AP16110

XC2000 Family

Programming the on-chip Flash using the SSC Bootstrap Loader

Microcontrollers

Never stop thinking

Edition 2007-09

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED 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.

Information

For further information on technology, delivery terms and conditions and prices please contact your 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 your 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.

AP16110		
Revision History:	2007-09	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents Page

1 Abstract

Most microcontrollers have non volatile flash memory, where application code and data (user code) are stored. During production this flash memory has to be programmed. This can either be done in system or on a separate programming stage prior to assembling the system.

This application note describes an implementation of a tool capable of programming a microcontroller's internal flash memory in mass production applications, costing no more than 10,-€. The tool has the size of a memory stick and can program the controller in-system, rather than requiring the chip to be programmed prior to installing it into the system. The primary advantage of this feature is that it allows manufacturers to program the controllers in their own system's production line. A separate and expensive programming stage is not required.

The **hardware** of this programming tool is basically a serial flash memory. It contains the user code, which is to be programmed. The target microcontroller itself programs this user code into its internal flash memory. The required programming routines are stored in an EEPROM, which is also part of the tool's hardware. The microcontroller uses the SPI protocol to communicate with the two memory devices. A power supply, a configuration switch and a signal LED complete the programming tool.

The tool's **software** distinguishes between two separate operating modes.

The first operating mode is the **setup mode**. This part of the software is needed to set up the programming tool. The programming routine must be stored in the EEPROM and the user code must be stored in the serial flash memory. The approach taken here is to use a XC2000 microcontroller starter kit, and Infineon's Memtool. Memtool is a software programming tool normally used to program a controller's on-chip flash memory. The standard driver is being adapted here in order to program memory devices on the SPI interface instead.

The second operating mode is the **programming mode**. Here the target microcontroller's on-chip flash memory is programmed. The required programming routine is downloaded from the EEPROM and executed automatically by the microcontroller. The programming software then reads user data from the serial flash memory, and stores it in the controller's internal flash memory. In order to assure correct programming a CRC value is calculated and compared to the value stored in the serial flash. Success or failure of the programming process is signaled by the LED.

2 Hardware

2.1 Description

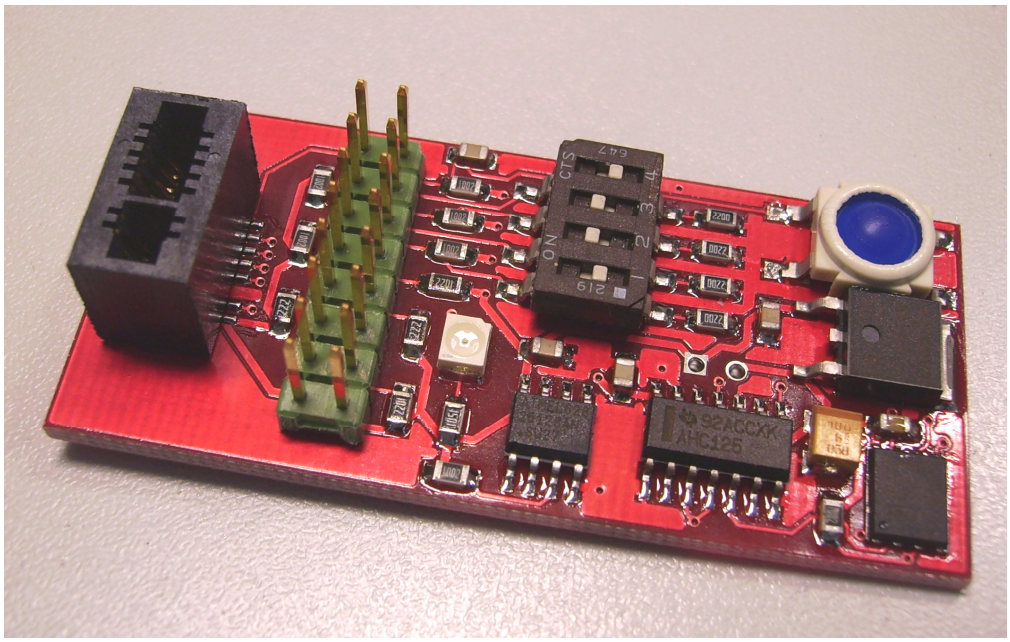


Figure 1 The SSC Programmer (Top View)

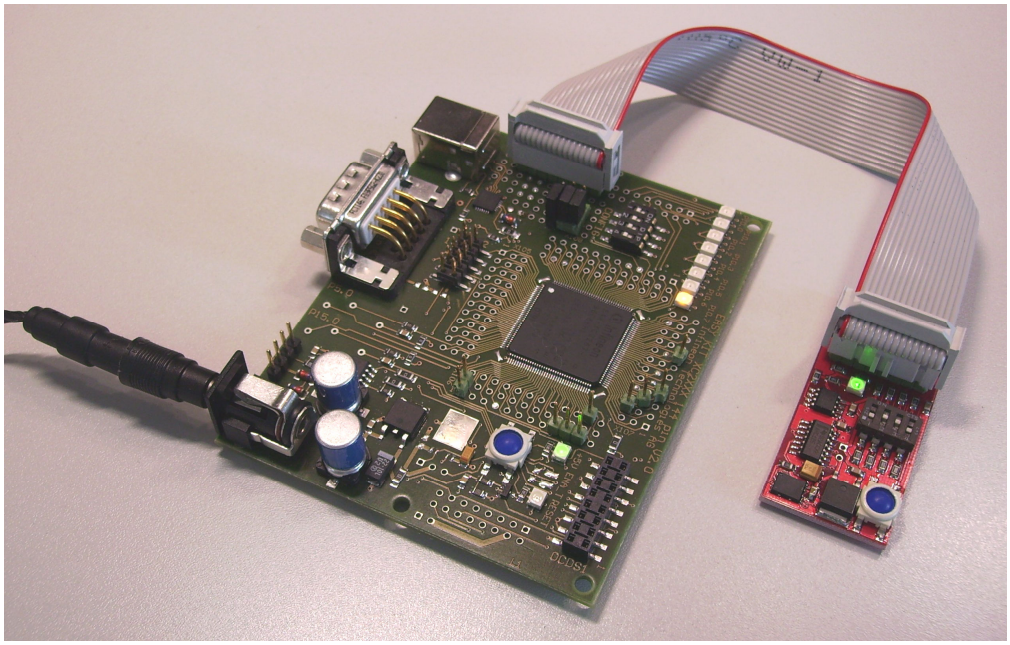


Figure 2 SSC Programmer connected to XC2000 Starter Kit

2.2 Schematic and Technical Details

The main parts of the tool's hardware are the 1 Mbyte serial flash (M25P80) and the 128 Kbyte EEPROM (AT25128N). The buffer gate (SN74AHC125) works as interface between the microcontroller and the serial flash module. It is required here because the M25P80 operates with 3.3 V, while the microcontroller uses a 5 V signal level. The pull-up resistors R3 – R6 provide safe and stable signal levels at all times. This is especially important for the two chip select lines; /CS is a low-active signal, the pull-up resistors assure that both memory devices are disabled while the microcontroller does not actively drive a low signal (e.g. during reset).

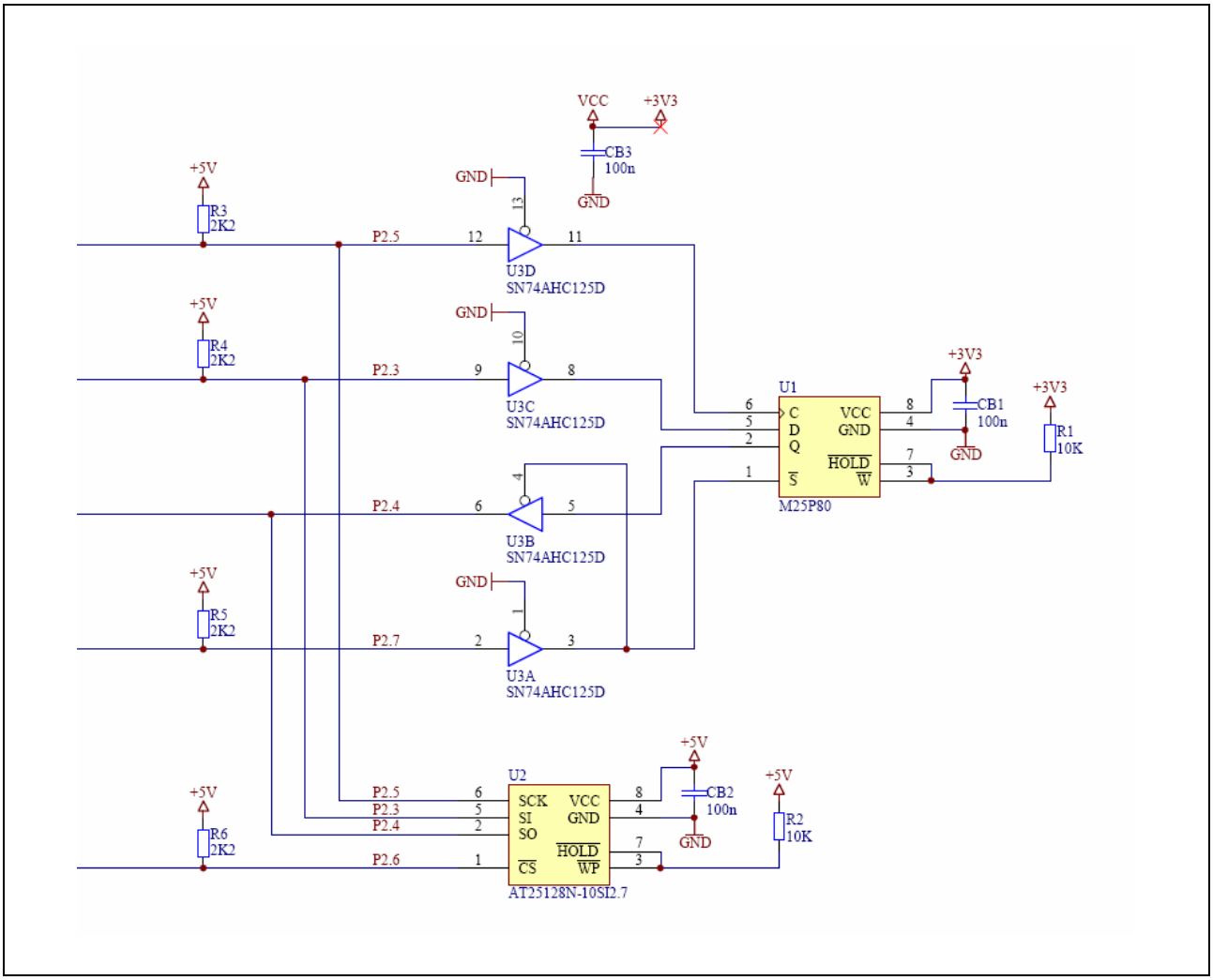


Figure 3 Schematic: Memory Devices and Buffer Gate

The configuration switch (S1) together with the pull-up and pull-down resistors R8 - R15 provides the boot configuration for the microcontroller. By applying a certain pattern to port 10 of the XC2000, the required boot mode (SSC for programming, ASC for setup) can be selected.

Depending on the intended target hardware it may be necessary to adjust the values of these pull-up and pull-down resistors in order to provide the system with stable logical values during reset. Information on the required logical values must be taken from the respective user's manual or data sheet. The XC2000 microcontroller rates a voltage level of less than 1.5 V as "low", while more than 3.5 V is considered "high". Here the resistors' values are chosen in such a way as to provide compatibility to the XC2000 starter kit.

The reset button (S2) allows the user to start the microcontroller according to the provided boot configuration.

In this example the signal LED (D1) is connected to port 10.7, but any other unused general purpose I/O port could also be used. A low signal level on the port turns the LED on.

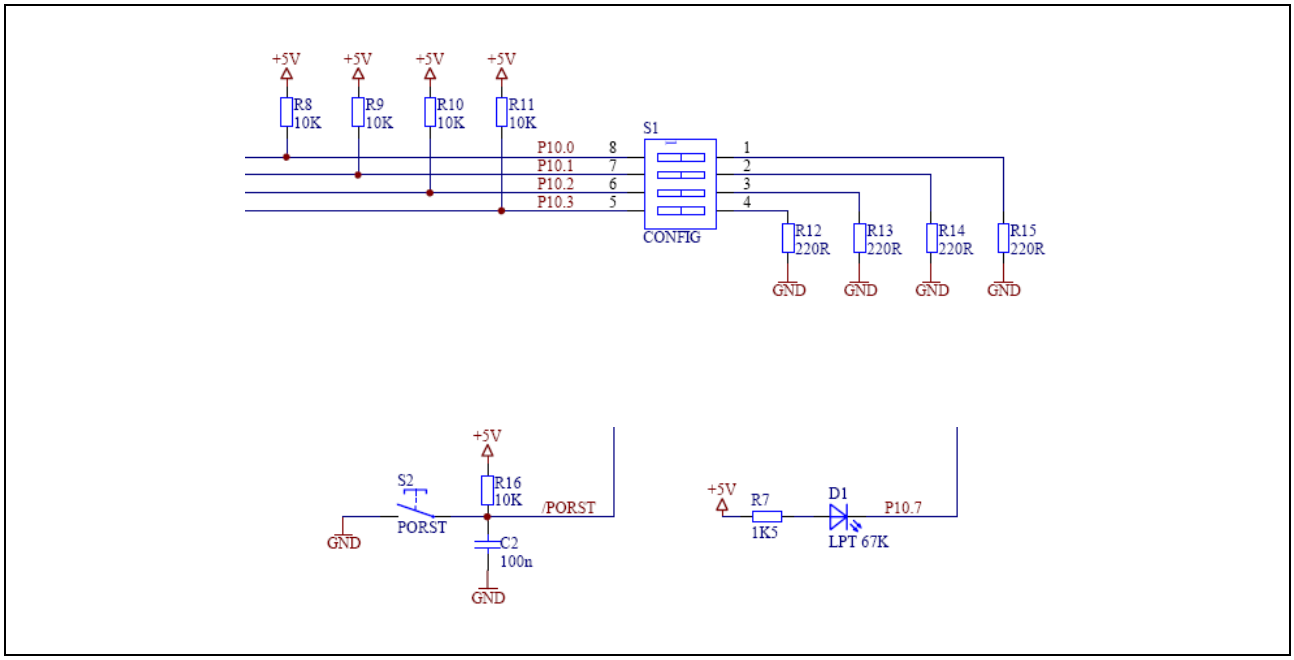


Figure 4 Schematic: Configuration Switch, Reset Switch, Signal LED

The programmer comes with two separate connectors, a 16-pin header, and an optional PCB connector. It is up to the user to provide the necessary means to connect the programmer to the target system, either directly on-board, or by using a suitable adapter cable.

For the complete schematic see the appendix (**6.6**).

2.3 Description of the SPI Bus

The serial peripheral interface bus (SPI) is a relatively simple synchronous serial interface for connecting external devices to a microcontroller using only four signal lines. SPI is a de facto standard defined by Motorola on the MC68HCxx line of microcontrollers. A synchronous clock shifts serial data into and out of a device in blocks of 8 or 16 bits.

The SPI bus is a master / slave interface. Whenever two devices communicate, one is referred to as the "master" (in this case the XC2000 microcontroller is always the master) and the other as the "slave" device. The master provides the serial clock and controls the chip select lines. It is important to understand that data is simultaneously transmitted and received, making SPI a full-duplex protocol.

The naming of the SPI signals varies depending on the device manufacturer; Table 1 shows the corresponding signals of all used devices, as well as the original Motorola names.

Table 1 SPI Bus Naming Conventions

Motorola Naming	Infineon XC2000	ST M25P80	Atmel AT25128N
MISO	MRST	Q	SO
MOSI	MTSR	D	SI
SCLK	SCLK	C	SCK
/SS	/CSx	/S	/CS

Figure 5 shows the general connection scheme for a single master and two slaves SPI bus application. Each slave has one dedicated chip select line, the other three signal lines are shared between the slave devices.

In order to avoid data collision on the MRST (Master Receive Slave Transmit) line the master has to assure that only one chip select line is active at a time. Slaves with inactive /CS lines ignore inputs; output lines are set to high impedance.

During each clock cycle one bit is shifted into the slave device on the rising edge, while one bit is shifted out on the falling edge of the clock. This means that each time a byte is transmitted also a byte is being received. The SPI master is responsible for handling the received data.

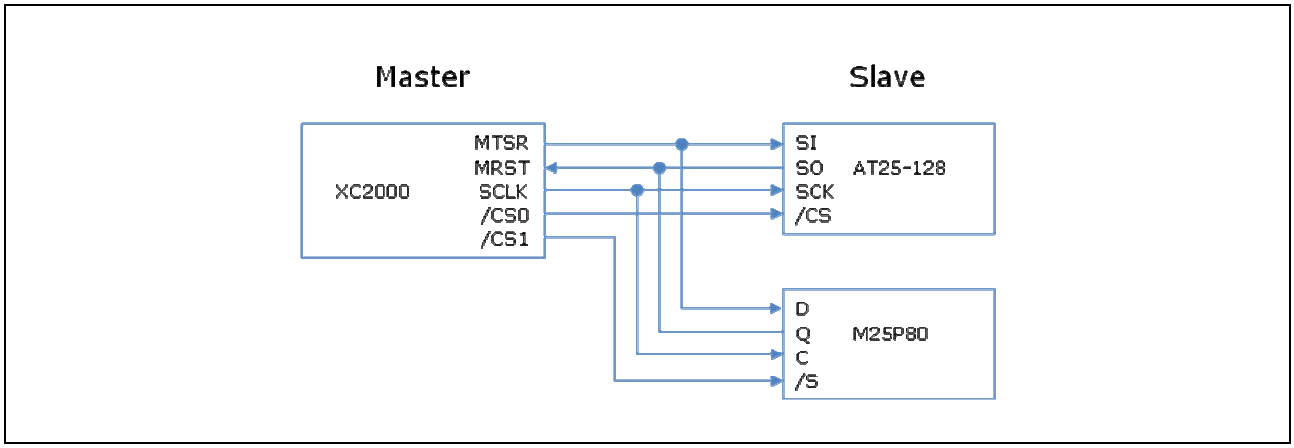


Figure 5 SPI Bus: Single Master and two Slaves

3 Software

Basically there are two operating modes, the setup mode and the programming mode.

The **setup mode** is used to configure the programming tool. The programming routine must be stored in the EEPROM and user code, which is to be programmed to the target device later, must be stored in the serial flash module. For this purpose Memtool, a software programming tool provided by Infineon, is used. Also during the setup process additional control information is stored in the serial flash, most important a calculated CRC value.

In the **programming mode** the previously stored user code is loaded from the serial flash module and programmed into the microcontroller's internal flash memory. This is done by the programming routine stored in the EEPROM. The routine is downloaded and run automatically after reset by the bootstrap loader. The programming process is being verified by calculating a CRC value and comparing it to the stored value.

3.1 Setup Mode

In the setup mode the programming tool itself is being programmed. First the loader software has to be stored in the EEPROM (step 1). Then the user code must be stored in the serial flash module (step 2). User code is the software that will be programmed to the target microcontroller later (see chapter 3.2). The setup steps can be done several times by the user, depending on application requirements.

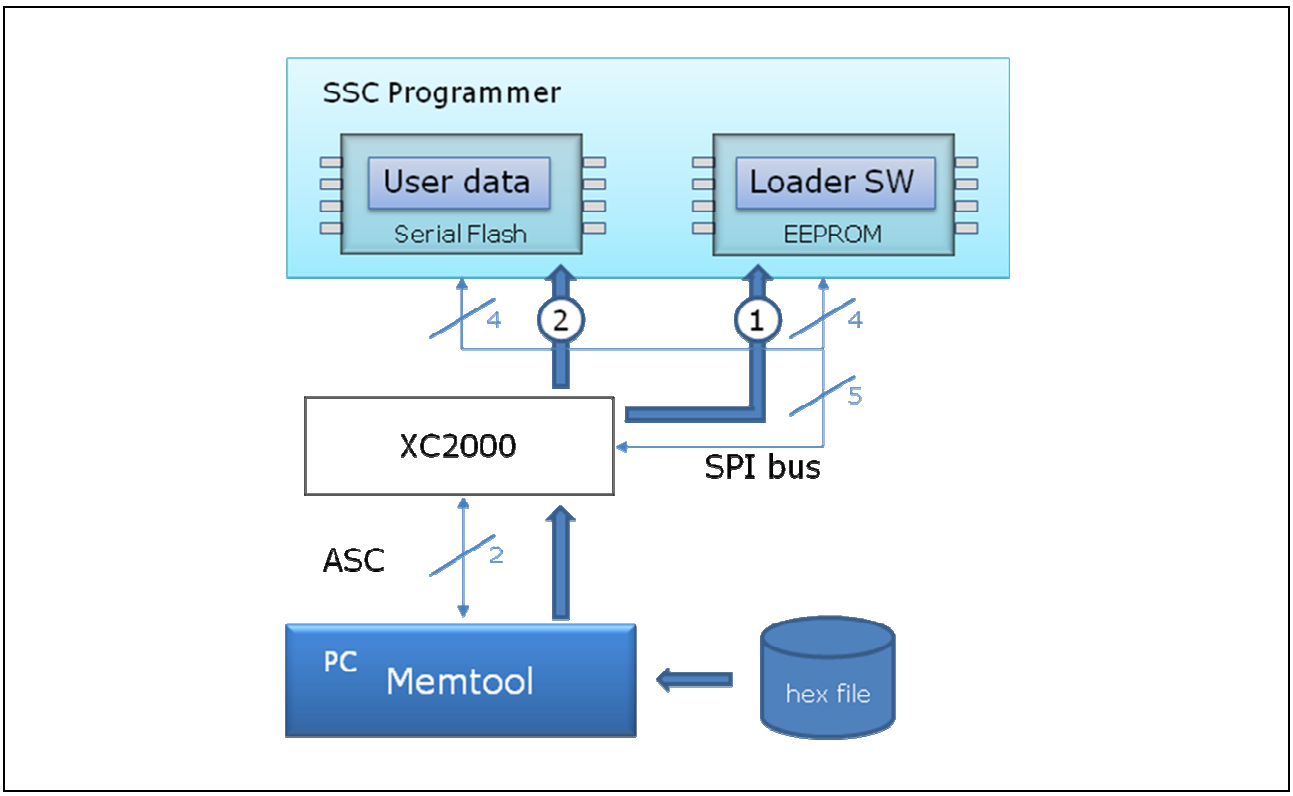


Figure 6 Data Flow in Setup Mode

3.1.1 Memtool

The Infineon on-chip memory programming tool Memtool is intended to handle the on-chip flash and OTP memory devices of Infineon microcontrollers. According to the capabilities of the respective on-chip memory device, the program can erase, program, verify and protect the module. To program a memory device one

can open an Intel-Hex file and completely or partly write its contents into the memory device. Infineon Memtool can be used on PC's running Microsoft Windows™. The target is connected to the host PC using a standard RS232 port.

During the programming process, Memtool:

- sets up and manages the communication between the host PC and the microcontroller by downloading the so called monitor
- downloads the driver to the controller's RAM
- downloads the hex-file to be programmed into a transmit buffer area in the controller's RAM
- refills the transmit buffer if required until all data is programmed
- calls the required functions from the driver.

While the driver:

- initializes the microcontroller
- provides the necessary routines to erase or program the on-chip memory devices.

The actual programming works in two steps. First Memtool fills the transmit buffer area with data from the Intel-Hex file. Then Memtool calls the programming function of the driver and provides all required information like target address and number of bytes to program in system registers. The driver's programming routine now transfers the contents of the transmit buffer to the provided destination address.

3.1.2 Memtool Driver Implementation

In order to write data to the external serial flash or EEPROM, the standard driver has been modified. There are two separate drivers for the programming tool, one for programming the serial flash module and one for programming the EEPROM. The original routines for programming the on-chip memory have been replaced by routines for programming the external SPI devices.

Figure 7 shows the memory mapping used by the standard driver. Here data is stored in the on-chip flash memory ($C0'0000_H$ to $CB'FFFF_H$). The user code is mapped to this area, which means that the code is designed to be executed directly from the flash memory, code execution starts at the address $C0'0000_H$.

The modified drivers write data targeted to $C0'0000_H$ - $CB'FFFF_H$ to the serial flash module starting from address $00'0000_H$ there. Data targeted to the internal RAM ($E0'0000_H$ – $E1'0000_H$) is written to the EEPROM.

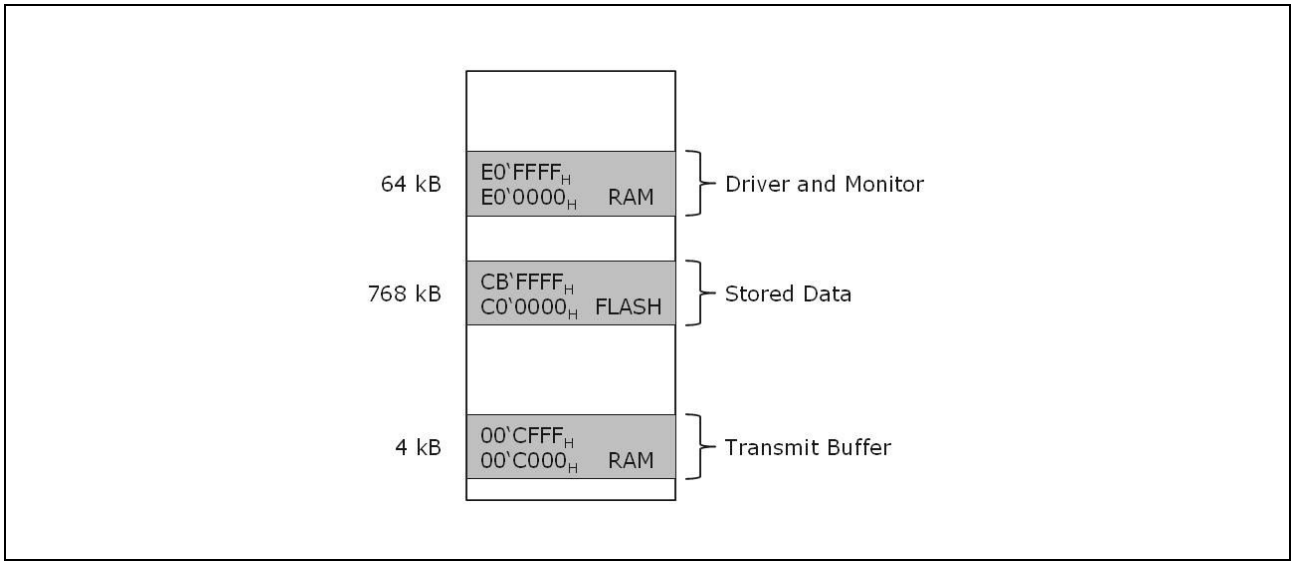


Figure 7 Memory Map

The modified programming routine first sets up the communication with the SPI device using channel 1 of USIC module 0 (U0C1). Then the target address is modified to match the address space of the SPI device. Data is written in blocks, the block size depends on the SPI device used. For the serial flash module a block size of 256 bytes is used, for the EEPROM 64 bytes per block is used.

The driver for the serial flash module also calculates the CRC value during data storage. After the last byte of the transmit buffer is transferred, this CRC value is stored in the 16 byte info block. For details on the info block see chapter 3.2.3.

3.1.3 CRC-Algorithm

A cyclic redundancy check (CRC) is a type of hash function used to produce a checksum against a block of data. The checksum is used to detect errors after transmission or storage. In this case the CRC value is computed during the setup process of the programming tool and afterwards stored in the info block. It is verified after the data has been stored in the microcontroller's internal flash (see chapter 3.2) to confirm that no data errors occurred on transit.

Table 2 CRC Definition

Length	Polynomial	Direction	Preset
16 bits	$x^{16} + x^{12} + x^5 + 1$ (8408_H)	Backward	$FFFF_H$

There are two separate implementations for the CRC algorithm here. The first one is written in assembly language and is part of the Memtool driver. It computes the CRC value and stores it in the info block in the serial flash memory during the setup process. The second implementation is written in C language. It is part of the main programming algorithm, and calculates the CRC value after the user data is stored in the microcontroller's internal flash. Both implementations follow the flow in Figure 8.

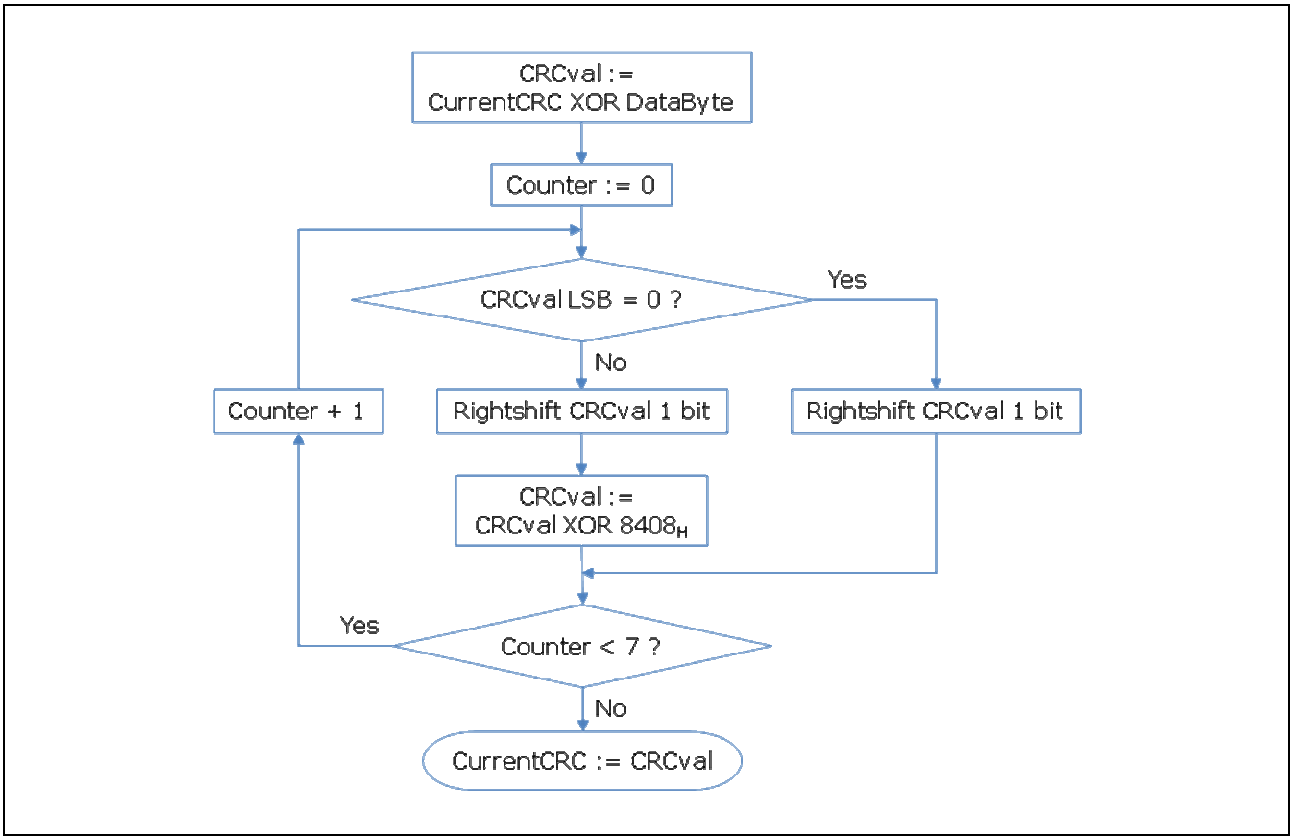


Figure 8 CRC Flow Chart

3.1.3.1 Assembly Language Implementation

```
;-------------------------------------------------------------------------------
;Calc_CRC
;IN:           Data in Register R1
;              Current CRC Value on address 0xD002
;OUT:          New CRC Value on address 0xD002
;DESCRIPTION:  calculates CRC using polynomial x^16+x^12+x^5+1
;              (0x8408 backward, start value 0xFFFF)
;-------------------------------------------------------------------------------
Calc_CRC PROC NEAR
     MOV       R3,#0D002h     ;current CRC is stored on 0xD002
     MOV       R8,[R3]        ;current CRC -> R8
     MOVBS     R6,RL1         ;data -> R6
     XOR       R8,R6          ;xor CRC value with new data
     MOV       R4,#00H        ;initialize counter

Calc_1:
     JNB       R8.0,Calc_2    ;CRC LSB = 0?
     SHR       R8,#01H        ;right shift CRC by 1 bit
     XOR       R8,#08408H     ;xor with polynomial
     JMPR      cc_UC,Calc_0   ;
Calc_2:
     SHR       R8,#01H        ;right shift CRC by 1 bit
Calc_0:
     CMPI1     R4,#07H        ;compare counter to 7 and increase by 1
     JMPR      cc_SLT,Calc_1  ;jump if counter < 7

     MOV       [R3],R8        ;store new CRC on 0xD002
     RET
Calc_CRC ENDP
```

3.1.3.2 C Language Implementation

```
void vCalcCRC(ulong ulFlashDestAddress, uword uwNbytes)
{
   int i,j;
   uword rx_w;
   char rx_c;
   uword uwDPP3, uwAddLo;

   for (i=0; i<(uwNbytes / 2); i++)
   {
   //the pragma asm sequence reads data from internal flash
      //upper bits of DestAddress = DPP3
      uwDPP3 = (uword)(ulFlashDestAddress >> 14);
      //lower 14 bits = short address; set bits [14,15] for DPP3
      uwAddLo = (uword)((ulFlashDestAddress & 0x3FFF) | 0xC000);
      #pragma asm (@w1=uwAddLo, @w2=uwDPP3, @3)
         PUSH DPP3              ;save original DPP3 to stack
         MOV  DPP3,@w2          ;set DPP3
         MOV  @3,[@w1]          ;load data word from flash
         POP  DPP3              ;restore DPP3
      #pragma endasm (rx_w=@3)

   //first byte (Low Byte)
      rx_c = (char) (rx_w);
      crcval = crcval ^ ((unsigned int) rx_c);
      for (j=0; j<8; j++)
      {
         if (crcval & 0x0001)
         {
            crcval = (crcval >> 1) ^ 0x8408;
         }
         else
         {
            crcval >>= 1;
```

```
        }
      }
    //second byte (High Byte)
      rx_c = (char) (rx_w >> 8);
      crcval = crcval ^ ((unsigned int) rx_c);
      for (j=0; j<8; j++)
      {
        if (crcval & 0x0001)
        {
          crcval = (crcval >> 1 ) ^ 0x8408;
        }
        else
        {
          crcval >>= 1;
        }
      }
      //only locally increased, original value in main remains unchanged.
      ulFlashDestAddress += 2;
    }
}
```

This implementation follows the implementation presented in Annex D: Cyclic Redundancy Check (CRC) of the ISO/IEC FCD 15693-3.

3.2 Programming Mode

In the programming mode user data from the serial flash module is stored in the microcontroller's internal flash memory. Figure 9 shows how data is handled in this mode.

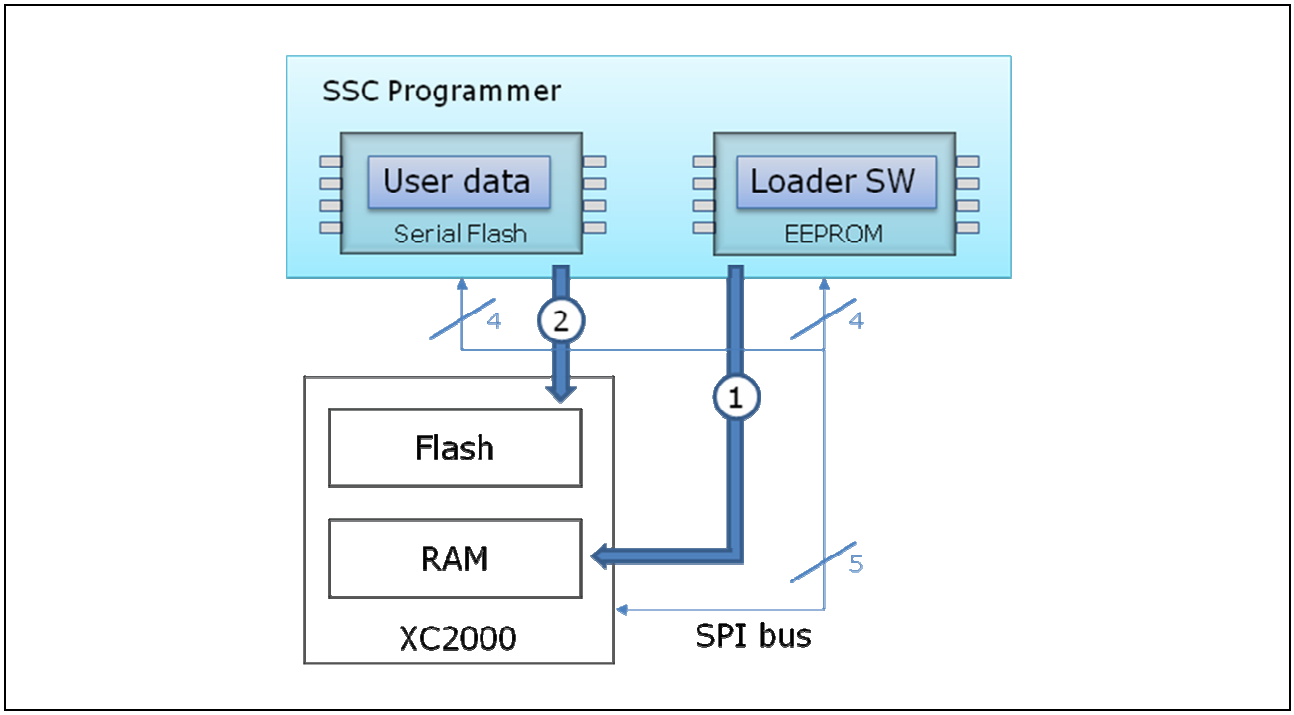


Figure 9 Data Flow in Programming Mode

After system startup the loader software is downloaded to the controller's RAM by the bootstrap loader (step 1). This loader routine is then executed and transfers the user data from the external serial flash to the internal flash memory (step 2).

3.2.1 Basic Flow

After connecting the programmer to the target microcontroller the programming mode is entered by a power-on reset. Because of the startup configuration provided by the hardware (see 2.2 / Figure 4) the controller starts in the SSC bootstrap loader mode. The bootstrap loader is a built-in startup routine, that downloads the actual programming routine (Main Program) from the external EEPROM to the internal RAM. The main program is then executed. It programs the internal flash memory with the data stored in the external serial flash.

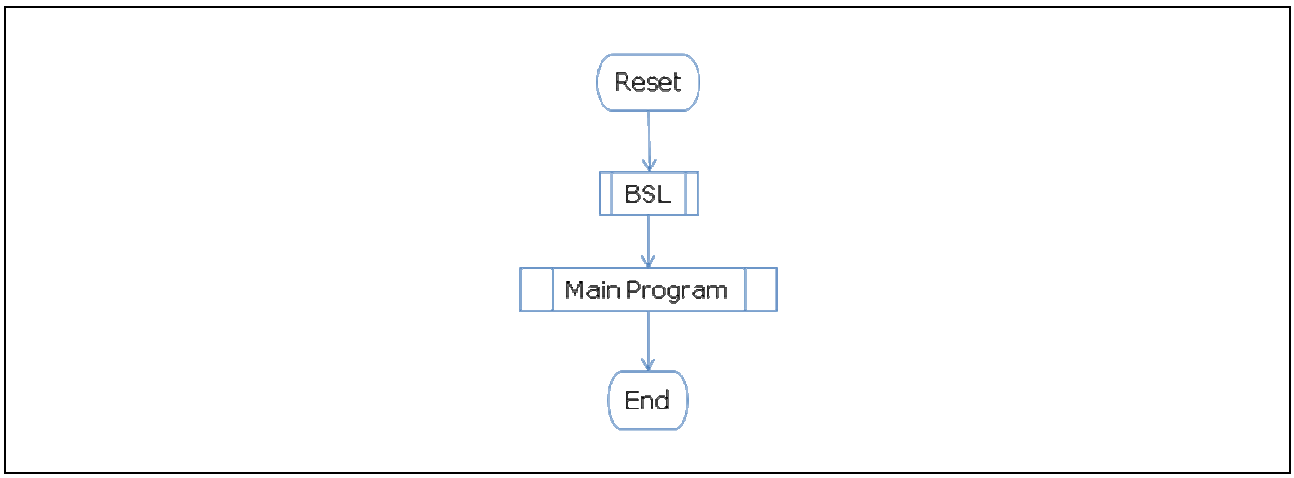


Figure 10 Basic Flow in Programming Mode

3.2.2 Bootstrap Loader

The SSC bootstrap loader is part of the startup software stored in the controller's internal boot ROM. It provides a mechanism to load a user defined program via the SSC interface. The bootstrap loader moves code/data into the internal Program SRAM (PSRAM) starting at location $E0'0000_H$. The XC2000 enters the SSC BSL mode when a special configuration pattern is applied to pins [0:3] of port 10 during a hardware reset. The starter kit provides a switch to do so. An equal switch is part of the SSC programmer hardware.

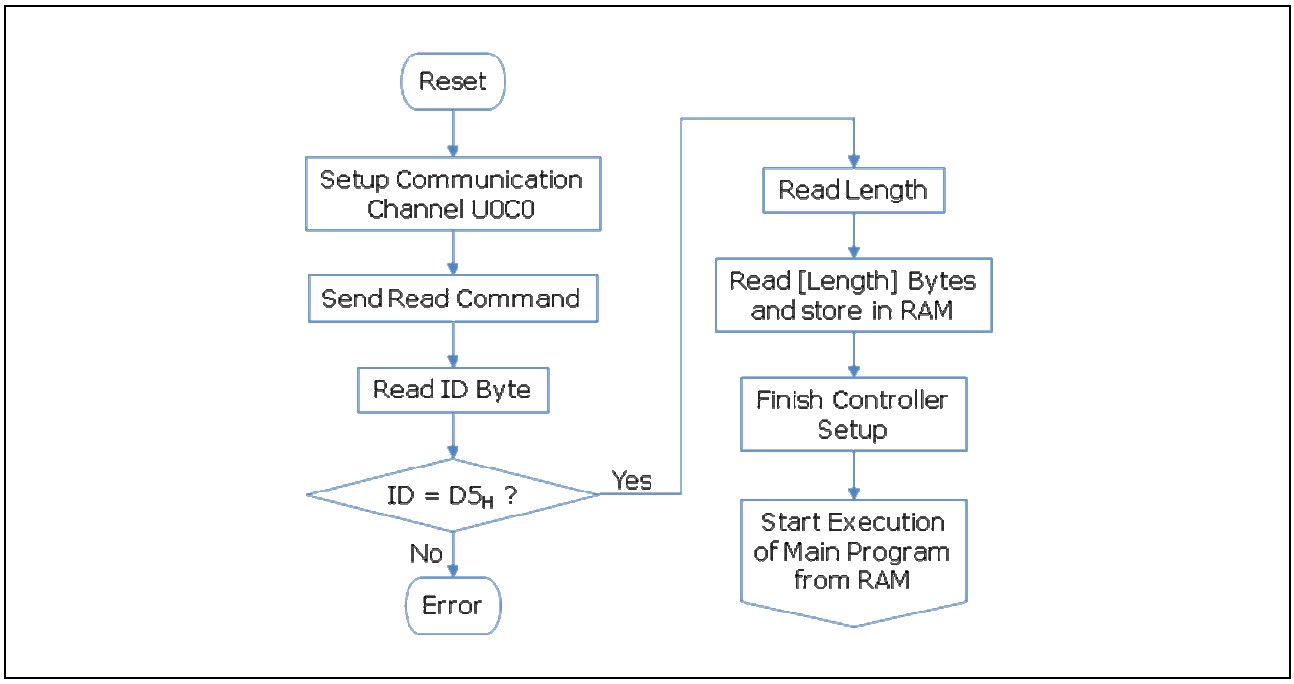


Figure 11 SSC Bootstrap Loader

The XC2000 bootstrap loader software basically performs the following steps:

- Set up the communication using the U0C0 channel in SPI-mode.
- Send Read command (03H) to the connected SPI compatible device (EEPROM) in order to set it to read mode.
- Read the first byte (identification byte) and check it.
- If the identification byte is ok (D5H) read second and third byte (containing the number of bytes to be loaded).
- Read the appropriate number of bytes and store them sequentially from the beginning of PSRAM (address E0'0000H).

After the last byte is received and stored, the startup software continues making the final chip settings and afterwards starts code execution beginning at address $E0'0000_H$.

For more details on this process see the XC2000 User's Manual.

3.2.3 Main Programming Routine

The task of the programming routine is to transfer the data stored in the external serial flash to the controller's internal flash memory. The software provides:

- read access to the serial flash, using the USIC module of the XC2000
- read and write access to the internal flash memory
- a CRC algorithm for validation purposes.

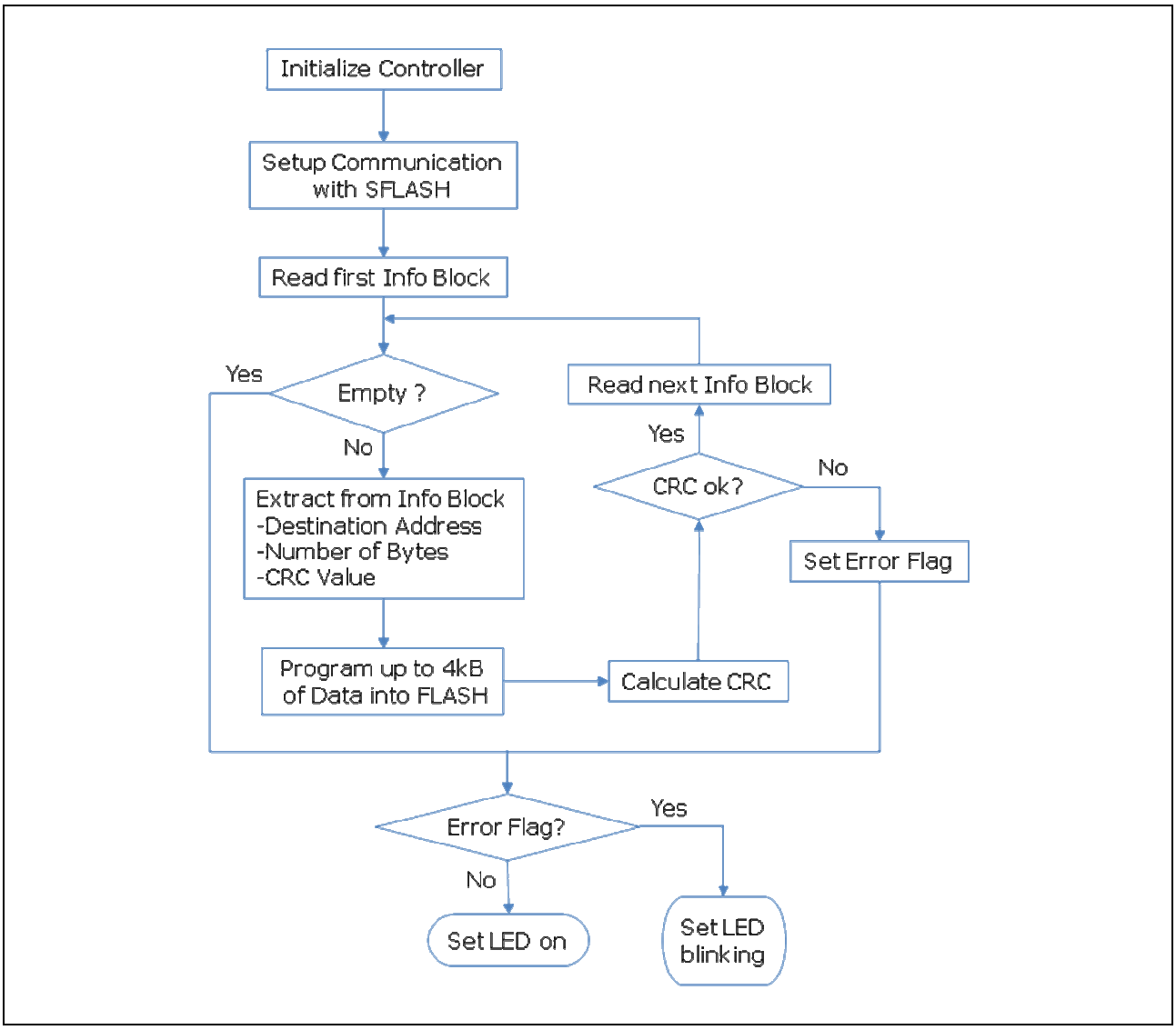


Figure 12 Main Programming Routine

Initializing the controller includes setting up the internal clock frequency (80 MHz) and globally enabling the interrupt system. The serial communication is set up by a separate function. This function activates the USIC module (U0C0), a flexible interface module covering several serial communication protocols and sets the operating mode to SPI. It then configures the SPI protocol, the required I/O pins and the used interrupts.

In the next step the first info block is read from the serial flash module. Each info block is 16 bytes long. The info blocks are stored in the address range from $00'F000_H$ to $00'FFFF_H$. The corresponding 4 Kb sector on the microcontroller ($C0'F000_H$ to $C0'FFFF_H$) is not accessible to the user. This so called configuration sector is accessible only during system startup or in test mode to restore flash parameters and redundancy settings. This sector can never contain user code, so it is the perfect place to store the info blocks.

InfoBlock

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	CRC		NBURSTS		ADDRESS			

Field	Bytes	Description
ADDRESS	[3:0]	**Flash Destination Address** 32 bit value containing the destination base address
NBURSTS	[5:4]	**Number of Bursts** Number of Bytes to program is NBURSTS x 256 Bytes
CRC	[7:6]	**CRC Value** 16 bit CRC value for the current 4kByte block of user code

An empty info block contains only 16 x FF_H. Each info block corresponds to one block of data with a maximum of 4096 bytes (NBURSTS = 16).

If the current info block is not empty, the programming software reads the destination address and the number of bytes to be transferred. The associated data is stored in the on-chip flash module. Then the CRC value is calculated over the previously stored data and compared to the value in the info block. If both are equal the next info block is read, else the programming tool signals an error.

3.2.4 Software Implementation Details

To describe the software implementation on an abstract level a layered model is used. The layered approach allows easy and comfortable programming on C-level, while at the same time providing the means to easily adapt the software to other SPI devices, or even to other target microcontrollers without the need to change the main programming algorithm. The model consists of three abstraction layers. Figure 13 shows how these layers interact with each other and with the hardware.

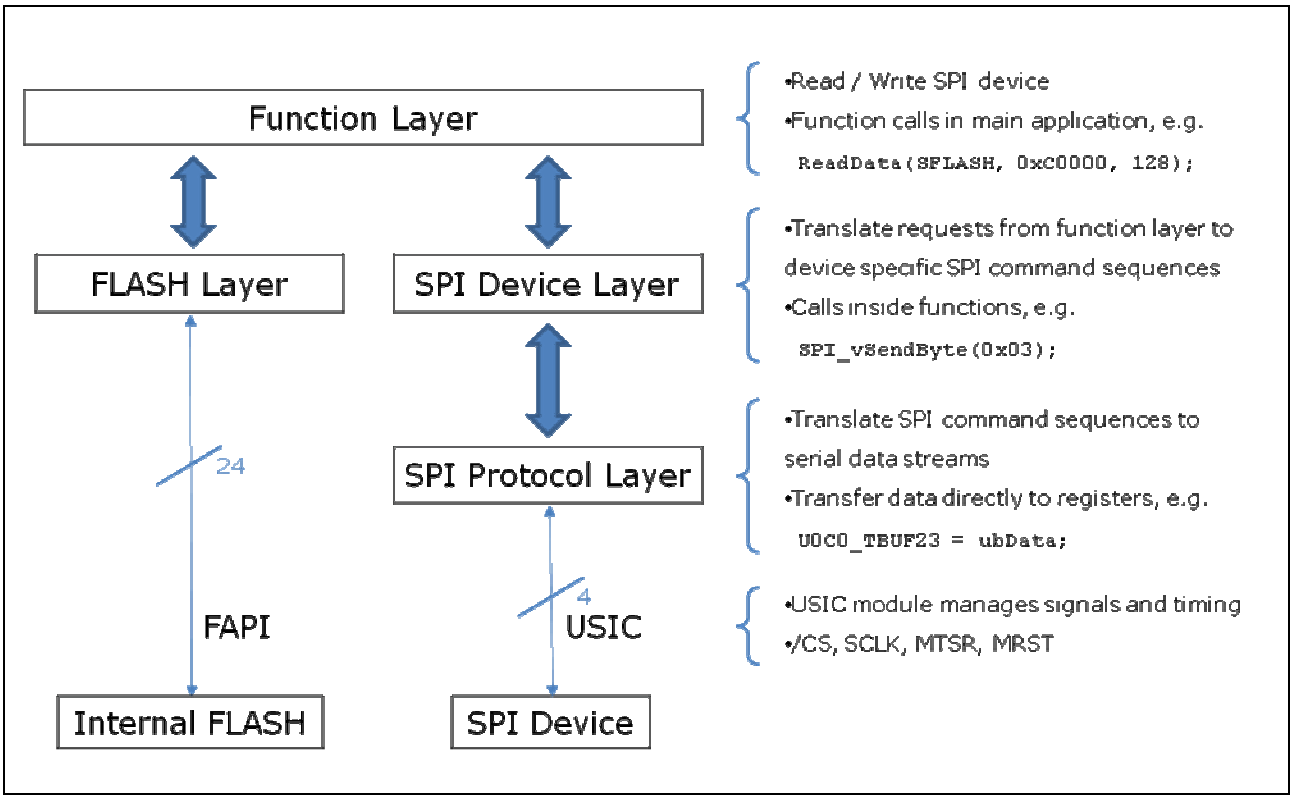


Figure 13 Abstraction layer model

3.2.4.1 Function Layer

The function layer is the top level abstraction layer in this project. In principle this is the main function of the program. It is responsible for initializing all necessary interfaces, moving data from the serial flash to the internal flash and calculating the CRC value.

usic_spi_init

Syntax	void usic_spi_init(void)
Parameters (in)	none
Parameters (out)	none
Description	Initialize the USIC module: • turn on module U0C0 • set operating mode to SPI • configure used I/O pins (Port 2.3 – 2.7) • setup baud rate (1 MHz) • configure SPI protocol (shift unit, /CS handling, frame control) • configure used interrupts (receive interrupt)

vCalcCRC

Syntax	void vCalcCRC(ulFlashDestAddress, uword uwNbytes)
Parameters (in)	ulong ulFlashDestAddress uword uwNbytes
Parameters (out)	none
Description	Calculates the CRC value, starting from the provided flash address over [uwNbytes] bytes. The current CRC value is stored in a global variable.

3.2.4.2 SPI-Device Layer

The SPI-device layer is responsible for handling device specific requests from the function layer. The SPI-device layer translates these requests into the instruction sequences needed for the serial flash module. All needed functions of the serial flash module's instruction set are contained in this layer. The device layer relies upon the SPI-protocol layer to perform the actual SPI transmit and receive operations.

ubReadDataBytes

Syntax	ubyte *ubReadDataBytes(ubyte ubTarget, ulong ulAddress, uword uwLength)
Parameters (in)	ubyte ubTarget ulong ulAddress uword uwLength
Parameters (out)	ubyte ubReadData
Description	Read [uwLentgh] bytes from the SPI device ubTarget (SFLASH, EEPROM), starting from address ulAddress. Return a pointer to an array with the read data.

3.2.4.3 FLASH Layer

The FLASH layer is on the same abstraction level as the SPI-device layer. It is the interface between the function layer and the internal FLASH module. The required functions to erase and program the internal flash are provided here. All read and write operations to the flash memory are controlled by the Flash Application Programming Interface (FAPI). The functions on this abstraction layer communicate with the FAPI by writing command sequences to certain flash addresses.

For details on programming the internal flash see the XC2000 User's Manual.

vEnterPageMode

Syntax	void vEnterPageMode(ulong ulAddress)
Parameters (in)	ulong ulAddress
Parameters (out)	none
Description	Page mode is the required mode to be able to program the internal flash.

vLoadPageWord

Syntax	void vLoadPageWord(ulong ulAddress, ubyte *pubData)
Parameters (in)	ulong ulAddress ubyte *pubData
Parameters (out)	none
Description	Fill the assembly buffer of the required page. *pubData is a pointer to an array.

vProgramPage

Syntax	void vProgramPage(ulong ulAddress, ubyte *pubData)
Parameters (in)	ulong ulAddress ubyte *pubData
Parameters (out)	none
Description	Program one page of data (128 bytes) at the provided address. *pubData is a pointer to an array. The function calls vLoadPageWord to fill the assembly buffer.

vEraseSector

Syntax	void vEraseSector(ulong ulAddress)
Parameters (in)	ulong ulAddress
Parameters (out)	none
Description	Erase one sector of the internal flash.

3.2.4.4 SPI-Protocol Layer

The SPI-protocol layer is the low level layer responsible for the actual communication between the microcontroller and the SPI device. Low level means that these functions write data directly to system registers and transmit buffers.

The basic data flow control is done by the USIC module, depending on the selected configuration (see 3.2.4.1). Only the target (SFLASH, EEPROM) needs to be selected. This feature is included for greater flexibility, since for the programming algorithm itself only the serial flash is needed. Data reception is handled by an interrupt service routine, each time a byte is received, the interrupt request is raised by the USIC module.

SPI_vSetCS

Syntax	void SPI_vSetCS(ubyte ubCS_Line)
Parameters (in)	ubyte ubCS_Line
Parameters (out)	none
Description	Set chip select line for all future SPI transfers (until changed or cleared). e.g. EEPROM, SFLASH

SPI_vClrCS

Syntax	void SPI_vClrCS(void);
Parameters (in)	none
Parameters (out)	none
Description	Clear all chip select lines.

SPI_vSendByte

Syntax	void SPI_vSendByte(ubyte ubData)
Parameters (in)	ubyte ubData
Parameters (out)	none
Description	Send one byte to the SPI device.

SPI_vSendData

Syntax	void SPI_vSendData(ubyte *pubData, uword uwLength)
Parameters (in)	ubyte *pubData uword uwLength
Parameters (out)	none
Description	Send [uwLength] bytes to the SPI device in one single frame. *pubData is a pointer to an array.

void SPI_viRx

Syntax	interrupt (SPI_RINT) void SPI_viRx(void)
Parameters (in)	none
Parameters (out)	none
Description	Interrupt service routine that is called each time a byte has been received. Stores the last received byte from the receive buffer register in an array and clears the receive interrupt flag afterwards.

4 User's Guide

This chapter describes setup and usage of the programming tool. For the purpose of an example an XC2000 Easy Kit is being used for setup and as target system. A 16-pin connection header has to be placed in the free area of the board. Figure 14 shows the wiring diagram. For this example one single Easy Kit is being used for setup and programming mode. This requires a way to switch between some of the port pins:

Table 3 Port Pins in Setup and Programming Mode

Function	Setup Mode	Programming Mode
MTSR	P2.9	P2.3
MRST	P2.10	P2.4
SCLK	P2.8	P2.5

The reason for this is that in setup mode Memtool uses pins P2.3 – P2.5 for ASC communication, so they cannot be used for the SSC communication between controller and SPI memory devices. On the other hand in programming mode the SSC bootstrap loader uses exactly these pins. Of course if you are using separate boards for setup and programming, the jumpers J1 – J3 are not required, you can connect the respective port pins directly to the header.

You should also remove the 0R resistor connecting the CAN2 transceiver to P2.4 (here R151).

The diagram in Figure 14 refers to the Easy Kit XC2000 144 pin V2.0. If you are using a different board version please carefully check the manual to ensure correct wiring.

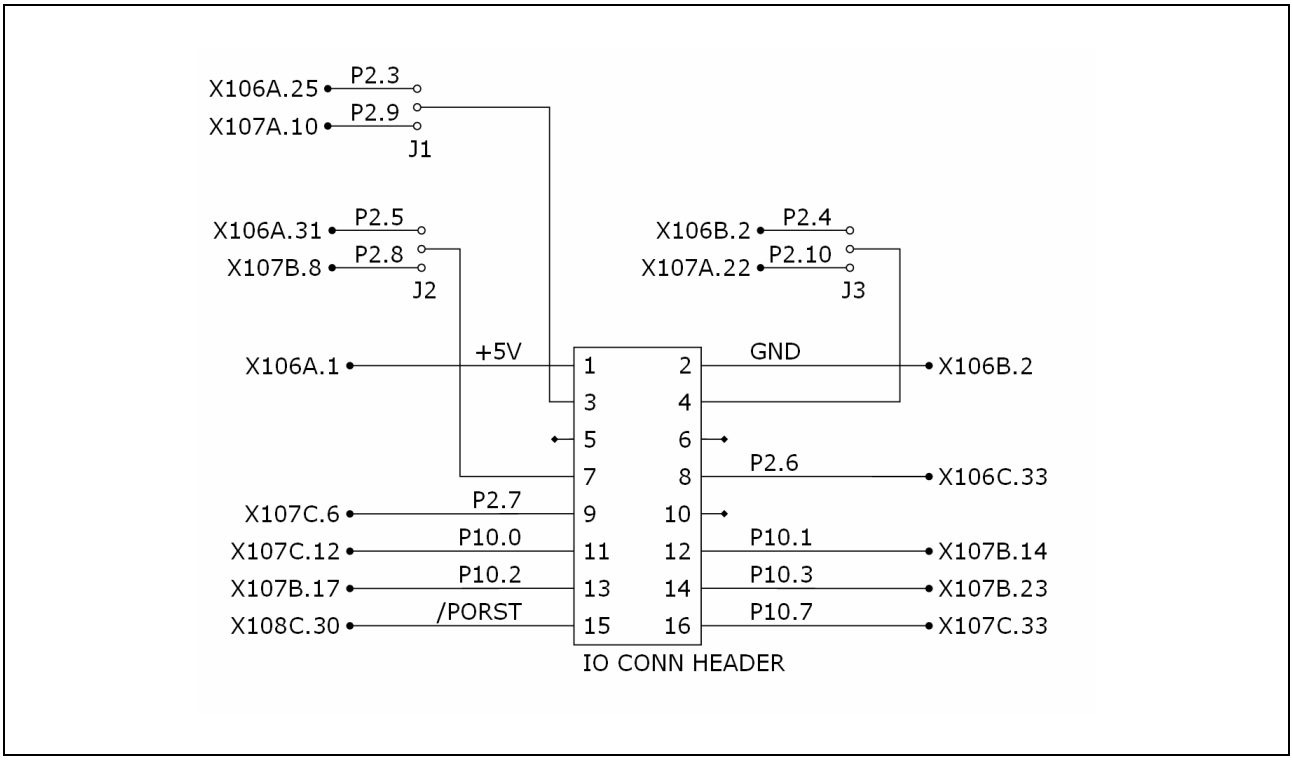


Figure 14 Easy Kit Connector Wiring Diagram

4.1 Setup Mode

By default Memtool does not support the programming of external memory devices. As a first step the modified drivers and config files must be copied to the Memtool directory:

Copy the file `XC2000_SPI.info` to `[…]\Memtool4\`

Copy the files `Easykit_XC2287_SFLASH.cfg`, `XC2XXX_mon.hex`, `XC2287_EEPROM.DAT` and `XC2287_SFLASH.DAT` to `[…]\Memtool4\Targets\`

`[…]` is the path to the `Memtool4` directory, by default this is `C:\Program Files\Infineon\`

Start Memtool. Select 'Target' -> 'Change…' from the menu and choose **Easy Kit with XC2287 SPI SFLASH** as target:

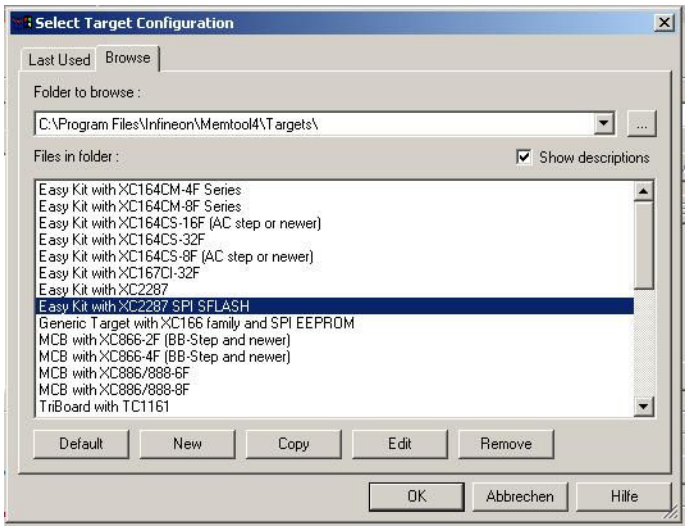


Figure 15 Memtool Target Selection

Connect the programmer to the Easy Kit. Connect the Easy Kit to the PC (USB). Select the ASC bootstrap loader mode with the config switch (ON-OFF-OFF-OFF).

Note: You can either use the config switch on the Easy Kit, or the config switch on the programming tool. The unused switch must be set to OFF-OFF-OFF-OFF!

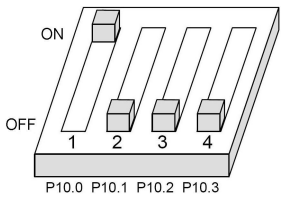


Figure 16 Config Settings for ASC Bootstrap Loader Mode

Provide the Easy Kit with power, press the reset button afterwards.

In Memtool you click the 'Connect' button. After a connection has been established you can choose between the two SPI memory devices.

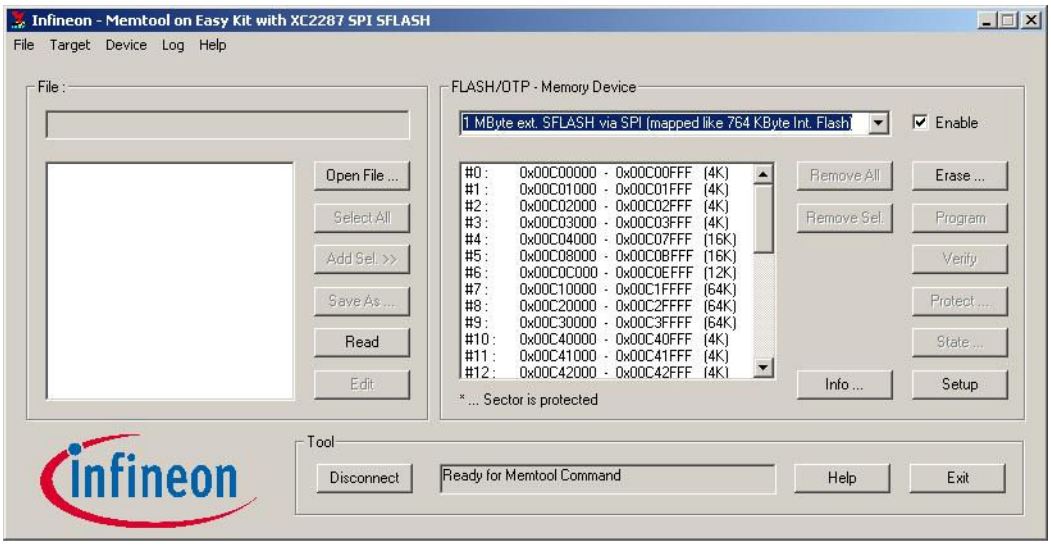


Figure 17 Serial Flash - mapped to on-chip Flash (top); EEPROM - mapped to PSRAM (bottom)

4.1.1 Programming the EEPROM

To program the loader routine into the EEPROM (see Figure 18) you open the appropriate hex-file (1), select and add all parts on the left side (2), then you simply click 'Program' (3).

The loader routine needs to be mapped to the memory area starting from $E0'0000_H$.

Note: This driver is designed to program the EEPROM in order to work with the SSC BSL. Therefore it writes the loader routine with 3 bytes offset! See description of SSC BSL for details.

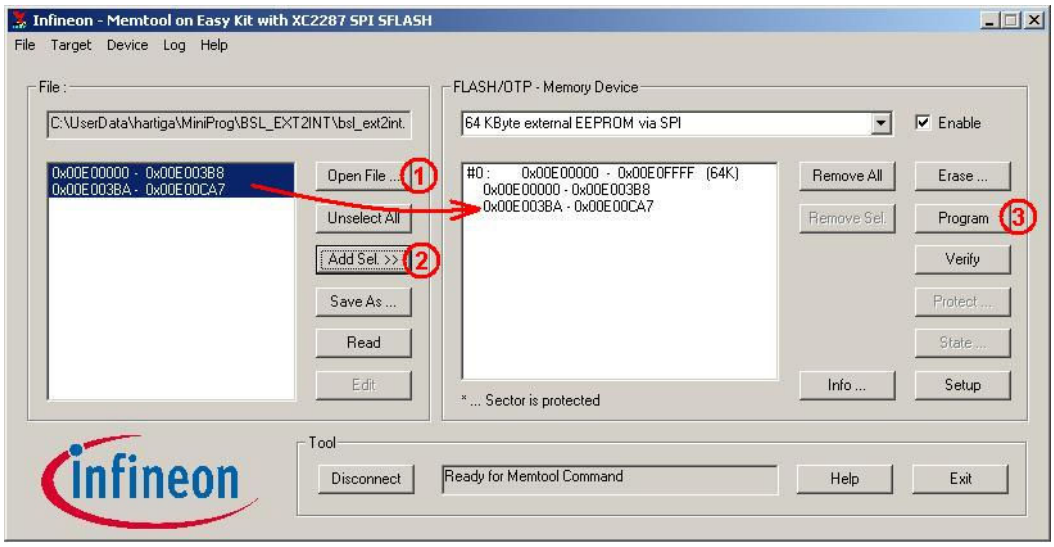


Figure 18 Programming the EEPROM

A bulk erase function is provided, but it is not necessary to erase the EEPROM before programming.

4.1.2 Programming the Serial Flash Module

For the serial flash it is important to erase the module first, otherwise correct programming cannot be guaranteed. In this version of the driver a bulk erase function is implemented, but because of the provided memory mapping it will be called once for each sector if you select 'Erase whole FLASH Module'. So in order **to erase the serial flash module** you **choose to erase only one** (no matter which) **sector**:

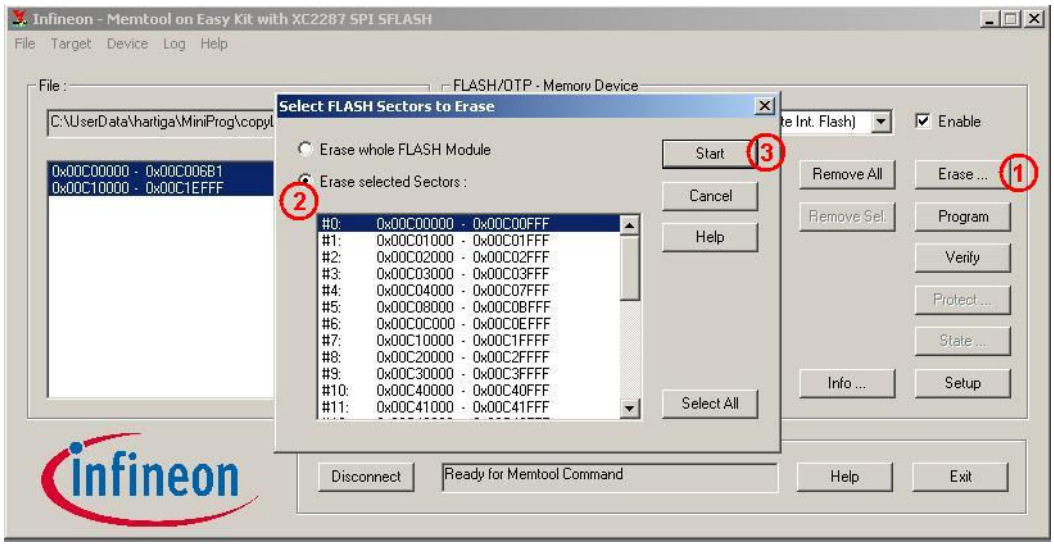


Figure 19 Erasing the SFLASH

To program the user code into the serial flash module (see Figure 20) you open the appropriate hex-file (1), select and add all parts on the left side (2), then you simply click 'Program' (3).

The user code needs to be mapped to the memory area starting from $C0'0000_H$ (on-chip FLASH).

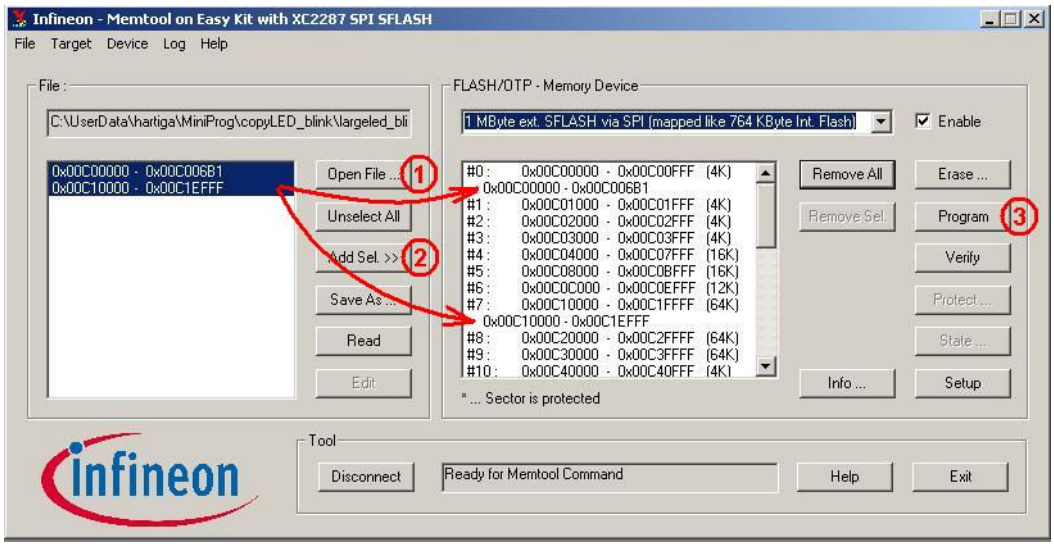


Figure 20 Programming the SFLASH

4.2 Programming Mode

Connect the programmer to the target system (in the example of the single Easy Kit set the jumpers to Programming Mode, see Table 3). Select the SSC bootstrap loader mode with the config switch (OFF-ON-ON-OFF).

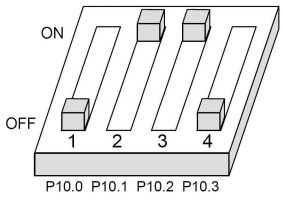


Figure 21 Config Settings for SSC Bootstrap Loader Mode (for Programming Mode)

Supply the target with power.

Programming starts automatically. If you are programming larger files you will see the LED flicker for a short time during programming (approx. 1 sec / 30 Kbytes).

If programming is successful the LED will be set permanently 'on', otherwise the LED will blink rapidly.

5 Conclusion and Outlook

This application note shows that it is possible to build a fast and easy to use programming tool for less than 10,- €. At a quantity of 100 pieces the price per unit is 8,35 €. A detailed bill of materials can be found in the appendix (6.5). The price will further decrease for larger quantities, e.g. the PCB will cost only 1,20 € at 500 pieces.

Programming is very fast, it only takes about 28 seconds to program 768 Kbytes (whole flash module). The on-chip flash module consists of three independent 256 Kbytes blocks. For speed optimization the programming process could be modified in order to program all three blocks in parallel. The programming time would then decrease to about 10 seconds.

In real applications some things will have to be considered, which were not part of this application note:

- The security aspect needs further attention

This means that some kind of protection mechanism must be implemented in the programming routine in order to prevent software hacking. Otherwise the mechanism described here could easily be used to modify a system's firmware, e.g. to avoid copy protection. The XC2000 microcontroller provides a protection system for the on-chip flash memory which could be used for this purpose.

- The target hardware must provide a connector for the programming tool

For demonstration purposes the required connections have been made on an XC2000 starter kit by simply wiring the used controller pins to a 16 pin connector manually. In a real application these connections need to be included into the design of the PCB.

The high speed and the extremely low price are especially useful in small batch production applications, during development, or for prototypes. For example the developing engineer could program the tool and immediately pass it on to the production line. Software changes during development, updates for prototypes, or even firmware updates in the final product can quickly and easily be applied. The programming process could even be delayed to a later moment: For example a generic control unit could be used for several different design models of an electronic system. With this programming tool the unit could be programmed just-in-time in order to fit into the desired model, a feature which simplifies stock-keeping.

The concept presented here for the XC2000 can also be extended to other microcontrollers. The basic requirement is a SSC bootstrap loader. Of course the signal lines will have to be adjusted according to the requirements of the new target controller, and the programming software will have to be adapted as well. But because of the layered software model and the simple hardware layout, these changes can be done fast and with little effort.

6 Appendix

6.1 Abbreviations

ASC	Asynchronous Serial Channel
CRC	Cyclic Redundancy Check
FAPI	Flash Application Programming Interface
OTP	One Time Programmable
PCB	Printed Circuit Board
SPI	Serial Peripheral Interface
SSC	Synchronous Serial Channel
USIC	Universal Serial Interface Channel

6.2 List of Tables

6.3 List of Figures

6.4 Literature / Sources

Infineon Technologies: *XC2000 User's Manual, Volume 1 (of 2): System Units, V1.0 June 2007*

Infineon Technologies: *XC2000 User's Manual, Volume 2 (of 2): Peripheral Units, V1.0 June 2007*

Infineon Technologies: *AP16095 – Interfacing the XC16x Microcontroller to a Serial SPI EEPROM.* V1.0, 2006-07 – Application Note

Intel: *Hexadecimal File Format Specification*. Rev. A, 1988-01 - Specification

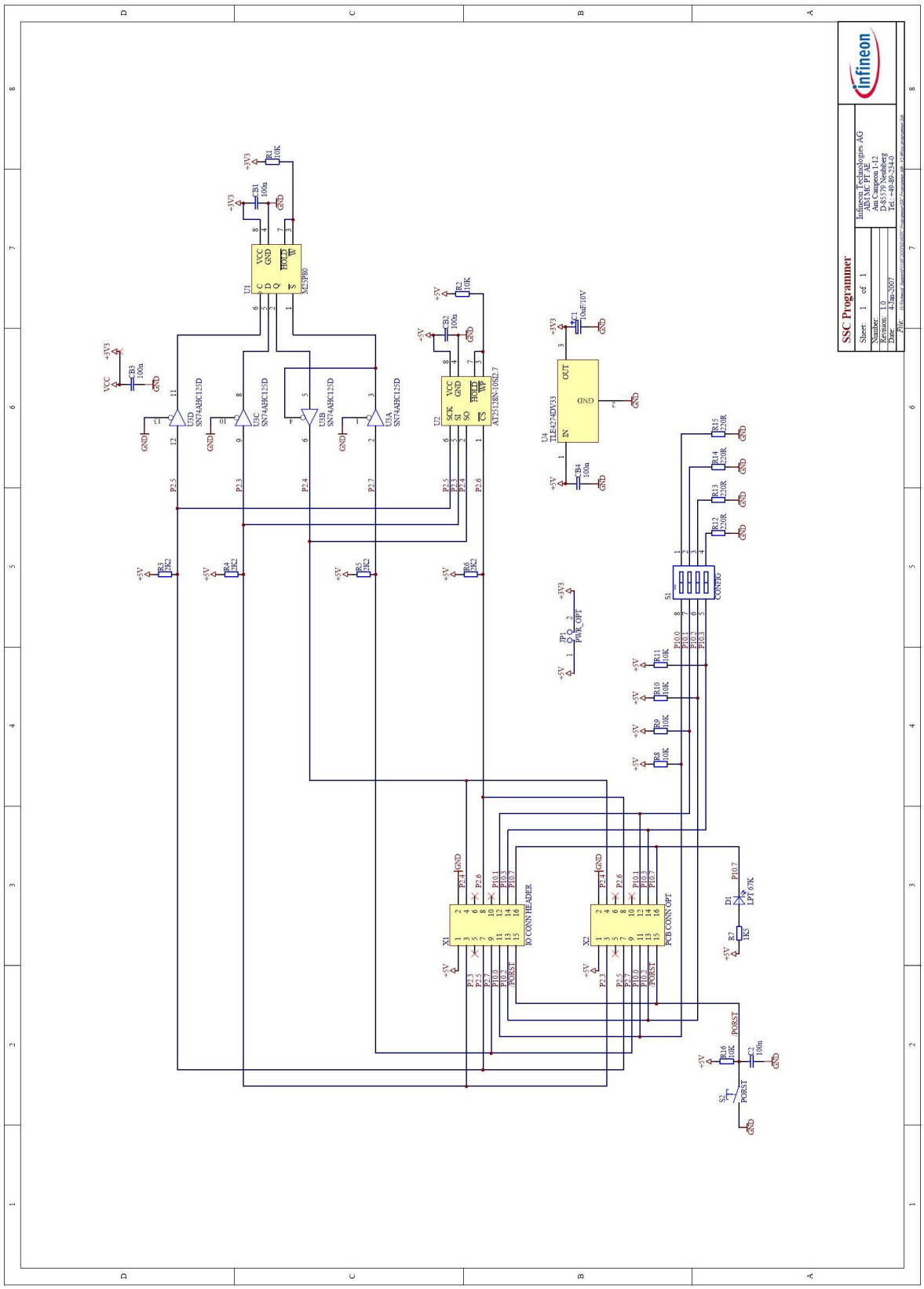
ISO/IEC FCD 15693-3 2000-03-10. *Identification cards - Contactless integrated circuit(s) cards - Vicinity cards - Part 3: Anti-collision and transmission protocol. Annex D: Cyclic Redundancy Check (CRC)*

6.5 Bill of Materials

Table 4 Components and Prices

Part	Component	Price per Piece (€)
Printed Circuit Board	-	1,78
M25P80 (Serial Flash)	U1	2,88
AT25128N (EEPROM)	U2	0,66
SN74AHC125D (Buffer Gate)	U3	0,25
TLE4274DV33 (Voltage Regulator)	U4	0,70
Config Switch	S1	0,38
Reset Button	S2	0,70
LED	D1	0,09
Resistors, Capacitors	R1-R16, C1-C2, CB1-CB4	0,62
16 pin Connection Header	X1	0,29
PCB Connector (optional)	X2	3,00
Sum (w/o optional connector)		**8,35**

6.6 Schematic



http://www.infineon.com

Published by Infineon Technologies AG