AP08013

# XC866

## BB Step

# XC866 Flash Download Using BSL

Microcontrollers

Never stop thinking

**Edition 2006-01-01**

**Published by Infineon Technologies AG,**
**St.-Martin-Strasse 53,**
**81669 München, Germany**

**© Infineon Technologies AG 2006.**
**All Rights Reserved.**

**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 only be used in life-support devices or systems with the express written approval of Infineon Technologies. 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:** **2006-01** V 1.1

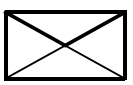
Previous Version: none

| 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**



# 1 Introduction

A built-in bootstrap loader is implemented in the XC800 family to provide a mechanism to load data / code into the internal memory of the device (XRAM or FLASH) via a UART interface. In variants that supports external memory, it is also possible to load data / code to the external memory.

The protocol used in the XC800 family is standard, although there might be a slight variant because of the structure of the flash memory.

## 1.1 Overview of the Documents

This document provides a cookbook on the BSL mode for XC866. It will shows a detail steps on the Flash Downloading. For XRAM downloading, the step to erase the flash can be skipped.

In general, XC866 device has 2 type of variants for BSL mode:

- LIN Variant (e.g: XC866L)
- UART Variant (e.g: XC866)

The access modes for these 2 variants are different

The Users Manual contains detailed information about the BSL Mode Protocol.

The example code is also provided for the reference. Even though the example code is tested, there is no warranty provided.

There are 3 files that comes with this documents:

| | |
|---|---|
| **xc866_bsl.c** | : code that contains the API for bsl mode. |
| **xc866_bsl.h** | : header file |
| **xc866_example.c** | : main example code on how to use the API. |

These files can be compiled in the Visual C 6.0

In order to run the example code, the PC host will need to be connected with a UART cable with the starterkit.

The following files may also be included for the Tips & Recommendations:

(Please note that these files are optional)

| | |
|---|---|
| **xc800_verify.a51** | : assembly code that can be downloaded to verify the content of the Flash. |
| **xc800_verify.hex** | : hex file generated from the above assembly code. |

# 2 Supported BSL Mode in XC866

In order to gain access to the **BSL mode**, the chip must be **reset with MBC and TMS pins pulled low externally**.

Please note the following hardware pins are used:

**P1.0**: Used as RXD (IN)

**P1.1**: Used as TXD (OUT)

## 2.1 UART Mode

For UART variants the following step has to be done by the PC Host:

1. Send the first byte: **0x80** for the auto baud rate recognition.

   Note: Because of the accuracy, it is recommended to use only a range of baud rate from **9600** to **57600** baud rate.
2. Wait for 1 byte **acknowledgement**: **0x55**
3. Send the **bsl_erase_flash()** function to erase the sectors to be downloaded.

   Parameter to be used:

   bankNumber = <Flash Bank to be erased>

   sectorNumber = <Sector to be erased>

   The flash erase will commence before the acknowledgement is sent from microcontroller. Hence, there is a waiting time for about **100 ms**.[1)]

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).
4. Send the **bsl_uart_header()** function.

   Parameter to be used:

   bslHeader.mode = 2

   bslHeader.dataLength = 32

   bslHeader.startAddr = <Starting Address - must by 32-byte aligned>

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

   Details about the header block and the type of acknowledgement is described in the BSL section User Manual.
5. Send the **bsl_uart_data()**

   Parameter to be used:

   bslData.dataLength = 32

1) If P-Flash (Address 0x0000 - 0x2FFF) is selected only, it will take about 100 ms. Likewise, if only D-Flash (0xA000 - 0xAFFF) is selected, it will take about 100 ms. If both P-Flash and D-Flash are selected, the erasing will take about 200 ms.

bslData.cDataArray = <Pointer to the Data Array>

It is recommended to send **only 32-bytes** of data for the Flash Programming.

The flash programming will commence before the acknowledgement is sent from microcontroller. Hence, there is a waiting time for about **2 ms**.

Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

6. Repeat the data sending until all data is sent (**if and only if the address is continuous**).
7. Send the **bsl_uart_eot()**.

   This routine must have an empty data buffer (when this API is used for Flash Download).

   Parameter to be used:

   bslEOT.dataLength = 32

   bslEOT.lastCodeLength = 0

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).
8. Complete the flash download by repeating Step 4 - Step 7 (in case the address are discontinuous).

***Note: Step3 - Step7 is performed in the file_download_xc866() function (see API in chapter 4).***

## 2.2 LIN Mode

For LIN variants, the following step has to be done by PC Host:

1. Send the **bsl_lin_header()**.

   Parameter to be used:

   bslHeader.NAD = 0x7F

   bslHeader.dwbaudrate = 9600

   bslHeader.startAddr = 0

   bslHeader.mode = 0

   bslHeader.fastLin = 1 (to enter the Fast LIN mode)

   Wait for 9 bytes acknowledgement (embedded inside the function).

   The routine will check that the **second byte** has a value of **0x55** to indicate a success.

   Note: Because of the accuracy, it is recommended to use only a range of baud rate from **9600** to **57600** baud rate.
2. Send the **bsl_erase_flash()** function to erase the sectors to be downloaded.

Parameter to be used:

bankNumber = <Flash Bank to be erased>

sectorNumber = <Sector to be erased>

The flash erase will commence before the acknowledgement is sent from microcontroller. Hence, there is a waiting time for about **100 ms.**[1)]

Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

3. Send the **bsl_uart_header()** function.

   Parameter to be used:

   blHeader.mode = 2

   bslHeader.dataLength = 32

   bslHeader.startAddr = <Starting Address - must by 32-byte aligned>

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

   Details about the header block and the type of acknowledgement is described in the BSL User Manual.

4. Send the **bsl_uart_data()**

   Parameter to be used:

   bslData.dataLength = 32

   bslData.cDataArray = <Pointer to the Data Array>

   It is recommended to send **only 32-bytes** of data for the Flash Programming.

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

   The flash programming will commence before the acknowledgement is sent from microcontroller. Hence, there is a waiting time for about 2 ms.

5. Repeat the data sending until necessary (for the continuous address).
6. As the last procedure to complete the whole cycle, send bsl_uart_eot().

   This routine must have an empty data buffer.

   Parameter to be used:

   bslEOT.dataLength = 32

   bslEOT.lastCodeLength = 0

   Wait for 1 byte acknowledgement: 0x55 (embedded inside the function).

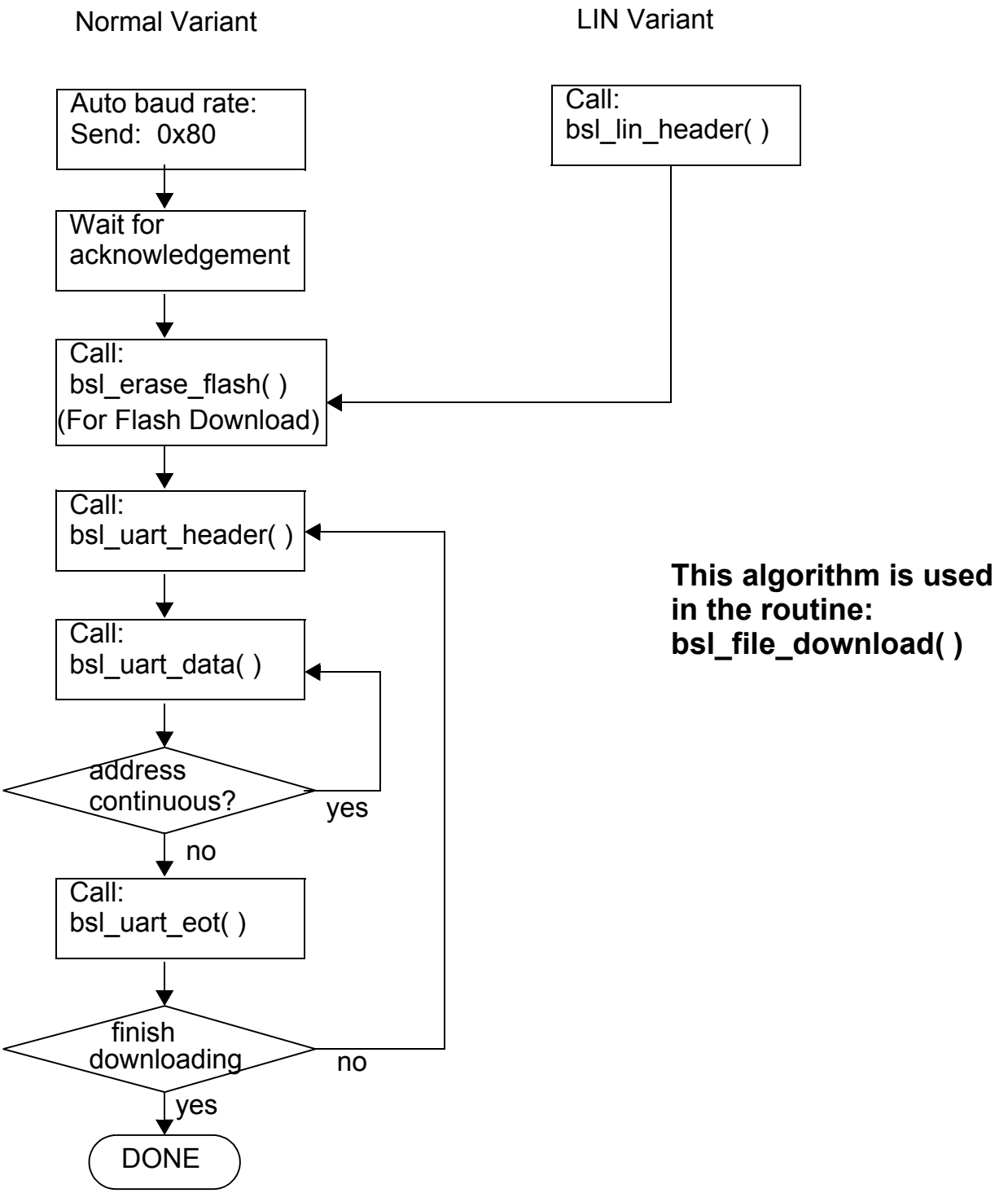
7. Complete the flash download by repeating Step 3 - Step 6 (in case the address are discontinuous).

1) If P-Flash is selected only, it will take about 100 ms. Likewise, if only D-Flash is selected, it will take about 100 ms. If both P-Flash and D-Flash are selected, the erasing will take about 200 ms.

***Note:*** ***Step2 - Step6 is performed in the file_download_xc866() function (see API in chapter 4).***

***Note:*** ***For the LIN Variant, if single wire is used, then the echo to the PC Host must be taken care of. In this case, the provided function will not work and needs to be slightly adapted.***

## 2.3 BSL Diagram for code download



# 3 Flash Protection / Un-protect

XC866 BB step allows the Flash to be protected by password after the download.

The same method, if repeated (with the same password) will unprotect the flash and erase all of the flash content automatically.

After the Flash Protection is done, no further Flash Downloading can be done.

The steps to protect / unprotect the Flash is the same for both variants (LIN and non-LIN).

After downloading is done, the following instruction can be send:

1. Send **bsl_uart_header()**.

   Parameter to be used:

   | | |
   |---|---|
   | bslHeader.mode | = 6 |
   | bslHeader.password | = <1 character password> |

If the same step is repeated, the flash will be unprotected and the content of Flash Memory will be automatically erased.

***Note: After sending this header, the device will exit from BSL mode. If there is a need to perform further activity, it is necessary to perform the reset sequence again.***

# 4 API Description of XC866_BSL

## 4.1 Data type structure in the XC866_BSL:

```
typedef struct BSL_HEADER {
    unsigned char NAD;              //  NAD for LIN.
    unsigned char mode;             //  0  = Download to XRAM
                                    //  1  = Run from XRAM (F000)
                                    //  2  = Download to FLASH
                                    //  3  = Run from FLASH (0000)
                                    //  6  = Password protect/unprotect
    unsigned int  startAddr;        //  Starting Address.
    unsigned char dataLength;       //  UART Mode:
                                    //  Data Byte Length to be
                                    //  written in a subsequent
                                    //  DATA Block.
    unsigned char fastLIN;          //  Only for LIN mode (0 = Normal
                                    //  LIN, 1 = Fast LIN)
    unsigned char password;         //  1 byte password to protect
                                    //  or unprotect Flash (Mode 6 only)
    DWORD            dwBaudrate;    //  Baud rate (Only necessary for
                                    //  LIN Mode)
} BSL_HEADER;

typedef struct BSL_DATA {
    unsigned char NAD;              //  NAD for LIN.
    unsigned char *cDataArray;      //  Pointer to the data to be loaded.
    unsigned char dataLength;       //  Data Byte Length to be loaded (MUST
                                    //  be the same value as in BSL HEADER)
    DWORD         dwBaudrate;       //  Baud rate (Only necessary for LIN
                                    //  Mode)
} BSL_DATA;

typedef struct BSL_EOT {
    unsigned char NAD;              //  NAD for LIN.
    unsigned char *cDataArray;      //  Pointer to the data to be loaded.
    unsigned char lastCodeLength;   //  Data Byte Length to be loaded
                                    //  (lastCodeLength < dataLength).
    unsigned char dataLength;       //  Data Byte Length as stated in BSL
                                    //  HEADER and BSL DATA
    DWORD         dwBaudrate;       //  Baud rate (Only necessary for LIN
                                    //  Mode)
} BSL_EOT;

typedef struct BSL_DOWNLOAD {
    char *hexFileName;              //  Hex File Name.
```

```
    unsigned eraseFlash;          //  0 = Flash will NOT be erased before
                                      downloading
                                  //  1 = Flash will be erased before
                                      downloading
    unsigned eraseOnly;           //  0 = Flash will be downloaded after
                                      erasing.
                                  //  1 = Flash will NOT be downloaded
                                      after erasing.
    unsigned verbose;             //  0 = No message will be displayed.
                                  //  1 = (Default) Message will be
                                      displayed.
    unsigned *xram_valid;         //  0 = No Download to XRAM is done
                                  //  1 = Download to XRAM is done
    unsigned char singleWire;     //  Set to 1 if Single Wire connection
                                      is used.
} BSL_DOWNLOAD;

typedef struct BSL_ERASE {
    unsigned bankNumber;          //  Bank Number to indicate the bank to
                                      be erased.
    unsigned sectorNumber;        //  Sector Number to indicate the
                                      sector to be erased.
    unsigned deviceNumber;        //  0 = XC866 (and its derivatives).
                                      1 = XC886 (and its derivatives).
    unsigned startAddress;        //  Reserved
    unsigned char option;         //  Reserved
} BSL_EOT;


typedef struct AUTO_BAUDRATE {
    unsigned char detection;      //  Detection Mode:
                                  //  0 = UART Mode (No Auto Detection)
                                  //  1 = LIN  Mode (No Auto Detection)
                                  //  2 = Auto Detection Mode
                                  //       Byte 0x80 will be send first
                                  //      and wait for 10ms.
                                  //       If no response, then proceed
                                  //       with LIN auto detection.
    unsigned char mode;           //  Same as BSL HEADER
    unsigned char fastLIN;        //  Only for LIN mode (0 = Normal LIN,
                                  //  1 = Fast LIN)
    DWORD         dwBaudrate;     //  Baud rate
} AUTO_BAUDRATE;
```

## 4.2 Function prototype in XC866_BSL:

```
/*---------------------------------------------------------------------
Function Name          : bsl_init_uart()
Description            : Responsible to initialize the UART Port (COM
                         PORT)
                         Following actions are done:
                         -) Initialize the chosen COM PORT with the
                         selected baud rate
                         -) Return the hComm handle.
                       The parameter *uiError will be updated accordingly.
Function Called        : None
Input Parameter        : *cPortName     => Port Name (e.g: "COM1", "COM2"
                                           etc)
                         dwBaudrate     => Baud rate
Output Parameter       : *hComm         => Communication Handle
                         *uiError       => Error Code
Return Value           : None
---------------------------------------------------------------------*/
/*---------------------------------------------------------------------
Function Name          : bsl_autobaudrate()
Description            : Sending the first byte for auto baud rate
                         detection.
                         If UART or LIN auto detection is enabled, the
                         function will first send byte: 0x80 to detect if
                         it is UART or not.  If there is no responds within
                         the specified time-out value, then send the first
                         LINE header (the chip must be reset again!!)
Function Called        : bsl_lin_header()
Input Parameter        : *hComm         => Communication Handle.
                         autobaud       => AUTO_BAUDRATE structure.
Output Parameter       : *uiError       => Error Code
Return Value           : None
---------------------------------------------------------------------*/
/*---------------------------------------------------------------------
Function Name          : bsl_lin_header()
Description            : Sending the LIN Header Block.
Function Called        : None
Input Parameter        : *hComm         => Communication Handle.
                         bslHeader      => BSL_HEADER structure.
Output Parameter       : *uiError       => Error Code
Return Value           : None
---------------------------------------------------------------------*/

/*---------------------------------------------------------------------
```

```
Function Name          :  bsl_erase_flash()
Description            :  Responsible to erase the flash sector
Input Parameter        :  *hComm         =>Communication Handle.
                          bslErase       =>BSL Erase Structure
                          bankNumber:
                          Bit 0 => Indicate Bank 0
                          Bit 1 => Indicate Bank 1
                          Bit 2 => Indicate Bank 2
                          Bit 3 => Indicate Bank 3
                          sectorNumber:
                           Bank 0:           Bank 1:
                           Bit 0 => Sector0  Bit 4 => Sector0
                           Bit 1 => Sector1  Bit 5 => Sector1
                           Bit 2 => Sector2  Bit 6 => Sector2

                           Bank 2:           Bank 3:
                           Bit 8 => Sector0  Bit 12 => Sector0
                           Bit 9 => Sector1  Bit 13 => Sector1
                           Bit 10=> Sector2  Bit 14 => Sector2
                                             Bit 15 => Sector3
                                             Bit 16 => Sector4
                                             Bit 17 => Sector5
                                             Bit 18 => Sector6
                                             Bit 19 => Sector7
                                             Bit 20 => Sector8
                                             Bit 21 => Sector9
Output Parameter       :  *uiError       =>Error Code
Return Value           :  None
------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Function Name          :  bsl_uart_header()
Description            :  Sending the UART Header Block.
Function Called        :  None
Input Parameter        :  *hComm         =>Communication Handle.
                          bslHeader      =>BSL_HEADER structure.
Output Parameter       :  *uiError       =>Error Code
Return Value           :  None
------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------
Function Name          :  bsl_uart_data()
Description            :  Sending the UART Data Block.
Function Called        :  None
Input Parameter        :  *hComm         =>Communication Handle.
                          bslData        =>BSL_DATA structure.
Output Parameter       :  *uiError       =>Error Code
Return Value           :  None
------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------
Function Name         : bsl_uart_eot()
Description           : Sending the UART EOT Block.
Function Called       : None
Input Parameter       : *hComm        => Communication Handle.
                        bslEOT        => BSL_EOT structure.
Output Parameter      : *uiError      => Error Code
Return Value          : None
-------------------------------------------------------------------*/
/*-------------------------------------------------------------------
Function Name         : close_interface()
Description           : Responsible to close all of the communication
                        channel (UART or JTAG)
Function Called       : None
Input Parameter       : *hComm        => Communication Handle.
Output Parameter      : *uiError      => Error Code
Return Value          : None
-------------------------------------------------------------------*/
/*-------------------------------------------------------------------
Function Name         : file_download_xc866()
Description           : The function to download the hex file.
Function Called       : bsl_erase(), bsl_uart_header(), bsl_uart_data(),
                        bsl_uart_eot()
Input Parameter       : *hComm        => Communication Handle.
                        bslDownload   => BSL_DOWNLOAD structure.
Output Parameter      : *uiError      => Error Code
Return Value          : None
-------------------------------------------------------------------*/
```

# 5 Tips and Recommendations

## 5.1 Flash Verification

Since it is not possible to use BSL mode to read out the memory content, there is a need to download the code to the SRAM and run it from there. For this purpose, sample code is provided: xc800_verify.a51 (xc800_verify.hex)

The hex file will need to be downloaded to the XRAM first and executed from there.

The sample code will do the following:

1. Auto baud rate detection.
2. Wait for the bsl_uart_header( ) from HOST.

   This is to find out the starting address to do verification.
3. Wait for the bsl_uart_data( ) from HOST.

   The data sent during this time will be used to verify the content of the memory, whose address was specified with bsl_uart_header( )
4. Wait for the bsl_uart_eot( ) from HOST.

   This is to indicate that the address is not continuous.
5. Repeat again to step 2 until all code / data is verified.
6. **No Flash Protection** is possible after the code / data verification.

   In order to perform flash protection, chip needs to be reset again and the protection can be done after the auto baud rate detection.

The same steps as described in Chapter 2.1 can be followed for both normal and LIN variants.

In case there is any error, the following responses will be sent for each byte that has comparison error:

| | |
|---|---|
| Verification Error Byte | : 0xFC |
| High Byte Address | : 0xXX |
| Low Byte Address | : 0xXX |
| Actual Data Byte | : 0xXX |
| Expected Data Byte | : 0xXX |

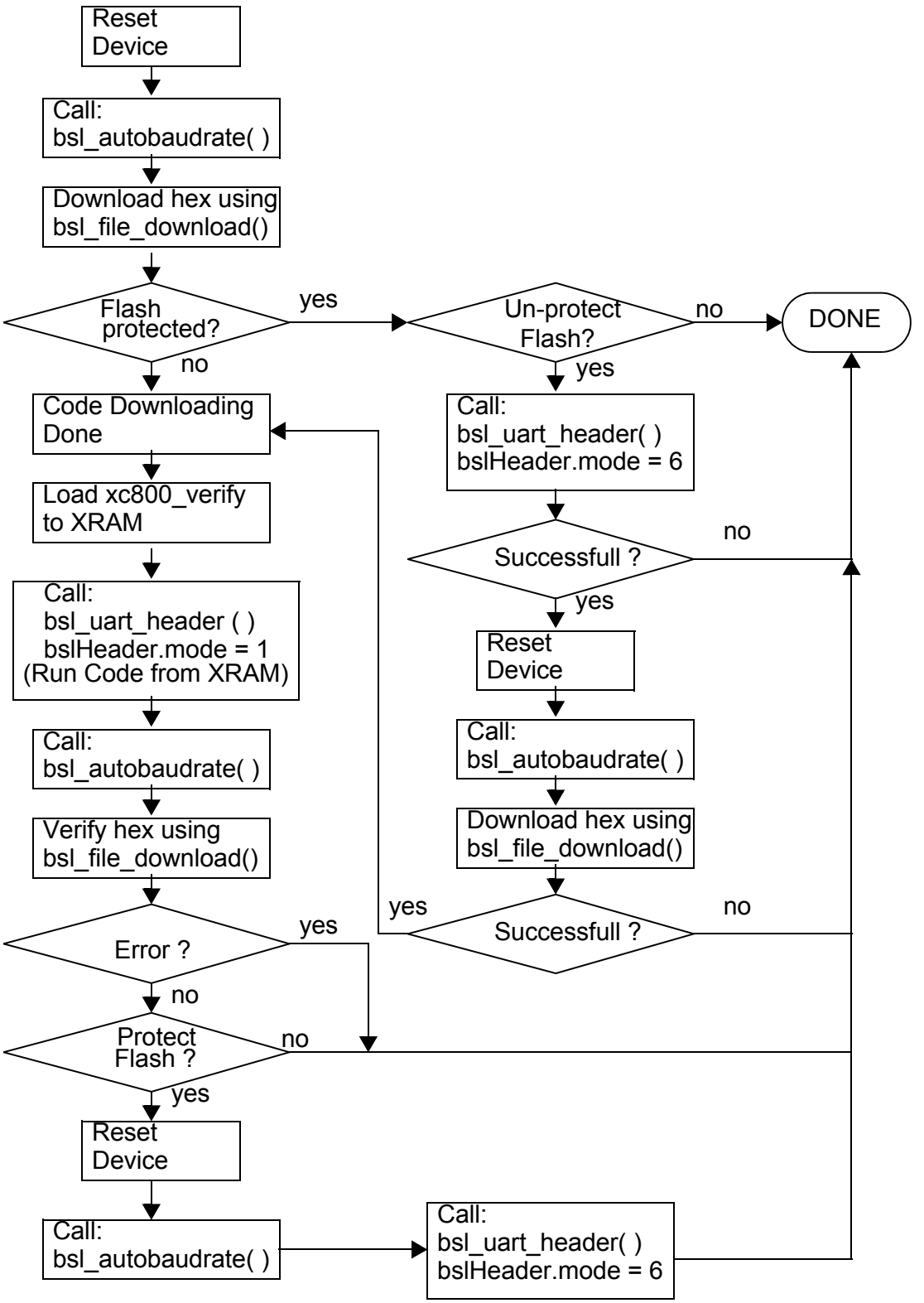
## 5.2 Flash Memory Mapping

The flash can be erased sector-wide. The table below shows the address range for each flash sector.

**Table 1 Flash Memory Mapping**

| BANK | SECTOR | Address Range | Size |
|---|---|---|---|
| 0 | 0 | 0x0000 - 0x0EFF | 3.75 Kbytes |
| | 1 | 0x0F00 - 0x0F7F | 128 Bytes |
| | 2 | 0x0F80 - 0x0FFF | 128 Bytes |
| 1 | 0 | 0x1000 - 0x1EFF | 3.75 Kbytes |
| | 1 | 0x1F00 - 0x1F7F | 128 Bytes |
| | 2 | 0x1F80 - 0x1FFF | 128 Bytes |
| 2 | 0 | 0x2000 - 0x2EFF | 3.75 Kbytes |
| | 1 | 0x2F00 - 0x2F7F | 128 Bytes |
| | 2 | 0x2F80 - 0x2FFF | 128 Bytes |
| 3 | 0 | 0xA000 - 0xA3FF | 1 KBytes |
| | 1 | 0xA400 - 0xA7FF | 1 KBytes |
| | 2 | 0xA800 - 0xA9FF | 512 Bytes |
| | 3 | 0xAA00 - 0xABFF | 512 Bytes |
| | 4 | 0xAC00 - 0xACFF | 256 Bytes |
| | 5 | 0xAD00 - 0xADFF | 256 Bytes |
| | 6 | 0xAE00 - 0xAE7F | 128 Bytes |
| | 7 | 0xAE80 - 0xAEFF | 128 Bytes |
| | 8 | 0xAF00 - 0xAF7F | 128 Bytes |
| | 9 | 0xAF80 - 0xAFFF | 128 Bytes |

## 5.3 Flow chart of the example code



## 5.4 BSL Protocol Used

### 5.4.1 Flash / XRAM code download / verification:

**Header Block**

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|
| **TYPE (0)** | **MODE** | **Start Address High** | **Start Address Low** | **Block Length** | **Not Used** | **Not Used** | **checksum** |

**Data Block**

| Byte 0 | Byte 1 - (Block_Length - 2) | Byte (Block_Length - 1) |
|---|---|---|
| **TYPE (1)** | **Program Codes ((Block_Length - 2) byte)** | **checksum** |

**EOT Block**

| Byte 0 | Byte 1 | Byte 2 - ... | ... | ... |
|---|---|---|---|---|
| **TYPE (2)** | **Last_Codelength** | **Program Codes (Last_Codelength) byte** | **Not Used (Block_Length-3-Last_Codelength) byte** | **checksum** |

### 5.4.2 Flash Erasing:

**Header Block**

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|
| **TYPE (0)** | **MODE (0x04)** | **Bank 0 (Sector0-2)** | **Bank 1 (Sector0-2)** | **Bank 2 (Sector0-2)** | **Bank 3 (Sector0-7)** | **Bank 3 (Sector8-9)** | **checksum** |

### 5.4.3 Flash Protection / Unprotection

**Header Block**

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|
| **TYPE (0)** | **MODE (0x06)** | **Password** | **Not Used** | **Not Used** | **Not Used** | **Not Used** | **checksum** |

| | Description |
|---|---|
| **TYPE** | 0 = Header Block<br>1 = Data Block<br>2 = EOT Block |
| **MODE** | 0 = Download to XRAM<br>1 = Execute Code from XRAM (leave BSL mode)<br>2 = Download to FLASH<br>3 = Execute Code from Flash (leave BSL mode)<br>4 = Erase Flash<br>6 = Flash Protection / Un-protection (leave BSL mode) |
| **Start Address High / Low** | 16 bit starting address for the programming. |
| **Block Length** | The number of bytes for the subsequent Data and EOT blocks (including Type and checksum) |
| **Program Codes** | The bytes to be programmed into the memory. |
| **Last_CodeLength** | The number of bytes of the next program codes to be programmed into the memory. |
| **Bank x (sector y - z)** | Each bit in this byte, represent the sector to be erased (starting from the LSB) |
| **password** | The password for the protection / un-protection of Flash. |
| **Response (Acknowledge Frame)** | 0x55 = OK<br>0xFE = Checksum Error<br>0xFD = Flash is protected<br>0xFC = Verification Error |

### 5.4.4 LIN Auto baud rate Detection

**Master Request Header + Command**

Byte 0

| SYN Break: 0x0 (Half baud rate) |
|---|

→ Wait 1 ms

| Byte 0 | Byte 1 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte7 | Byte8 | Byte9 | Byte10 | Byte11 |
|---|---|---|---|---|---|---|---|---|---|---|
| SYN Char (0x55) | ID (0x3C) | NAD (0x7F) | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | Fast LIN 0x01 | checksum (0x80) |

→ Wait 50 ms

**Slave Response Header**

Byte 0

| SYN Break: 0x0 (Half baud rate) |
|---|

→ Wait 1 ms

| Byte 0 | Byte 1 |
|---|---|
| SYN Char (0x55) | ID (0x7D) |

→ Wait for acknowledgement frame (9 bytes)

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

Published by Infineon Technologies AG