AN12393

LPC845 I2C Secondary Bootloader

Rev. 0 — April 2019 Application Note

Contents

1 Introduction

The LPC845 is an Arm® Cortex®-M0+ based, low-cost 32-bit MCU family operating at CPU frequencies of up to 30 MHz. The LPC845 supports up to 64 KB of flash memory and 16 KB of SRAM.

The LPC845 supports Arm Serial Wire Debug (SWD) mode for Cortex-M0+ core. Device programming can be achieved through the SWD port. In addition, the LPC845 contains an on-chip boot ROM that supports In-System Programming (ISP) when the part resides in the end-user board and In-Application Programming (IAP) as directed by the end-user application code.

A Secondary BootLoader (SBL) is a piece of code that allows a user application code to be downloaded using alternative channels other than the standard UART0 used by the internal bootloader. The following steps describe how SBL boots a program to a flash location.

1. The primary bootloader is the firmware that resides in the microcontroller's boot ROM block and is executed on power-up and resets.
2. After the boot ROM's execution, the SBL is executed. It utilizes the boot ROM's IAP functionalities and allows programming the LPC845 flash through I^2C slave interface which can be used between the host processor and LPC845.
3. Then SBL executes the end-user application.

Figure 1. on page 1 shows an example of a system setup where the host processor can program the LPC845 via I^2C interface assisted by the SBL code.

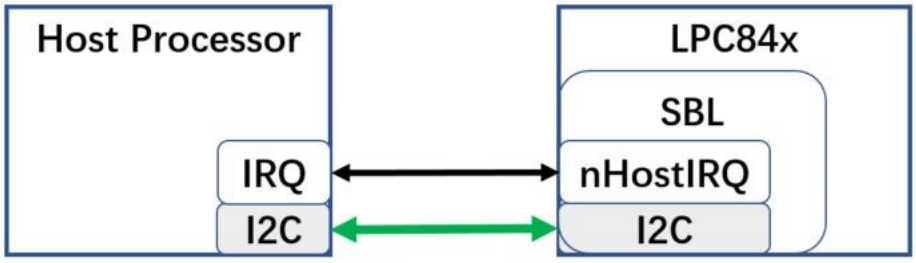

Figure 1. SBL for LPC845 application

2 SBL functionalities and boot process with SBL

2.1 Memory map with applications boot with SBL

The flash size of LPC845 is 64 KB, and is divided into 32 sectors. The corresponding address space is `0x0000 0000` - `0x0001 0000`. Some IAP and ISP commands operate on sectors and specify sector numbers. In addition, a page erase command is available. The size of a sector is 1 KB and the size of a page is 64 byte. One sector contains 16 pages.

The LPC845 I²C SBL is downloaded to the first eight sectors in flash, so the user application starts from `0x0000 2000` in flash. Therefore, applications that boot from the SBL must be initially set up with the vector table at the address of `0x0000 2000`.

An application that boots from the SBL must meet the following requirements:

- Uses no flash memory in the region of `0x0000 0000` - `0x0000 1FFF`.
- Requires an image header at the address of `0x0000 2100` in flash.

Figure 2. on page 2 shows the flash memory allocation of the SBL and the user application.

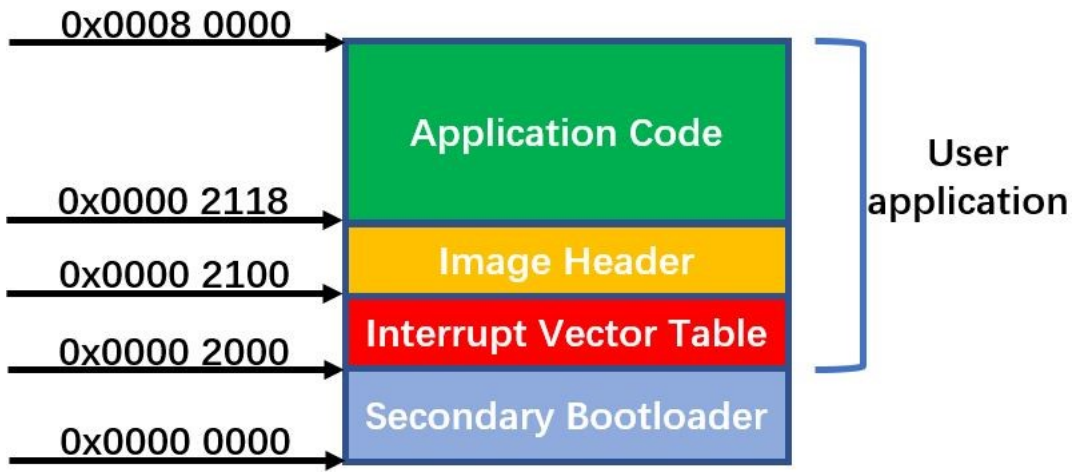

Figure 2. Flash memory map for application boot with SBL

2.2 Boot process with SBL

Figure 3. on page 3 shows the boot sequence that all LPC845 parts with an SBL flashed.

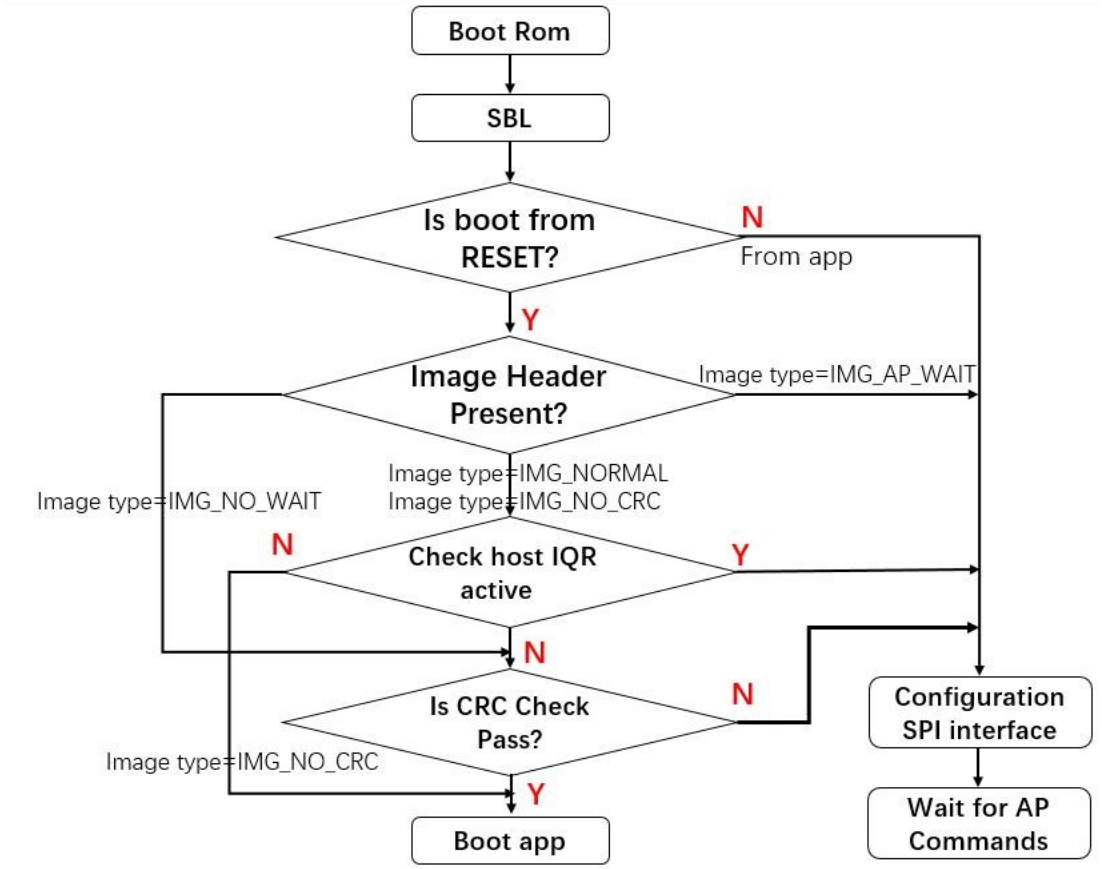

Figure 3. Boot flow of LPC845 with SBL

- After the reset (power-on reset, watchdog reset, external reset, BOD reset, or software system reset), the Boot ROM will run and pass the control to the SBL.
- To allow proper handshaking between the SBL and the application, an image header is required in the application image at the offset of `0x100` (`0x00002100` absolute flash address). Before booting the application, the SBL checks for the presence of the image header.
- If the image header is absent, the SBL configures the I^2C interface and then enters the state of waiting for the AP command.
- If the image header is present, the SBL checks the image type.
- Depending on the image type, the SBL either checks the image integrity and boots the image automatically or enters an AP command processing loop (where the AP controls to boot the application).

2.3 SBL flash IAP programming support

FOr detailed command description, see *LPC5410x I2C SPI Secondary Bootloader* (AN11610). For detailed information about IAP command and usage, refer to sections 5.6, 5.7, and 5,8 in *LPC84x User manual* (UM11029). When working with the SBL, it is not necessary for the user to check the detailed implementation of these commands. However, you can review Chapter 5 in *LPC84x User manual* (UM11029) for a background of the SBL implementation.

2.4 Emulated host processor/slave processor communication

2.4.1 Hardware and software environment

The sample test application can be tested using Keil MDK IDE v.5.25 along with LPCXpresso 845 MAX board (#OM13097) and LPCXpresso 54102 board (#OM13077) used as USB-to-I2C tool. The `I2C-Util` tool uses I^2C protocol in OM13077 board to send firmware updates to LPCXpresso 845 MAX board.

Figure 4. on page 4 to Figure 6. on page 4 show the hardware platform and the connection between LPCXpresso 845 MAX board and LPCXpresso 54102 board.

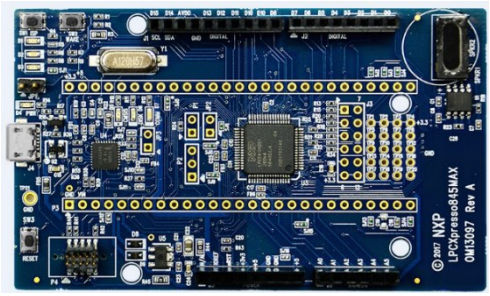

Figure 4. LPCpresso 845 MAX board

Figure 5. LPCpresso 54102 board as USB-to-I2C tool

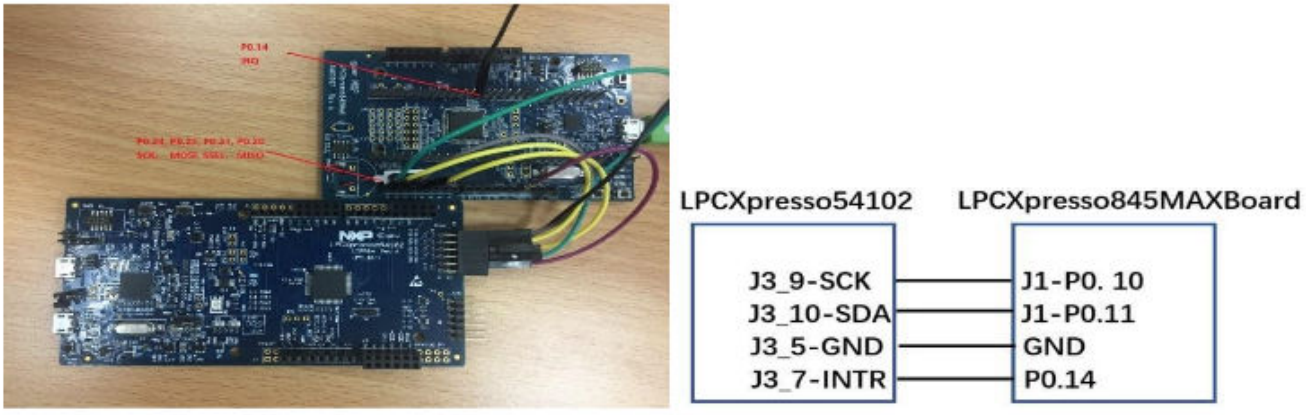

Figure 6. Connection between LPCpresso 54102 board and LPCpresso 845 board

2.4.2 Downloading SBL to LPC845

SBL can be downloaded to LPC845 through many ways depending on the situation. In production, the SBL can be pre-programmed to the LPC845 via a debugger or UART ISP mode whichever is available. In prototyping, the part can most likely be programmed after fitted on the board assume the SWD or ISP UART port is accessible.

There are two ways are recommended to users to download SBL to flash:

- Use LPCXresso845 onboard debugger.
- Use the **Flash Magic** tool.

Users can use the **Flash Magic** tool to download the SBL to flash by performing the steps as follows if they don't have an onboard debugger.

1. Put the LPC845 into ISP mode by pressing down the ISP button (**SW1**) and then toggle the **Reset** button (**SW3**) (Low to High).
2. Choose the SBL hex file from the application note package and allow Flash Magic to successfully program the SBL.
3. After downloading the SBL hex file, press the **Reset** button on LPCXpresso845MAXBoard to boot ROM and SBL.

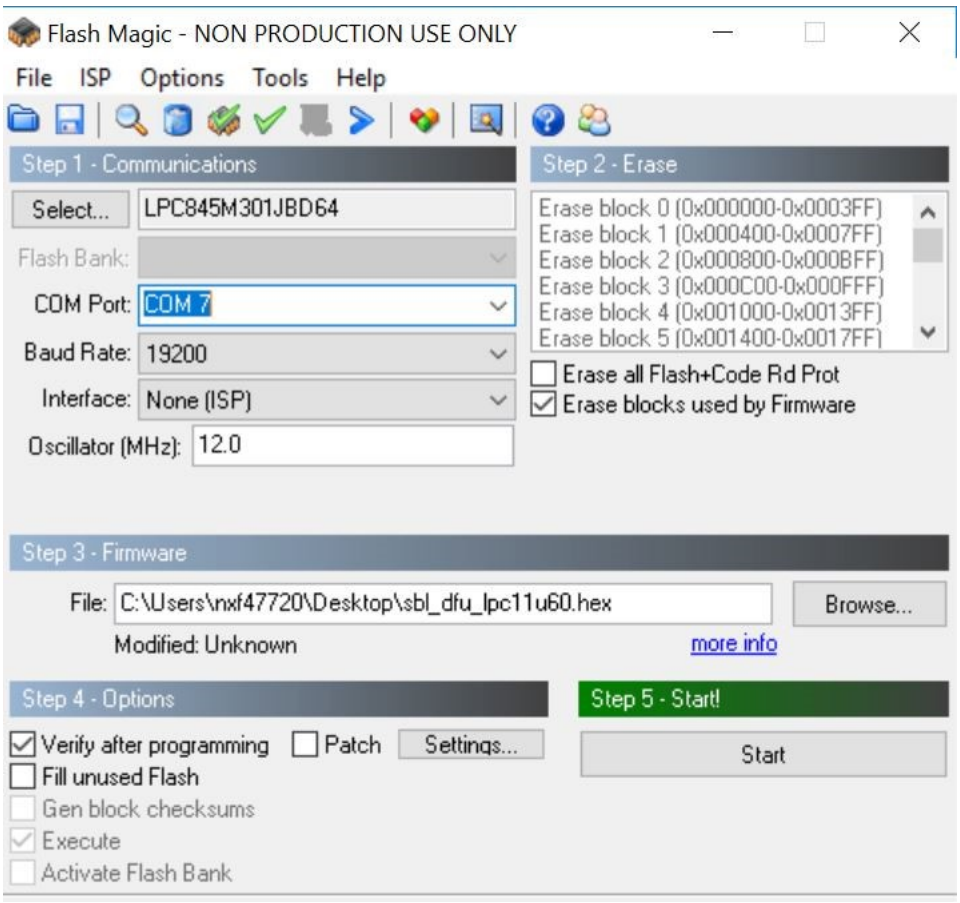

Figure 7. Download SBL to LPC845 with flash magic

2.4.3 System introduction

The windows PC application talks to the SBL through the USB to I^2C/SPI Bridge (implemented with LPC43xx) via NXP's USBSerialIO library (for more information about LPCUSBSIO library, go to http://www.lpcware.com/search/gss/libusbsio).

NOTE

The onboard debugger for the LPCXpresso54102 board is LPC4322, which has been downloaded with the CMSIS-DAP firmware already. The CMSIS-DAP firmware allows debugging from any compatible toolchain, including IAR EWARM, Keil MDK and NXP's MCUXpresso IDE.

Figure 8. on page 6 shows the the high-level block diagram of the system.

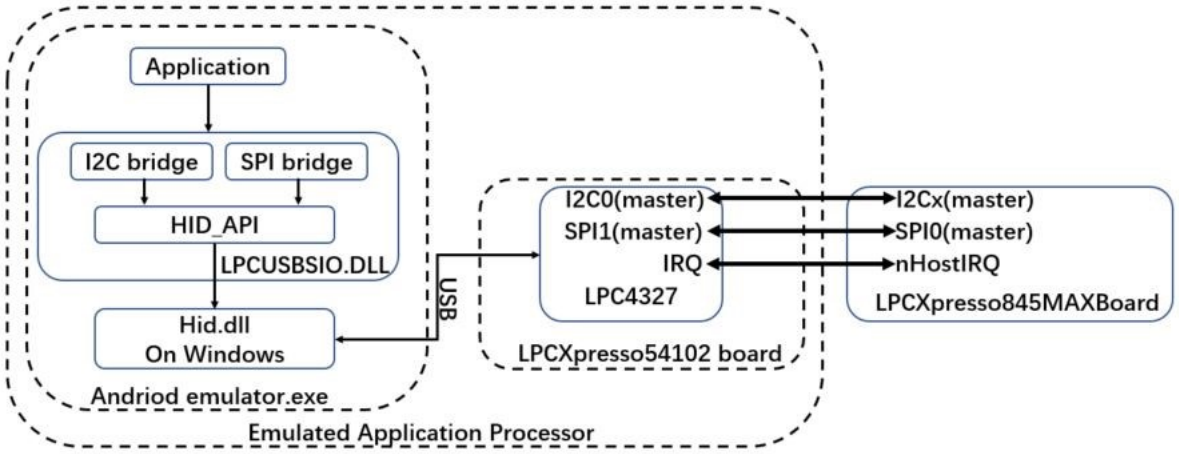

Figure 8. Emulated host processor communication with LPCpresso845MAXBoard

As seen from Figure 8. on page 6, running the Emulated Host Processor (`I2C-util`) from the PC allows the user to communicate with the LPC43xx and they work together as the host processor.

After successfully downloading the SBL by following instructions in Downloading SBL to LPC845 on page 5 and pressing the **RESET** button, you can open the **Command prompt** as administrator to run the `I2C-util.exe` to get the options to communicate with the LPC845 via I^2C or SPI. In this example, the I^2C interface is chosen to communicate with LPC845.

3 Contents of package

Figure 9. on page 6 shows the extracted contents of the package.

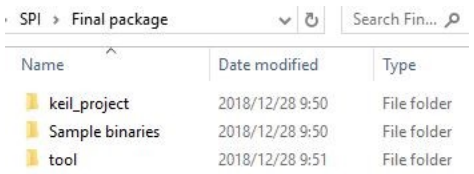

Figure 9. Package contents

A brief description for each of the folders is as follows.

- **tool** – This folder contains the `I2C-util.exe` and `lpc845_secimgcr.exe`.
  - `I2C-util.exe` – This tool is used to interface with the SBL through I^2C.
  - `lpc845_secimgcr.exe` – This tool is used to generate and insert a valid CRC.
- **Sample binaries** – This folder contains sample binary files that can be generated with the image creator tool.
  - `lpc845_I2C_sbl.bin` – Sample application binary that was used to create the sample firmware images with CRC in this folder
  - `lpc845_I2C_sbl_crc.bin` – Application binary with CRC generated and inserted.
- **Keil project** – This folder contains two Keil projects for the `lpc845_I2C_sbl` and test application.

4 Test application

The test application is an LED blinky example. It toggles the blue LED on the LPCXpresso845MAXBoard.

4.1 Building app binary file

When generating the binary file of the test application, users can use the `firmware1.sct` file as the linker file.

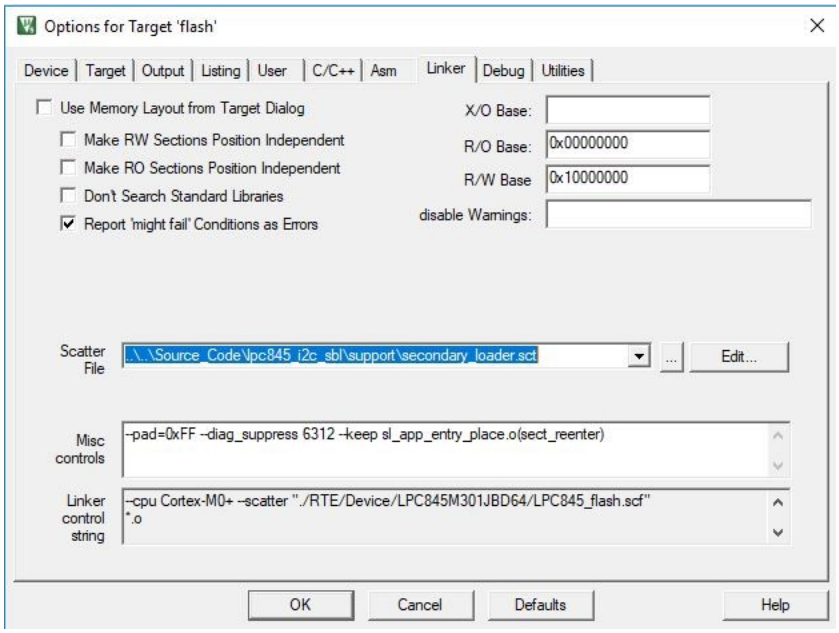

Figure 10. Choose firmware1.sct file as linker file for test application

The `firmware1.sct` file will lead test application to flash at `0x2000`. Figure 11. on page 7 shows the linker script.

```
LR_IROM1 0x00002000 0x00002000  {    ; load region size_region
  ER_IROM1 0x00002000 0x00002000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x10000000 0x00001000  {  ; RW data
   .ANY (+RW +ZI)
  }

}
```

Figure 11. Linker script of test application

4.2 Re-invoking I²C SBL from test application

The SBL supports re-invoke SBL from app. After calling the `bootSecondaryLoader(psetup)` fucntion in the app, the program can jump to SBL. Figure 12. on page 8 defines the `bootSecondaryLoader()` function.

```
typedef bool (*InBootSecondaryLoader)(const SL_PINSETUP_T *pSetup);

/* Address of indtrect boot table */
#define SL_INDIRECT_FUNC_TABLE      (0x00001F00)

/* Placement addresses for app call flag and app supplied config daa
   for host interface pins. Note these addresses may be used in the
   startup code source and may need values changed there also. */
#define SL_ADDRESS_APPCALLEDFL      (0x10000000)
#define SL_ADDRESS_APPPINDATA       (0x10000004)

/* Function for booting the secondary loader from an application. Returns with
   false if the pSetup strructure is not valid, or doesn't return if the
   loader was started successfully. */
static INLINE bool bootSecondaryLoader(const SL_PINSETUP_T *pSetup)
{
  InBootSecondaryLoader SL, *pSL = (InBootSecondaryLoader *) SL_INDIRECT_FUNC_TABLE;
  SL_PINSETUP_T *pAppPinSetup = (SL_PINSETUP_T *) SL_ADDRESS_APPPINDATA;

  *pAppPinSetup = *pSetup;

  SL = *pSL;
  return SL(pSetup);
}
```

Figure 12. BootSecondaryLoader() function definition

After executing the `bootSecondaryLoader()` function, the program jump to execution at `0x00001F00`.

The `indrectAppJump` pointer is defined in the SBL project as follows:

```
__attribute__ ((at(0x00001F00))) const uint32_t * indrectAppJump = (uint32_t *) &secondaryLoaderEntry;
```

The `IndrectAppJump` pointer is placed at `0x0001F00` and points to the `secondaryLoaderEntry()` function. The `secondaryLoaderEntry()` function calls the `secondaryLoaderAppEntry()` function.

4.3 Image creator tool

Before the app's binary file is downloaded to the target board, we need to use the lpc845_secimgcr.exe tool to add the CRC check code to the app's binary file. The SBL uses the CRC check code to check whether the app is valid. The specific steps are as follows:

1. Open `lpc845_secimgcr.exe`: Open the **cmd** command window as an administrator and switch to the path to `lpc845_secimgcr.exe` tool.
2. Enter the following in the command window:

```
C:\<path>\lpc845_secimgcr.exe <input filename.bin> <output filename.bin>
```

Figure 13. on page 9 shows the syntax to generate the CRC for the input application binary file of `lpc845_I2C_sbl_app.bin` and creates an output file of `lpc845_I2C_sbl_crc.bin`.

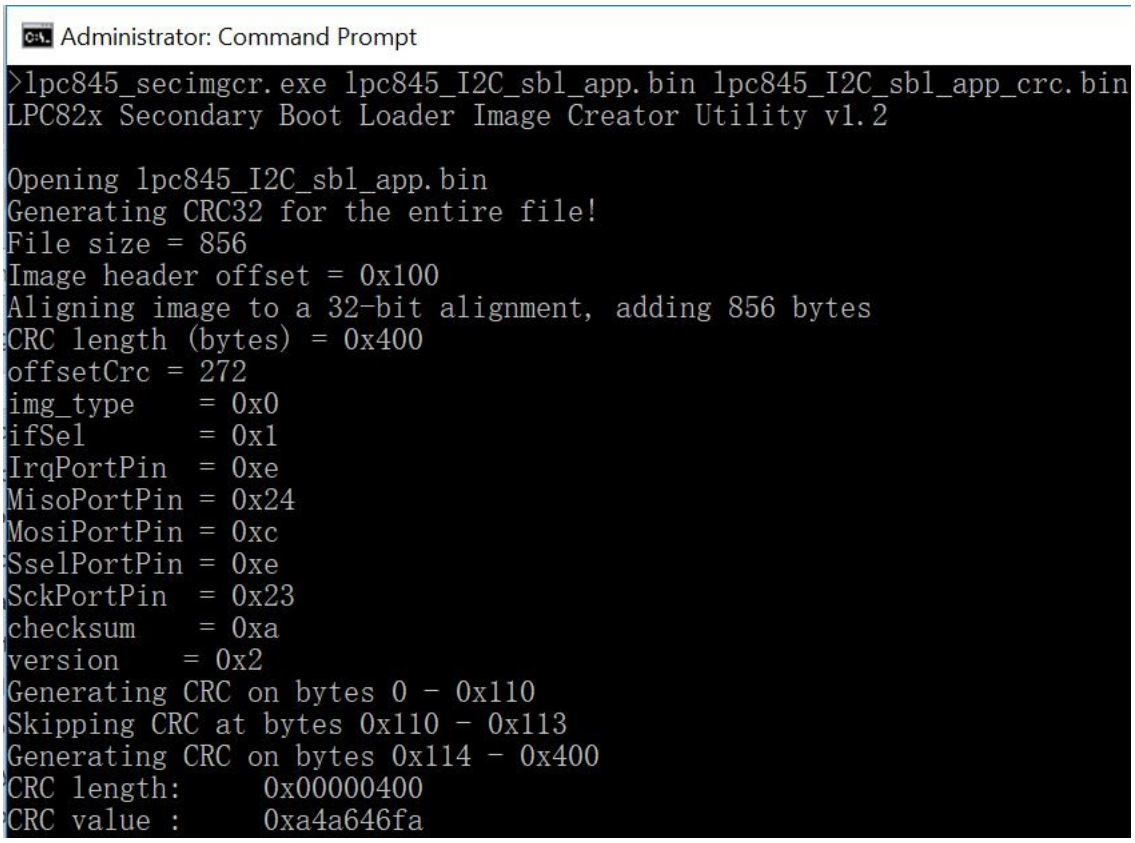

Figure 13. Image with CRC header

The CRC can be generated over the image header or over the entire length of the image. The syntax is

```
C:\<path>\ lpc80x_secimgcr.exe -n[1,2] <input filename.bin>> <output filename.bin>
```

-n indicates the length of image over which CRC is generated. **n1** is the full application image and **n2** is just the image header. If **–n[1,2]** parameter is not specified, the default value is **nl**.

5 Programming and updating firmware

For `IMG_NORMAL` and `IMG_NO_CRC` image boot, the host processor uses the `nHostIRQ` line to stop booting the image and reprogram the part. In this case, the host I/O line connected to the LPC845 first works as an output and pulls low.

The `nHostIRQ` line on the LPC845 first works as an input to sense that the host has pulled this line low. When the SBL senses this line being pulled low, it stops proceeding to check the CRC32 of the image.

Then the Host needs to reconfigure the nHostIRQ line to be switched to being an input to allow the nHostIRQ line on the LPC845 to drive it.

With the emulated Android AP/Sensor Hub environment as described in SBL functionalities and boot process with SBL on page 1, the usage of `nHostIRQ` in `IMG_NORMAL` image booting is as shown in Figure 14. on page 10.

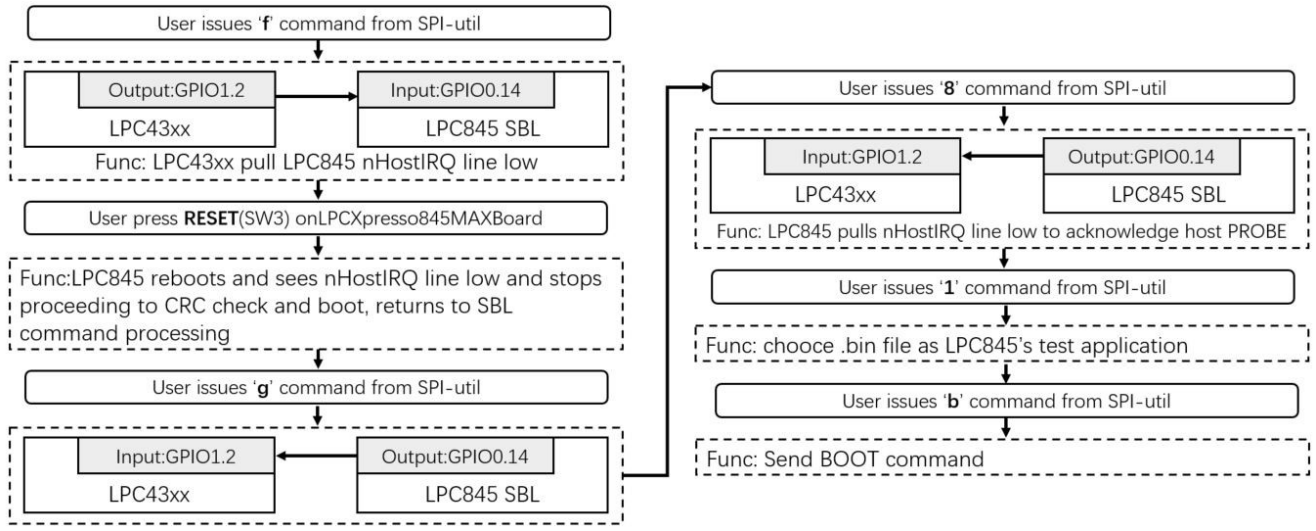

Figure 14. Usage of nHostIRQ and main steps to update firmware in flash

1. Program the sample application image (any of the Keil/IAR or MCUXpresso version).
2. Press the **Reset** button to boot the application image.
3. Issue the **f** command to pull `nHostIRQ` low.
4. Press the **Reset** button to reset the LPCXpresso845MAXBoard.
5. Issue the **g** command to program `nHostIRQ` as input.
6. Issue the **8** command to send `GetVerision`.
7. Issue the **1** command to update the Firmware, and then input the Firmware anme.
8. Issue the **b** command to BOOT the current Firmware.

If the newest test application is booted successfully, the BLUE LED will blink.

With the related project attached with this application note, users can understand this handshaking process by following the procedure shown in Figure 15. on page 11.

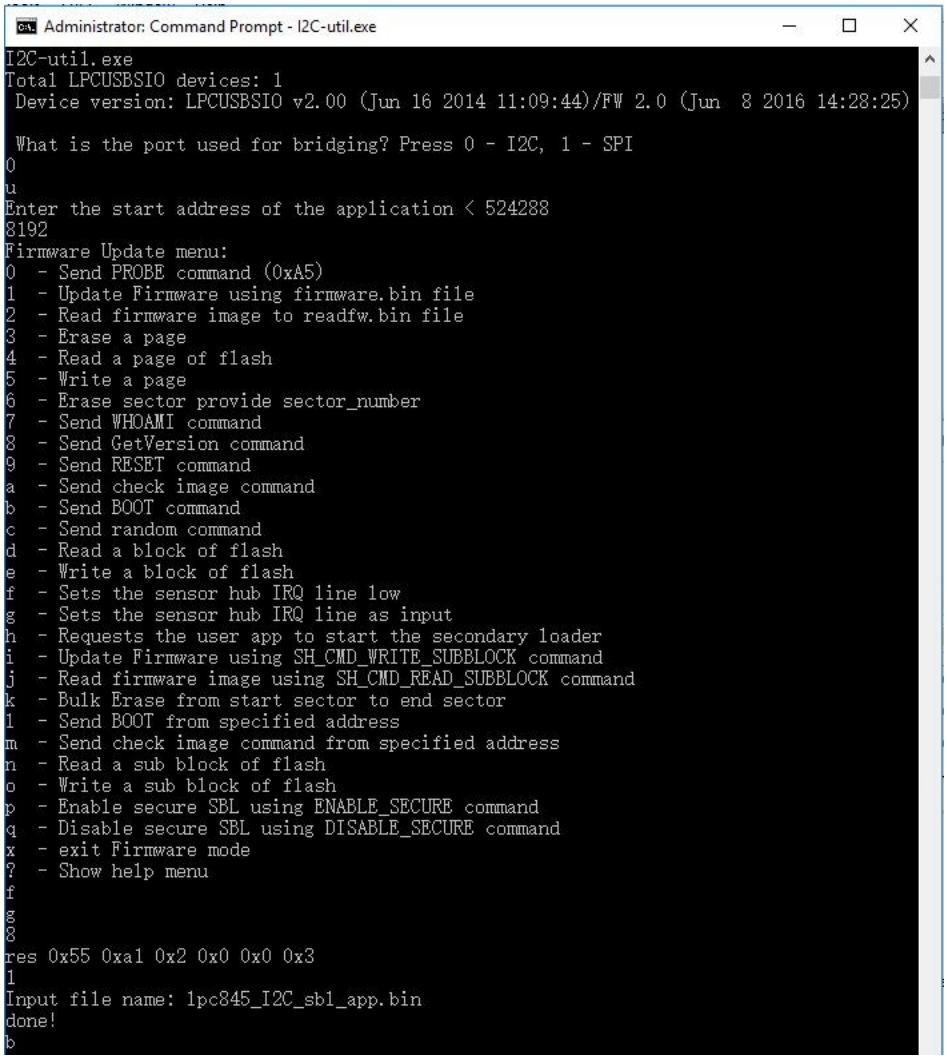

Figure 15. Handshakeing process during field firmware update

6 Conclusion

This application note provides a reference design for firmware updating for bug fixes or product updates using IAP via secondary bootloader using SBL. The user can refer to this application note to easily customize the host system and application.

7 References

- *LPC5410x I2C SPI Secondary Bootloader* (AN11610)
- *LPC82x I2C secondary bootloader* (AN11780)
- *LPC84x User manual* (UM11029)

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: April 2019
Document identifier: AN12393

arm