

AN11257

SPI secondary boot loader

Rev. 1 — 10 September 2012

Application note

Document information

Info	Content
Keywords	LPC11xx, LPC17xx, SBL, SPI
Abstract	This application note describes how to implement a Secondary Boot Loader (SBL) using SPI Communication.



Revision history

Rev	Date	Description
1	20120910	Initial version.

Contact information

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This document describes the functional requirement specifications, implementation and usage of the SPI Secondary Boot Loader (SBL) on LPC11xx and LPC17xx family.

1.1 Project overview

This project can run on the LPC11xx and LPC17xx MCUs. In these MCUs, the primary boot loader resides in the boot block. The boot loader is executed every time the part is powered on or reset. It can execute the ISP command handler or the user application code, which is stored in sector 0 of internal flash memory.

The SBL in this project refers to a user-defined application that provides the user with an option to update the User Application Firmware or execute the previously programmed User Application Firmware. It is placed from the address 0x00 so that when the primary boot loader runs user application, it executes first.

1.2 System components

The system components are shown in [Fig 1](#).

UART communication between the PC and the master MCU is added to give user control over the firmware upgrade process.

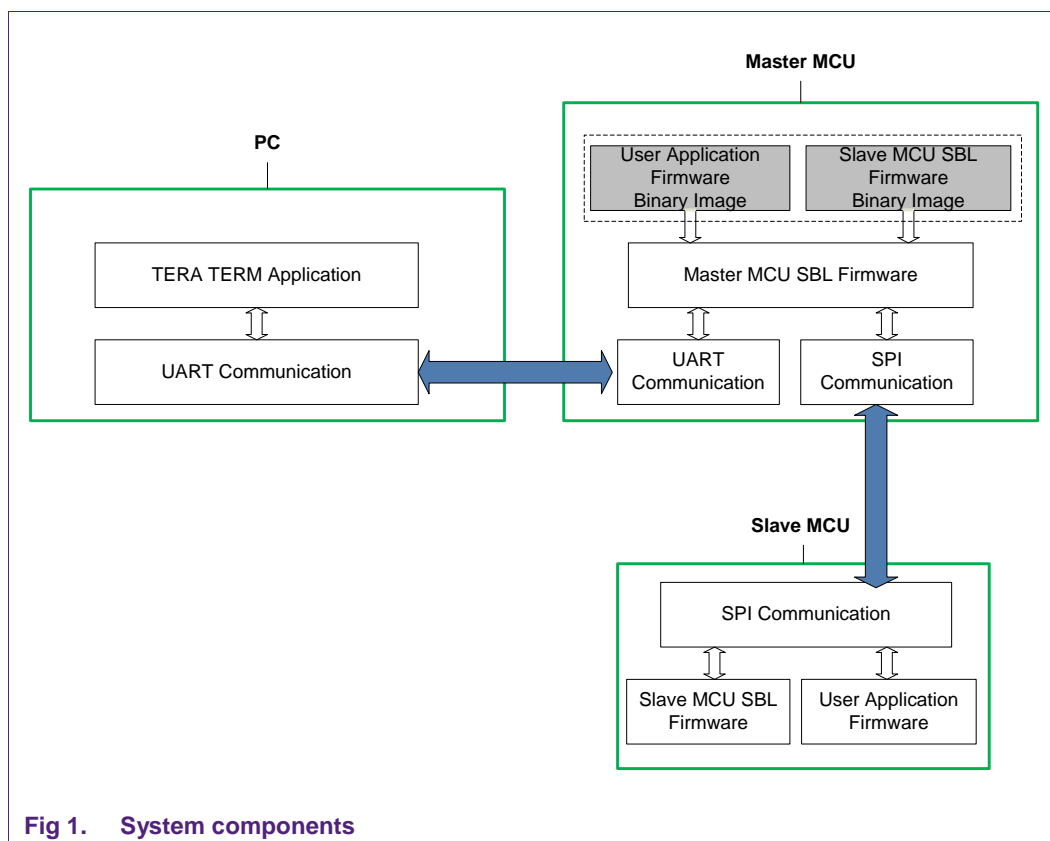


Fig 1. System components

The user interacts with master MCU using Tera Term. The master MCU supports three commands: Read User Application Firmware ID from slave MCU, Upgrade User Application Firmware, and Upgrade SBL Firmware for slave MCU. Firmware data is stored in flash memory of master MCU, from sector 1 to the last sector.

Once the master MCU receives a command, it communicates to the slave MCU to perform the relevant actions. All transmissions between master MCU and slave MCU are done using SPI. The protocol in this case is covered in section 3.

Slave MCU has an SBL located at sector 0 of the flash memory. At startup, SBL validates user code by checking the firmware version ID (the last 4 bytes of flash memory). If the ID is invalid (0xFF), the slave MCU enters SBL mode and waits for new firmware over the SPI bus. If not, the MCU enters Application mode and executes the Reset Handler of User Application Firmware at address 0x1004.

In Application Mode, the slave also supports functions to read its User Application firmware version ID and upgrade the User Application firmware. Moreover, the firmware of SBL can also be upgraded. Using a special command, the user can ask the master MCU to communicate to the slave MCU in order to perform this function.

1.3 SPI communication

SPI is a full duplex serial interface. Device communicates using master/slave relationships, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data are transferred in both directions. It is up to the master and slave devices to know whether a received byte is meaningful or not.

*In this project, the slave MCU will send a **READY** byte (0xAA) on SPI bus once it enters idle state. The master MCU must read this byte before sending any function to Slave.*

A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determines the edges of the clock signal on which the data are driven and sampled. Each of the two parameters has two possible states, which allows for four possible combinations, all of which are incompatible with one another. So a master/slave pair must use the same parameter pair values to communicate.

2. System overview

2.1 Project configuration

SPI configuration:

SPI clock rate: 1 MHz

Transfer size: 8 bits per transfer

Clock phase (CPHA): '1' - Data is sampled on the second clock edge of the SCK.

Clock polarity (CPOL): '1' - SCK is active-low

LSB First (LSBF): '0' - SPI data is transferred MSB (bit 7) first.

UART Configuration:

Baud rate: 115200

Data: 8 bits

Stop bits: 1

Parity bit: None

Flow Control: None

2.2 Memory map

2.2.1 Memory map for master MCU SBL

The master MCU's firmware resides on sector 0 of its flash memory. The remaining sectors are reserved to store firmware data. When master MCU has to upgrade slave MCU SBL firmware or User Application firmware in the slave, it will read data from these sectors.

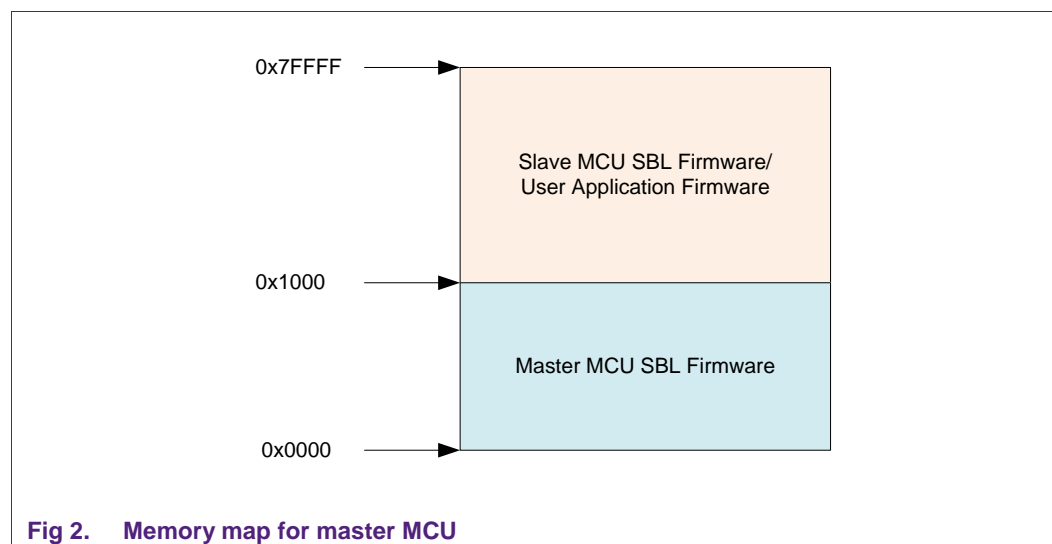


Fig 2. Memory map for master MCU

2.2.2 Memory map for slave MCU SBL

The flash memory is divided into two regions. One region is for the placement of the User Application Firmware and the other region is for the placement of Secondary Boot Loader. The SBL is placed at the first 4 kB of flash memory and runs first when the system resets.

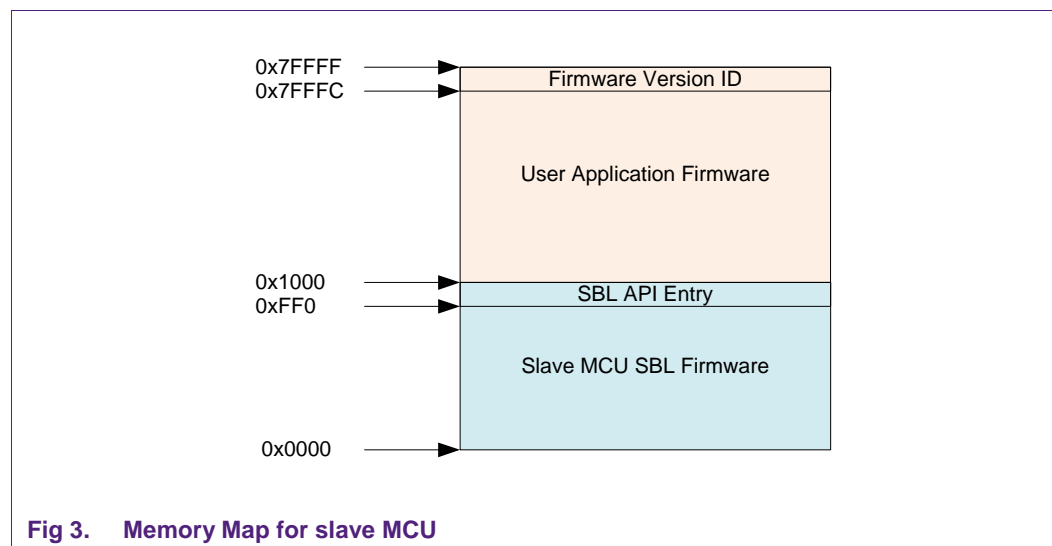


Fig 3. Memory Map for slave MCU

As shown in [Fig 3](#), the Firmware Version ID is stored in the last 4 bytes of flash memory; from 0x7FFFC for LPC17xx MCU and 0x7FFC for LPC11xx MCU. The SBL also provides an entry point for calling SBL APIs. The address of the entry is from 0xFF0.

3. Functional description

3.1 SBL mode

3.1.1 Read firmware version ID

Function ID: 0x31

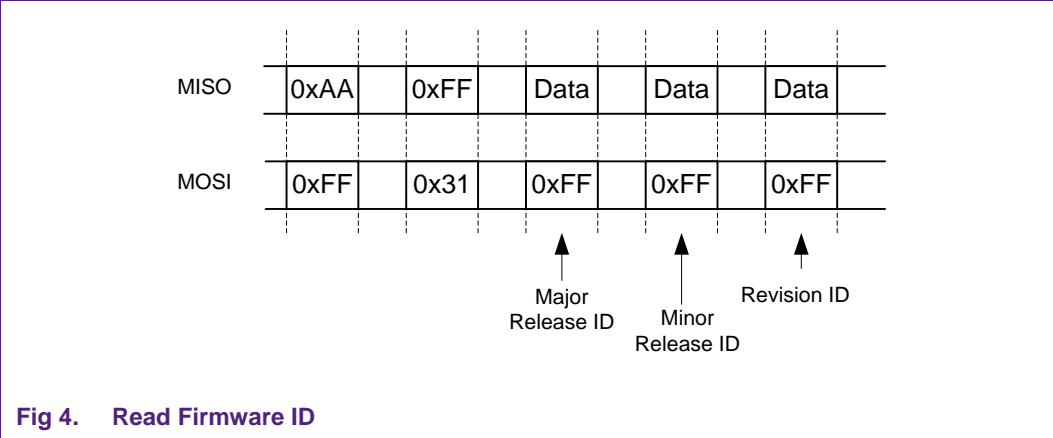
Description:

- This function allows the SBL master to read the firmware version ID of slave MCU SBL.

Implementation:

- The master MCU SBL waits for a READY byte (0xAA) over SPI.
- The master MCU SBL makes a request to read the firmware version ID by issuing a function ID (0x31) to the slave MCU SBL.
- Upon receiving the request, the slave MCU SBL will prepare the firmware version ID information.
- The master MCU SBL will then retrieve the firmware version ID information by issuing an SBL read instruction.

SPI Communication Protocol:



Major release ID:

- Range of ID: 0 - 255

Minor release ID:

- Range of ID: 0 - 255

Revision ID:

- Range of ID: 0, a-f

3.1.2 User application firmware upgrading

Function ID: 0x32

Description

- This function performs programming of the flash memory by the application program in a running system with an aim to upgrade the firmware.

Implementation

- The master MCU SBL waits for a READY byte (0xAA) over SPI.
- The master MCU SBL makes a request to enter IAP mode by issuing a function ID (0x32) to the slave MCU SBL.
- The slave MCU SBL erases the last sector of Flash Memory, which stores the Firmware Version ID information. Then it enters to IAP mode.
- The master MCU SBL transfers firmware data to the slave MCU SBL one block at a time via SPI. Each block of data is 1 kB in size.
- The slave MCU SBL programs the flash one block at a time. The flash on the slave MCU SBL is divided into sectors. Each sector has a size of 4 kB. The slave MCU SBL will always prepare and erase the contents of the flash sector at the beginning of every new sector.
 - When the first block of data has been programmed, the slave MCU SBL is ready to receive the next block of data.
 - When the second block of data is received, since the contents of the entire flash sector have already cleared, the slave MCU SBL shall directly prepare and program the second block of data at a location that is 1 kB offset from the start of the sector.
- This process continues until the master MCU SBL sends the last block of flash data.
- In order to inform the slave MCU SBL that there are no more data, the master MCU SBL would send Block num (MSB) and Block num (LSB) bytes having a value of 0x00.
- When the IAP process is completed, the slave MCU SBL will execute Reset Handler of User Application at address 0x1004.

SBL communication protocol

1. The master MCU SBL follows this communication protocol to send the first 1 kB block of data.

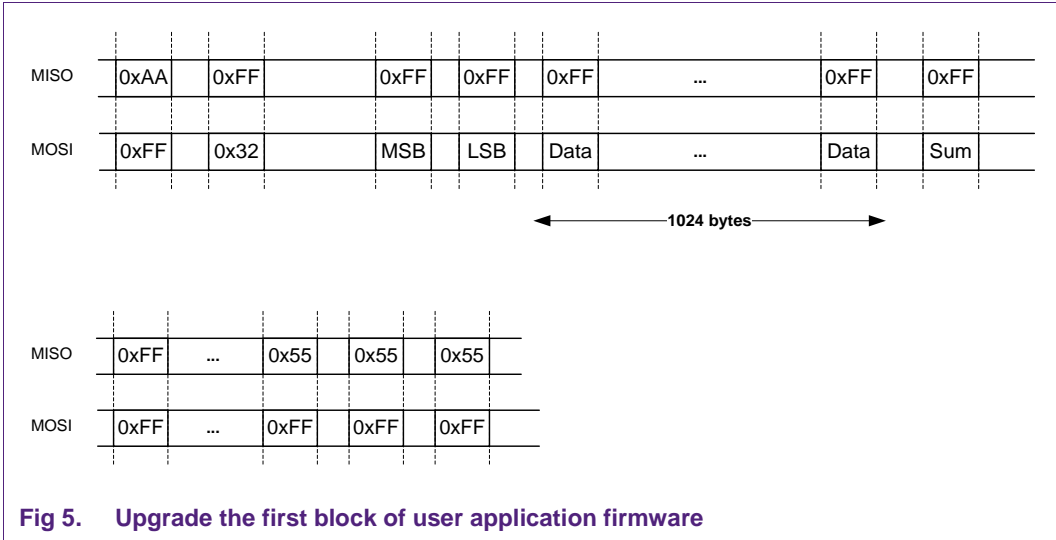


Fig 5. Upgrade the first block of user application firmware

Function ID:

- The function ID used to perform IAP is 0x32. The master MCU SBL is required to send the function ID as the first byte so that the slave MCU SBL can enter IAP mode.

Block num (MSB & LSB)

- Represents the block number that the master MCU SBL is sending.
E.g. For block number 1: Block num (MSB) – 0x00, Block num (LSB) – 0x01
For block number 257: Block num (MSB) – 0x01, Block num (LBD) – 0x01

1024 bytes of data

- Represents the 1 kB block of data that is used for programming the flash on the slave MCU SBL.

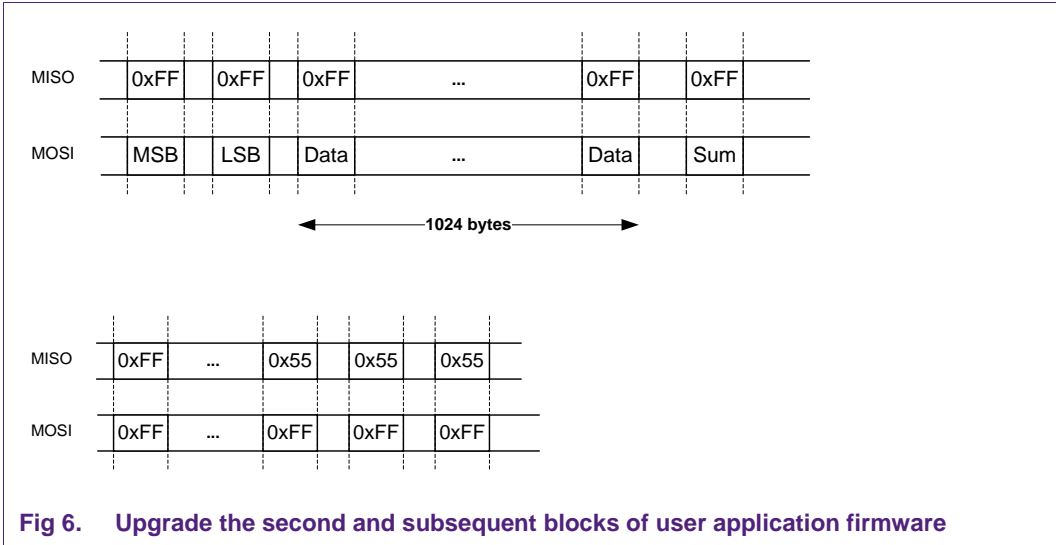
Checksum byte

- Checksum is used to check the integrity of the SBL transfer.
- Checksum is initialized as 0.
- Computation of Checksum:
$$\text{Checksum} = 0 - \text{Block num (MSB)} - \text{Block num (LSB)} - \text{Data}_0 - \dots - \text{Data}_{1023}$$

When the master MCU SBL completes the transmission of the block of data, the master MCU SBL will read continuously the status of the Secondary Boot Loader. When 3 bytes of 0x55 are received, the master MCU SBL understands that the slave MCU SBL has received data successfully. If not, the master MCU SBL will resend the block.

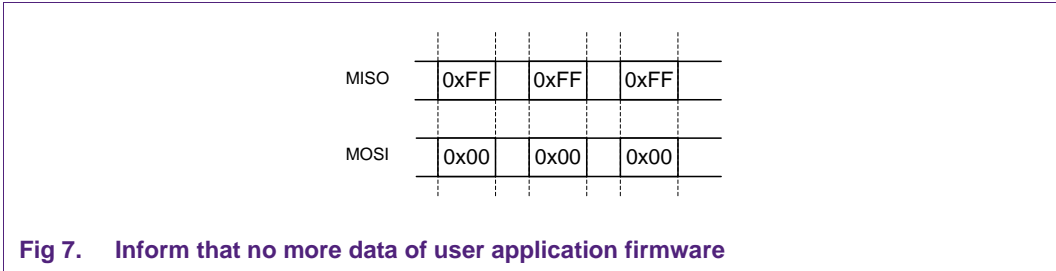
SBL Status

- If the three SBL status bytes received are '0x55', this means that programming is successful for the 1 kB of data.
 - If the three SBL status bytes received are '0xFF', this means that there are errors during programming of the 1 kB of data.
2. The master MCU SBL follows this communication protocol to send the second and subsequent blocks of data.



Fn ID

- The function ID is not required here, since the slave MCU SBL is now in IAP mode.
 - The absence of function ID is the only difference in the SBL communication protocol between the transmission of the first and subsequent blocks of data.
3. The master MCU SBL follows this communication protocol to inform the slave MCU SBL that there are no more data to be sent from the master MCU SBL.



Block num (MSB) and (LSB)

- Block num (MSB): 0x00
- Block num (LSB): 0x00

Checksum byte

- Generated using Command ID, Block num (MSB) and (LSB)
- Computation: Checksum = 0 - Block num (MSB) - Block num (LSB)

3.1.3 SBL firmware upgrade

Function ID: 0x33

Description:

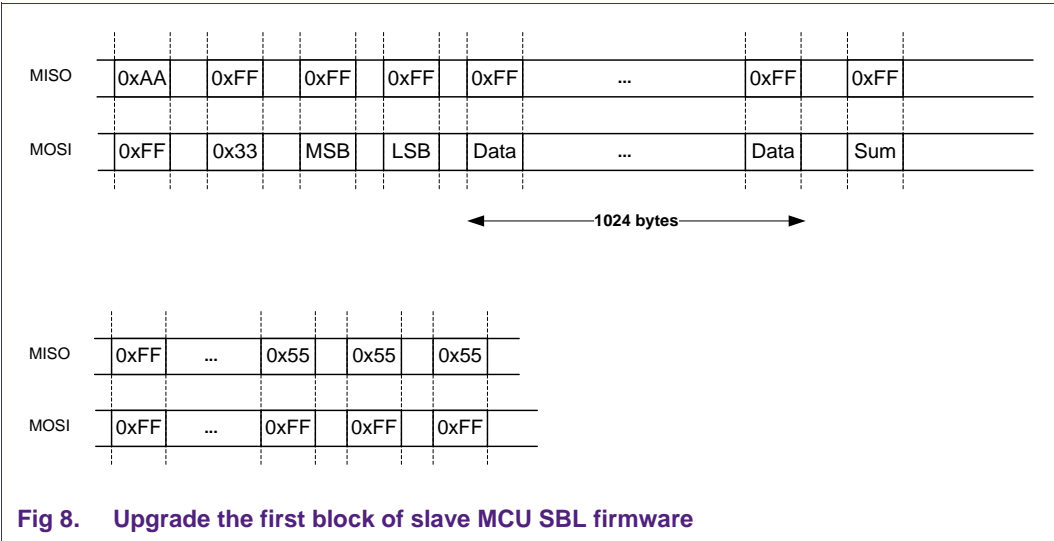
- This function performs programming of Sector 0 by the application program in a running system.

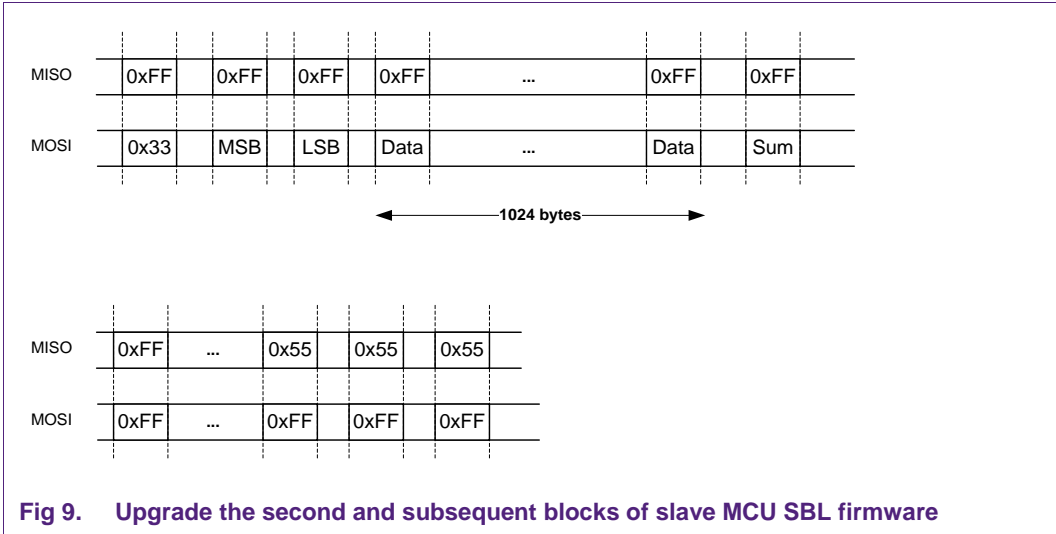
Implementation:

- The master MCU SBL waits for a READY byte (0xAA) over SPI.
- The master MCU SBL makes a request to program sector 0 of the flash memory by issuing a function ID (0x33) to the slave MCU SBL.
- Next, the master MCU SBL transfers firmware data to the slave MCU SBL one block at a time via SPI. Each block of data is 1 kB in size.
- The slave MCU SBL will always prepare and erase the contents of the flash sector at the beginning of every new sector.
- The slave MCU SBL programs the flash one block at a time (each sector of the flash memory contains four blocks of 1 kB of memory).
- After the master MCU SBL sends the last block of flash data, it would send Block Num (MSB) and Block Num (LSB) bytes having a value of 0x00 to inform the MCU that there are no more data.
- When the IAP process is completed, the slave MCU SBL will reset the system.

SBL communication protocol:

The SPI communication protocol for programming the SBL region on Flash Memory is similar to the SPI communication protocol for programming the User Application region.





3.2 Fail-safe implementation

The fail-safe mechanism is to ensure that in the event of power loss during In-Application Programming, the MCU is able to re-start and complete the IAP process when the power comes back on.

Implementation:

There are three important implementations in achieving the fail-safe mechanism for IAP.

1. Have a main function that decides if the flow of execution should go to the User Application or to the In-Application Programming Process.
2. Create and embed a Firmware ID at an address located at the end of the flash memory.
 - The Firmware ID is used as a determinant to identify if the User Application Code is intact. If the check is successful, it will enter the User Application. Otherwise, it enters IAP mode to program the User Application Code.
 - The Firmware ID is located at the end of the flash memory to ensure that this is the last data to be programmed during IAP process. In the event when the Firmware ID is invalid (0xFF), this means that the User Application Code has defects. One reason for this is that the previous IAP process is not successful.
 - In the main function, the MCU will always try to enter IAP mode if the Firmware ID is not valid.
3. The first task into the IAP mode is to erase the Firmware ID (the last sector to commence the IAP process).

4. Implementation

4.1 Top-level flow diagram

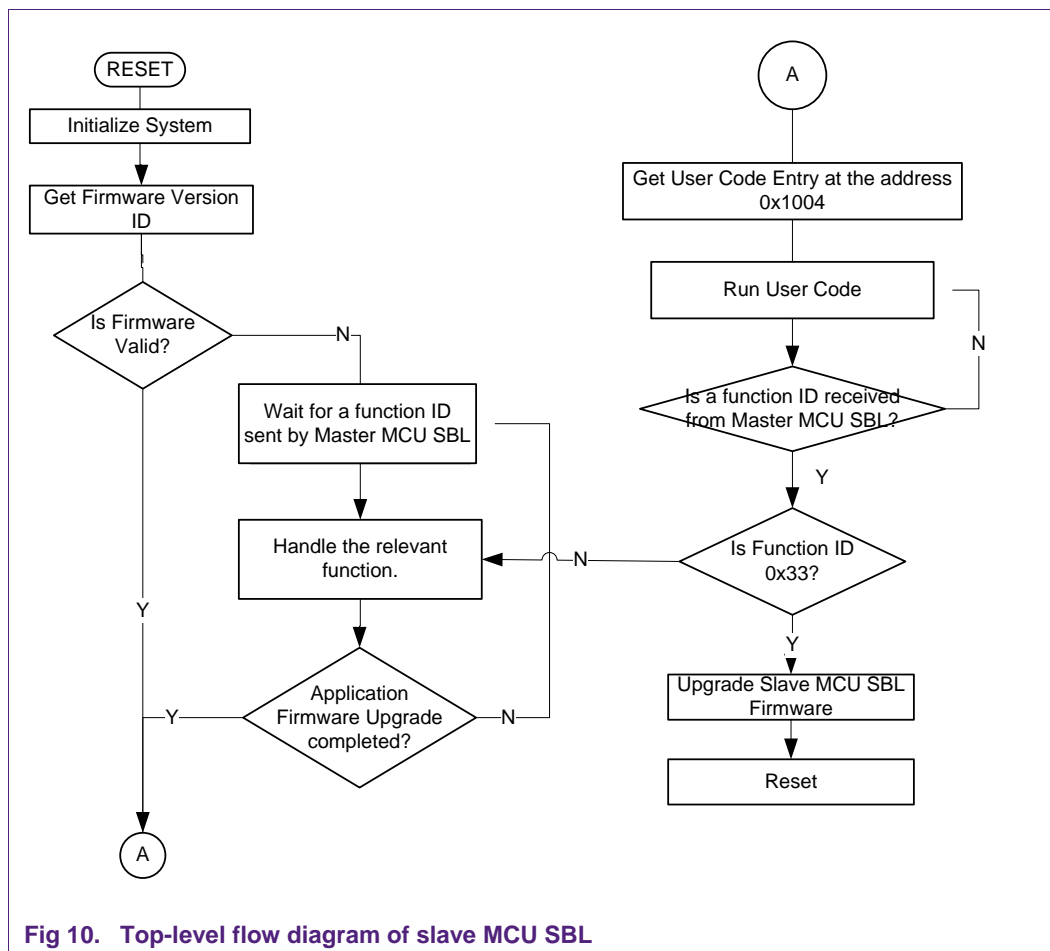


Fig 10. Top-level flow diagram of slave MCU SBL

4.2 SBL mode

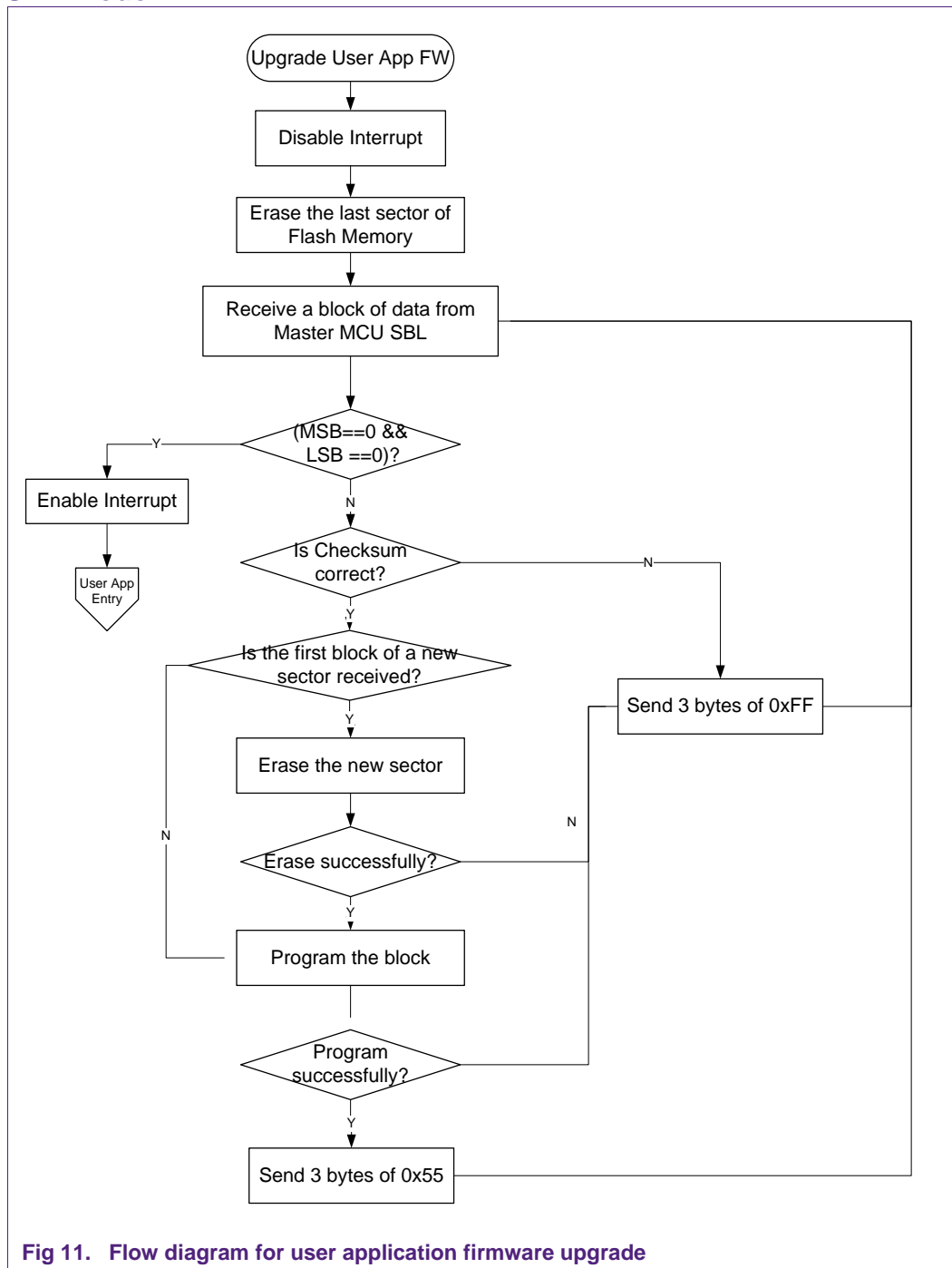
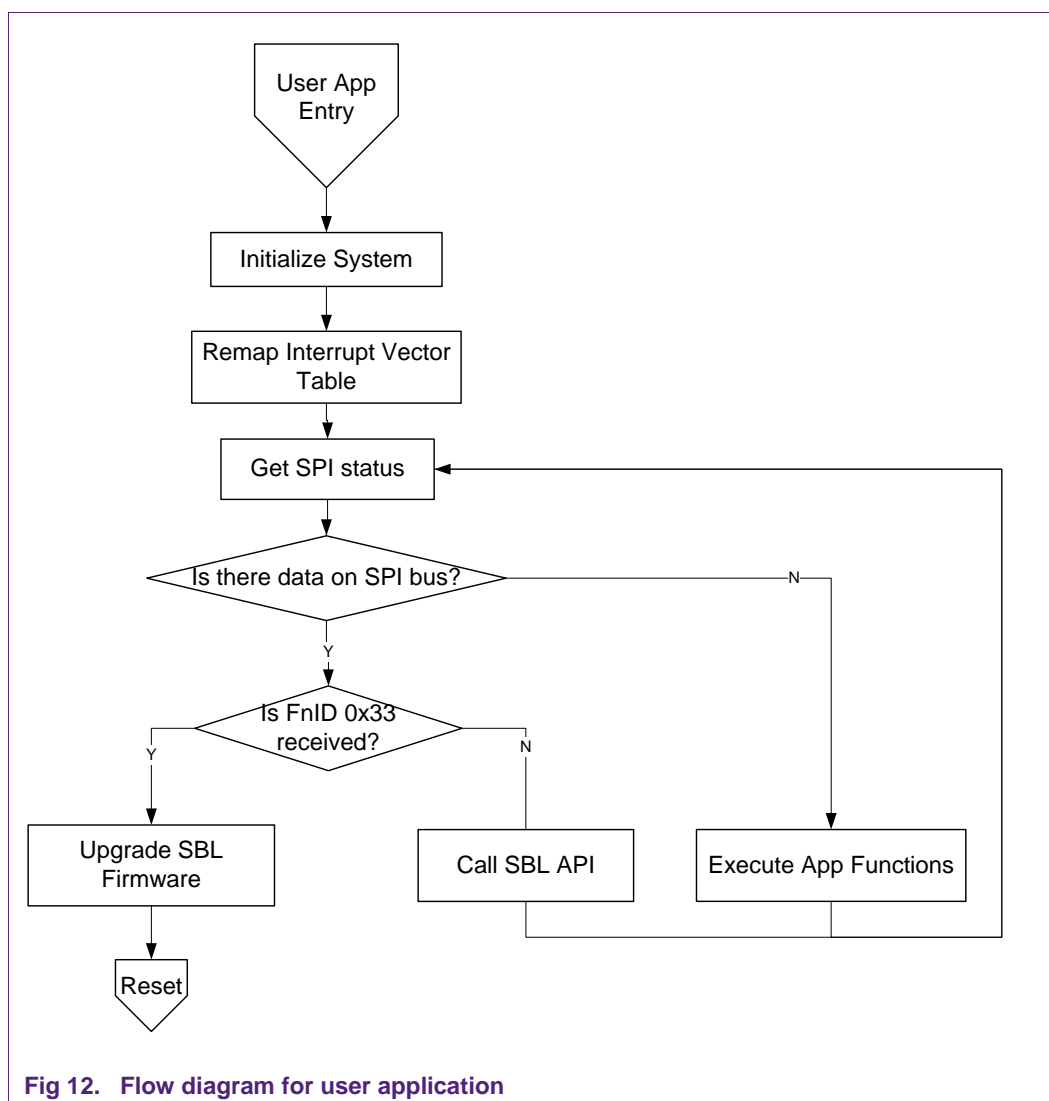


Fig 11. Flow diagram for user application firmware upgrade

4.3 Application mode



The SBL function could be called in the following way using C.

```
#define SBL_LOCATION 0xFF1
typedef void (*sbl_api)(uint32_t API_Code, uint8_t* pParam);
```

Create function pointer and call it:

```
sbl_api entry;
entry = (sbl_api) (SBL_LOCATION);
(entry)(API_Code, Params);
```

The parameters will be passed as shown in [Table 1](#):

Table 1. SBL APIs

API_Code	Params	Description
0x01	None	Enter SBL Mode.
0x02	None	Exit SBL Mode.
0x03	Function ID (Size = 1 byte)	Handle command received over SPI bus.

4.4 Scatter loading description file

To support In-Application Programming, a scatter loading description file is used to specify the memory map of the image to the linker, in order to control the grouping and placement of the image components.

This is important, because the new image is placed at its intended location and not any other location that can corrupt the codes used to perform IAP.

4.4.1 Scatter loading description file for SBL

The SBL resides on the first sector of the slave MCU. It is divided into two regions: one for storing the firmware itself, the other is used to public the SBL API for User Application.

Table 2. Scatter loading description file for SBL (LPC17xx)

Load Region	Execution Region	Input Sections
0x00000000 - 0x00000FEF	0x00000000 – 0x00000FEF	Interrupt Vector Table, Code and RO data.
	0x10000000 – 0x10007BFF	RW data, ZI data.
	0x2007C000 – 0x20083FFF	RW data, ZI data.
0x00000FF0 - 0x00000FFF	0x00000FF0 - 0x00000FFF	SBL API.

Table 3. Scatter loading description file for SBL (LPC11xx)

Load Region	Execution Region	Input Sections
0x00000000 - 0x00000FEF	0x00000000 - 0x00000FEF	Interrupt Vector Table, Code and RO data.
	0x10000000 – 0x10001BFF	RW data, ZI data.
0x00000FF0 - 0x00000FFF	0x00000FF0 - 0x00000FFF	SBL API.

The last 1 kB of internal RAM, starting from 0x10007C00 for the LPC17xx and from 0x10001C00 for the LPC11xx, is used to store 1 kB of firmware data sent by the master MCU while upgrading firmware for the User Application.

4.4.2 Scatter loading description file for user application

The User Application resides in flash memory, from sector 1 to the last sector of the slave MCU. It is also divided into 2 regions; 1 for storing firmware data itself and the other for storing Firmware Version ID.

Table 4. Scatter loading description file for user application (LPC17xx)

Load Region	Execution Region	Input Sections
0x00001000 - 0x0007FFFB	0x00001000 – 0x0007FFFB	Interrupt Vector Table, Code and RO data.
	0x10000000 – 0x10007BFF	RW data, ZI data.
	0x2007C000 – 0x20083FFF	RW data, ZI data.
0x0007FFFC- 0x0007FFFF	0x00000FF0 - 0x00000FFF	Firmware Version ID

Table 5. Scatter loading description file for user application (LPC11xx)

Load Region	Execution Region	Input Sections
0x00001000 - 0x00007FFB	0x00000000 – 0x00007FFB	Interrupt Vector Table, Code and RO data.
	0x10000000 – 0x10001BFF	RW data, ZI data.
0x00007FFC- 0x00007FFF	0x00007FFC- 0x00007FFF	Firmware Version ID

The last 1 kB of internal RAM is also used to store 1 kB firmware data sent by master MCU while upgrading firmware for SBL.

5. How to run

This project is tested on Keil's MCB1700 with LPC1768 version 1 and Keil's MCB1000 with LPC1114. It requires two boards: one master and one slave.

5.1 Directory contents

The folder structure of the project is described in [Fig 13](#).

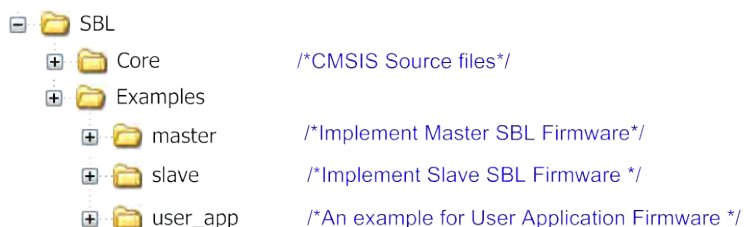


Fig 13. Project directory contents

In each example folder, there are four sub-folders:

- Folder “Keil”: includes Keil project file, scatter file and debug file. After compiling, output files are placed in a sub-folder of this folder. The name of the sub-folder is the same as the name of the selected target.
- Folder “drivers”: includes source code for controlling peripherals including SPI, UART, and GPIO.
- Folder “sbl”: includes SBL source code (Master/Slave role).
- Folder “main”: includes main program.

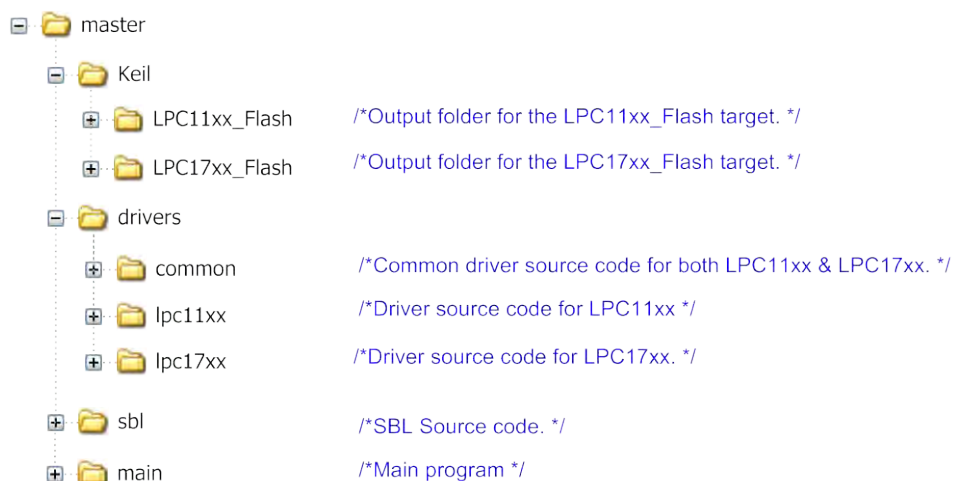


Fig 14. Example directory contents

5.2 Hardware configuration

5.2.1 MCB1700

These jumpers must be configured as:

- VDDIO: ON
- VDDREGS: ON
- Remaining jumpers: OFF

5.2.2 MCB1000

These jumpers must be configured as:

- J2 – VDD: ON

5.2.3 SPI connection

SPI pins:

Function	MCB1700	MCB1000
SCK	P0.15	P0.6
SSEL	P0.16	P0.2
MISO	P0.17	P0.8
MOSI	P0.18	P0.9

SPI connection:

- SCK on master connects SCK on slave board
- SSEL on master connects to SSEL on slave board
- MISO on master connects to MISO on slave board
- MOSI on master connects to MOSI on slave board

Common ground must be connected together between two boards.

5.3 Software configuration

5.3.1 Tera Term setting

Serial display configuration

- 115200 bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control

5.4 Steps to run

Step 1: Open master MCU SBL Project in **SBL\Examples\master** folder.

- Select the target which relevant to the master board.
- Determine the size of User application firmware using SBL_FW_DATA_BLOCK_NUM macro in file “sbl.h”.
- Build master MCU SBL Project.
- Burn hex file into master board. (If run on ROM mode)

Step 2: Open slave MCU SBL Project in **SBL\Examples\slave** folder and User Application project in **SBL\Examples\user_app** folder.

- Select the target relevant to the slave board.
- Build the projects.
- Erase all sectors on flash memory of slave board.
- Burn the hex file of slave MCU SBL Project into slave board.
- Burn the hex file of User Application project to master board.

Step 3: Connect UART0 on master board to COM port on your computer.

Step 4: Configure hardware, connect master board and slave board as above instruction.

Step 5: Configure serial display as above instruction.

Step 6: Reset Slave board. The LED at P2.2 is on to notify that the system is in SBL mode.

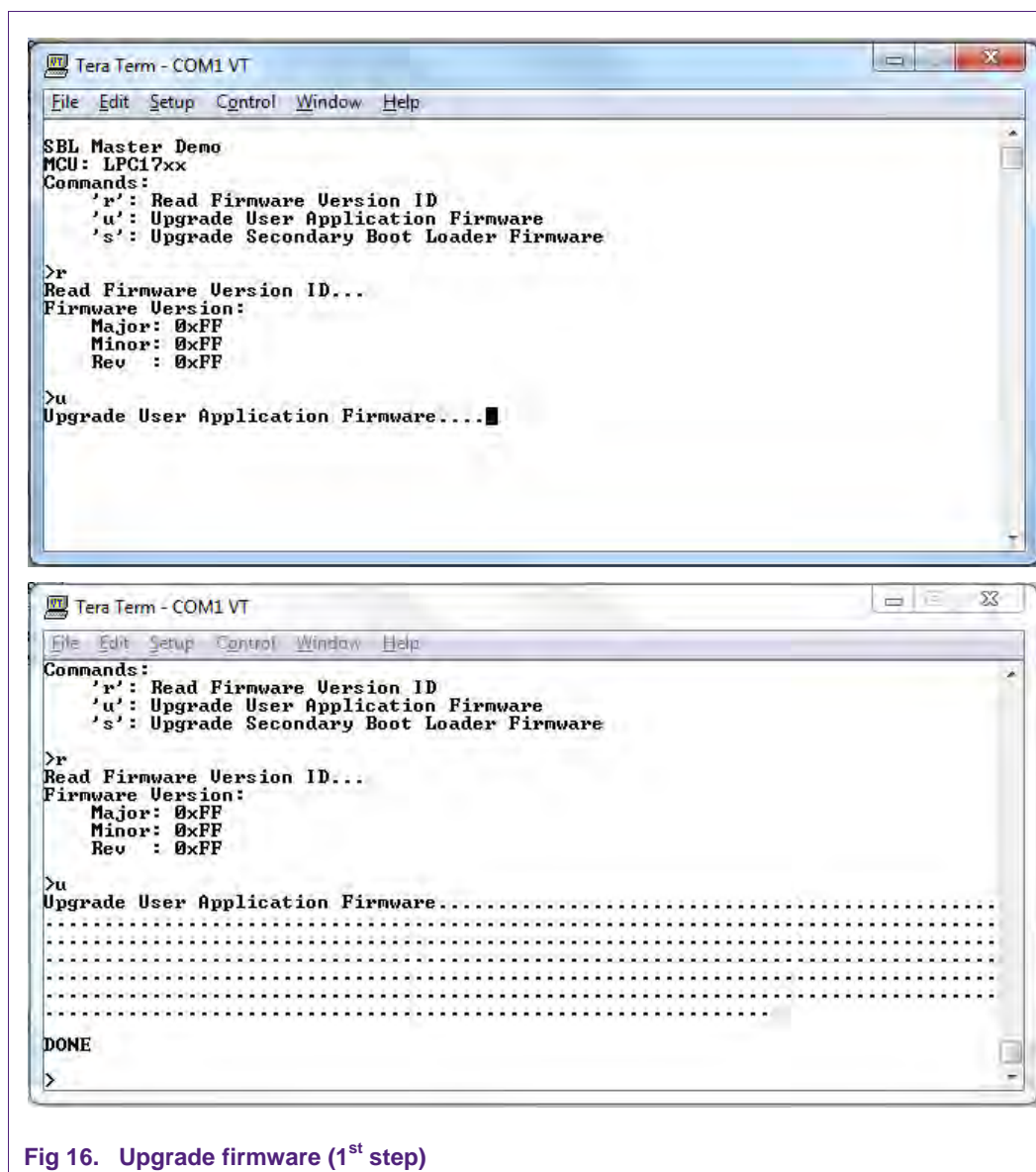
Step 7: Reset the Master board. The welcome message is shown.

A screenshot of a Tera Term terminal window titled "Tera Term - COM1 VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The terminal output displays the following text:

```
SBL Master Demo
MCU: LPC17xx
Commands:
  'r': Read Firmware Version ID
  'u': Upgrade User Application Firmware
  's': Upgrade Secondary Boot Loader Firmware
>
```

Fig 15. Terminal Output when Master board resets.

- Press 'r' to read the firmware version ID from slave. If this is the first time upgrading firmware, 0xFF will be displayed for all version details.
- Press 'u' to start upgrading firmware for slave. The master gets firmware data from sector 1 of its flash memory and is then sent to the slave. The size of firmware data is determined by the SBL_FW_DATA_BLOCK_NUM macro.



- After upgrading ends, the user code is executed. LED at P2.2 is OFF. LEDs at P2.3~P2.6 blink.
- Press 'r' to display version of the new firmware.

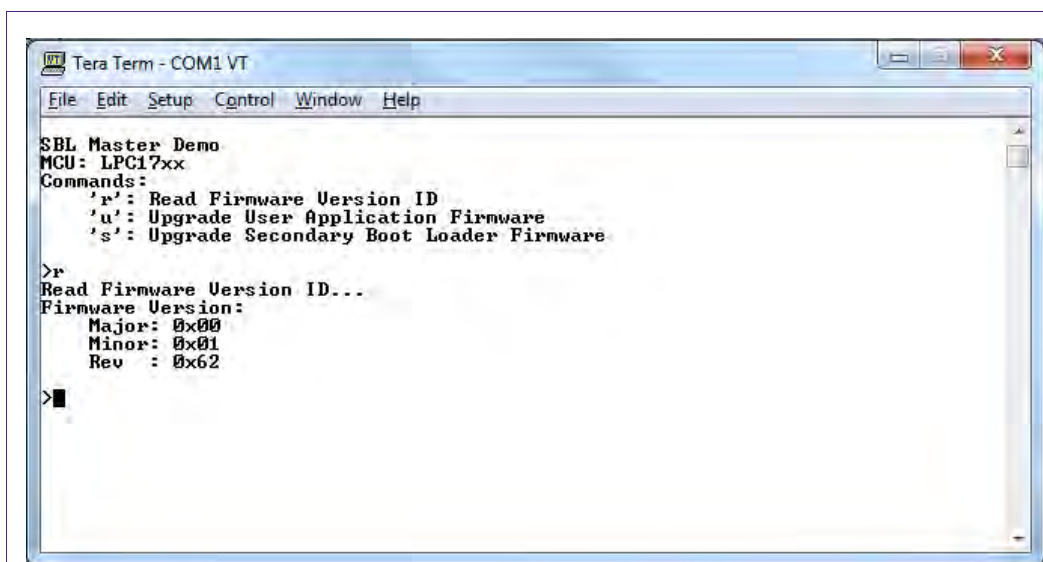


Fig 17. Upgrade firmware (5th step)

- To upgrade SBL firmware:
 - First, download the binary firmware file of the slave MCU SBL project (stored in the path **SBL\Examples\slave\Keil<Target Name>**) to sector 1 of flash memory in master MCU:
 - Open slave MCU SBL Project.
 - Open Target Option Dialog, select Utilities Tab.
 - Choose External Tool for Flash Programming as below.

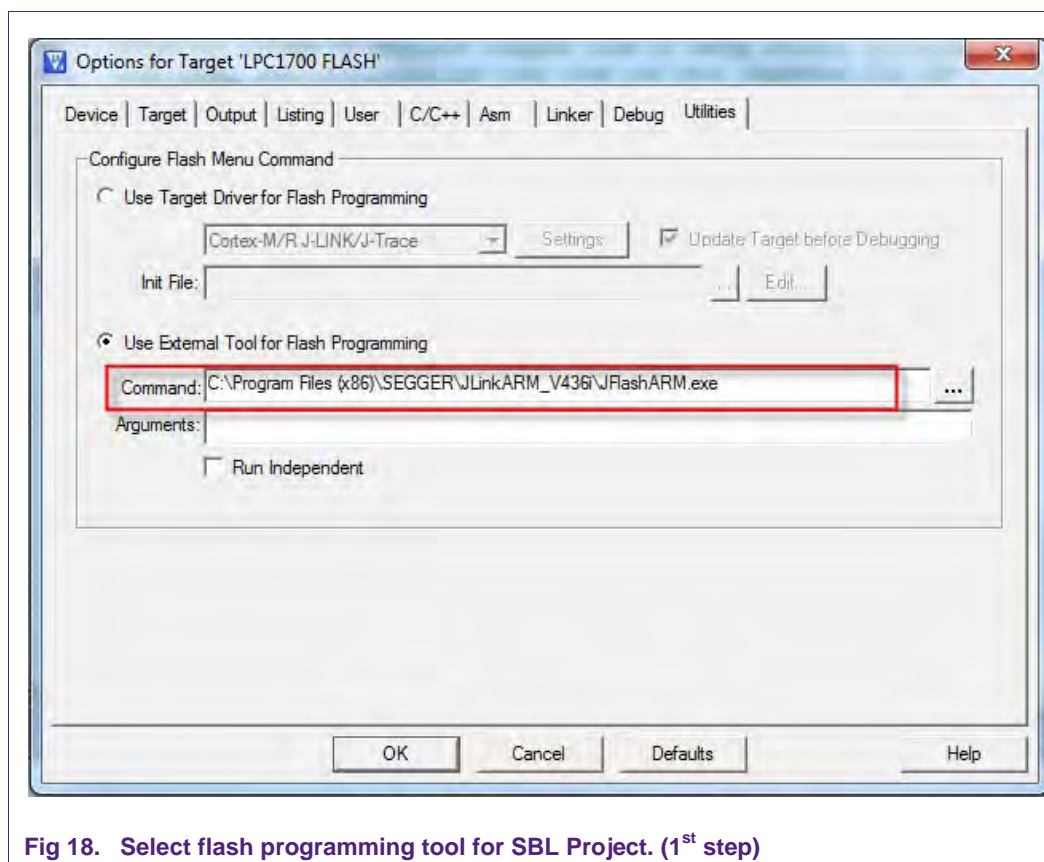


Fig 18. Select flash programming tool for SBL Project. (1st step)

- Select Flash→ Download. The J-Flash ARM Application is opened.
- Select File→ Open Project; browse to file “LPC1768.jflash” if the master is LPC17xx MCU. If it is LPC11xx MCU, please select “LPC1114.jflash”.

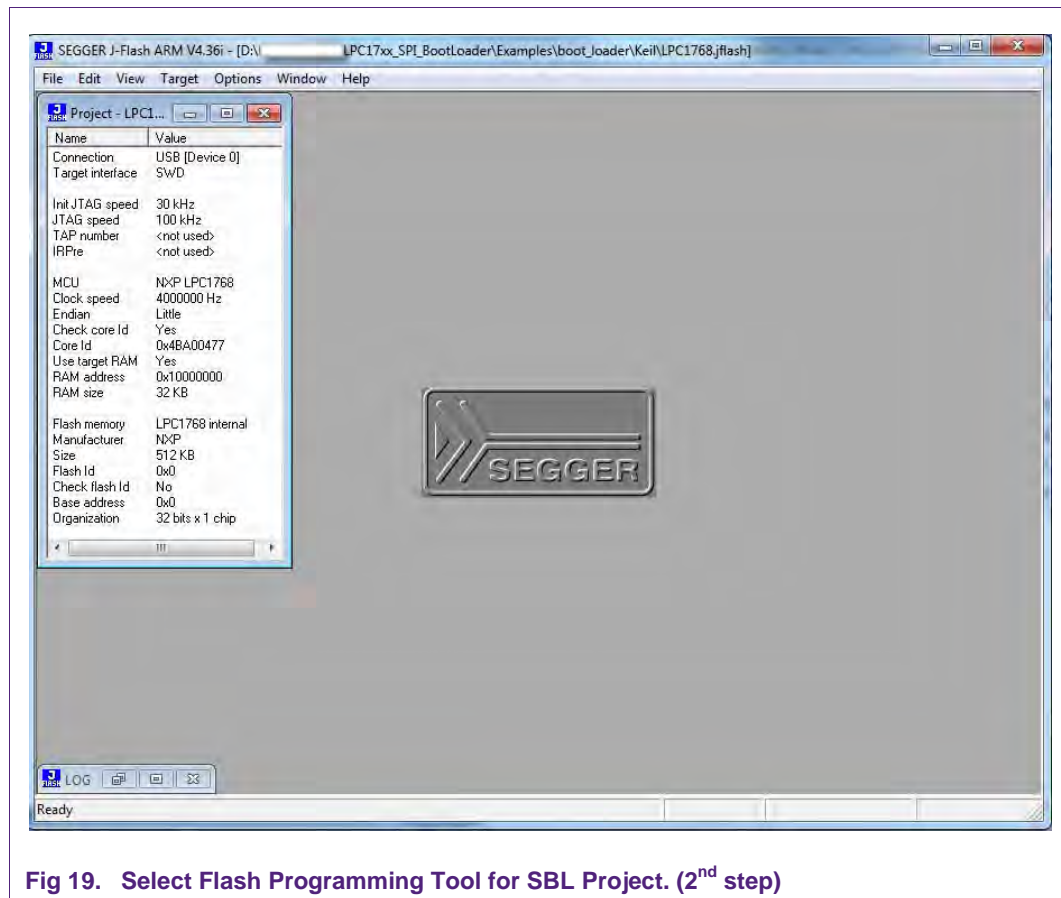


Fig 19. Select Flash Programming Tool for SBL Project. (2nd step)

- Select File→Open data file, browse to **sbl_slave.bin** file in the relevant output folder of SBL Project. Set the Start Address as 0x1000.

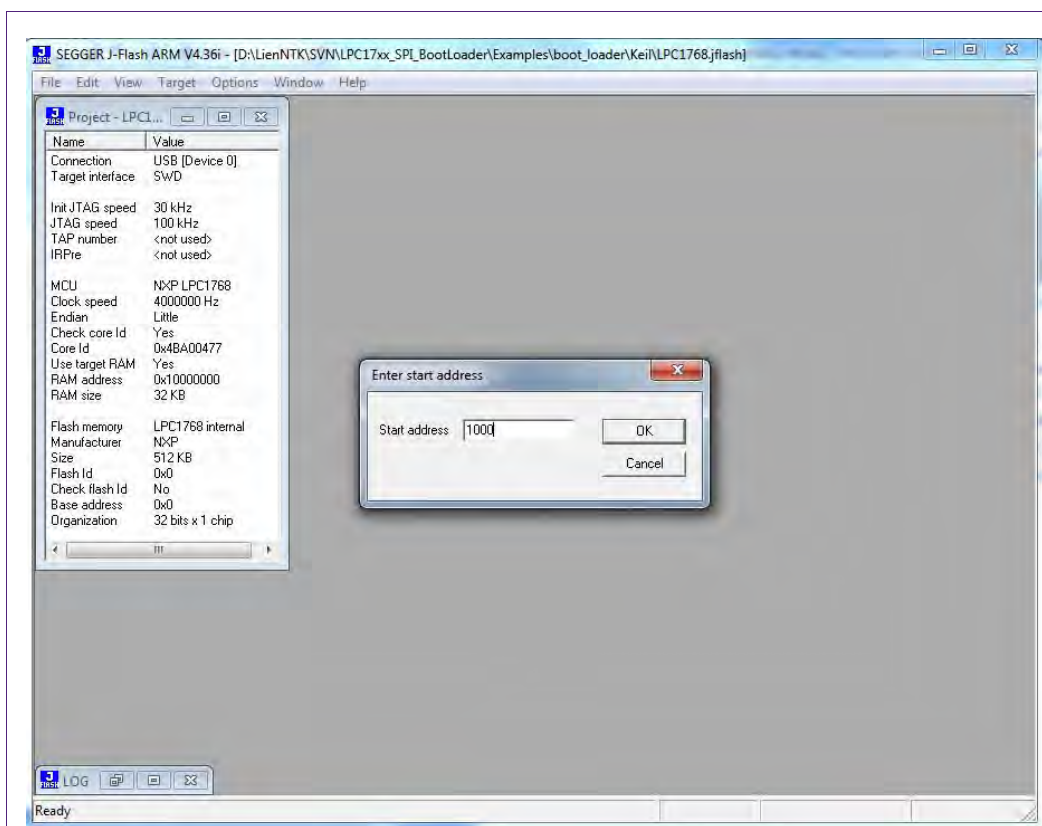


Fig 20. Select Flash Programming Tool for SBL Project. (3rd step)

- Select Target→Program to program the master board.
- Reset the master board.
- Press 's' to upgrade firmware for SBL.

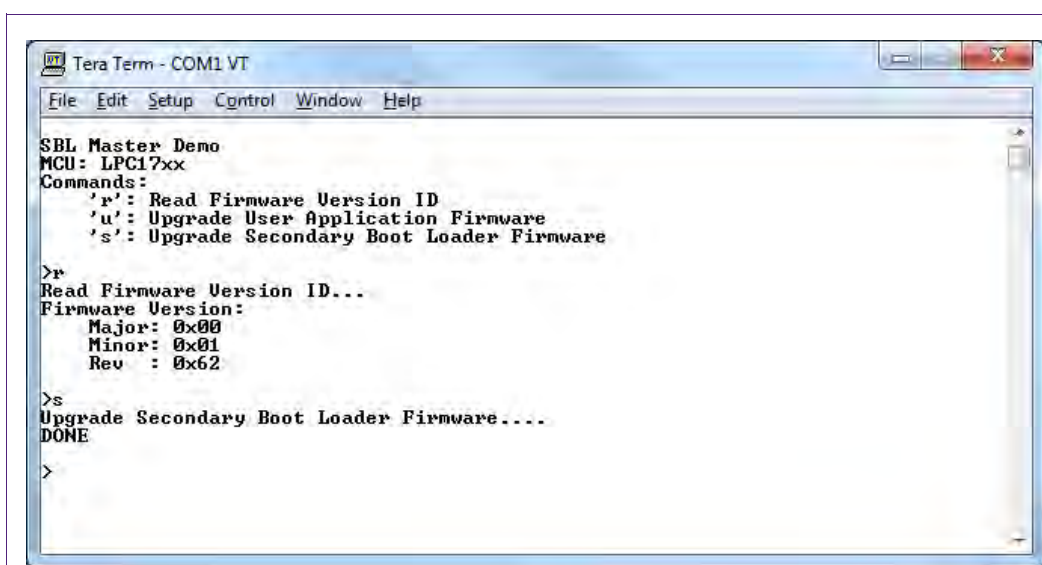


Fig 21. Upgrade SBL firmware

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP

Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out of the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US\$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. List of figures

Fig 1.	System components	4
Fig 2.	Memory map for master MCU	5
Fig 3.	Memory Map for slave MCU	6
Fig 4.	Read Firmware ID	7
Fig 5.	Upgrade the first block of user application firmware	9
Fig 6.	Upgrade the second and subsequent blocks of user application firmware	10
Fig 7.	Inform that no more data of user application firmware	10
Fig 8.	Upgrade the first block of slave MCU SBL firmware	11
Fig 9.	Upgrade the second and subsequent blocks of slave MCU SBL firmware	12
Fig 10.	Top-level flow diagram of slave MCU SBL	13
Fig 11.	Flow diagram for user application firmware upgrade	14
Fig 12.	Flow diagram for user application	15
Fig 13.	Project directory contents	18
Fig 14.	Example directory contents	18
Fig 15.	Terminal Output when Master board resets	20
Fig 16.	Upgrade firmware (1 st step)	21
Fig 17.	Upgrade firmware (5 th step)	22
Fig 18.	Select flash programming tool for SBL Project. (1 st step)	23
Fig 19.	Select Flash Programming Tool for SBL Project. (2 nd step)	24
Fig 20.	Select Flash Programming Tool for SBL Project. (3 rd step)	25
Fig 21.	Upgrade SBL firmware	25

8. List of tables

Table 1. SBL APIs..... 16

Table 2. Scatter loading description file for SBL
(LPC17xx) 16

Table 3. Scatter loading description file for SBL
(LPC11xx) 16

Table 4. Scatter loading description file for user
application (LPC17xx) 17

Table 5. Scatter loading description file for user
application (LPC11xx) 17

9. Contents

1.	Introduction	3
1.1	Project overview	3
1.2	System components	3
1.3	SPI communication	4
2.	System overview	5
2.1	Project configuration	5
2.2	Memory map	5
2.2.1	Memory map for master MCU SBL	5
2.2.2	Memory map for slave MCU SBL	6
3.	Functional description	6
3.1	SBL mode	6
3.1.1	Read firmware version ID	6
3.1.2	User application firmware upgrading	8
3.1.3	SBL firmware upgrading	11
3.2	Fail-safe implementation	12
4.	Implementation	13
4.1	Top-level flow diagram	13
4.2	SBL mode	14
4.3	Application mode	15
4.4	Scatter loading description file	16
4.4.1	Scatter loading description file for SBL	16
4.4.2	Scatter loading description file for user application	17
5.	How to run	18
5.1	Directory contents	18
5.2	Hardware configuration	19
5.2.1	MCB1700	19
5.2.2	MCB1000	19
5.2.3	SPI connection	19
5.3	Software configuration	19
5.3.1	Tera Term setting	19
5.4	Steps to run	19
6.	Legal information	26
6.1	Definitions	26
6.2	Disclaimers	26
6.3	Trademarks	26
7.	List of figures	27
8.	List of tables	28
9.	Contents	29

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.
