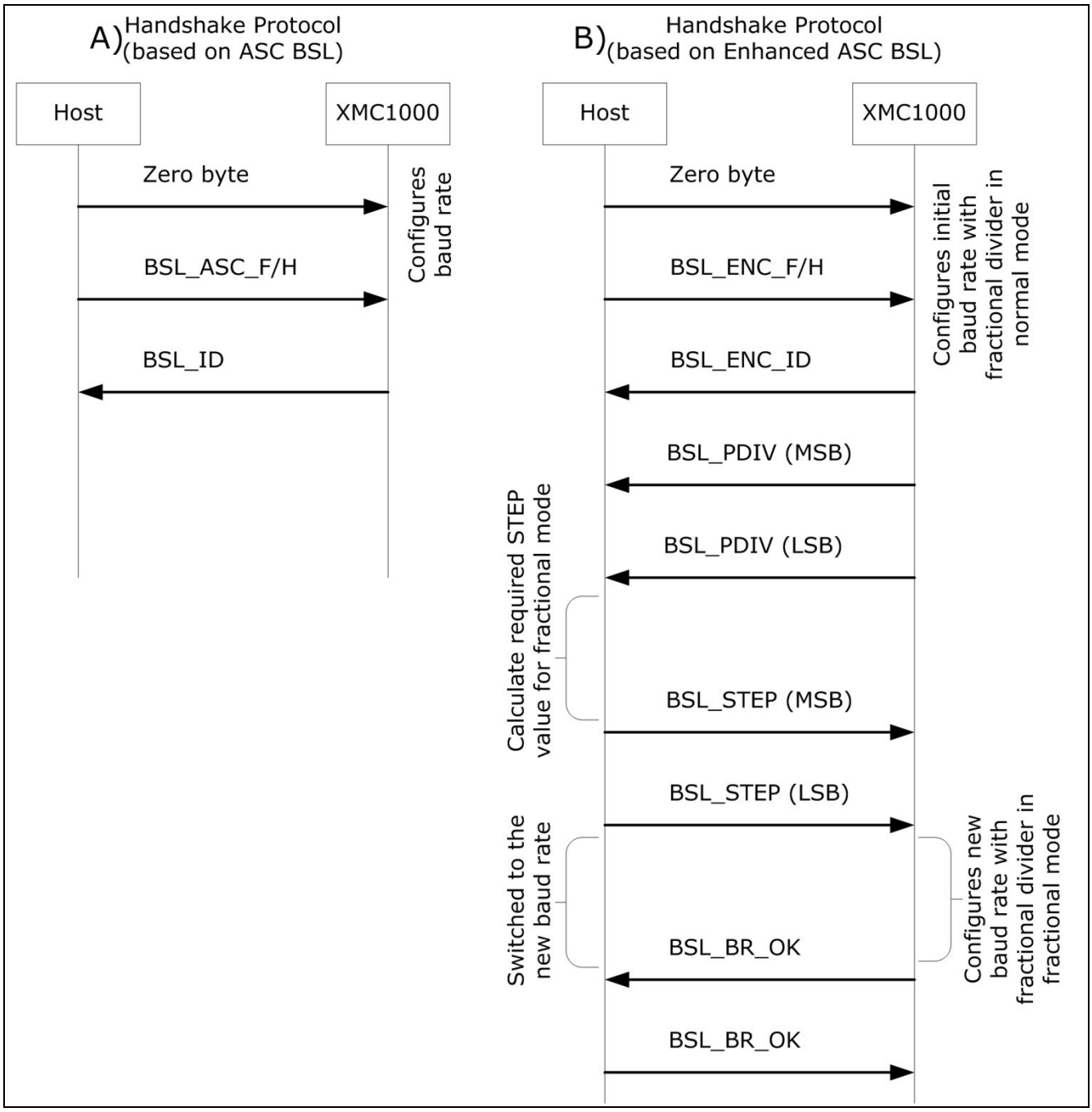

Figure 2 Handshake protocol for XMC1000 AA step device

For XMC1000 AB step device, there is only a small change in the Handshake protocol in enhanced ASC BSL mode as shown in Figure 3. 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.

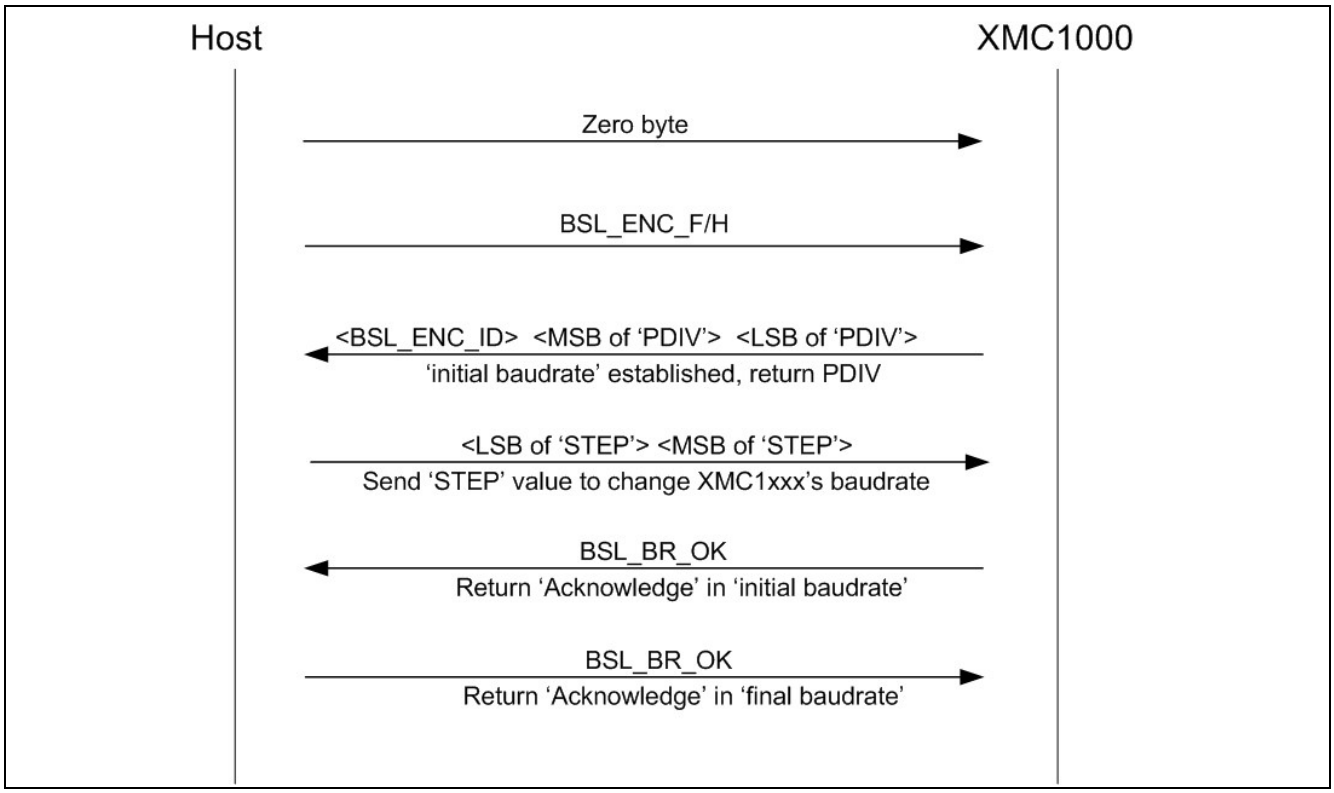

Figure 3 Handshake protocol for XMC1000 AB step device

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 4. The scaling factor is rounded to the nearest integer in order to meet a frequency deviation of +/- 3%.

Table 4 Table 4 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.

2.2.2 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 as shown in Figure 4. The least significant byte is received first. This download sequence is the same for both standard and Enhanced ASC Bootstrap Loader mode.

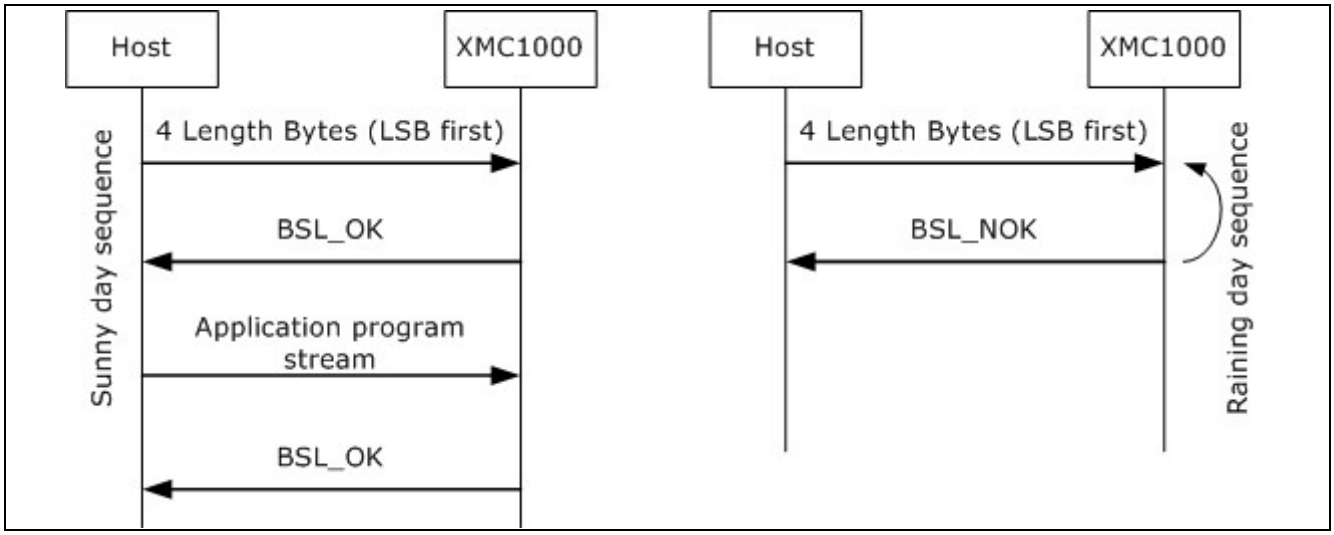

Figure 4 ASC Bootstrap Loader mode Application downloading sequence

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.

Principle of ASC Bootstrap Loading

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

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

3.1 ASC Bootstrap Loading

The communication between PC and the target device is established via the ASC interface.Figure 5. 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)

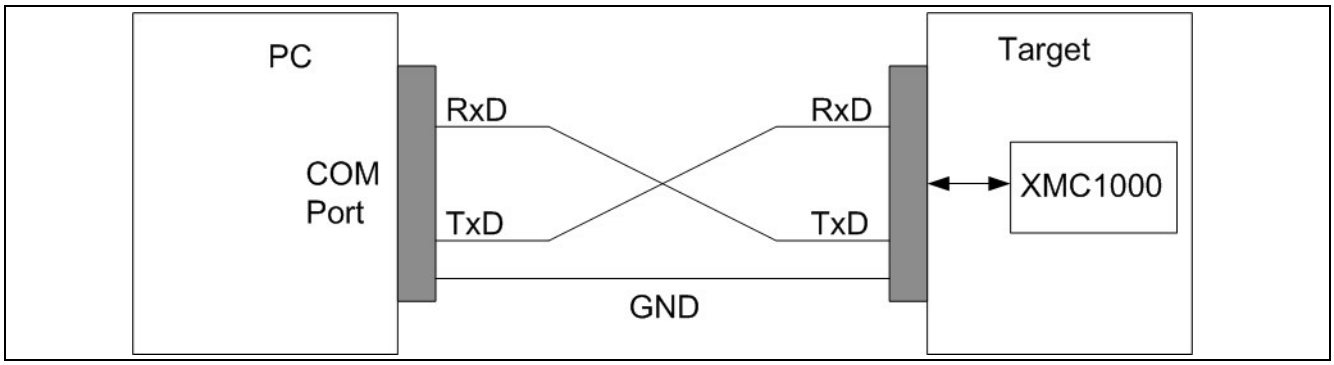

Figure 5 Connection between PC and target system for XMC1000 Bootstrap Loading

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 shown below in Figure 6.

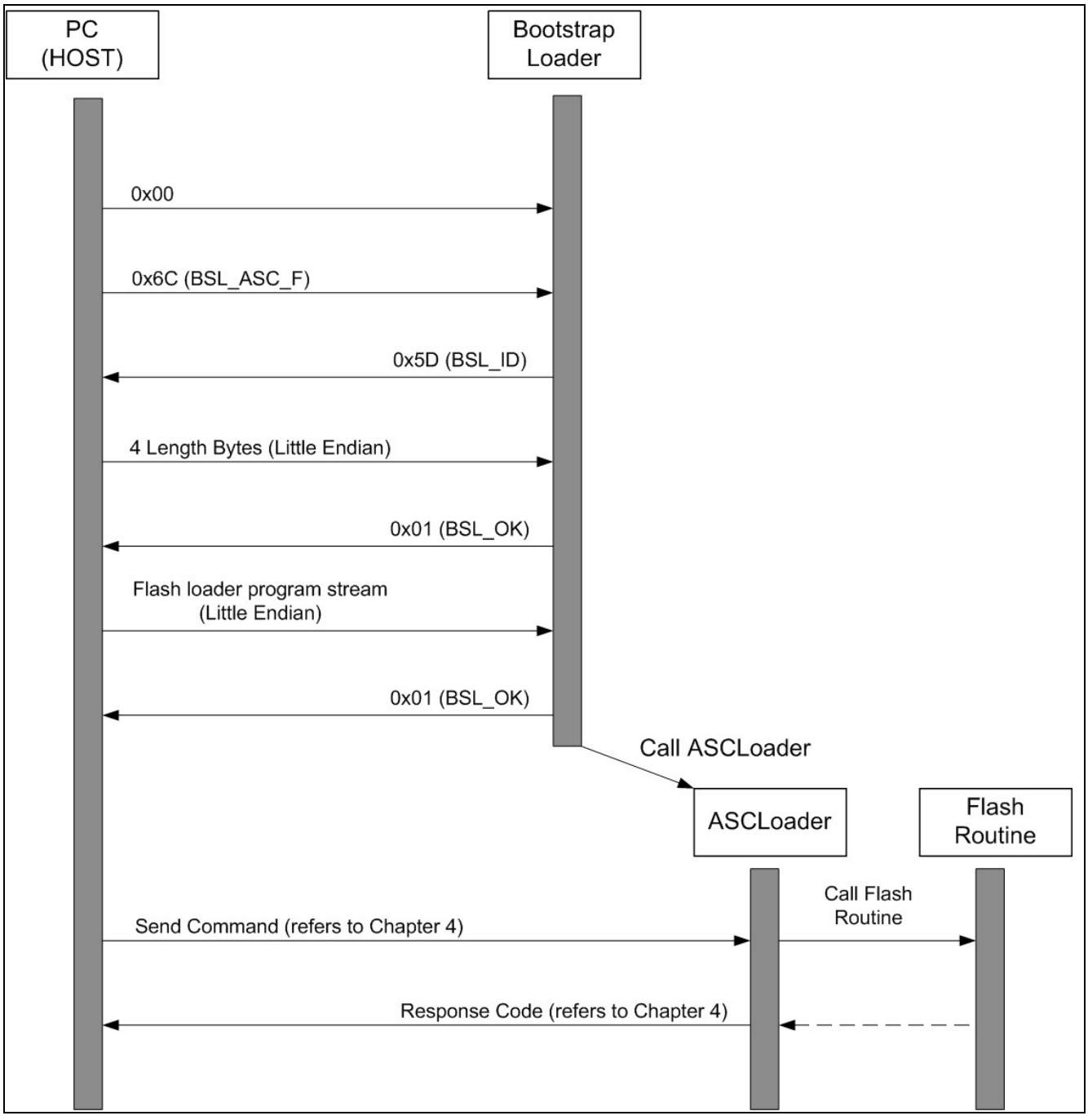

Figure 6 ASC Bootstrap loader procedure for Flash programming

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.

3.2 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
-

3.3 DAVE4 Project Settings

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

- Open the DAVE™ IDE
- Import the Infineon DAVE™ 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.

3.3.1 Modification of DAVE™ 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™ 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.

3.4 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 5. Derived devices (see the XMC1000 Data Sheet) can have less Flash memory. The FLASH bank shrinks by cutting-off higher numbered physical sectors.

Table 5 Table 5 Flash Memory Map

Range Description	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 Table 6 lists the logical sector structure in the XMC1000 family of products.

Table 6 Table 6 Sector Structure of Flash

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

Sector	Address Range	Size
38	0x10026000 – 0x10026FFF	4 KB
39	0x10027000 – 0x10027FFF	4 KB
40	0x10028000 – 0x10028FFF	4 KB
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

3.5 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.

3.5.1 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.

3.5.2 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.

3.5.3 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.

3.5.4 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.

3.5.5 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 listed in Table 7.

Table 7 Table 7 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

3.6 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 functions listed in Table 8:

Table 8 Table 8 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 (XMC1400_Blinky.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() shown in Figure 7.
- The Flash programming procedure is implemented in bl_download_flash() shown in Figure 8.

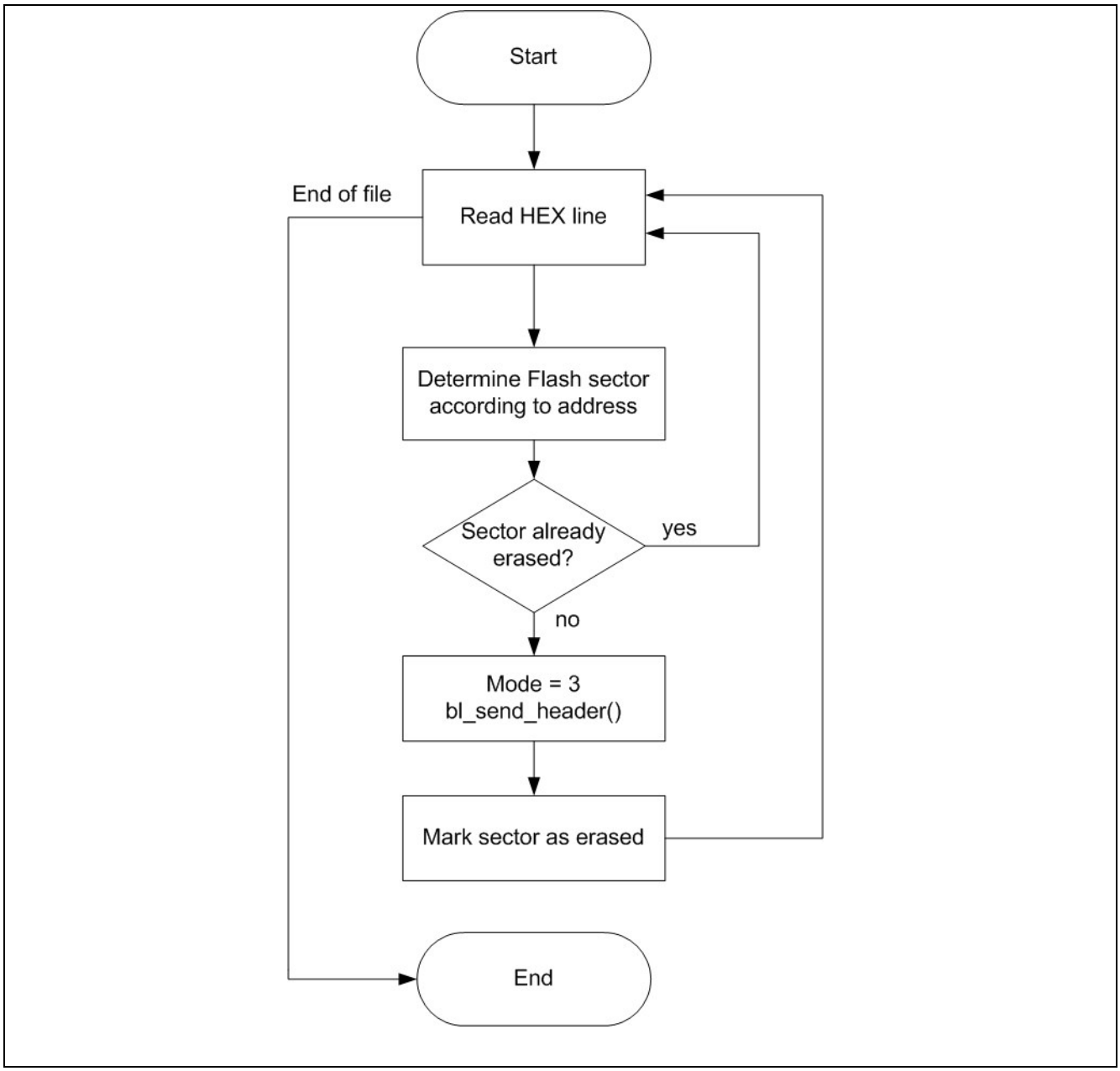

Figure 7 Flash erase procedure implemented in bl_erase_flash()

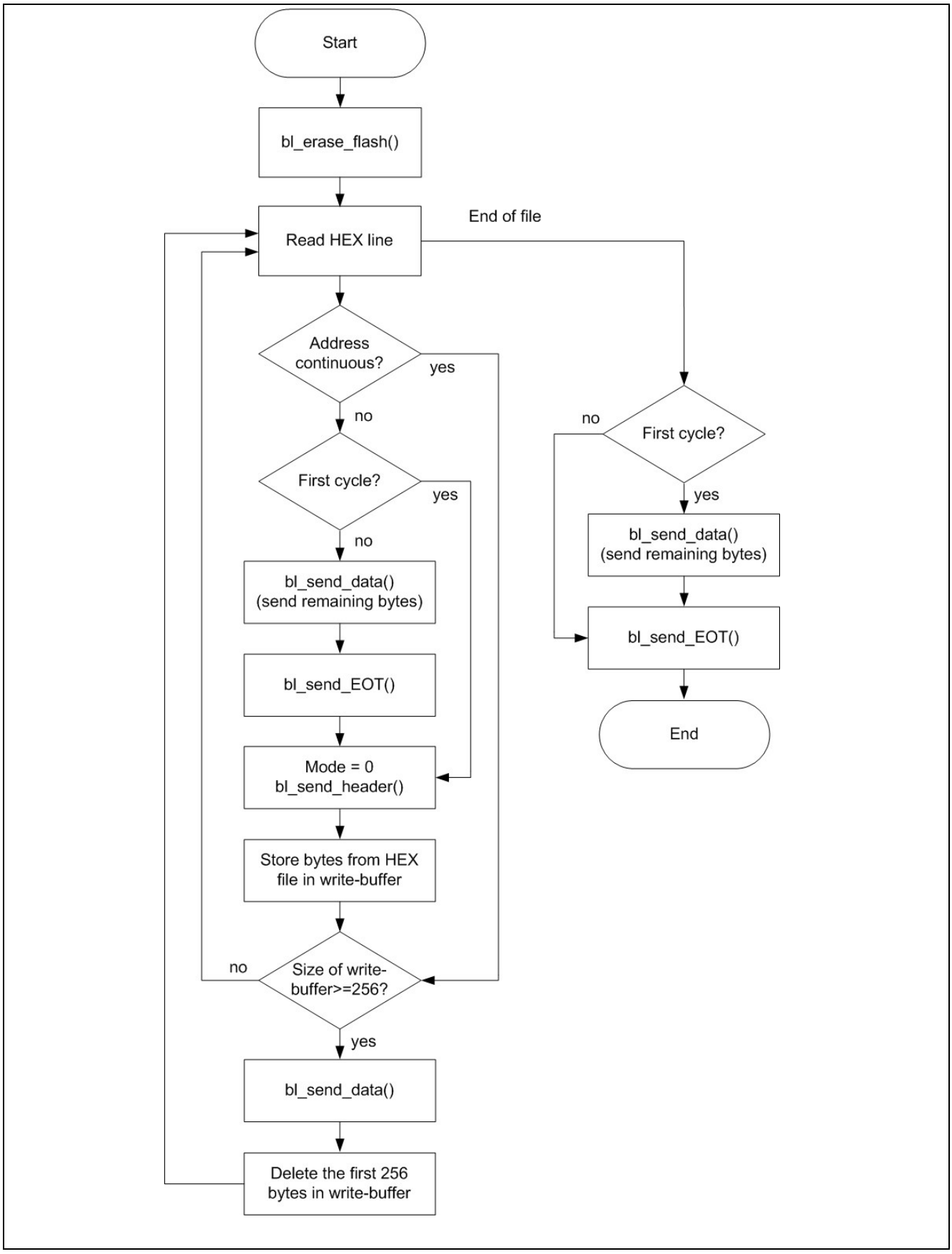

Figure 8 Flash programming procedure implemented in bl_download_flash()

3.7 Using the Demonstrator

The example programs have been tested on Infineon XMC1100/1200/1300/1400 boot kits. 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.

3.7.1 Hardware Setup

The XMC1x00 boot kits are configured to User Mode (Debug) SWD_0, hence, we need BMI set tool to change the BMI value to ASC Bootstrap Load Mode. First, connect the XMC1x00 boot kit to the PC host via a USB cable. Then, open the BMI set tool. After selecting “ASC Bootstrap Load Mode (ASC_BSL), no debug” click “Set BMI”.

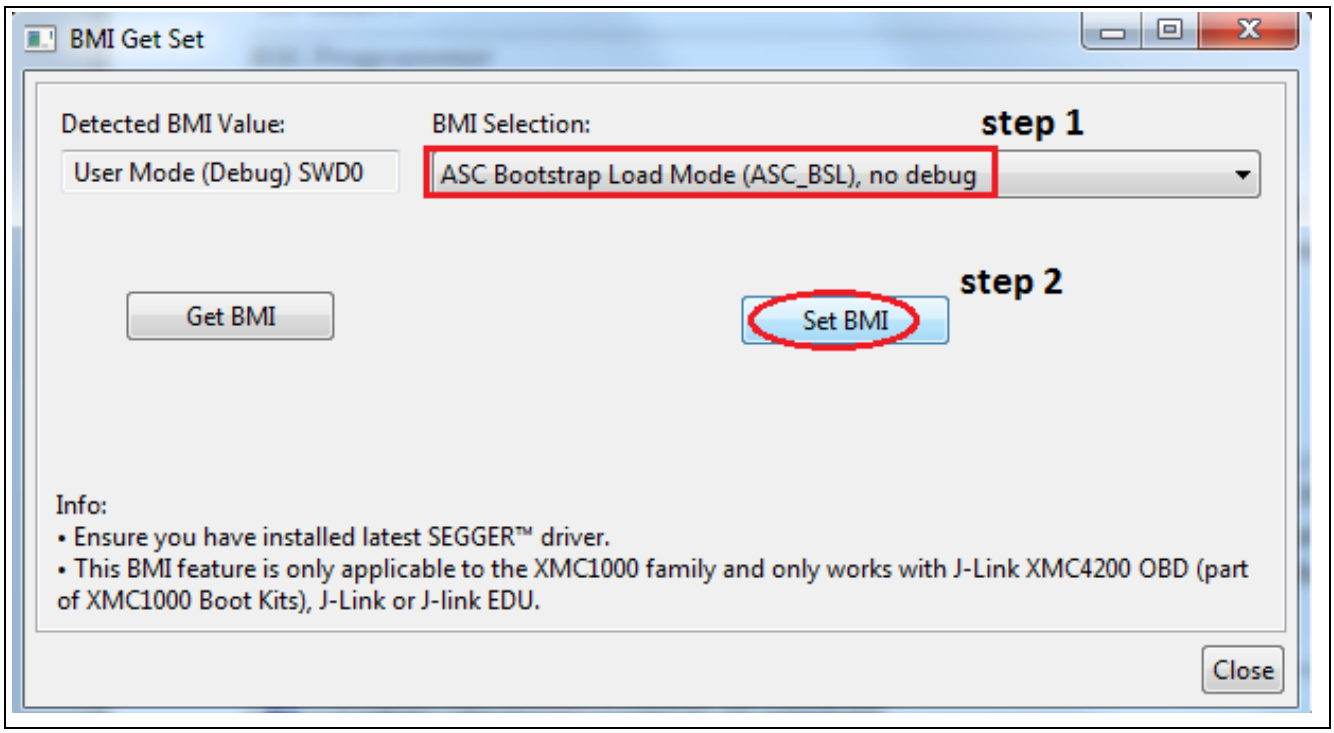

Figure 9 Using BMI Set tool to change the BMI value of the XMC1x00 device

3.7.2 Demonstrator File Structure

Figure 10 shows the file structure in the example programs.

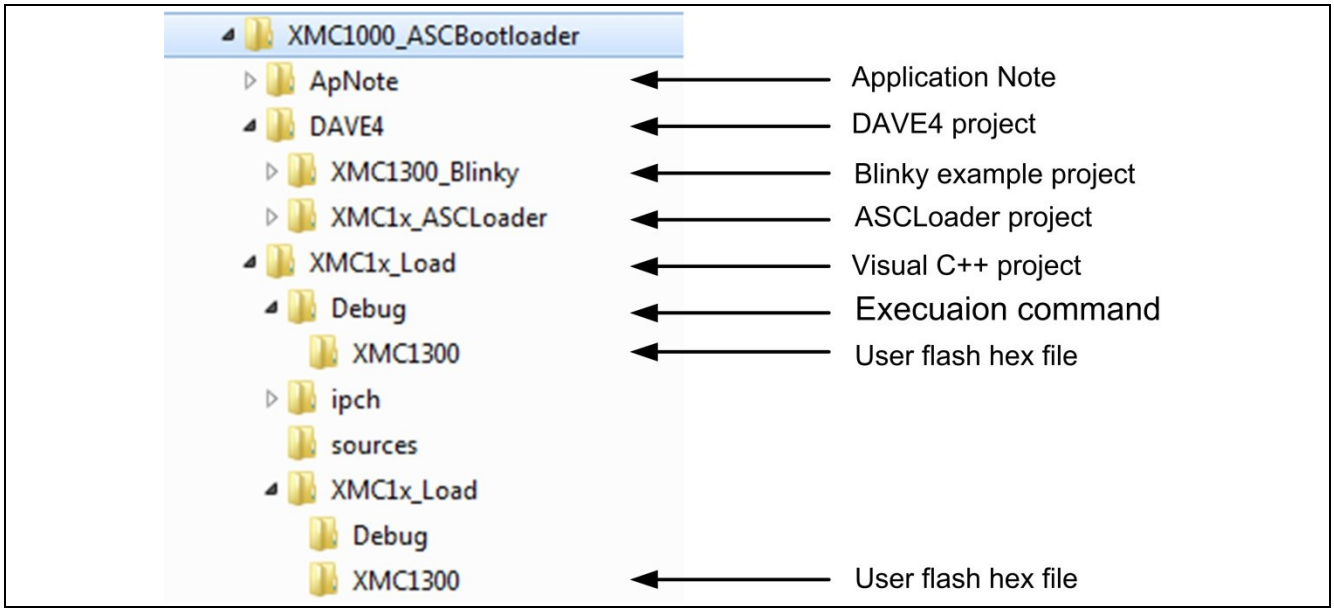

Figure 10 File structure of example programs

- 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

3.7.3 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 shown in Figure 11:

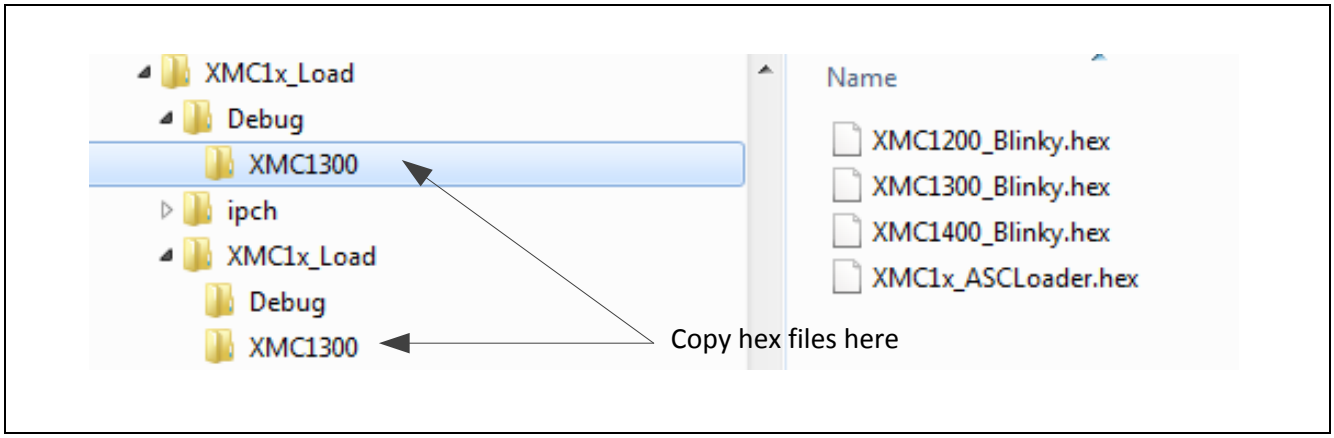

Figure 11 Location of object hex files to be flashed

There are two ways to start the demonstrator.

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

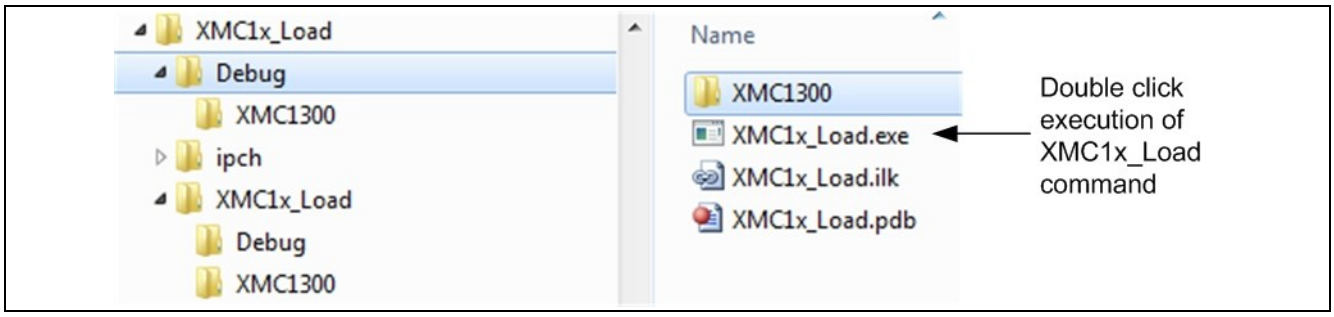

Figure 12 Direct start of demonstrator Example figure

2. 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.

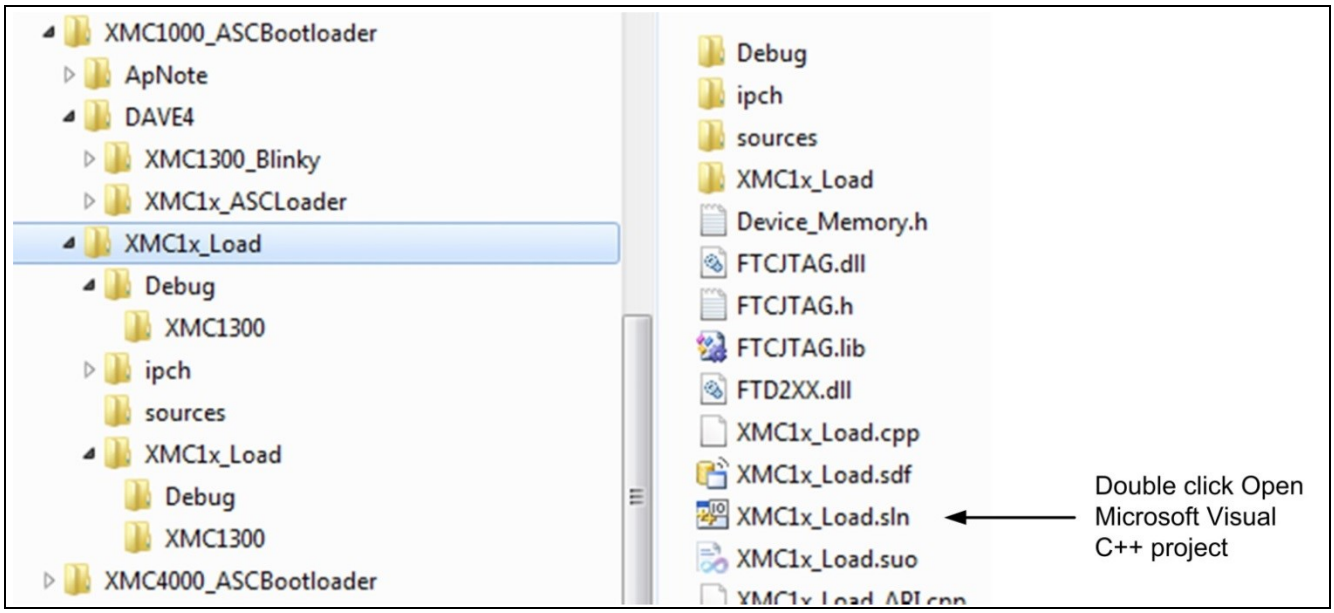

Figure 13 Start using Microsoft Visual project

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

On starting the demonstrator the following window is displayed:

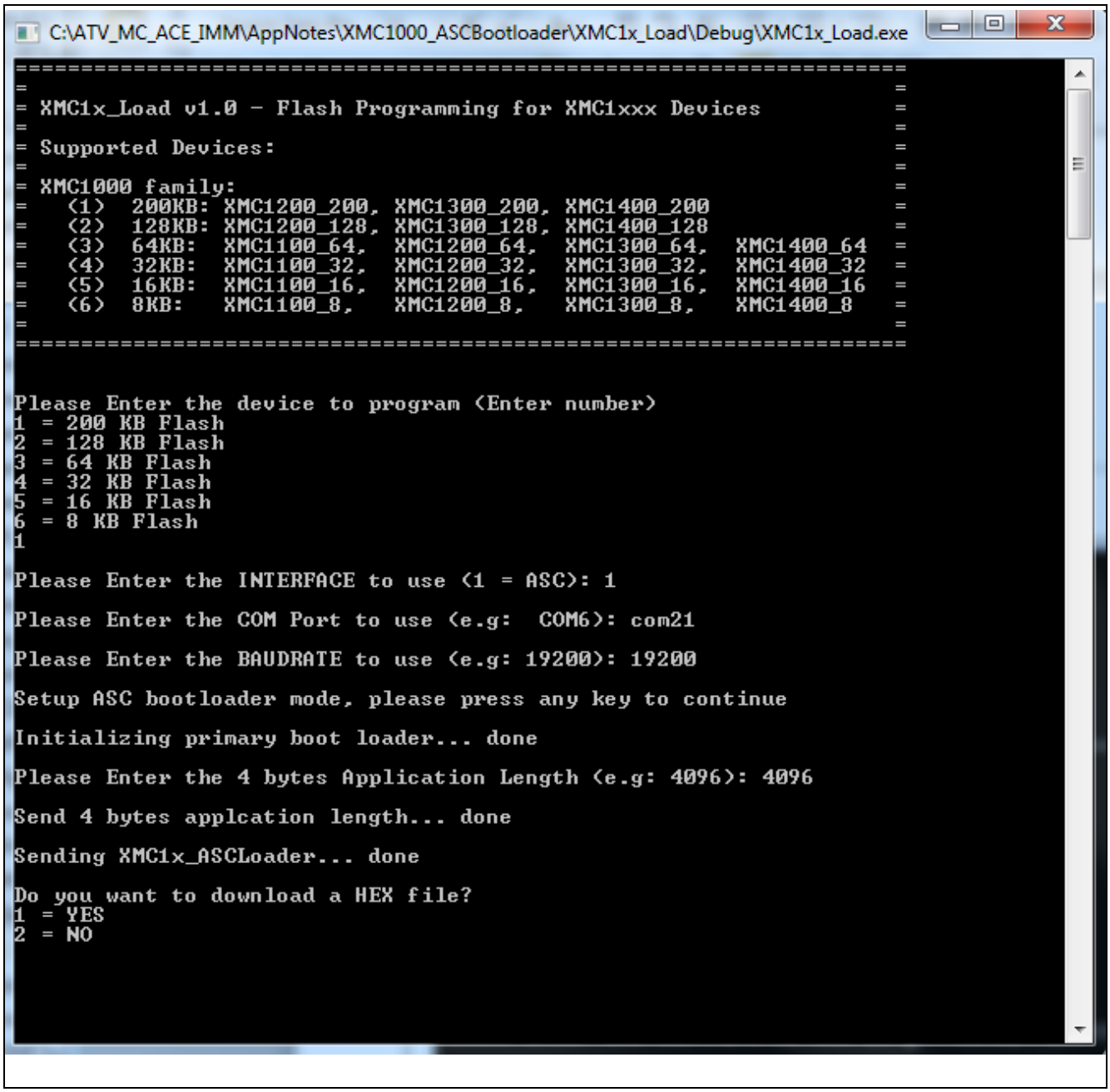

Figure 14 Start Window from Visual Project

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 as shown in Figure 15.

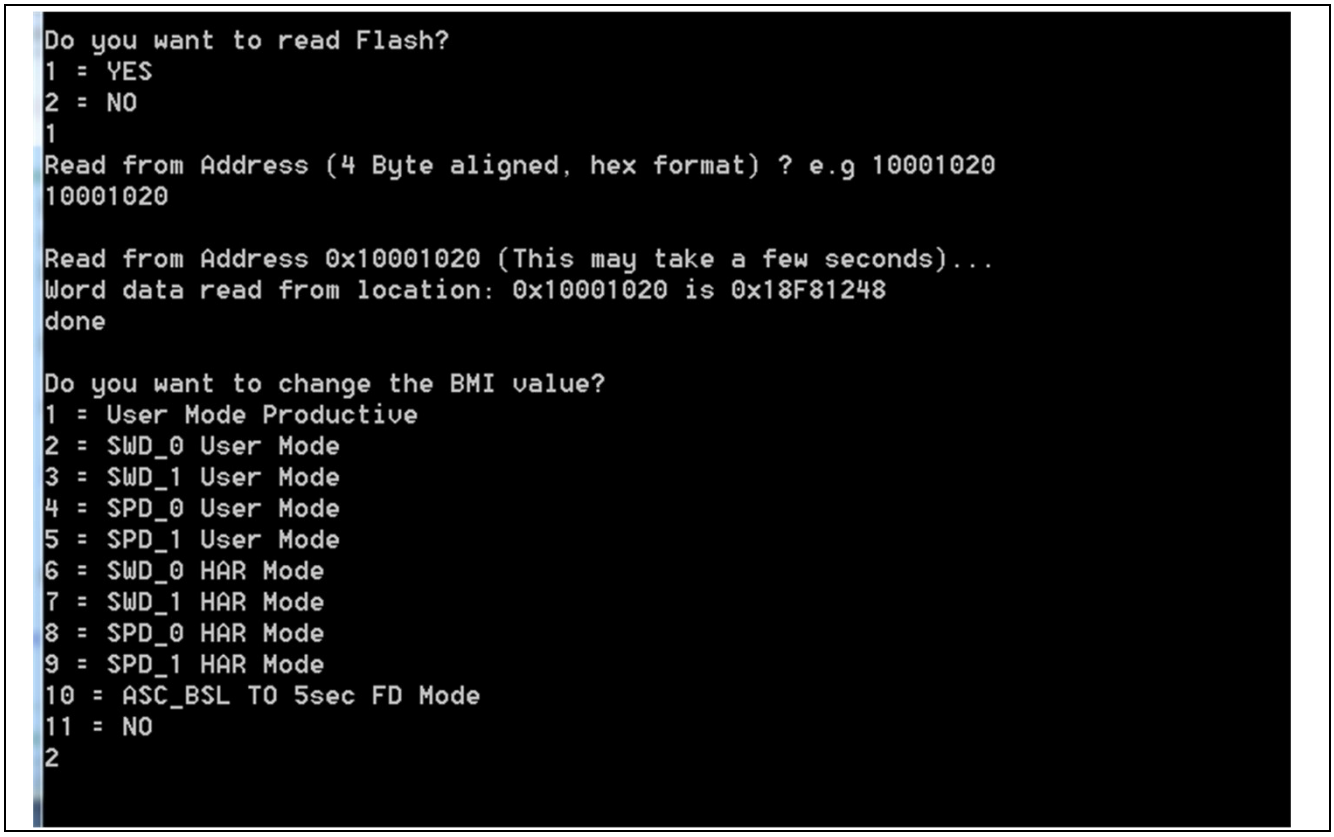

Figure 15 Window GUI illustrates the reading of Flash data and changing of BMI to SWD_0 User Mode

3.8 Reference Documents

Table 9 Table 9 References

Document	Description	Location
XMC1x00 User's Manual	User's Manual for XMC1x00 device	http://www.infineon.com/xmc1000
Bootloader ASC	Tooling Guide for XMC4000	http://www.infineon.com/xmc4000

Revision History

Current Version is 1.3, 2015-11

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
V1.3, 2015-11	
	1. Add support for XMC1400 2. Remove the chapter of BMI description

Trademarks of Infineon Technologies AG
AURIX™, C166™, CanPAK™, CIPOS™, CoolGaN™, CoolMOS™, CoolSET™, CoolSiC™, CORECONTROL™, CROSSAVE™, DAVE™, DI-POL™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, HITFET™, HybridPACK™, Infineon™, ISOFACE™, IsoPACK™, i-Wafer™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OmniTune™, OPTIGA™, OptiMOS™, ORIGA™, POWERCODE™, PRIMARION™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, ReverSave™, SatRIC™, SIEGET™, SIPMOS™, SmartLEWIS™, SOLID FLASH™, SPOC™, TEMPFET™, thinQ!™, TRENCHSTOP™, TriCore™.

Trademarks updated August 2015

Other Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2015-11-12

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2015 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32277

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.