XMC4000

Microcontroller Series
for Industrial Applications

Bootloader

✓ ASC

Tooling Guide

V1.1 2013-11

Microcontrollers

Edition 2013-11

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2014 Infineon Technologies AG
All Rights Reserved.

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, 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.

Information

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

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Revision History	
Page or Item	**Subjects (major changes since previous revision)**
V1.1, 2013-11	
V1.0, 2013-04	Added “Flash Protection” sections **Error! Reference source not found.** and **Error! Reference source not found.**. Added support for XMC4400, XMC4200/4100 devices. Corrected link to Infineon web site.

Trademarks of Infineon Technologies AG

AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, EconoPACK™, CoolMOS™, CoolSET™, CORECONTROL™, CROSSAVE™, DAVE™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, HITFET™, HybridPACK™, I²RF™, ISOFACE™, IsoPACK™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OptiMOS™, ORIGA™, PRIMARION™, PrimePACK™, PrimeSTACK™, PRO-SIL™, PROFET™, RASIC™, ReverSave™, SatRIC™, SIEGET™, SINDRION™, SIPMOS™, SmartLEWIS™, SOLID FLASH™, TEMPFET™, thinQ!™, TRENCHSTOP™, TriCore™.

Other Trademarks

Advance Design System™ (ADS) of Agilent Technologies, AMBA™, ARM™, MULTI-ICE™, KEIL™, PRIMECELL™, REALVIEW™, THUMB™, µVision™ of ARM Limited, UK. AUTOSAR™ is licensed by AUTOSAR development partnership. Bluetooth™ of Bluetooth SIG Inc. CAT-iq™ of DECT Forum. COLOSSUS™, FirstGPS™ of Trimble Navigation Ltd. EMV™ of EMVCo, LLC (Visa Holdings Inc.). EPCOS™ of Epcos AG. FLEXGO™ of Microsoft Corporation. FlexRay™ is licensed by FlexRay Consortium. HYPERTERMINAL™ of Hilgraeve Incorporated. IEC™ of Commission Electrotechnique Internationale. IrDA™ of Infrared Data Association Corporation. ISO™ of INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. MATLAB™ of MathWorks, Inc. MAXIM™ of Maxim Integrated Products, Inc. MICROTEC™, NUCLEUS™ of Mentor Graphics Corporation. Mifare™ of NXP. MIPI™ of MIPI Alliance, Inc. MIPS™ of MIPS Technologies, Inc., USA. muRata™ of MURATA MANUFACTURING CO., MICROWAVE OFFICE™ (MWO) of Applied Wave Research Inc., OmniVision™ of OmniVision Technologies, Inc. Openwave™ Openwave Systems Inc. RED HAT™ Red Hat, Inc. RFMD™ RF Micro Devices, Inc. SIRIUS™ of Sirius Satellite Radio Inc. SOLARIS™ of Sun Microsystems, Inc. SPANSION™ of Spansion LLC Ltd. Symbian™ of Symbian Software Limited. TAIYO YUDEN™ of Taiyo Yuden Co. TEAKLITE™ of CEVA, Inc. TEKTRONIX™ of Tektronix Inc. TOKO™ of TOKO KABUSHIKI KAISHA TA. UNIX™ of X/Open Company Limited. VERILOG™, PALLADIUM™ of Cadence Design Systems, Inc. VLYNQ™ of Texas Instruments Incorporated. VXWORKS™, WIND RIVER™ of WIND RIVER SYSTEMS, INC. ZETEX™ of Diodes Zetex Limited.

Last Trademarks Update 2011-02-24

Table of Contents

ASC Bootloader **5**

1 Introduction **6**
1.1 Tool-chains 6
1.2 Example Flash program 6

2 ASC Bootstrap Loading **7**
2.1 Flash Loader 8
2.2 DAVE3 Project Settings 9
2.3 Keil Project Settings 9
2.4 IAR Project Settings 10
2.5 Modification of startup.s File 10

3 Flash Memory Organization **10**
3.1 XMC4500 11
3.2 XMC4400 11
3.3 XMC4200 12

4 Communication Protocol **13**
4.1 Mode 0: Program Flash Page 13
4.2 Mode 1: Execute User Program from Flash 14
4.3 Mode 2: Execute User Program from iCache 14
4.4 Mode 3: Erase Flash Sector 15
4.5 Mode 4: Read Flash Protection Status 15
4.6 Mode 5: Protect or Unprotect Flash 15
4.7 Response Code to the HOST 16

5 HOST PC Program Example **17**

6 Using the Demonstrator **19**
6.1 Hardware Setup 19
6.2 Demonstrator File Structure 19
6.3 Run the Demonstrator 20

7 Reference Documents **21**

ASC Bootloader

1 Introduction

The XMC4000 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 XMC4000 Reference Manual. However the XMC4000 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 XMC4000 family supports both Asynchronous Serial Interface (ASC) BSL and Cotroller Area Network (CAN) 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-chains:

- DAVE3 development platform v3.1.6
- Keil Toolchain v.4.7
- IAR Toolchain v6.5.2

The project files for these three tool-chains provided in this example are independent from each other and user can choose to use any of the 3.

1.2 Example Flash program

An example Flash program, the project LED_Blinky that toggles an LED controlled by P3.9, is provided for all 3 tool-chains. 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:

- .\DAVE3\XMC4x00\ASCLoader, contains the ASC BSL Loader developed using the GCC compiler.
- .\Keil\XMC4x00\ASCLoader, contains the ASC BSL Loader developed using the Keil compiler.
- .\IAR\XMC4x00\ASCLoader, contains the ASC BSL Loader developed using the IAR compiler.
- .\DAVE3\XMC4x00\LED_Blinky, contains the Flash example program developed using the GCC compiler.
- .\Keil\XMC4x00\LED_Blinky, contains the Flash example program developed using the Keil compiler.
- .\IAR\XMC4x00\LED_Blinky, contains the Flash example program developed using the IAR compiler.

- .\XMCLoad\, 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 6 describes in detail how to use the demonstrator to download your own program into Flash and run it.

2 ASC Bootstrap Loading

The communication between PC and the target device is established via the ASC interface. Figure 1 shows a hardware setup for this application. On the target device side, the channel 0 of USIC0 (U0C0) is used as ASC. Ports P1.4 and P1.5 are used as RxD and TxD, respectively.

- receive pin RxD at pin P1.4 (USIC0_DX0B)
- transmit pin TxD at pin P1.5 (USIC0_DOUT0)

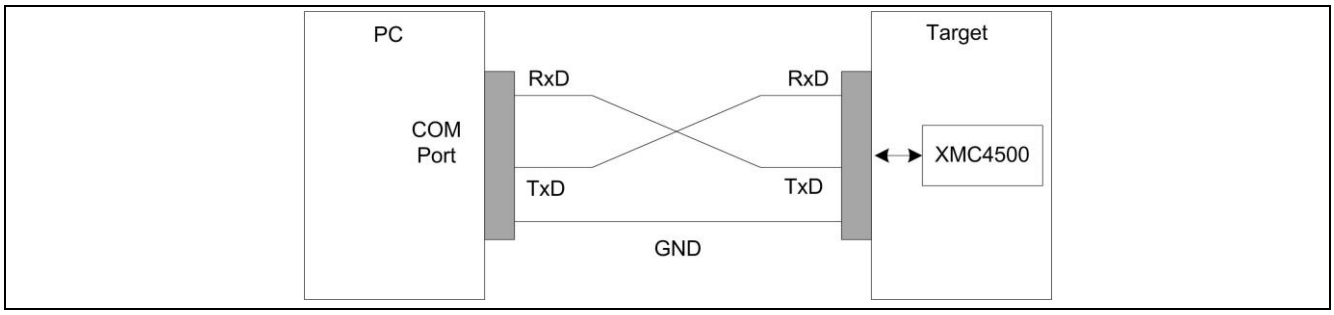


Figure 1 Connection between PC and target system for XMC4000 Bootstrap Loading

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

ASC Bootstrap Loader mode is entered upon a device reset, if the boot pins TMS=0 and TCK=0. These are configured by a DIP switch on the target board.

The configuration pins TCK and TMS in XMC4000 are usually connected to a DIP switch on the XMC4000 board.

Assuming that TMS is connected to switch pin 1 and TCK connected to switch pin 2, the DIP switch configuration is shown in Figure 2.

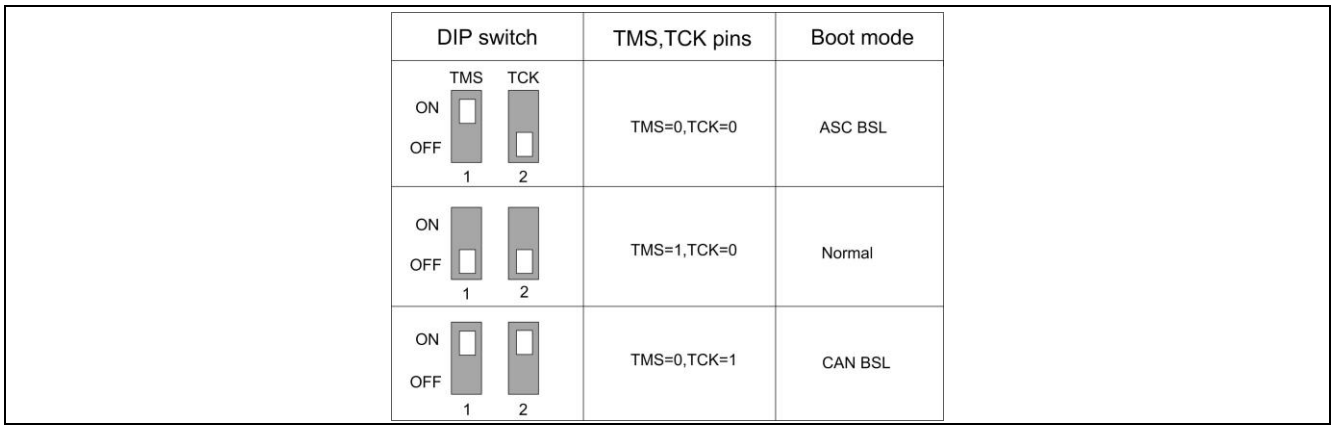


Figure 2 DIP switch configuration of boot modes

The bootstrap loader procedure is shown in Figure 3.

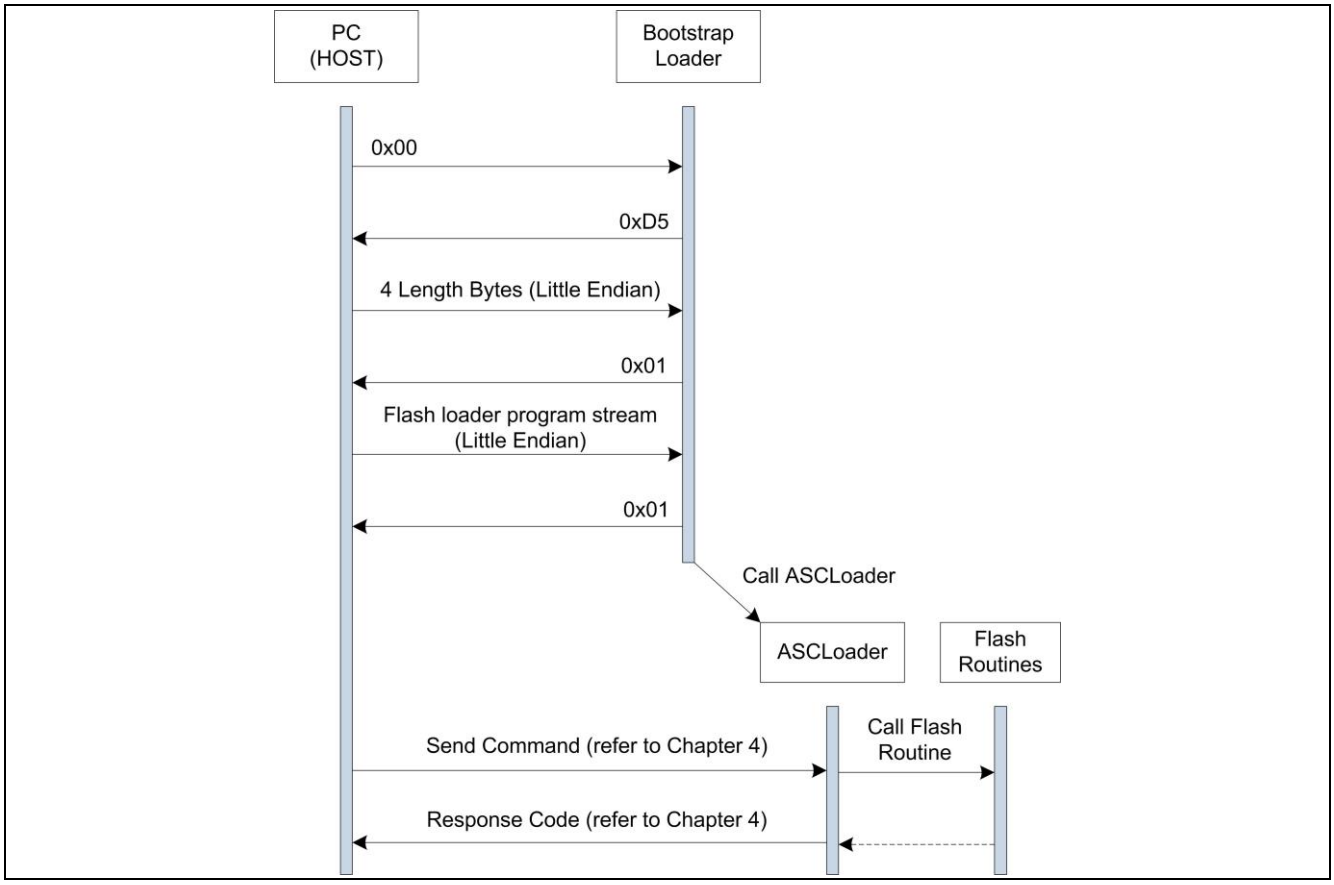


Figure 3 ASC Bootstrap loader procedure for Flash programming

The HOST starts by transmitting a zero byte to help the device detect the baud rate. The XMC4000 device supports baud rates of up to 115200 bits/s. The ASC interface will be initialized for 8 data bits and 1 stop bit.

After the baud rate is detected by the device, the bootstrap loader transmits an acknowledgement byte D5H 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, an 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 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 PSRAM size), a N_OK byte (0x02H) is transmitted back to the HOST and the BSL resumes it's wait for the correct length of bytes.

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

2.1 Flash Loader

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

- Erase Flash sectors
- Program Flash pages
- Verify a programmed Flash page

- Set flash protection
- Remove flash protection
- Read flash protection status
- Run the codes from both Flash and iCache

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 XMC4000 memory organization is described in the

Flash Memory Organization chapter.

Flash user codes can be executed starting from the Flash base address 0xC0000000 and the iCache base address 0x08000000.

2.2 DAVE3 Project Settings

The Flash loader DAVE3 project is available in the .\DAVE3\ASCLoader folder. The project can be imported into the DAVE3 IDE with the following steps:

- Open the DAVE3 IDE
- Import the Infineon DAVE project
- Select root directory as .\DAVE3\ASCLoader
- Finish the import

Note: The Flash Loader program must be located in the PSRAM starting at 0x10000000 (XMC4500) or 0x1FFFC000 (XMC4400/4200) because the Flash Loader program can only run from PSRAM. Therefore the default linker script file generated from DAVE3 cannot be used in the Flash loader project, because the default linker script file locates the codes in iCache starting at 0x80000000. The linker script file that locates the codes into PSRAM is provided in the XMC4x00_PSRAM.ld filer. 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 XMC4x00_PSRAM.ld into the field

The Linker Script Language file XMC4x00_PSRAM.ld, defines the ROM memory for codes in PSRAM starting from address 0x10000000 (XMC4500) or ox1FFFC000 (XMC4400/4200).

The stack, heap and global variables are located in DSRAM starting from address 0x20000000.

2.3 Keil Project Settings

The Keil project for Flash loader is available in the folder .\Keil\ASCLoader. The Keil compiler version is v4.7. The project can be imported into Keil µVersion as follows:

- Go to project->open project
- Go to folder .\Keil\Flash_Loader->choose project file “ASCLoader.uvproj”->open

Because the Flash Loader must be run from PSARM, the memory should be defined as follows:

- Go to Target
- IROM1 start 0x10000000 (XMC4500) or 0x1FFFC000 (XMC4400/4200), size 0x10000 (XMC4500) or 0x4000 (XMC4400/4200), Startup->yes
- IRAM1 start 0x20000000, size 0x10000

By default the Keil compiler generates the object file with ELF-format and the file extension .axf. But, the Flash loader needs HEX-format file. In order to get HEX file output go to:

- Open Option->Output->Create HEX file

2.4 IAR Project Settings

The IAR project for Flash loader is available in the folder .\IAR\ASCLoader. The compiler version v6.5.2. The project can be imported into an IAR Embedded Workspace IDE as follows:

- Open IAR Embedded Workspace
- Go to Project->Add Existing Project
- Go to folder .\IAR\ASCLoader->choose project file “ASCLoader.ewp->open

The IAR compiler uses a Linker script to locate the memory. In the .\config folder two linker script files are provided; XMC4500_Flash.icf and XMC4500_RAM.icf.

The file XMC4500_RAM.icf should be used to locate the codes in PSRAM.

To change the linker script file go to:

- Open Project Options->Linker->Config->Override default
- Go to folder .\config->Open “XMC4500_RAM.icf”->OK

In the file XMC4500_RAM.icf the ROM and Ram are defined:

- ROM_start = 0x10000000; ROM_end = 0x1000FFFF;
- RAM_start = 0x20000000; RAM_end = 0x2000FFFF;

The linker script files for XMC4400/4200 can be modified in similar way. The codes are located in PSRAM starting at 0x10000000 (XMC4500) or ox1FFFC000 (XMC4400/4200).

The stack, heap and global variables are located in DSRAM starting from address 0x20000000.

By default the IAR compiler generates the object file with ELF-format and the file extension .out. To generate HEX file output go to:

- Open Project Options->Output Converter->Generate additional output
- Select Output format as “Intel extended”
- Select “Override default”->OK

2.5 Modification of startup.s File

Attention: ***It is important to note that all clock setting functions in the startup_XMC4x00.s file used in all ASCLoader projects with different compilers, 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 DAVE3 startup_XMC4500.s file must be removed:***

- LDR R0, =SystemInit
- BLX R0
- LDR R0, =SystemInit_DAVE3
- BLX R0

These instructions must be removed because the functions SystemInit() and SystemInit:DAVE3() 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.

All startup.s files provided in the ASCLoader projects have been modified and the system init functions are removed.

3 Flash Memory Organization

The embedded Flash module in the XMC4x00 family includes maximal 1.0 MB of Flash memory for code or constant data (called Program Flash). The PMU contains one PFLASH bank, accessible via the cacheable or non-cacheable address space.

PFlash 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 1**. Derived devices (see the XMC4000 Data Sheet) can have less Flash memory. The PFLASH bank shrinks by cutting-off higher numbered physical sectors.

Table 1 Flash Memory Map

Range Description	Size	Start Address
PMU0 Program Flash Bank non-cached	1 Mbyte (XMC4500) 512 Kbyte (XMC4400) 256 Kbyte (XMC4200)	0xC000000H
PMU0 Program Flash Bank cached space (different address space for the same physical memory, mapped in the non-cached address space)	1 Mbyte (XMC4500) 512 Kbyte (XMC4400) 256 Kbyte (XMC4200)	0x8000000H

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

The following table lists the logical sector structure in the XMC4x00 family of products.

3.1 XMC4500

In XMC4500 the flash module PMU0 contains 1 MB Pflash memory. **Table 2** lists the flash logical sector structure in XMC4500.

Table 2 Sector Structure of PFLASH in XMC4500

Sector	Address Range	Size
0	0xC000000-0xC003FFF	16 KB
1	0xC004000-0xC007FFF	16 KB
2	0xC008000-0xC00BFFF	16 KB
3	0xC00C000-0xC00FFFF	16 KB
4	0xC010000-0xC013FFF	16 KB
5	0xC014000-0xC017FFF	16 KB
6	0xC018000-0xC01BFFF	16 KB
7	0xC01C000-0xC01FFFF	16 KB
8	0xC020000-0xC03FFFF	128KB
9	0xC040000-0xC07FFFF	256 KB
10	0xC080000-0xC0BFFFF	256 KB
11	0xC0C0000-0xC0FFFFF	256 KB

3.2 XMC4400

In XMC4400 the flash module PMU0 contains 512 KB Pflash memory. **Table 3** lists the flash logical sector structure in XMC4400.

Table 3 Sector Structure of PFLASH in XMC4400

Sector	Address Range	Size
0	0xC000000-0xC003FFF	16 KB
1	0xC004000-0xC007FFF	16 KB
2	0xC008000-0xC00BFFF	16 KB

Sector	Address Range	Size
3	0xC00C000-0xC00FFFF	16 KB
4	0xC010000-0xC013FFF	16 KB
5	0xC014000-0xC017FFF	16 KB
6	0xC018000-0xC01BFFF	16 KB
7	0xC01C000-0xC01FFFF	16 KB
8	0xC020000-0xC03FFFF	128KB
9	0xC040000-0xC07FFFF	256 KB

3.3 XMC4200

In XMC4200 the flash module PMU0 contains 256 KB Pflash memory. **Table 4** lists the flash logical sector structure in XMC4200.

Table 4 Sector Structure of PFLASH in XMC4200

Sector	Address Range	Size
0	0xC000000-0xC003FFF	16 KB
1	0xC004000-0xC007FFF	16 KB
2	0xC008000-0xC00BFFF	16 KB
3	0xC00C000-0xC00FFFF	16 KB
4	0xC010000-0xC013FFF	16 KB
5	0xC014000-0xC017FFF	16 KB
6	0xC018000-0xC01BFFF	16 KB
7	0xC01C000-0xC01FFFF	16 KB
8	0xC020000-0xC03FFFF	128KB

4 Communication Protocol

The Flash loader program “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	Byte 1	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	Byte 1	Bytes 2…257	Bytes 258…262	Byte 263
Block Type (0x01)	**Verifi-cation 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 ASCLoader. The modes and their corresponding communication protocol are described in the following sections of this chapter.

4.1 Mode 0: Program Flash Page

Header Block

Byte 0	Byte 1	Bytes 2…5	Bytes 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	Byte 1	Bytes 2…257	Bytes 258…262	Byte 263
Block Type (0x01)	**Verifi-cation 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.

4.2 Mode 1: Execute User Program from Flash

Header Block

Byte 0	Byte 1	Bytes 2…14	Byte 15
Block Type (0x00)	**Mode (0x01)**	**Not Used**	**Checksum**

The command causes a jump to the Flash base address **0xC000000**. The device exits BSL mode after sending **0x55** as acknowledgement.

4.3 Mode 2: Execute User Program from iCache

Header Block

Byte 0	Byte 1	Bytes 2…14	Byte 15
Block Type (0x00)	**Mode (0x02)**	**Not Used**	**Checksum**

The command causes a jump to the iCache base address **0x8000000**. The device will exit BSL mode after sending **0x55** as acknowledgement.

4.4 Mode 3: Erase Flash Sector

Header Block

Byte 0	Byte 1	Bytes 2…5	Bytes 6…9	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 (see Chapter 0). 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 (see Chapter 0).
  - 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.

4.5 Mode 4: Read Flash Protection Status

Header Block

Byte 0	Byte 1	Bytes 2…14	Byte 15
Block Type (0x00)	**Mode (0x04)**	**Not Used**	**Checksum**

- The command requires flash protection status. The device exits BSL mode after sending **0x55** as “flash unprotected” or **0xF8** as “flash protected”.
  -

4.6 Mode 5: Protect or Unprotect Flash

Header Block

Byte 0	Byte 1	Bytes 2…5	Bytes 6…9	Bytes 10	Bytes 11…12	Bytes 13…14	Byte 15
Block Type (0x00)	**Mode (0x05)**	**User Password 1**	**User Password 2**	**Flash Module**	**Protection Config**	**Not used**	**Checksum**

UserPassword1 (32bit): First user password. Byte 2 indicates the highest byte while Byte 5 indicates the lowest byte.

UserPassword2 (32bit): Second user password. Byte 6 indicates the highest byte while Byte 9 indicates the lowest byte.

FlashModule: Reserved

ProtectionConfig (16bit): Selection of the flash sectors to be protected. The protection configuration word has the following structure:

ProtectioConfig bit scheme

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Sn = 0: Sector n will not be protected.

Sn = 1: Sector n will be protected.

Note: In the case that sector n does not exist, Bit Sn should be set to 0. Please refer to ***Chapter 3*** *for detailed information about the flash sectorization.*

If the Flash is unprotected, it will be protected after sending this header block. The same block sent with the same passwords to a flash-protected device will unprotect the Flash. All erase or program commands sent to a flash-protected device will cause a protection error.

Attention: After sending the flash protect/unprotect command the device needs to be reset in order to make the command valid.

4.7 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 5 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

5 HOST PC Program Example

The XMC4000_Bootloader HOST program developed in C++ uses the communication structure described in **Chapter 4**.

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

Table 6 API Functions

API Function	Description
init_uart	Initialize PC COM interface
init_ASC_BSL	Initialize ASC BSL
send_loader	Send the ASCLoader
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 PFlash sectors
bl_download_pflash	Download code to PFlash
make_flash_image	Create a Flash image from HEX file

The main program (XMCLoad.cpp) initializes ASC and sends ASCLoader to the target device.

The user must specify the HEX file to be downloaded. An example HEX file (Blinky.hex) is provided. The user code is first downloaded to Flash and the user can then execute the downloaded code from both Flash and iCache.

- The Flash erase procedure, as shown in **Figure 4** is implemented in the function bl_erase_flash().
- The Flash programming procedure, as shown in **Figure 5**, is implemented in bl_download_pflash().

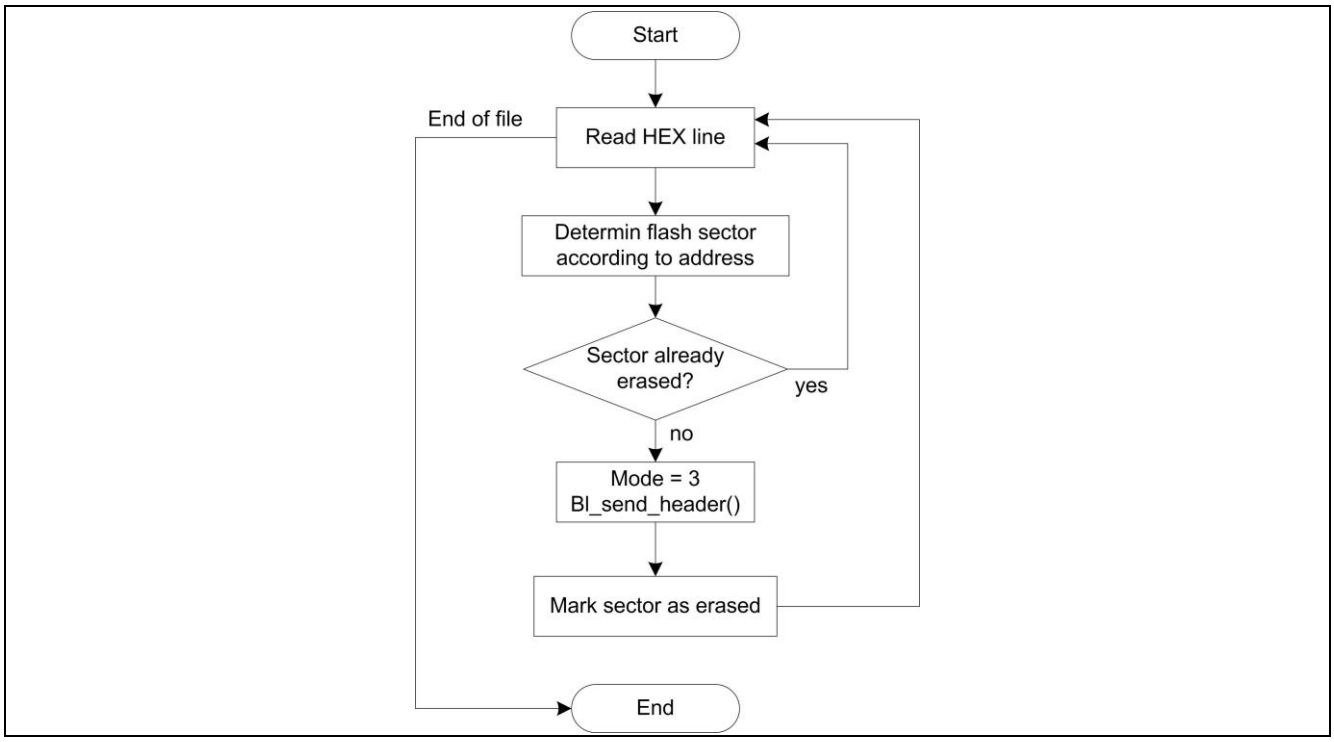


Figure 4 Flash erase procedure implemented in bl_erase_flash()

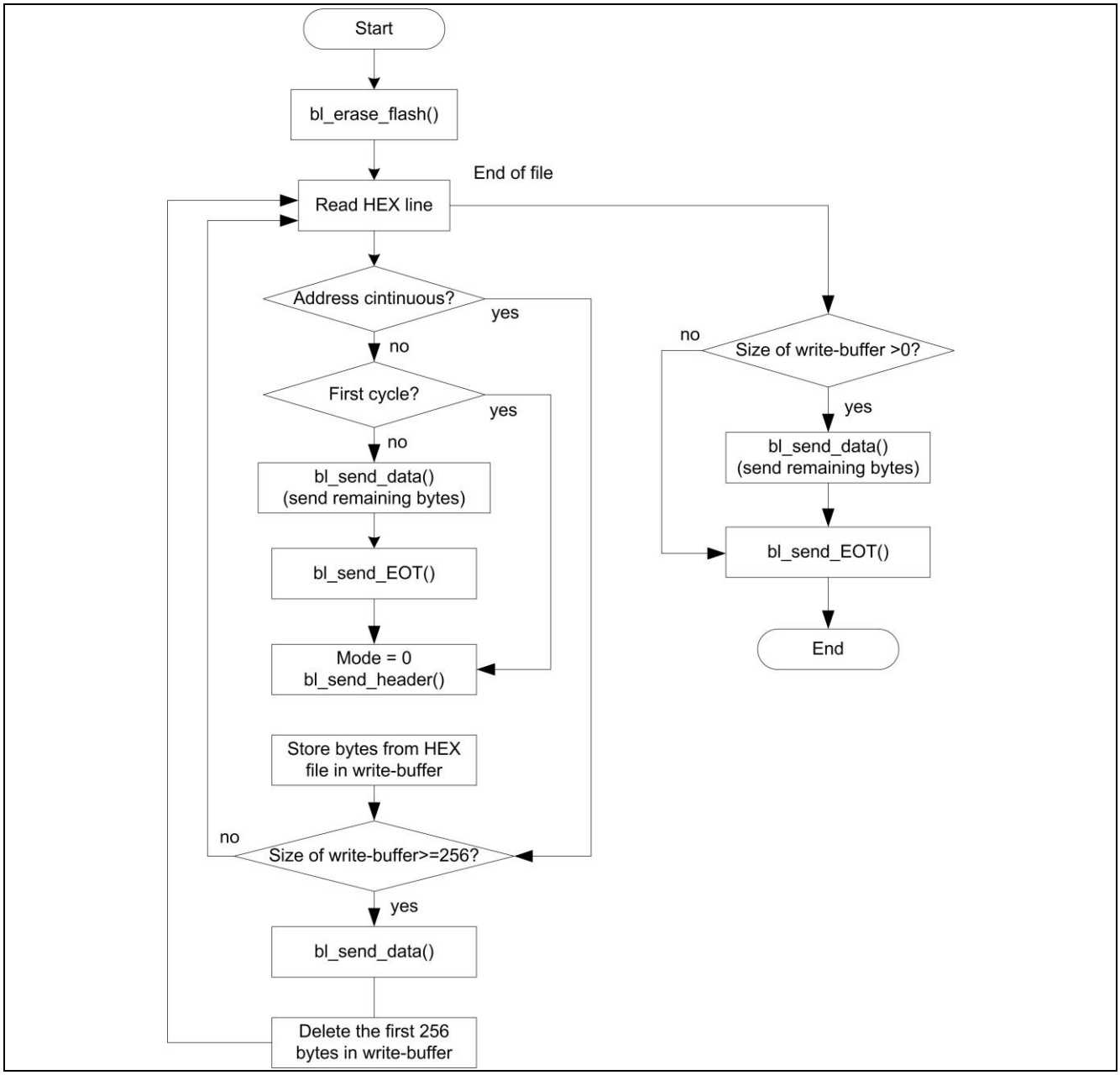


Figure 5 Flash programming procedure implemented in bl_download_pflash()

6 Using the Demonstrator

The example programs have been tested on an Infineon XMC4500 Hexagon board with XMC4500 AB step (A14). 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.

6.1 Hardware Setup

The ASC output Pin of the XMC4500 Hexagon board has just 3.3v, but the PC ASC output usually has 5v. In order to set up the communication between PC and XMC4000 board through the ASC interface, a voltage adapter (such as the Infineon Xspy-Adapter), is required to adjust the voltage difference.

If the Infineon Hexagon board with XMC4000 device is used in the test, the following hardware setup is required:

- Set the DIP switch jump on board as jump 1 (ON) and jump 2 (OFF) for ASC bootstrap load mode.
- Use an adapter to connect the on-board ASC interface to the PC ASC interface.

6.2 Demonstrator File Structure

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

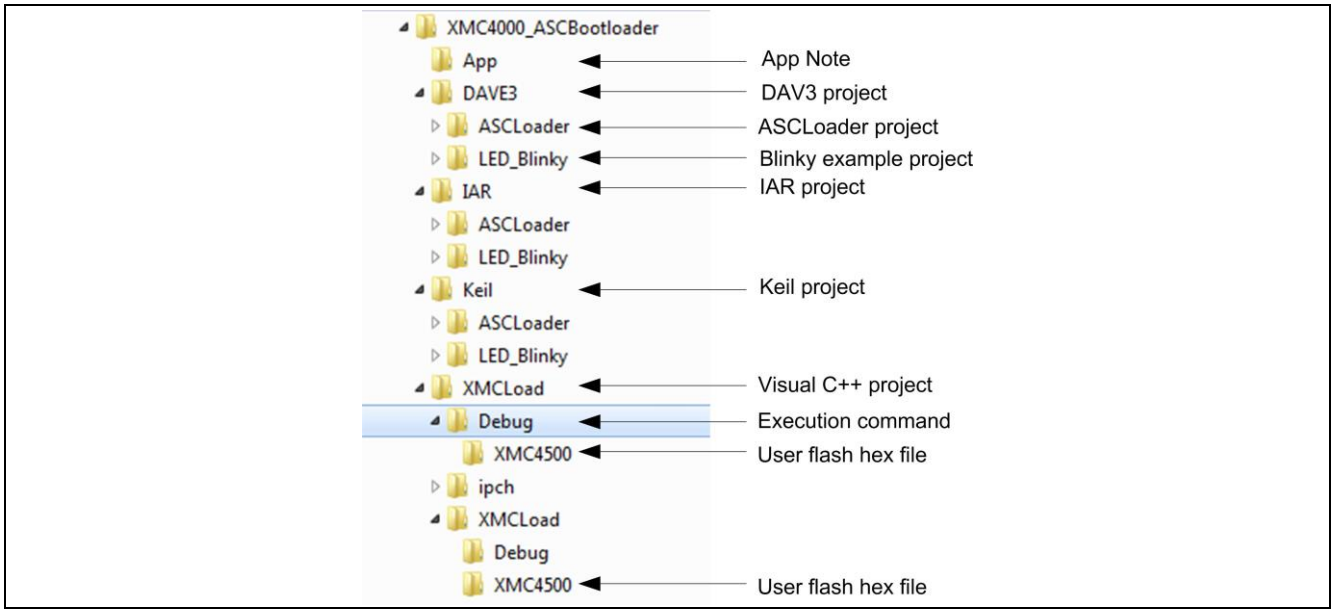


Figure 6 File structure of example programs

- This application note is contained in folder .\App.
- The folders .\DAVE3, .\IAR and .\Keil are the projects generated using the different compilers.
- The folders \XMC4500, \XMC4400 and \XMC4200 are the device folders, where the corresponding BSL flash loader program saved.
- ASCLoader project contains the ASC bootstrap loader program
- The LED_Blinky project is the example project for LED blinking
- .\XMCLoad contains the Microsoft Visual C++ 2010 project for the Host PC.

- The ASCLoader.hex and example LED Blinky hex files are saved in.\XMCLoad\Release\XMC4x00 and.\XMCLoad\XMCLoad\XMC4x00, separately.

6.3 Run the Demonstrator

Before starting the demonstrator, the hex file that needs to be downloaded into Flash should be copied into the folders .\XMCLoad\Debug\XMC4x00 and .\XMCLoad\XMCLoad\XMC4x00, depending on which device is used. For example, if the XMC4500 device is used, the hex file should be copied like:

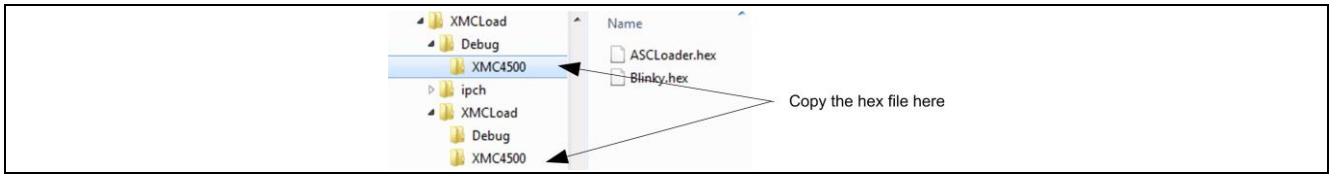


Figure 7 Location of object hex files to be flashed

There are two ways to start the demonstrator.

1. Double click the file XMCLoad.exe under .\XMCLoad\Release:

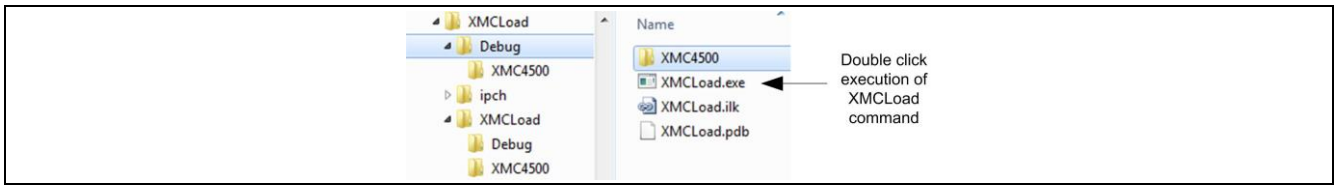


Figure 8 Direct start of demonstrator

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

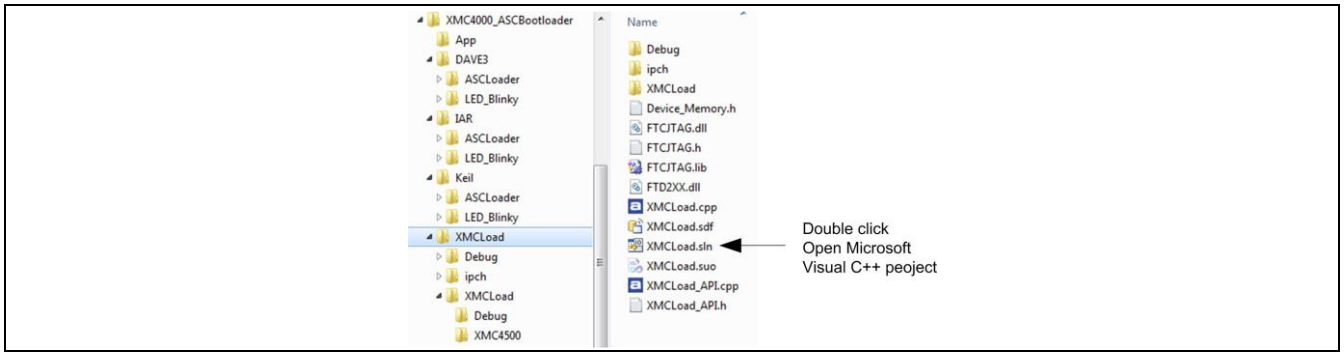


Figure 9 Start using Microsoft Visual project

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

On starting the demonstrator the following window is displayed:

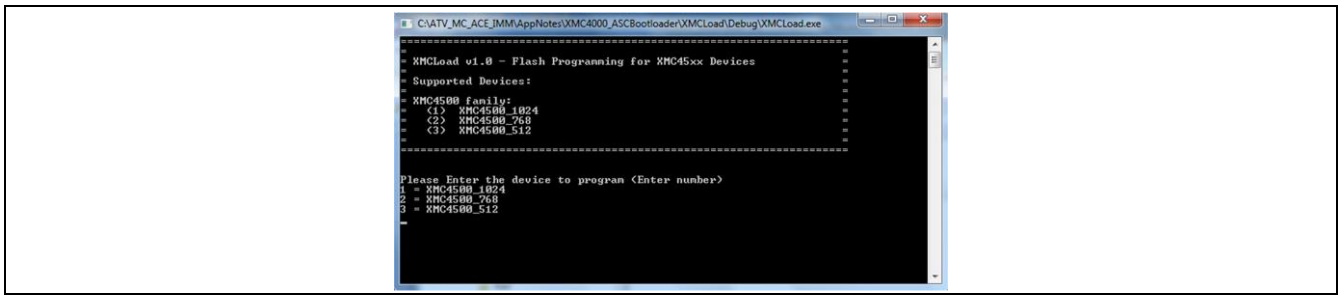


Figure 10 Start Window from Visual Project

Follow the instructions in the window to finish the Flash programming, set flash protection or remove the flash protection.

Note: The hex file name that will be programmed into Flash must be given completely with the file extension; e.g. Blinky.hex. Otherwise, the program does not know the file name. The Flash loader program accepts only hex file format. Furthermore, the ACLoader.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, the program can be executed from both Flash and iCache.

7 Reference Documents

Table 7 References

Document	Description	Location
XMC4500 Reference Manual	Hardware Reference	http://www.infineon.com/xmc4000
TriCore AUDO-F Flash Download Using Bootstrap Loader	Application Note AP32132	http://www.infineon.com/xmc4000

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG