XMC1000 S-series device

XMC1000

About this document

Scope and purpose

The XMC1000 S-series device provides Intellectual Property (IP) protection for your application code. The device has a boot mode which handles the download of the encrypted application code. In this document, we give an overview of the IP protection scheme and the command set, we describe the flash programming flow of the XMC1000 S-series device, and provide instructions for encryption and downloading the application code to the S-series device using the Secure Download Manager tool provided by Infineon.

Intended audience

This document provides the encryption concept of XMC1000 S-series device, so a field application engineer could explain the benefit of this feature to a potential customer. For the XMC1000 S-series programmer, this document provides a detailed description of the protocol for downloading encrypted code to the XMC1000 S-series device.

References

[1] The User’s Manual can be downloaded from http://www.infineon.com/XMC

Table of contents

1 Secure Bootstrap Loader (SBSL)

Secure Bootstrap Loader (SBSL) is a start-up mode available in the XMC1000 to support the secure transfer of encrypted application software, including code and data, and programming that Intellectual Property (IP) into the device Flash. The encryption is based on the Advanced Encryption Standard (AES) and a key size of 128 bits.

1.1 IP protection scheme

The IP protection scheme is a system level solution to prevent embedded software cloning. The principles behind the protection scheme are:

- IP is always transported in its encrypted form from the moment it leaves the provider's premises, until it is downloaded into the device.
- Only authorized devices allow the correct download and the subsequent operation of the IP.
  - For example, encrypted IP created for one XMC1000 S-series variant is not permitted to be downloaded into another variant of the XMC1000 S-Series device.

1.1.1 Fundamental building blocks

The protection scheme is based on these components:

- XMC1000 S-series device
- Software encryption tool
- Download tool with SBSL support

1.1.1.1 XMC1000 S-series device

The default start-up mode for the XMC1000 S-series devices in the device delivery state is the SBSL mode, and each variant of the devices is assigned a unique 128-bit identifier - SBSL ID.

The SBSL ID is stored in the flash configuration sector during device personalization. It can be read as part of the SBSL status information with a FLASH_GET_SBSL_STATUS command (see section 1.2.3.1).

For the application code to identify an XMC1000 S-series device, the Flash address 1000'0F1C$_H$ can be read. A read value of C0DE'1705$_H$ indicates that the device is an S-series device.

1.1.1.2 Software encryption tool

A PC-based software encryption tool called Secure Download Manager (SDM) is provided to:

1. Generate an AES 128-bit IP Key, K_{IP}.
2. Encrypt the IP with K_{IP}.

K_{IP} is generated from a smart card that is connected to the SDM through a PC/SC card reader. There is no limit to the number of keys that can be generated from the smart card.

K_{IP} is used to encrypt the IP, and also to enable the download of IP into the device.

To encrypt the required IP, K_{IP} and the SBSL ID of the target device are required as inputs. K_{IP} can be created as new, or selected from the list of previously generated keys.

The output of the encryption is a zipped file (<design>.zip for example) containing a further three files:

- '<design >.properties' file;

  - carries the SBSL ID of the target device.
- '<design >_kc.ldf' file;
  - carries the encrypted K_{IP}.
- '<design >_ip.ldf' file;
  - carries the encrypted IP.

1.1.1.3 Download tool with SBSL support

Functions

A download tool serves the following functions:

- Acts as an interface to the target device.
- Identifies the correct encrypted K_{IP} and IP for the device based on its SBSL ID.
- Downloads the encrypted K_{IP} and IP into the device.
- Handles any error response returned by the device.

Configuration

The SBSL uses the ASC (UART) protocol for communication between the host and the device, with the configuration:

- 8 data bits.
- 1 stop bit.
- No parity.
- LSB first.
- Channel selection based on which RxD pin the first Start and Header bytes are received.

Operation

Once the download tool has established the communication channel and baud rate, it uses the SBSL command set to carry out the necessary actions on the device. The detailed download flow, including baud rate negotiation, is described in section 2.1.

The SDM tool can also be used as a download tool. In this instance the user has to select the corresponding zipped file generated during the IP encryption phase, to be downloaded into the device by SDM.

1.1.2 IP protection flow

The IP protection flow consists of two main phases:

- Encryption phase.
- Download phase.

Encryption

The encryption phase is entered once the IP provider is provided with the SBSL ID of the target devices. During this phase, the IP provider uses the software encryption tool to encrypt the IP with an IP key. The output is a zipped file containing the encrypted IP and associated files.

Download

The download phase is entered once the device programmer receives the zipped file generated from the encryption phase. During the download phase, the device programmer uses the download tool to install the IP key in the device flash configuration sector, before downloading the encrypted IP. The encrypted IP will be decrypted by the SBSL, using the installed IP key, and programmed into the device flash memory.

Note: The SBSL ID of the target device must match with the one used during the encryption phase, otherwise the download tool will flag an error and data download to the unknown device will fail.

Once the IP is successfully programmed into the device flash memory, the SBSL switches the Boot Mode Index (BMI) to the User Productive (UP) mode and begins execution of the IP.

Note: In user productive mode, all external accesses to the device are disabled. If it is intended to support future updates of the IP, the IP must be able to call the Request BMI Installation user routine in the boot ROM to switch the BMI back to SBSL mode. When reverting back to SBSL mode, the flash memory will be restored to the delivery state and the download phase can be repeated with the new IP.

1.2 SBSL command set

The SBSL protocol is as follows:

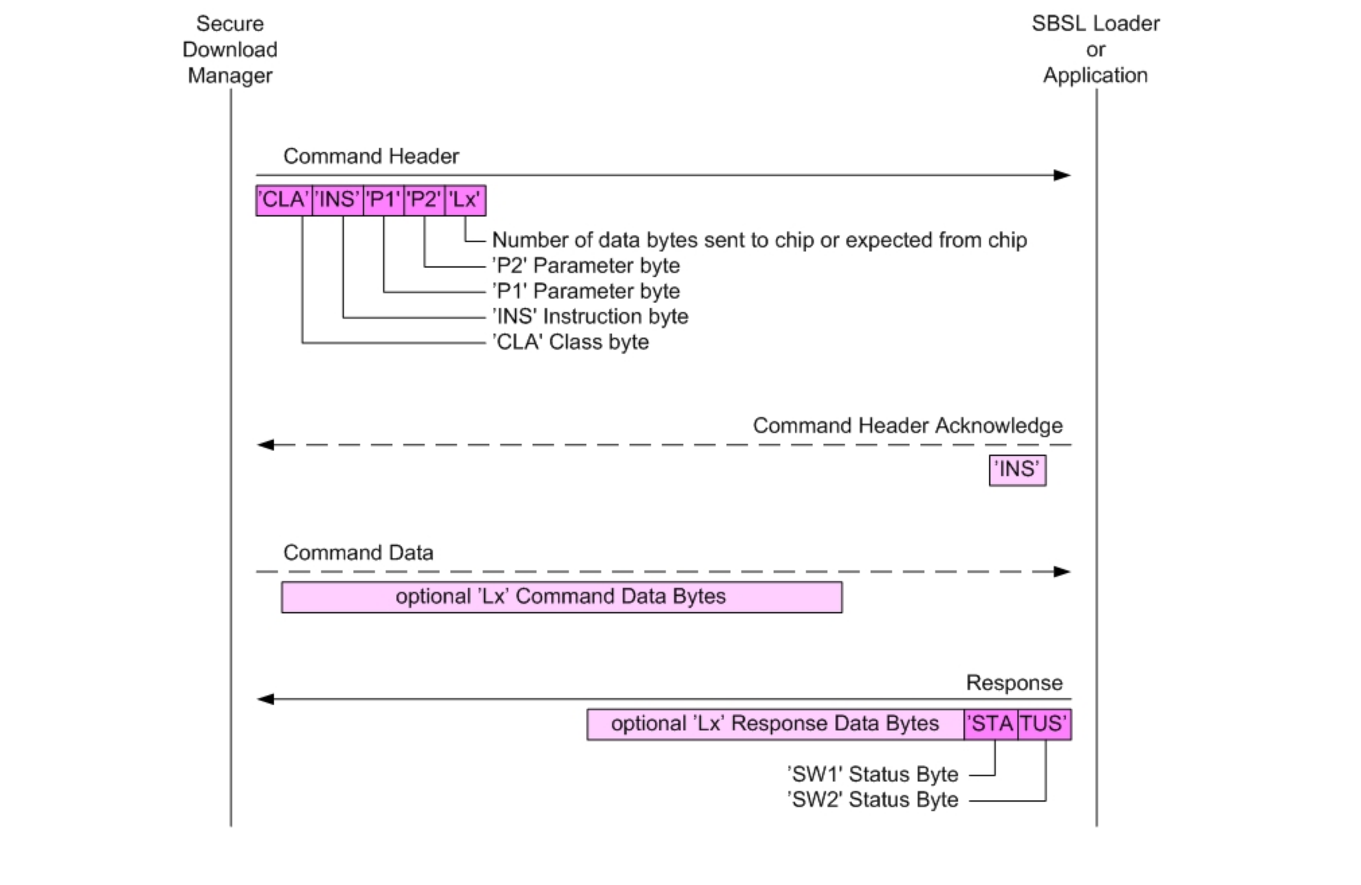


Figure 1 **Data flowchart for the SBSL Loader protocol**

When ‘Lx’ is not equal to zero, the SBSL device will returns the received ‘INS’ byte as ‘header acknowledge’. For example for FLASH_CHIP_RESET command (chapter 1.2.2.1), the L_c is 0x00, so there is no ‘header acknowledge’ from the SBSL device.

If the SBSL requires more time, it sends one or more 'Waiting Time Extension Requests' as illustrated by Figure 2 for the SBSL loader command ' FLASH_LOAD_CHECK_SIGNATURE ' ('A0 21 00 00 00').

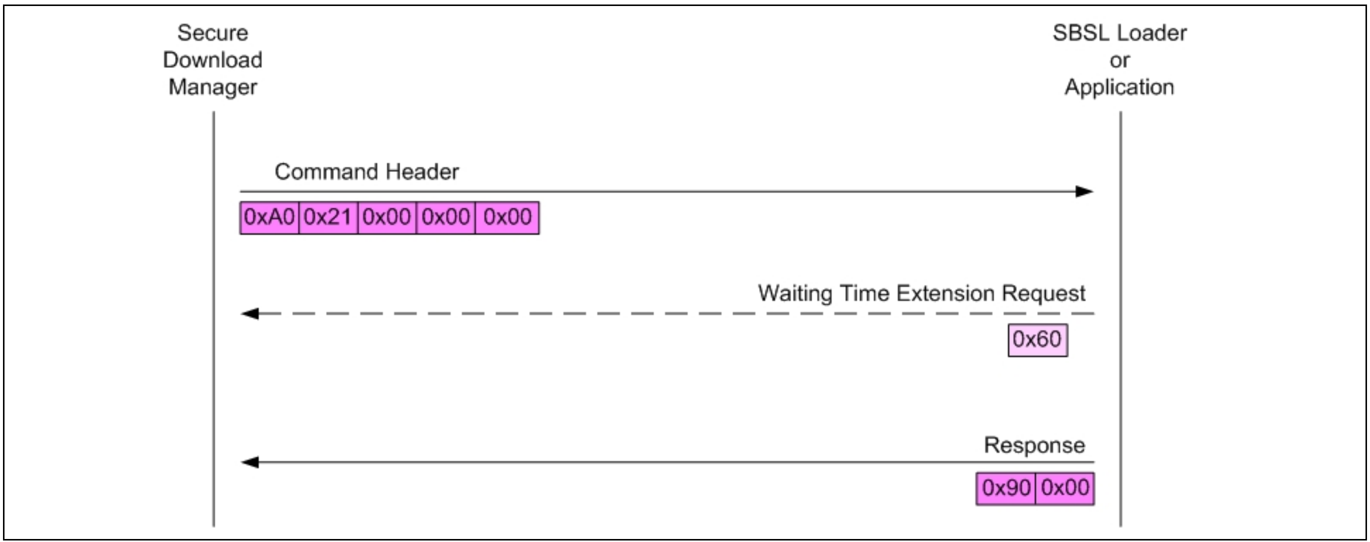


Figure 2 Data flowchart for waiting time extension request

About SBSL commands

An SBSL command is identified by two 8-bit integers representing the command class CLA and command instruction INS. It also contains:

- Two 8-bit command parameters P1 and P2.
- An 8-bit field L_c, indicating the number of bytes of the following command data.
- N_c bytes of command data.
- An 8-bit field L_e, indicating the maximum number of response bytes expected.

SBSL commands are designed to transport payload data only in one direction; i.e. to the SBSL in the command's data field, or from the SBSL in the response's data field, but not in both at the same time.

Table 1 SBSL commands

CLA	INS	Name	Description
0xA0	0x00	FLASH_CHIP_RESET	Triggers chip reset
0xA0	0x10	FLASH_GET_SBSL_STATUS	Retrieves the 39-byte SBSL status information, *rSbslStatus*
0xA0	0x12	FLASH_CHANGE_KEY	Updates the IP Key, K_{IP} and its label, L_{IP}
0xA0	0x20	FLASH_LOAD_DATA	Loads data to flash memory
0xA0	0x21	FLASH_LOAD_CHECK_SIGNATURE	Verifies checksum of downloaded data

1.2.1 General command status response

The SBSL always returns a two-byte status word, SW1 and SW2, and data (if applicable) in response to a SBSL command.

Table 2 lists the general command status response values. Any additional command-specific responses are listed in the respective command description.

Table 2 Command status response values SW1-SW2

SW1	SW2	Meaning	Processing status
0x90	0x00	Success	Normal
0x64	0x00	Execution error: NVM unchanged	Execution error
0x65	0x00	Execution error: NVM changed	
0x65	0x81	NVM is changed; memory failure	
0x67	0x00	Wrong length (L_c or L_e)	Checking error
0x69	0x82	Insufficient security state	
0x69	0x83	Authentication method blocked	
0x69	0x84	Reference data not usable	
0x69	0x85	Conditions of use not fulfilled	
0x6A	0x00	Wrong parameters P1 and P2	
0x6A	0x86	Wrong parameters P1 and P2	
0x6C	L'_e	Wrong length L_e, SW2 indicates the expected length L'_e	
0x6D	0x00	Invalid instruction byte (INS)	
0x6E	0x00	Invalid class byte (CLA)	

1.2.2 Protocol-specific commands

1.2.2.1 FLASH_CHIP_RESET

This command triggers a chip reset. The response is returned and the chip reset takes place.

Security

None

Parameters

None

Syntax

Table 3 FLASH_CHIP_RESET syntax

CLA	INS	P1	P2	L_c	Data field	L_e
0xA0	0x00	0x00	0x00	0x00	-	-

Response

Table 4 FLASH_CHIP_RESET response

Data field	SW1	SW2	Status
-	0x90	0x00	Success

Return value

The command always reports success.

1.2.3 Status and configuration

1.2.3.1 FLASH_GET_SBSL_STATUS

This command retrieves the 39-byte SBSL status information, *rSbslStatus*, from the chip. *rSbslStatus* is useful to determine further steps for handling the SBSL or for comparison to the expected state during personalization.

During SBSL status preparation, the SBSL executes the erase flash procedure, if the SBSL has been previously re-activated and the user flash area is therefore scheduled for erasure. Waiting Time Extension (WTX) requests are sent during the flash erase to obey protocol timing. A byte of value 0x60 is sent for each WTX request.

Security

None.

Parameters

None.

Syntax

Table 5 FLASH_GET_SBSL_STATUS syntax

CLA	INS	P1	P2	L_c	Data Field	L_e
0xA0	0x10	0x00	0x00	-	-	0x27

Response

Table 6 FLASH_GET_SBSL_STATUS response

Data field	SW1	SW2	Status
rSbslStatus	0x90	0x00	Success
rSbslStatus	0x65	0x81	Erase procedure failure
-	0x67	0x00	Wrong L_e

Return value

This command returns *rSbslStatus*.

Table 7 rSbslStatus structure

Offset	Bytes	Value	Description
0	4	“SBSL”	Magic name identifying structure
4	1	0xC0	SBSL version tag
5	1	0x04	Length of following data
6	1	0x06	XMC1000
7	3	vr rb bb	Software version (v), revision (r), build (b)
10	1	0xC1	SBSL patch version tag
11	1	0x03	Length of following data
12	3	vr rb bb	Patch version (v), revision (r), build (b)
15	1	0xC2	SBSL state tag

Offset	Bytes	Value	Description
16	1	0x04	Length of following data
17	1	ULC	**SBSL unified life cycle**
18	1	V	V.0 bit: 0 -> SBSL is not valid; 1 -> SBSL is valid V.1 bit: 0 -> K_{IP} is not valid or set; 1 -> K_{IP} is valid Others: reserved
19	1	0x00	Reserved
20	1	FDTC	**Flash download trial counter** Indicates the current remaining number of download attempts. Every start of a download sequence decreases the value of FDTC by one, upon receiving the first 'FLASH_LOAD_DATA' command. If the download ended successfully (verified by a checksum calculation, see section 2.4), FDTC is reset to its initial start value. If the download failed, FDTC remains on its decreased value. In case FDTC has reached 0, further flash downloads are irreversibly blocked and the affected chip needs to be replaced with a new one.
21	1	0xC3	SBSL ID tag
22	1	0x10	Length of following data
23	16	SBSL ID	SBSL ID

1.2.4 Key management

1.2.4.1 FLASH_CHANGE_KEY

This command updates the IP Key K_{IP}, and its label L_{IP}, in the chip with the ones contained in the 20-byte key field.

Security

None

Parameters

None

Syntax

Table 8 FLASH_CHANGE_KEY syntax

CLA	INS	P1	P2	L_c	Data field	L_e
0xA0	0x12	0x01	0x00	0x14	20-byte key field	-

Response

Table 9 FLASH_CHANGE_KEY response

Data Field	SW1	SW2	Status
-	0x90	0x00	Success
-	0x67	0x00	Wrong L_c
-	0x69	0x82	Insufficient security state
-	0x69	0x84	Mismatch between key and key label
-	0x6A	0x86	Wrong parameters P1 and P2

Return value

This command either returns success (0x90 0x00) or an error condition as shown in Table 9.

1.2.5 Flash download

1.2.5.1 FLASH_LOAD_DATA

This command delivers a configurable length of encrypted SBSL download data to the chip. This data is handed over to the decryption module of the SBSL.

After decryption, the data is decoded and complete pages are flashed into the Flash memory. A checksum is computed over all data blocks and validated with the FLASH_LOAD_CHECK_SIGNATURE command.

Note: The flash download sequence must be explicitly finished with a following FLASH_LOAD_CHECK_SIGNATURE command.

Security

The IP key K_{IP} has to be in place.

Parameters

D_l is the encrypted SBSL data of l bytes.

Syntax

Table 10 FLASH_LOAD_DATA syntax

CLA	INS	P1	P2	L_c	Data field	L_e
0xA0	0x20	0x00	0x00	l	D_l	-

Response

Table 11 FLASH_LOAD_DATA response

Data field	SW1	SW2	Status
-	0x90	0x00	Success
-	0x64	0x00	Fatal No fab-out state or K_{IP} is not set.
-	0x65	0x00	Error updating the SBSL state; for example FDTC.
-	0x65	0x01	Fatal Write to flash outside user flash range was suppressed, chip enters sleep mode.
-	0x65	0x81	Fatal NVM programming error occurred, chip enters sleep mode.
-	0x69	0x82	Insufficient security state. No download trials are left for example, or K_{IP} is missing.
-	0x69	0x84	Error in download stream was found.
-	0x69	0x85	Error interpreting a download record.

Return value

The command either returns success or an error status value.

In case the “no fab-out” state (0x64 0x00) is returned, the following causes may apply:

- The chip is not in a fab-out state anymore; i.e. data has already been flashed into the Flash memory using the SBSL. After re-activation of the SBSL, the FLASH_GET_SBSL_STATUS command described in section 1.2.3.1 must be run to erase the user flash.
- K_{IP}, the IP key, is not set. It must be set by using the command FLASH_CHANGE_KEY described in section 1.2.4.1.

In case of a fatal error during execution, the SBSL restarts the chip immediately after sending the command response.

1.2.5.2 FLASH_LOAD_CHECK_SIGNATURE

This command verifies the checksum computed over all data blocks and finishes the flash download procedure.

The actual download signature verification is performed within the download stream interpretation. Its result is kept in memory until it is retrieved with this command.

Immediately after reporting the status the Secure Boot Strap Loader activates the downloaded user flash software in the case of success, or otherwise restarts the chip.

Security

The IP key K_{IP} has to be in place.

Parameters

None.

Syntax

Table 12 FLASH_LOAD_DATA syntax

CLA	INS	P1	P2	L_c	Data field	L_e
0xA0	0x21	0x00	0x00	0x00	-	-

Response

Table 13 FLASH_LOAD_DATA response

Data field	SW1	SW2	Status
-	0x90	0x00	Success
-	0x65	0x00	Wrong hash value
-	0x65	0x81	Flash programming error
-	0x69	0x82	No download started

Return value

The command either returns success or an error status value.

2 Programming via secure BSL

The download of application software and data to an XMC1000 device can be performed via Infineon's secure download manager as well as via a proprietary download tool (developed by an OEM for example).

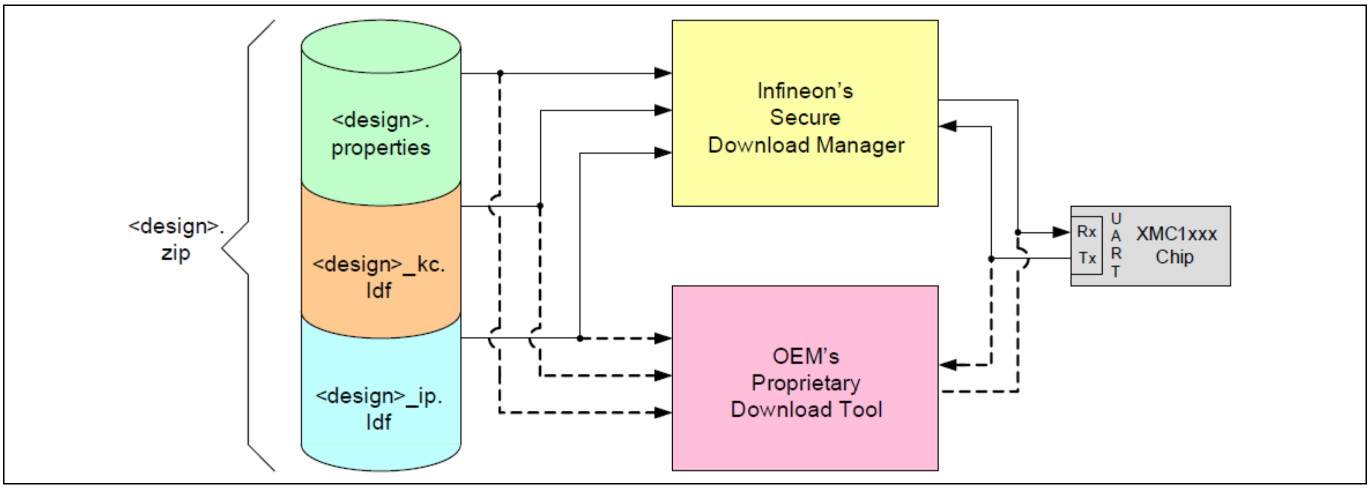


Figure 3 Components required for download of application software to XMC1000

Both tools use the same '<design>.zip' file as input. The download input file (*.zip) shall be generated with Infineon's Secure Download Manager tool.

Note: Please refer to section 1.1.1.2 for the functionality of the three files to be found in <design>.zip.

Figure 4 illustrates the operations executed during a download:

1. A baudrate for communication between the XMC1000 chip and the download tool has to be negotiated.
2. The SBSL Id's stored on-chip and used as the base for key generation and encryption of the download information have to be compared. In case of a miss-match, the download operation has to be aborted, because the decryption of the downloaded key and software would fail.
3. The SBSL Id specific key has to be downloaded.
4. The SBSL Id specific code and data have to be downloaded.

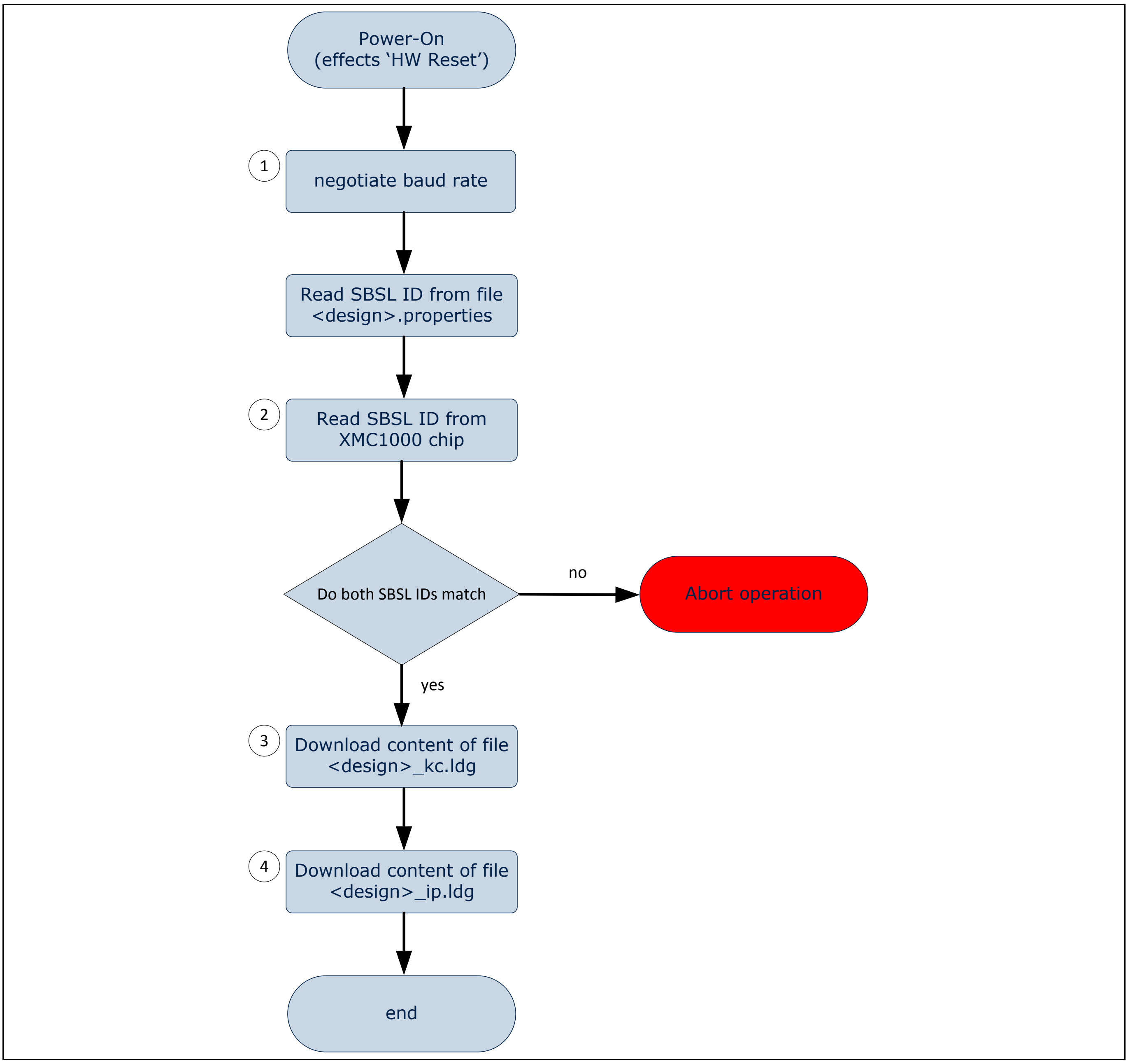


Figure 4 Simplified download flow

2.1 Detailed description of download flow

The following sections explain the download flow.

2.1.1 Programming pin

Two sets of pins/channels are available to choose for programming the XMC1000 S-series device. Both channel 0 and 1 are ready for UART communication after power up.

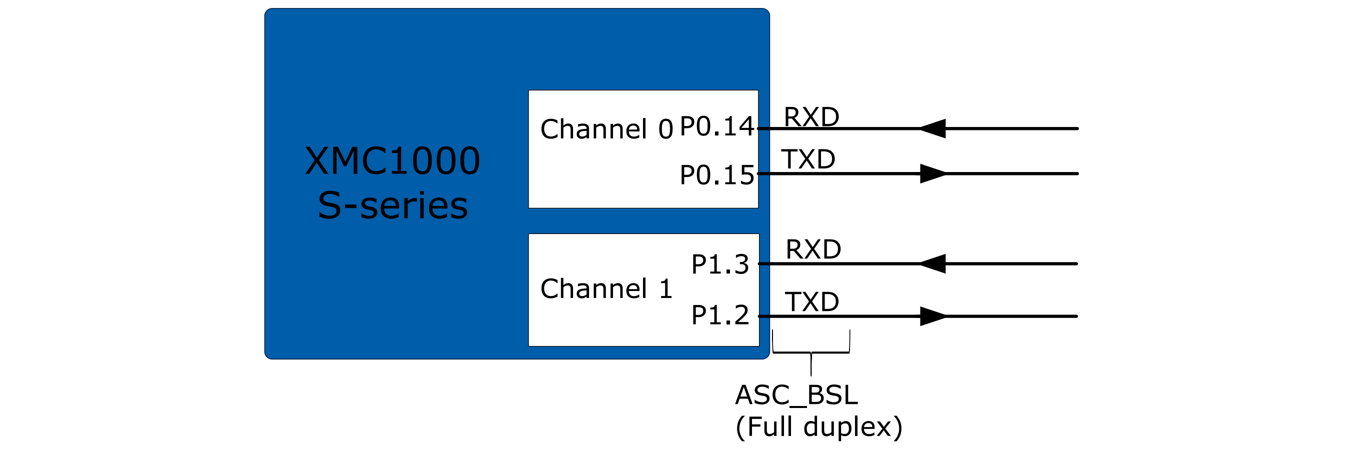


Figure 5 Pins used for UART communication with XMC1000 S-series device during programming

2.1.2 Baudrate negotiation

XMC1000 devices are equipped with baudrate recognition logic working in the range between 300 and 115,200 Baud, depending on the internal MCLK (see Table 14).

After a reset, XMC1000 devices can operate in different baudrate modes:

- Standard baudrate mode
  - In this mode the XMC1000 always operates with a fixed baudrate that is determined by the first bytes arriving on UART's Rx line.
- Enhanced baudrate mode
  - In this mode the XMC1000 starts with an initial baudrate that is also set by the first bytes arriving on UART's Rx line.
  - XMC1000's initial baudrate can be modified by the download tool to a 'working baudrate' up to 3,996,000 Baud depending on the internal MCLK (see Table 14).

2.1.3 Standard baudrate mode

The “Standard baudrate mode” is activated by the download tool on the ‘0x00 0x6C’ command. In response, the XMC1000 answers with ‘0x5D’. If a different reply is returned, the XMC1000 fails to recognize the baudrate used by the download tool.

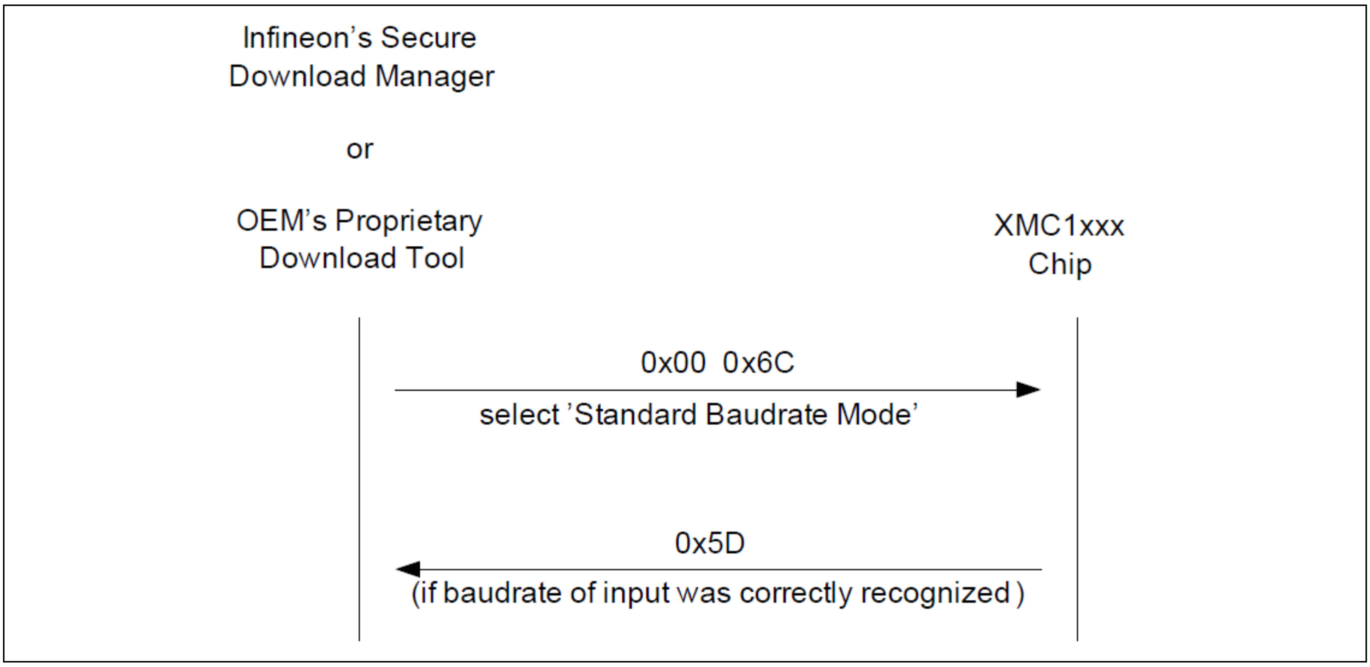


Figure 6 Protocol flow in standard baudrate mode

2.1.4 Enhanced baudrate mode

The Enhanced Baudrate mode is activated by the download tool via the ‘0x00 0x93’ command.

XMC1000 answers with ‘0xA2’ and a ‘PDIV’ value in the same baudrate.

The download tool analyses the received ‘PDIV’ value and uses it for the calculation of the ‘STEP’ value defining the requested ‘final baudrate’.

After transmission of ‘0xF0’ by both XMC1000 and the download tool, the switch to the ‘final baudrate’ has been successfully finished.

AA step devices

Note: For AA step devices, the ‘0xF0’ Acknowledge from an XMC1xxx device is transmit in the ‘final baudrate’ as shown in Figure 7.

AB step devices

Note: For AB step devices, the ‘0xF0’ Acknowledge from an XMC1xxx device is transmit in the ‘initial baudrate’ as shown in Figure 8.

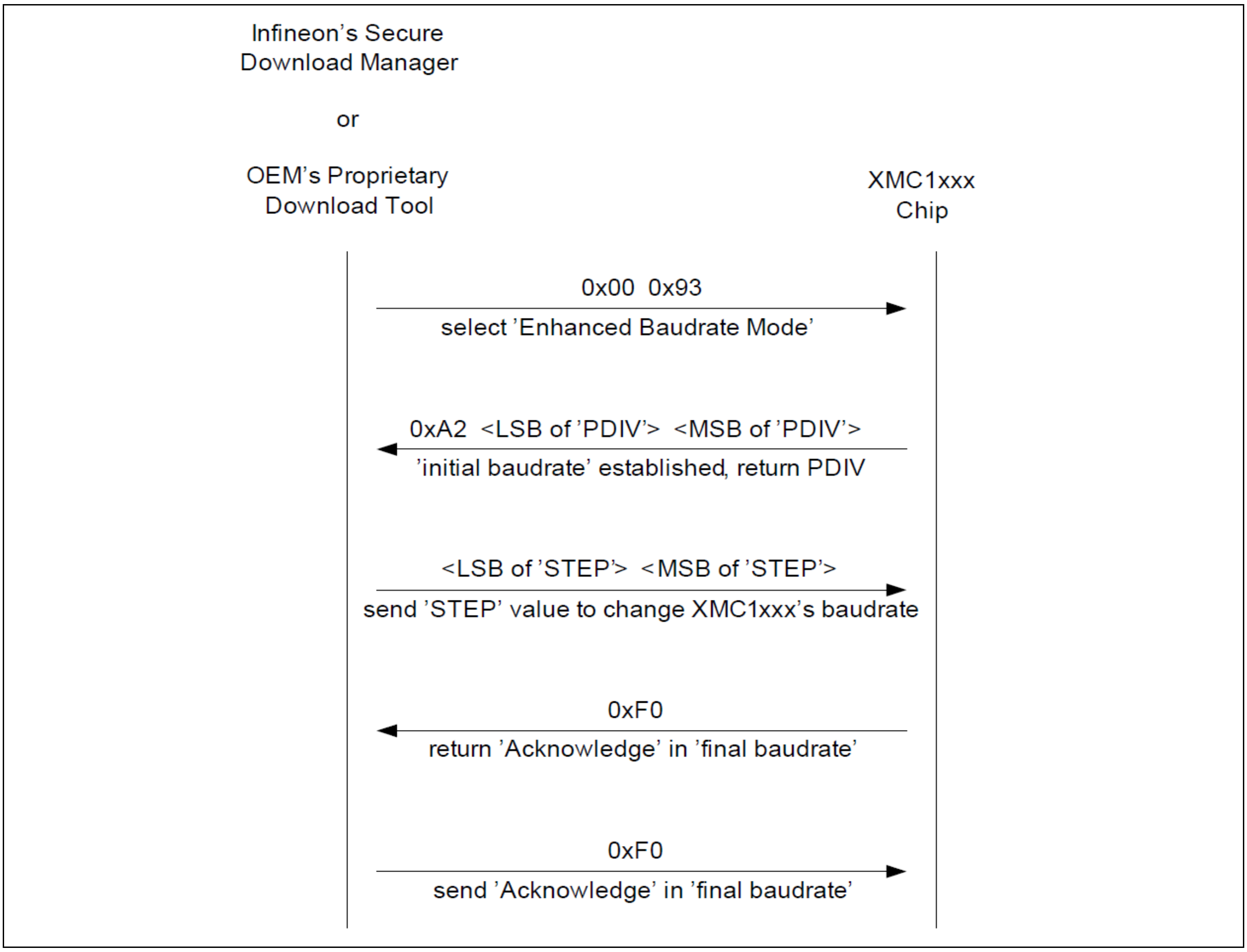


Figure 7 Protocol flow in enhanced baudrate mode for AA step devices

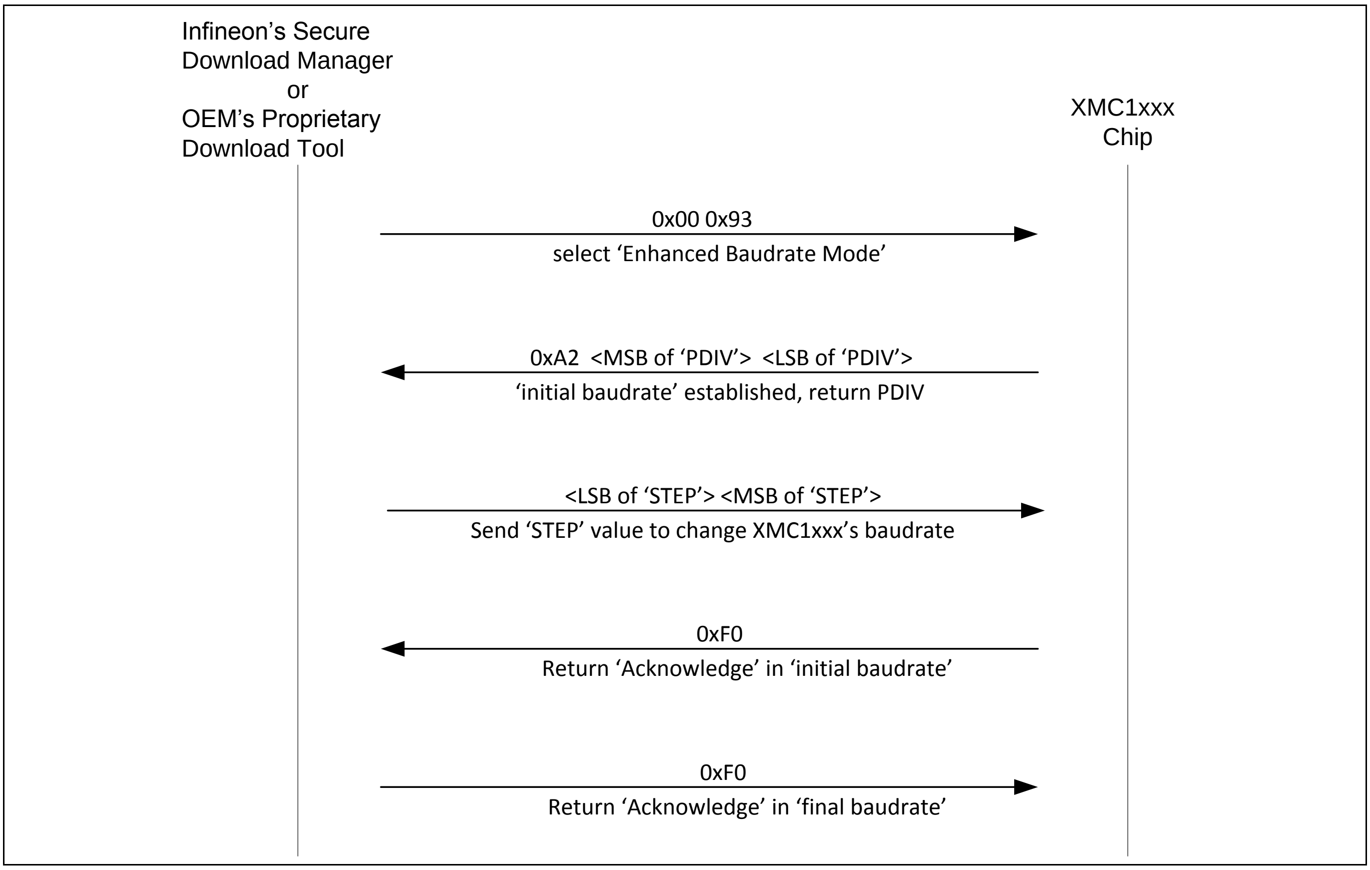


Figure 8 Protocol flow in enhanced baudrate mode for AB step devices

2.1.4.1 Analysis of PDIV value

The returned PDIV value is used to calculate the Master Clock Frequency (MCLK) in the XMC1000 device according to following formula:

MCLK = Initial baudrate x (PDIV + 1) x 8

Example

- Initial baudrate = 9,600 Baud = 9,600 Hz
- PDIV = 0x00 0x67 = 0x0067 = 103

Results in: MCLK = 9,600 Hz x 104 x 8 = 7.9872 MHz ~ 8 MHz.

As indicated by Table 14, there are different minimum and maximum baudrates that are allowed depending on the current MCLK value.

Table 14 Supported baudrates

MCLK	Minimum initial baudrate (Baud)	Maximum initial baudrate (Baud)	Maximum final baudrate (Baud)
2 MHz (min)	300	7200	249750
8 MHz	1200	28800	999000
16 MHz	2400	57600	1998000
32 MHz	4800	115200	3996000

2.1.4.2 Calculation of STEP value

The STEP value is used to adjust the final baudrate according to the formula:

STEP = 1024 x (Target Baudrate / Initial Baudrate) / (PDIV + 1)

Example

- Initial Baudrate = 9,600 Baud
- Target Baudrate = 115,200 Baud
- PDIV = 0x00 0x67 = 0x0067 = 103

STEP = 1024 x (115,200 / 9,600) / 104 = 118.16 ~ 118 = 0x0076 = 0x00 0x76.

2.2 Reading SBSL Id out of the XMC1000 chip

The SBSL Id and further information are read via the flash Get SBSL Status command from the XMC1000 device.

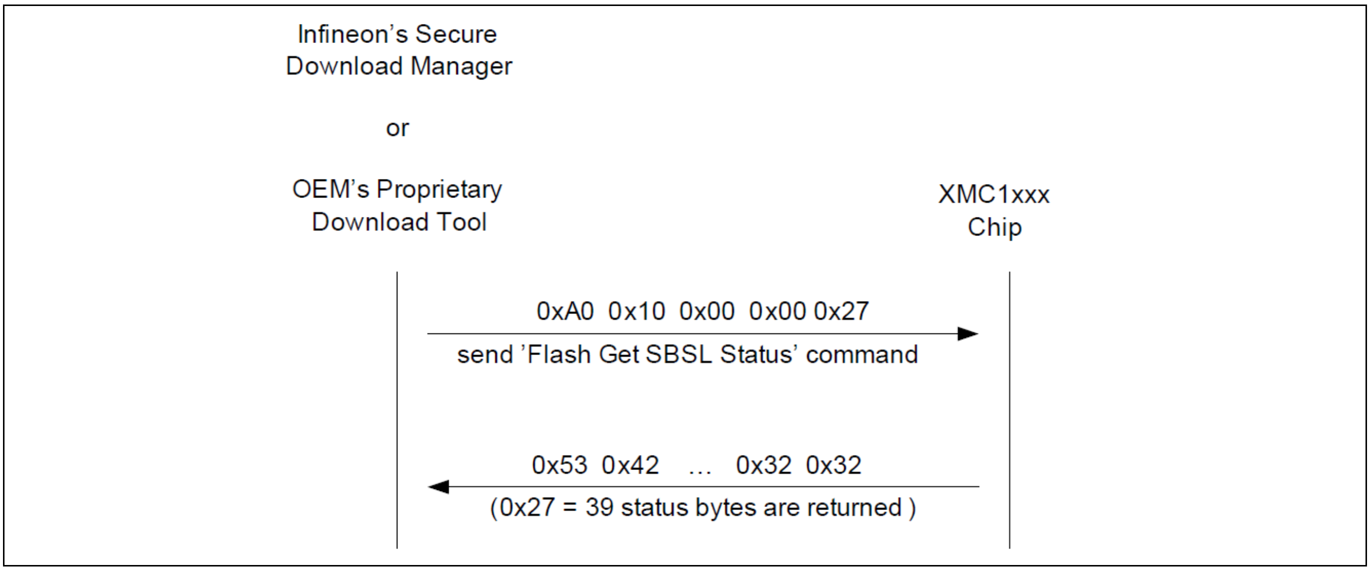


Figure 9 Protocol flow for SBSL Id evaluation

Note: For the layout of the flash get SBSL status reply, please refer to ***Table 7****.*

2.3 Download of new key

File <design>_kc.ldf contains the new key that has to be downloaded before the new application code. The file contains:

```
# SBSLID: 5342534C2D4944203D20323232323232
# LIP: 96377500
A0 12 01 00 14 96 37 75 00 36 BC ED 8E 91 B4 F5 75 E6 86 FA CA B1 BB CA C4
```

Note: Lines starting with '#' hold comments and must not be sent to a XMC1000 device.

The third line contains:

- a command APDU ('A0 12 01 00 14') announcing 0x14 = 20 following key bytes.
- the encrypted new key value.

When XMC1000 accepts the new key, it returns '90 00'. Otherwise, a 2 byte 'error SW1 SW2' (for example '67 xx') is returned.

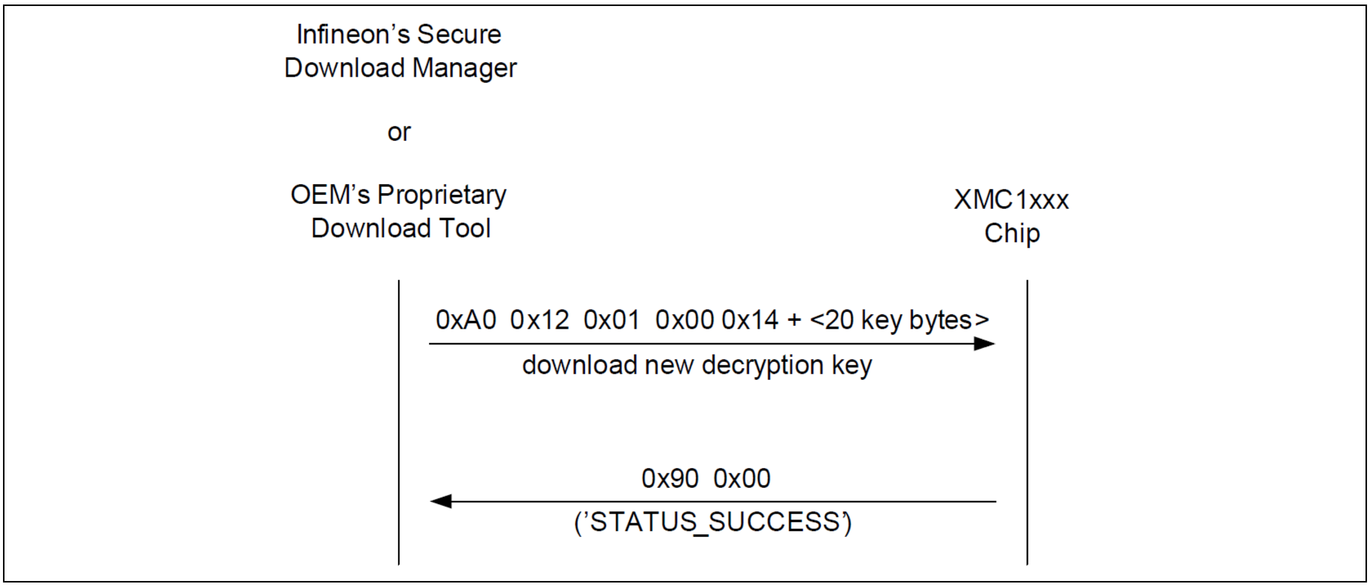


Figure 10 Protocol flow for download of new decryption key

2.4 Download of application code and data

File <design>_ip.ldf contains the encrypted application code and data, which have to be downloaded to XMC1000. The file layout is as follows:

```
# LIP: 96377500
A0 20 00 00 82 A0 00 00 00 16 ... 1E E3 D4 4E E6 68 CA 71 35 65
A0 20 00 00 82 A0 00 00 00 16 ... 2A A6 15 4C A5 5F DD 3E 40 9E
.                          .
.                          .
A0 20 00 00 22 A0 00 00 00 16 ... C5 DE
A0 21 00 00 00
```

Note: Lines starting with '#' hold comments and must not be sent to a XMC1000 device.

The following lines contain:

- a command APDU
  - 'A0 20 00 00 xx' announcing 'xx' following code and/or data bytes.
  - 'A0 21 00 00 00' initiating a checksum calculation over all downloaded bytes.
- the affiliated code and/or data bytes

When the XMC1000 successfully processes the new APDU and <code and/or data bytes> string, it returns '90 00'. Otherwise, a 2 byte 'error SW1 SW2' (e.g. '67 xx') is returned.

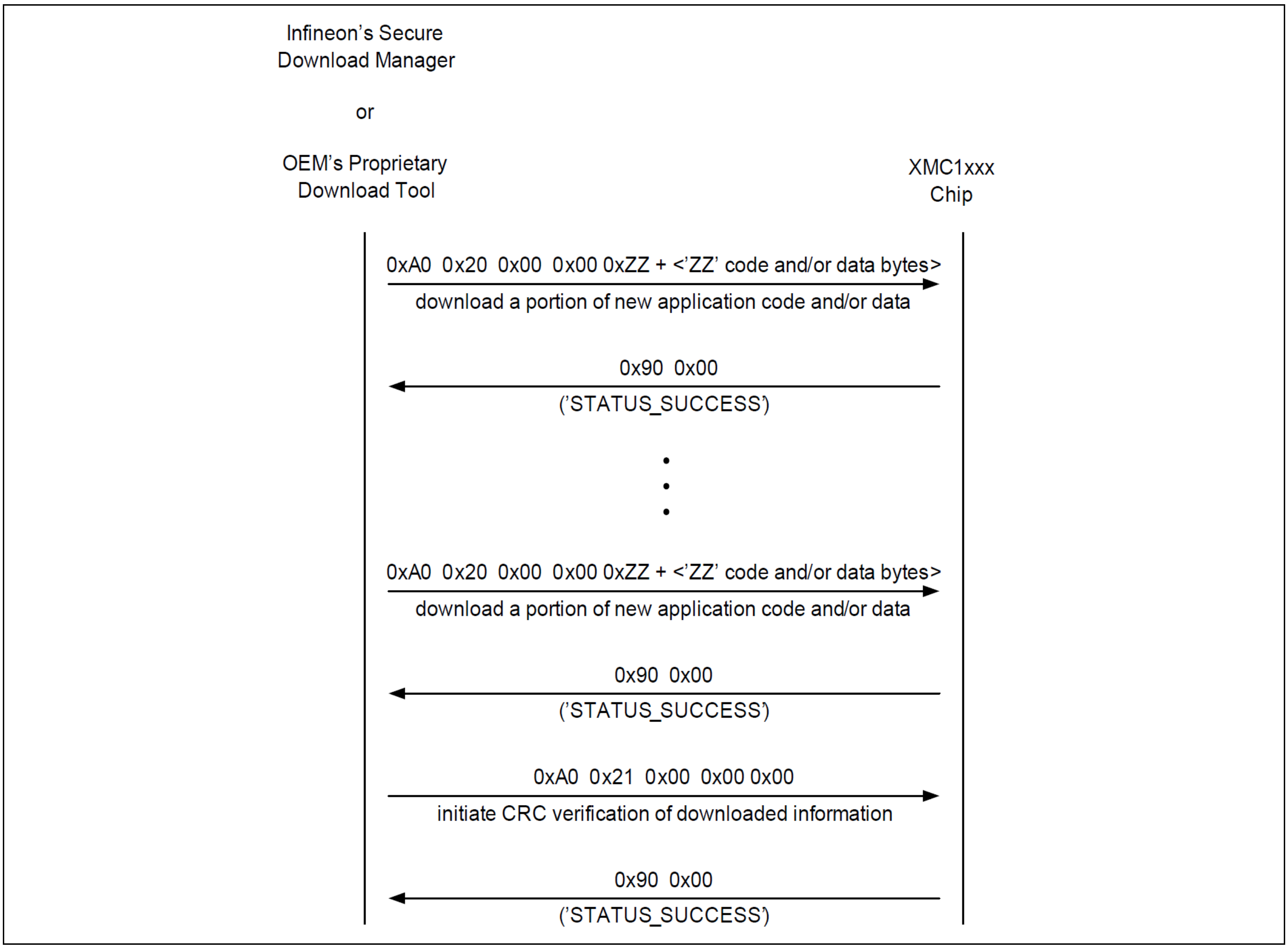


Figure 11 Protocol flow for download of new application code and data

2.5 Lifecycle of XMC1000 S-series device

State diagram of the XMC1000 S-series:

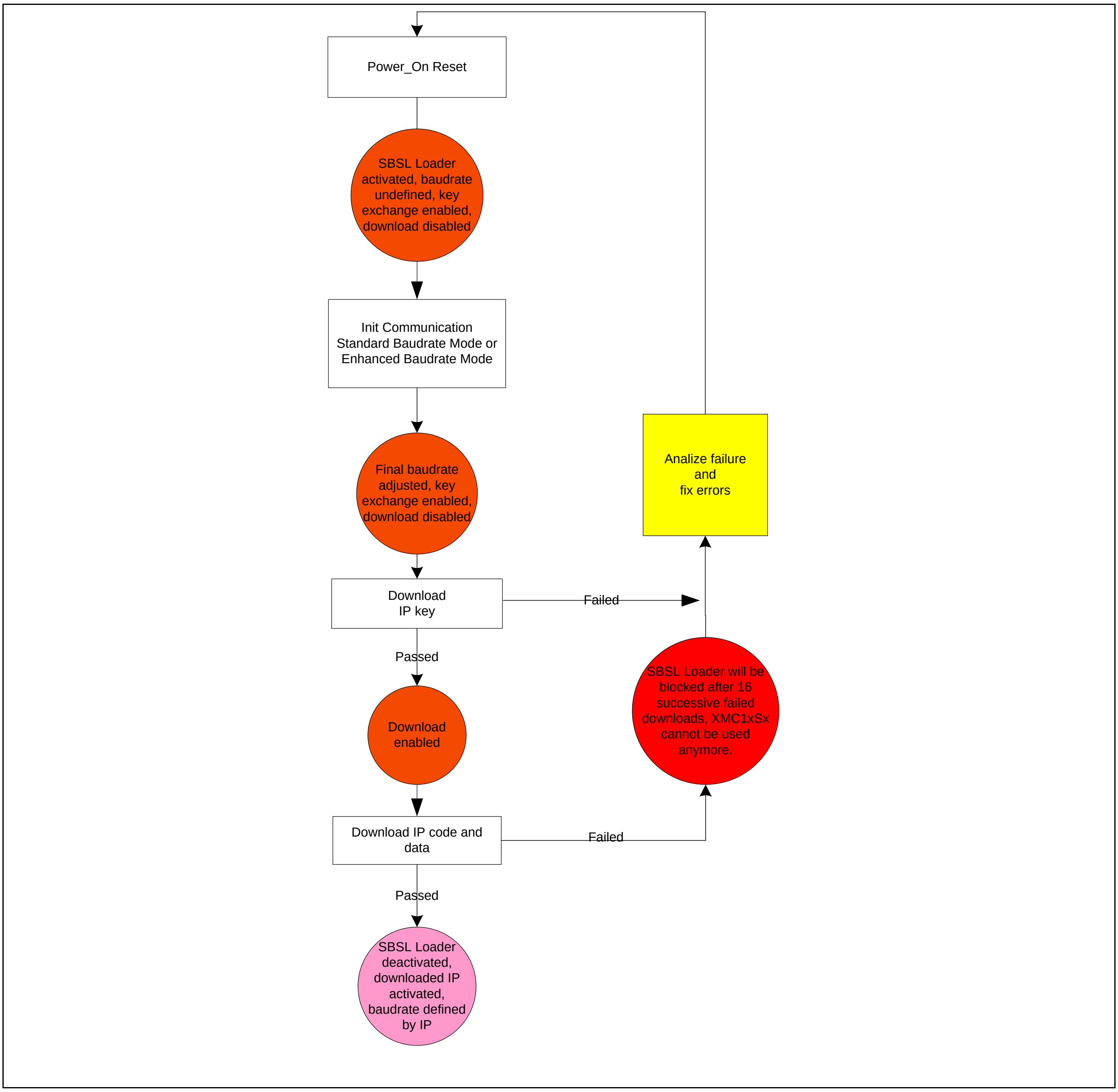


Figure 12 State diagram of SBSL flow

3 Use of Infineon’s Secure Download Manager

The download flow is automatically handled by Infineon’s secure download manager tool. The user only has to:

- create a ‘Download project’, bearing in mind all of the required information for the download.
- specify the path to the requested ‘<design>.zip’ file.
- click the ‘Start Icon’.

These actions can be made interactively or can be executed via a <JavaScript>.js file that is called by a batch process.

Note: For details of the <JavaScript>.js capabilities, see the online help in Infineon’s secure download manager tool.

The download project wizard is started from **File > New > Project** :

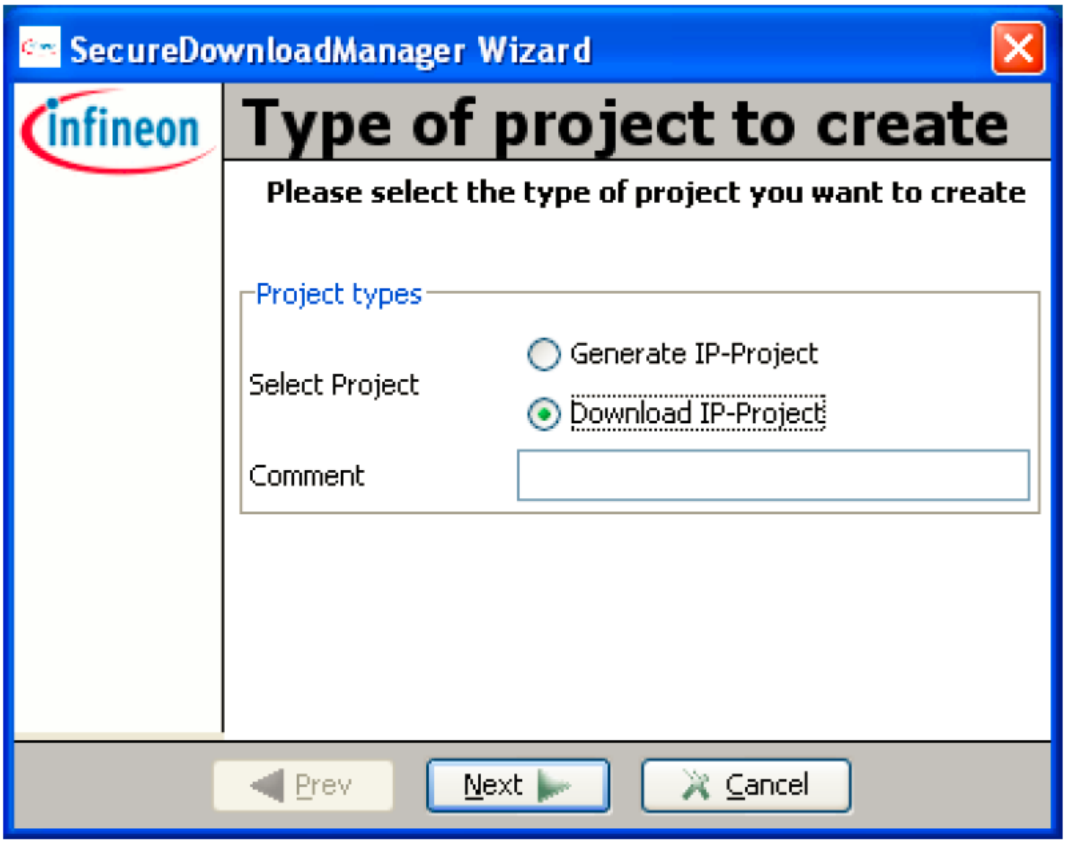


Figure 13 Initial wizard window for creation of a download IP-project

In the next window the selection of the ‘Interactive - use XMC1xxx-chip with serial interface’ option is recommended in order to read all information requested from the linked XMC1000 device. Alternatively, the required information can be manually defined via the ‘offline’ option.

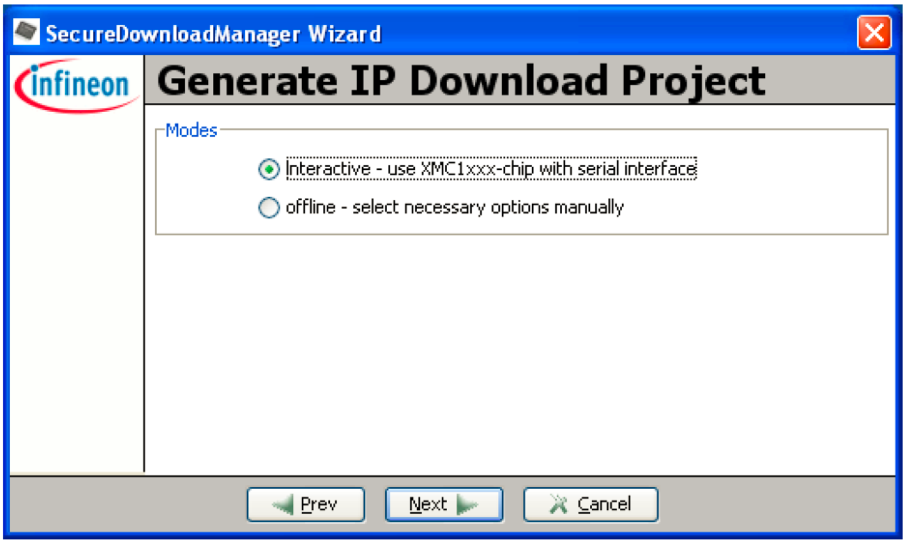


Figure 14 Collection of chip properties for the 'Download IP-Project'

For access to the target device, Infineon's Secure Download Manager tool requires the operation parameters of the 'Chiploader' that interfaces with the target XMC1000 device. This information is summarized in a *.cld file that has been pre-defined by Infineon, or has been generated by the user themselves via the **File > New > Chiploader** menu commands.

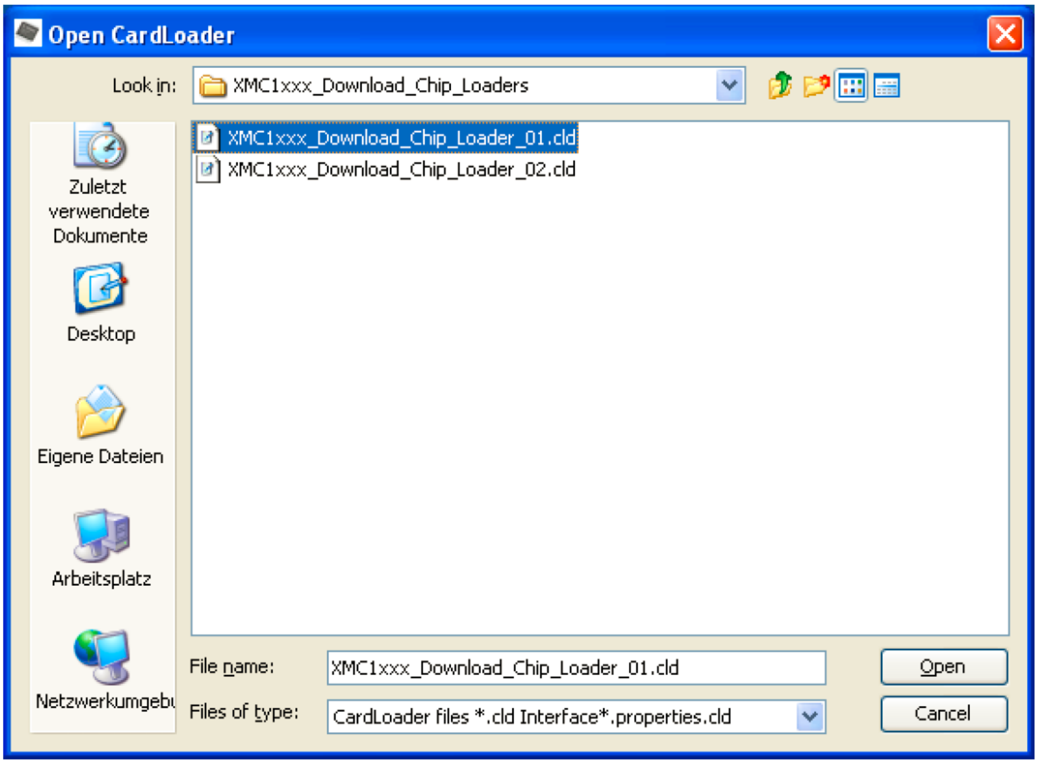


Figure 15 Query operation parameters of Chiploader

The path and name of the *.zip file containing the SBSL ID, the new decryption key, and the new application code and data, is queried by the Select IP File window.

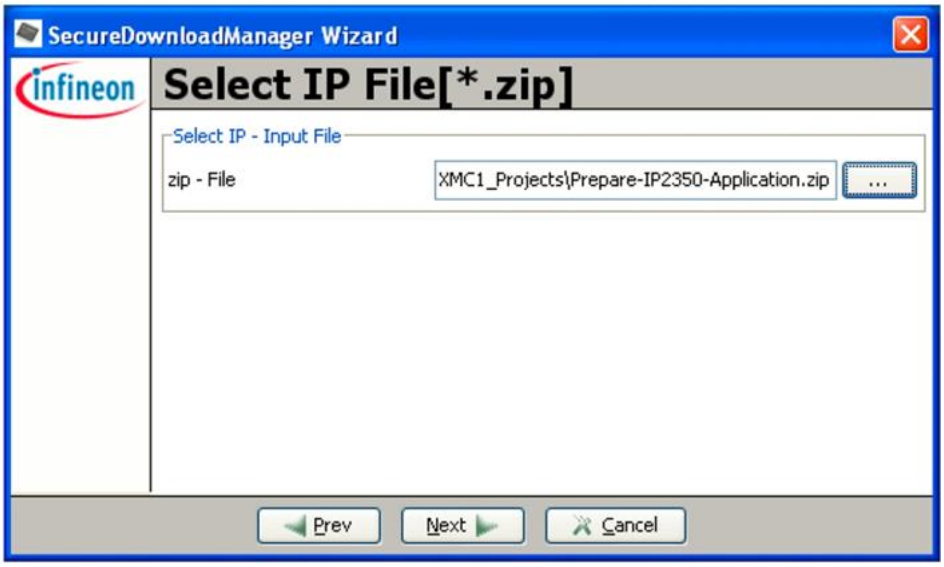


Figure 16 Query path and name of <design>.zip file

All information and references required for a successful execution of the Download Project are stored in a *.ldc file who’s path and name is queried by the **Wizard - Save file** window.

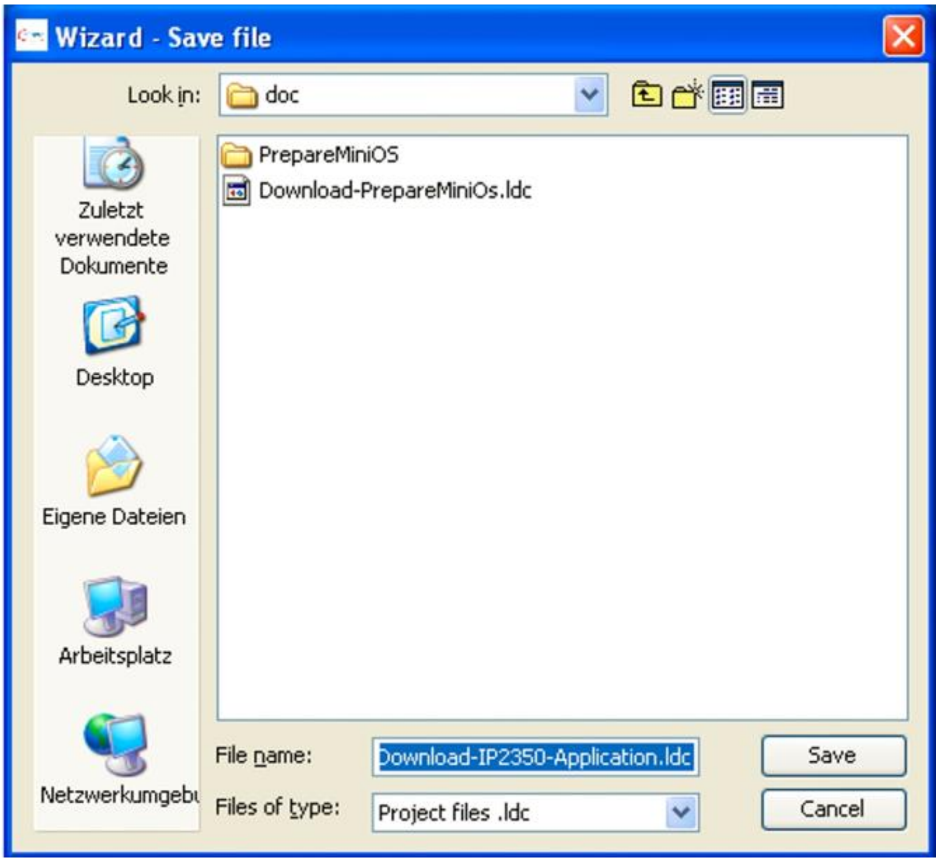


Figure 17 Select path and name of file storing download project properties

Finally, the download is started by a click on the ‘download icon’ indicated by the red colored arrow in the following figure:

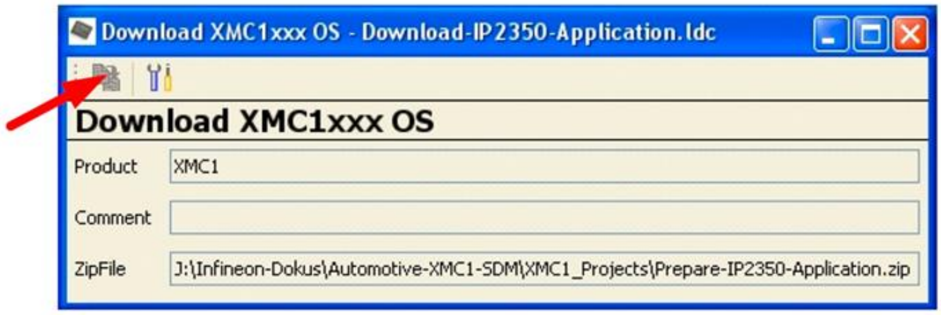


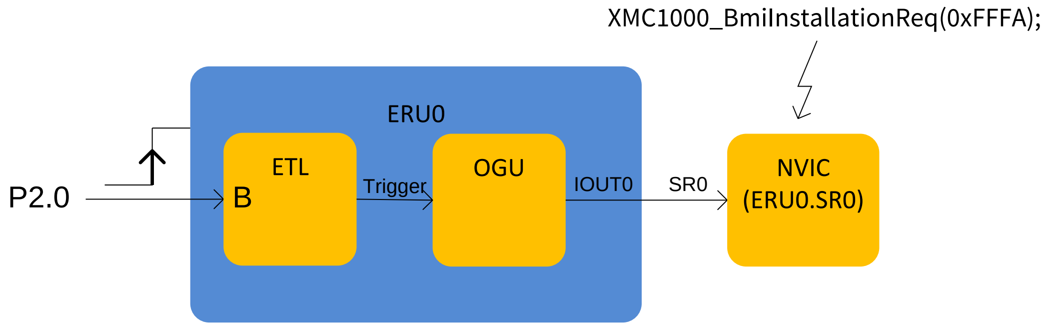
Figure 18 Start of download

4 Programming the BMI value in SBSL mode

Once the IP is successfully programmed into the device flash memory, the SBSL switches the Boot Mode Index (BMI) to the User Productive (UP) mode and begins execution of the IP. Unless the application code of the downloaded IP changes the BMI mode, there is no way to change the device BMI to ASC_SBSL mode.

If you need to change the IP and re-program the XMC1000 S-series device, you will need to embed code in the IP application code to allow changing the Boot Mode Index (BMI) value to Secure BootStrap Loader Mode (i.e. ASC_SBSL= 0xFFFA .) The user routine (XMC1000_BmiInstallationReq()) available inside the ROM allows application software to call and change the BMI value.

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



1. Initially, P2.0 is connected to GND.
2. To trigger a change in BMI to ASC_SBSL, P2.0 is connected to VCC.
3. This creates a rising edge on P2.0 and triggers an external interrupt event.
4. In the ISR, the routine for changing the BMI is called.

Figure 19 Changing the BMI from an external interrupt

4.1 Macro and variable settings

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

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

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

4.2 XMC™ Lib peripheral configuration structure

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

/* Event Trigger Logic Configuration - ERU0.0B0 (P2_0) selected */
XMC_ERU_ETL_CONFIG_t ERU0_ETL_Config =
{
 .input_a = (uint32_t)XMC_ERU_ETL_INPUT_A0, /* Event input selection for A(0-3) */
 .input_b = (uint32_t)XMC_ERU_ETL_INPUT_B0, /* Event input selection for B(0-3) */
 .enable_output_trigger = (uint32_t)1,
 .status_flag_mode =
(XMC_ERU_ETL_STATUS_FLAG_MODE_t)XMC_ERU_ETL_STATUS_FLAG_MODE_HWCTRL,

 /* Select the edge/s to convert as event */
 .edge_detection = XMC_ERU_ETL_EDGE_DETECTION_RISING,
 /* Select the source for event */
 .output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL0,
 .source = XMC_ERU_ETL_SOURCE_B
};

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

4.3 Interrupt service routine function implementation

```
/* Interrupt handler for external trigger interrupt */
//void IRQ3_Handler(void)
void ERU0_0_IRQHandler(void)
{
  /* BMI_installation routine to set BMI = ASC_SBSL */
 XMC1000_BmiInstallationReq(0xFFFA);
}
```

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

4.3.1 Main function implementation

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

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

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

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

 }

}
```

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

5 Revision history

Major changes since the last revision

Page or reference	Description of change
V1.0, 2016-2	Initial release

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 2016-04-22

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN_20161_PL30_0013