AN12327

Firmware Update Using Secondary Bootloader

Rev. 3 — 30 Octorber 2020

Application Note

Contents

1 Introduction

Dual image update (reliable update) is an important feature for advanced bootloaders. It assures that at least one image is bootable and works properly at any time. If any accident happens, the bootloader detects and uses the previous image as a bootable image.

However, the LPC55xx ROM bootloader does not support dual image feature yet. This application note implements a simple dual image update example on LPC55xx. It is useful to users for implementing a second dual image bootloader on LPC55xx series.

1.1 Glossary

Table 1 lists the abbreviation and acronymns used in the document.

Table 1. Glossary

Items	Description
SBL	Secondary Boot Loader
DSBL	Dual image Secondary Boot Loader
DSBL_APP	Example application demo to demonstrate dual image boot loader feature and work with DSBL
MCUBOOT	NXP unified bootloader solution, including protocol, PC software, documentation,and so on. It enables quick and easy programming through the entire product lifecycle. See MCUBOOT for details.
blhost	PC Command Line Interface (CLI) tools to implement MCUBOOT protocol. It is part of MCUBOOT software package.

2 Implementation

This section provides an overview of dual image layout implementation, boot flow, and application image format.

2.1 Overview

To ensure reliable update, a dual image layout is implemented. The idea is to download the image to a temporary region called the Receive Region. On every power cycle. The bootloader checks (integrity check passed) image in the Receive Region. If the downloaded new image has higher version number than the current image, DSBL copies the image from receive region into main region. A version flag located in the Image tracks the latest version in both regions.

In summary:

- Receive Region
  - — Bootloader downloads the new code to this area.
- Main Region
  - — Always store a correct image copied from Receive Region.
  - — DSBL jumps to image residing in the Main Region, if existing. For this, the image load address must be located in the Main Region.

The communication interface in this AN is via UART for demo purpose, the users can easily extend communication interface to others, such as I^{2}C SPI. The communication protocol follows NXP MCUBOOT protocol. which is compatible with LPC55xx ROM. Also, the following MCUBOOT protocol is helpful to users as they can reuse PC blhost software.

Figure 1 shows an overview of Flash partition.

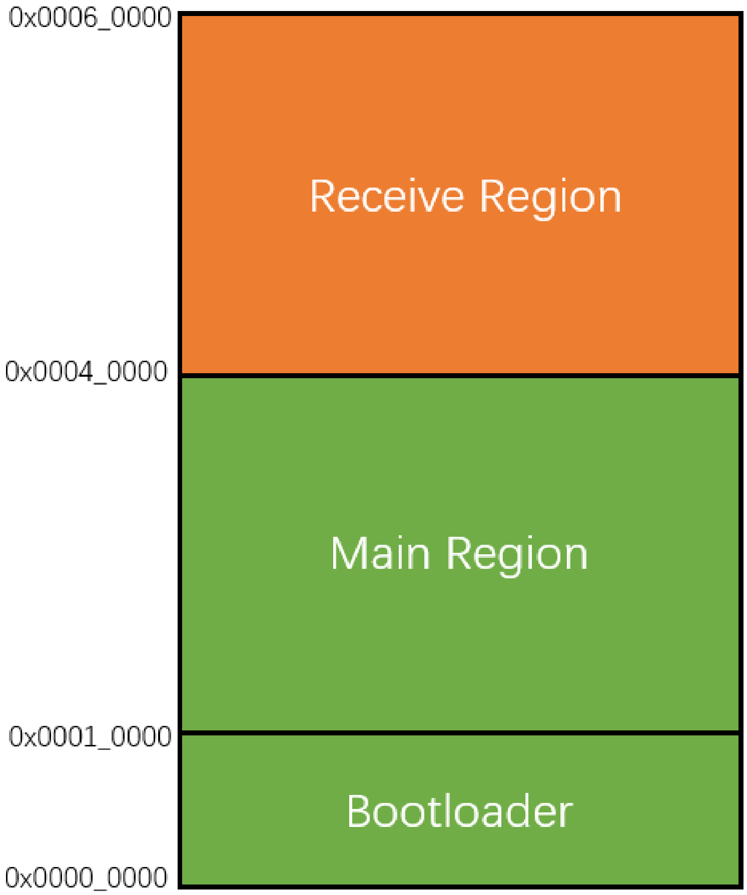


Figure 1. Flash partition

2.2 Boot flow

The DSBL is used to manage images and boot application. The DSBL code is executed every time the part is powered-ON or reset happens. Figure 2 shows the DSBL boot flow.

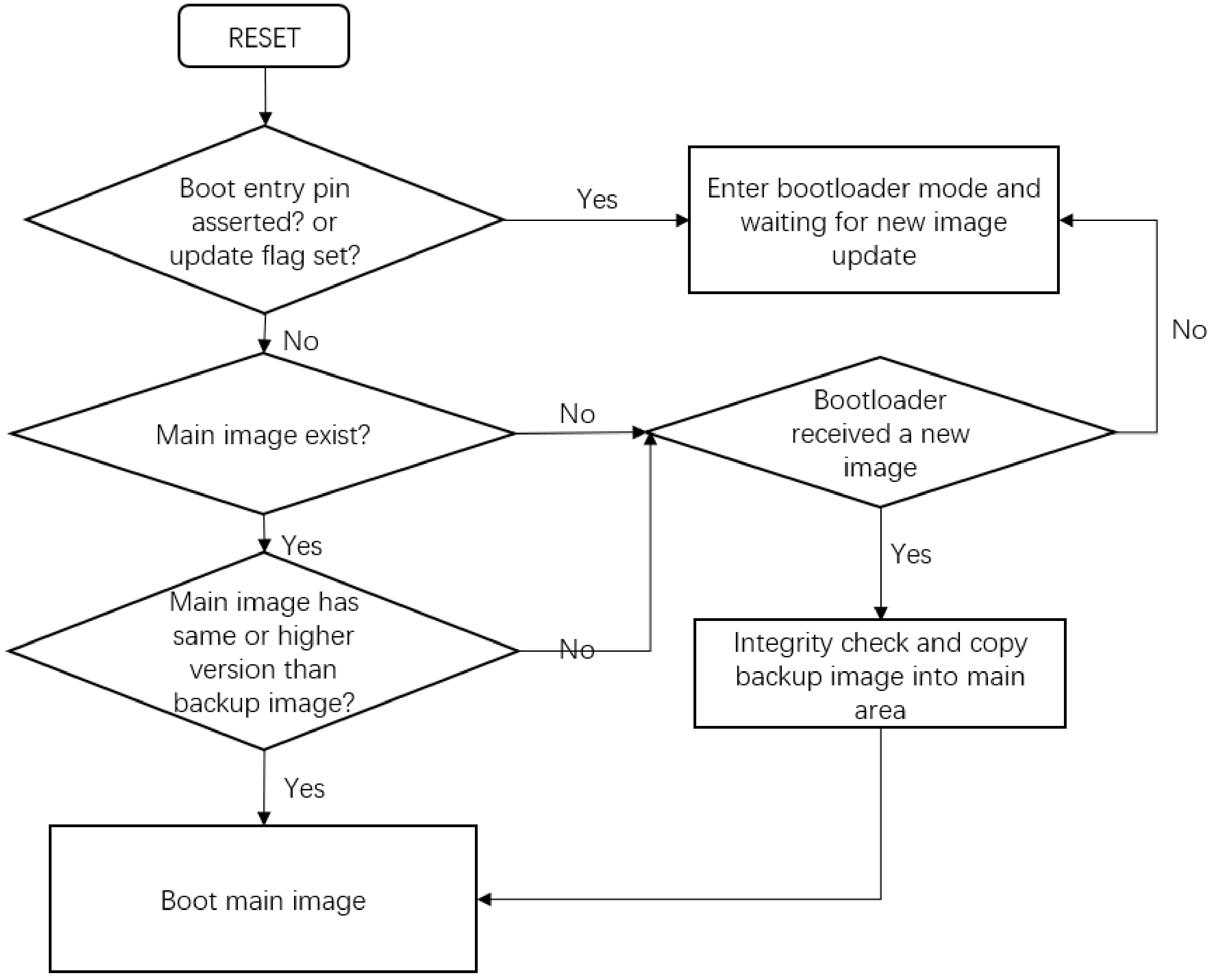

RESET
Boot entry pin asserted? or update flag set?
Yes
Enter bootloader mode and waiting for new image update
No
No
Main image exist?
No
Bootloader received a new image
Yes
Yes
Main image has same or higher version than backup image?
No
Integrity check and copy backup image into main area
Yes
Boot main image

Figure 2. DSBL boot flow

2.3 Application image format

This section describes the image memory layout and image creation steps. It takes LPC55S69 as example, and other LPC5500 series follow similar steps.

2.3.1 Image memory layout

Figure 3 shows the Dual Enhanced image type. It contains an image marker at offset `0x24`. It must also have a valid image header in the image pointed to at offset `0x28`. The starting address of the image must be fixed at `0x0001_0000`, the main region start address. The image header itself can reside in any area inside the image. In most case, the image header is put to the end of vector table. For LPC55xx series, it is offset `0x140`.

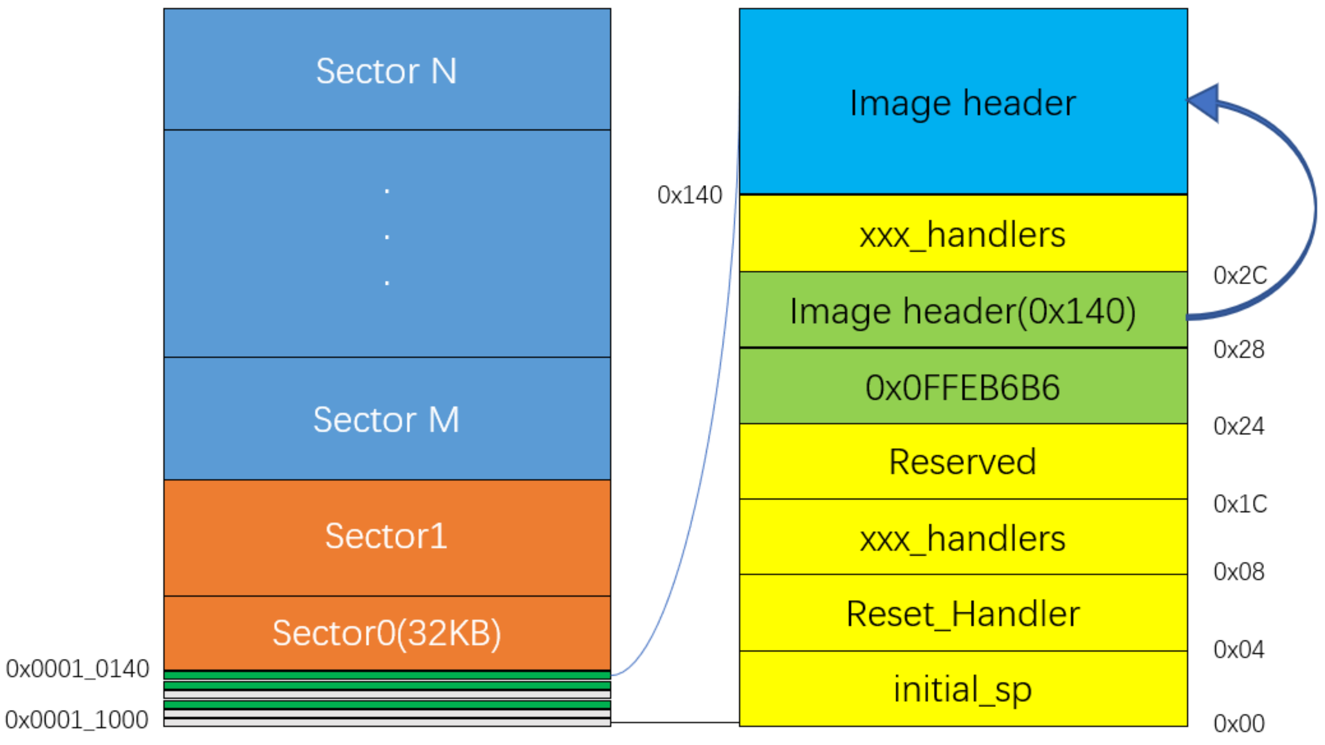


Figure 3. Application image layout

The image header itself is a 24-byte structure as listed in Table 2.

Table 2. Image header structure

Offset	Description
0x00	Header maker set to `0xFEEDA5A5`
0x04	Image Type (NORMAL = 0 or NO_CRC = 1)
0x08	Reserved
0x0C	Image length Length should be actual length, **4**, if CRC value field falls within the length.
0x10	CRC value
0x14	Version

With LPC55xx parts, CRC32 value for entire image binary is added in image header. The external tool, *image_generator.exe* file, helps add the image binary in the image header.

2.3.2 Image creation

This section describes the steps to modify the startup file in IDE and use external tools to add length and CRC value in image header.

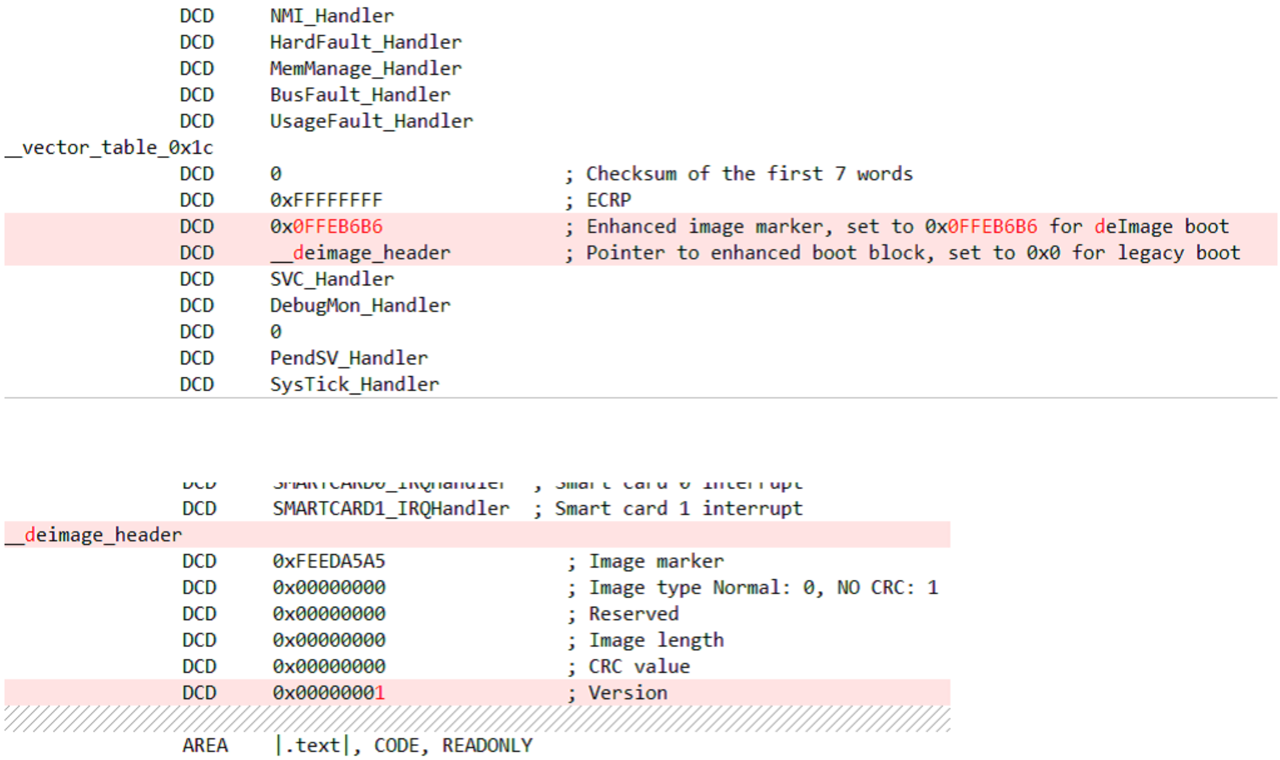
2.3.2.1 Modify startup file in IDE

Adding image marker and image header is done by modifying startup files.

- IAR

NOTE

Put the image header at the end of vector table.



```
                DCD     NMI_Handler
                DCD     HardFault_Handler
                DCD     MemManage_Handler
                DCD     BusFault_Handler
                DCD     UsageFault_Handler
__vector_table_0x1c
                DCD     0                          ; Checksum of the first 7 words
                DCD     0xFFFFFFFF                 ; ECRP
                DCD     0x0FFEB6B6                 ; Enhanced image marker, set to 0x0FFEB6B6 for deImage boot
                DCD     __deimage_header           ; Pointer to enhanced boot block, set to 0x0 for legacy boot
                DCD     SVC_Handler
                DCD     DebugMon_Handler
                DCD     0
                DCD     PendSV_Handler
                DCD     SysTick_Handler
```

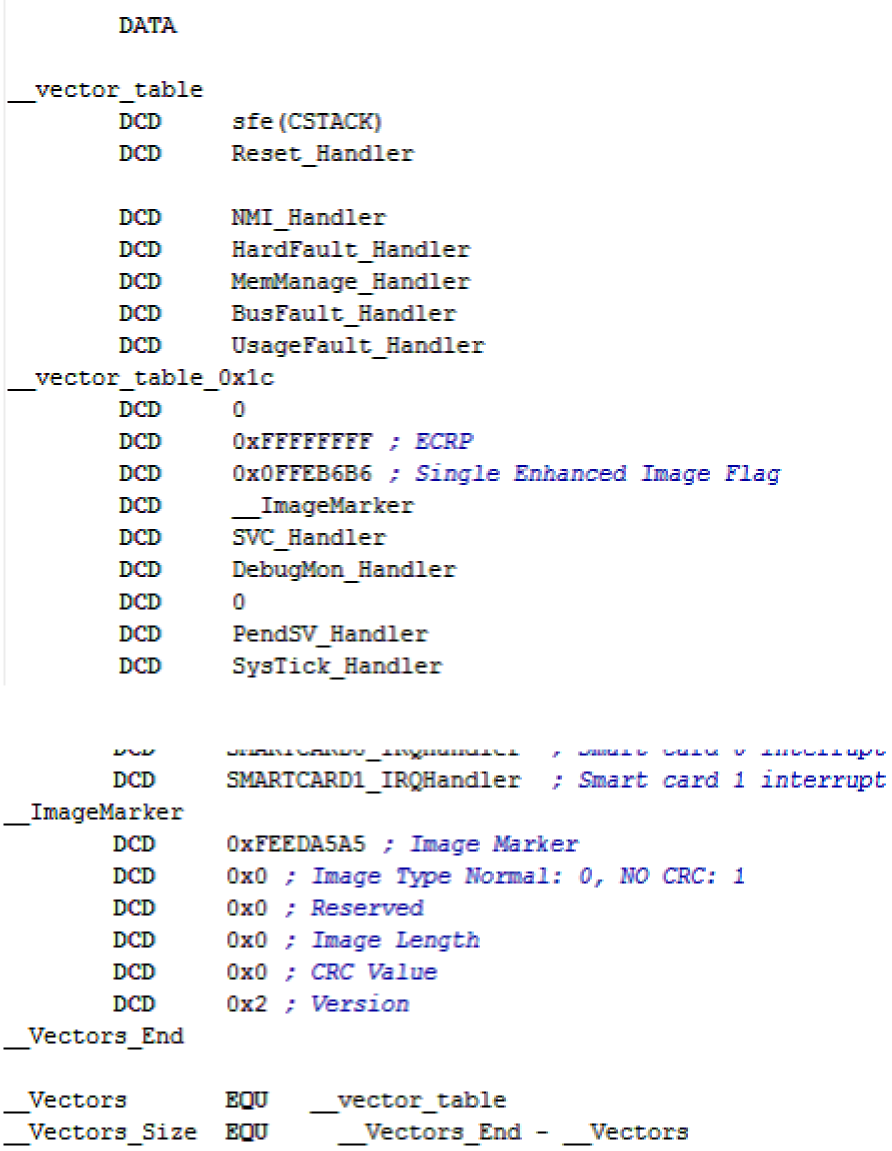
```
                DCD     SMARTCARD0_IRQHandler  ; Smart card 0 interrupt
                DCD     SMARTCARD1_IRQHandler  ; Smart card 1 interrupt
__deimage_header
                DCD     0xFEEDA5A5                 ; Image marker
                DCD     0x00000000                 ; Image type Normal: 0, NO CRC: 1
                DCD     0x00000000                 ; Reserved
                DCD     0x00000000                 ; Image length
                DCD     0x00000000                 ; CRC value
                DCD     0x00000001                 ; Version

                AREA    |.text|, CODE, READONLY
```

Figure 4. Adding image marker and image header in IAR

- KEIL

NOTE

Put image header at the end of vector table.



```
        DATA

__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler

        DCD     NMI_Handler
        DCD     HardFault_Handler
        DCD     MemManage_Handler
        DCD     BusFault_Handler
        DCD     UsageFault_Handler
__vector_table_0x1c
        DCD     0
        DCD     0xFFFFFFFF ; ECRP
        DCD     0x0FFEB6B6 ; Single Enhanced Image Flag
        DCD     __ImageMarker
        DCD     SVC_Handler
        DCD     DebugMon_Handler
        DCD     0
        DCD     PendSV_Handler
        DCD     SysTick_Handler

        DCD     SMARTCARD0_IRQHandler   ; Smart card 0 interrupt
        DCD     SMARTCARD1_IRQHandler   ; Smart card 1 interrupt
__ImageMarker
        DCD     0xFEEDA5A5 ; Image Marker
        DCD     0x0 ; Image Type Normal: 0, NO CRC: 1
        DCD     0x0 ; Reserved
        DCD     0x0 ; Image Length
        DCD     0x0 ; CRC Value
        DCD     0x2 ; Version
__Vectors_End

__Vectors       EQU   __vector_table
__Vectors_Size  EQU     __Vectors_End - __Vectors
```

Figure 5. Adding image marker and image header in Keil

2.3.2.2 Use external tools to add length and CRC value in image header

When the image type word in the image header is `0x00` (NORMAL), the image needs external tools to add length and CRC value into image header. The **image_generator.exe** file in the tool folder under the following location helps add length and CRC value into the image header.

```
\boards\<board_name>\dual_sbl\lpc55xx_dsbl_app\cm33_core0\tools
```

Double click **post_build.bat**, and the script calls **image_generator.exe** and generates the binary file named **dsbl_app_crc.bin** in this folder. Download the *.bin* file image to the receive region. For a step-by-step guide about how to use those tools, refer to Steps to run the demo.

3 Demo

The demo has two projects based on SDK, see Table 3 for details.

Table 3. Demo project description

Project name	Location in SDK	Description
`lpc55xx_dsbl`	*\boards\lpcxpresso55s69\dual_sbl\*	Dual image second boot loader project
`lpc55xx_dsbl_app`	*\boards\lpcxpresso55s69\dual_sbl\*	Demo application project

- The **lpc55xx_dsbl** stands for **lpc55xx dual image second boot loader**, which will be executed at boot-up. This program will handle Trust Zone configurations, communication with PC host, image check, and copying tasks. It is the first project you should download into EVK board.
- **lpc55xx_dsbl_app** stands for **lpc55xx dual image second boot loader application example**, which Is almost same as `hello_world`. The differences are:
  1. This image has an image marker and an image header resided after the vector table. So, it can be regionalized by DSBL.
  2. The linker starting address is modified from `0x0000_0000` to `0x0001_0000` to put loading/starting address into the main image region.

3.1 Hardware setup

The hardware uses LPC55S69 EVK board, as shown in Figure 6. Make sure you have read board user guide and familiar with basic function of the board, such as the positions of the **Reset** button and the debug connector, and so on.

This demo uses **Debug and UART USB connector (1)** as the debug interface and UART.

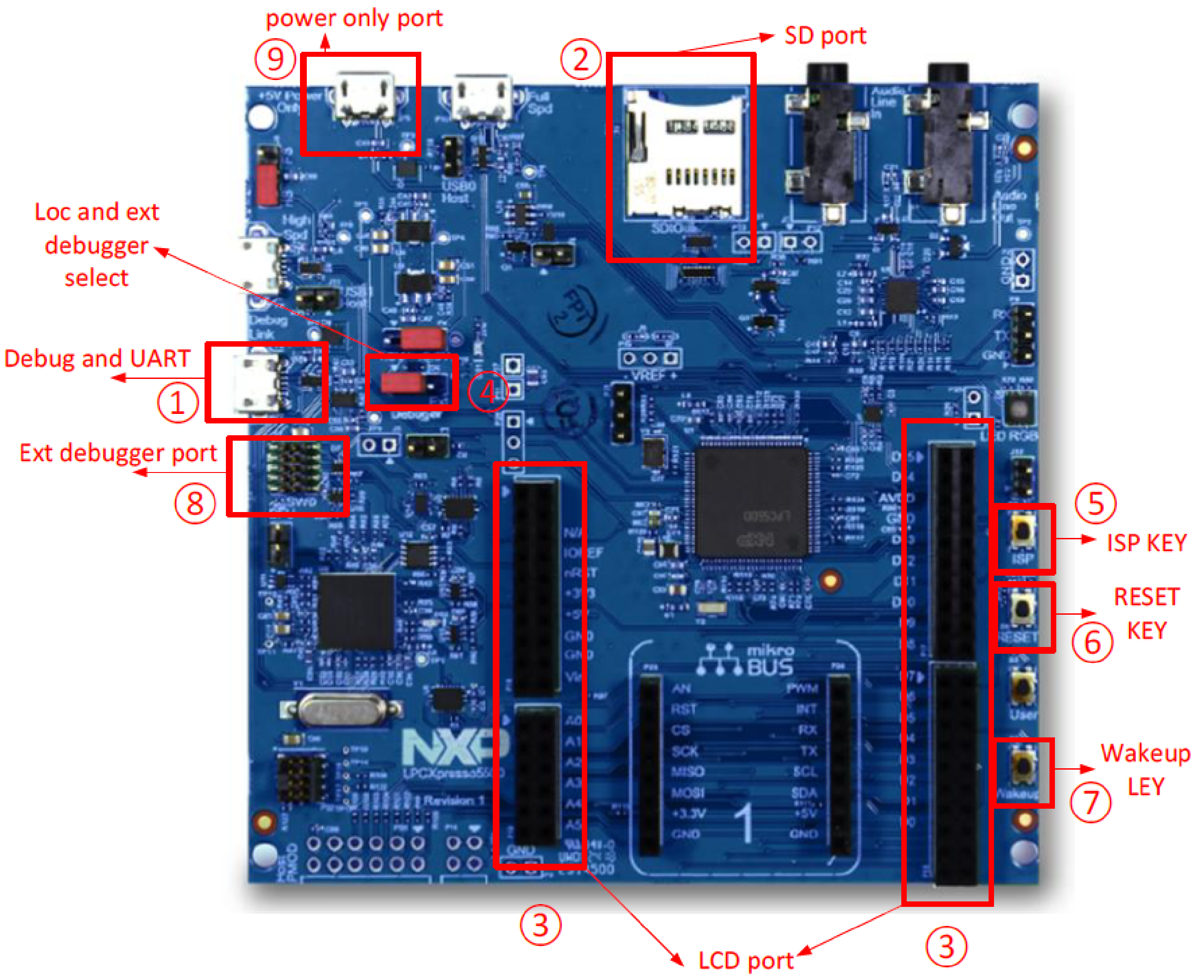


Figure 6. LPC55S69 EVK board

USB bridge and WAKEUP button (7) is used as the entry pin of the second bootloader.

The hardware uses LPCXpresso55S16 board, as shown in Figure 7. Make sure you have read board user guide and familiar with basic function of the board, such as the positions of the **RESET** button and the debug connector, etc. This demo uses **Debug and UART USB connector (J1)** as the debug interface and UART-USB bridge. Also, **WAKEUP** button (**SW1**) is used as the entry pin of the second bootloader.

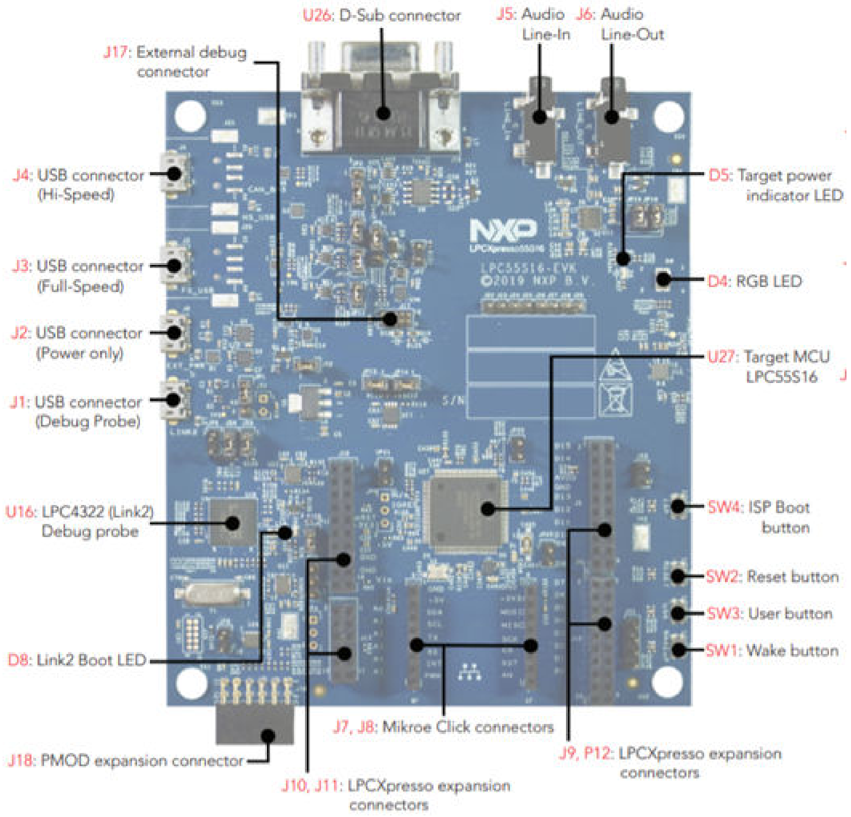

U26: D-Sub connector
J5: Audio Line-In
J6: Audio Line-Out
J17: External debug connector
J4: USB connector (Hi-Speed)
J3: USB connector (Full-Speed)
J2: USB connector (Power only)
J1: USB connector (Debug Probe)
U16: LPC4322 (Link2) Debug probe
D8: Link2 Boot LED
J18: PMOD expansion connector
J10, J11: LPCXpresso expansion connectors
J7, J8: Mikroe Click connectors
J9, P12: LPCXpresso expansion connectors
D5: Target power indicator LED
D4: RGB LED
U27: Target MCU LPC55S16
SW4: ISP Boot button
SW2: Reset button
SW3: User button
SW1: Wake button

Figure 7. LPCXpresso55S16 board

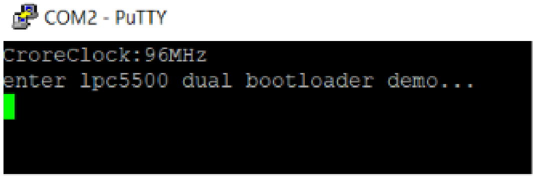
Other LPC55xx boards have similar setup steps. See EVK board user guide for details.

3.2 Steps to run the demo

NOTE

Ensure that you have basic knowledge about the LPC5500 series EVK board, have installed related LPC-Link II debugger driver, have successfully run the `hello_world` example in the SDK folder, and have verified the UART communication with PC.

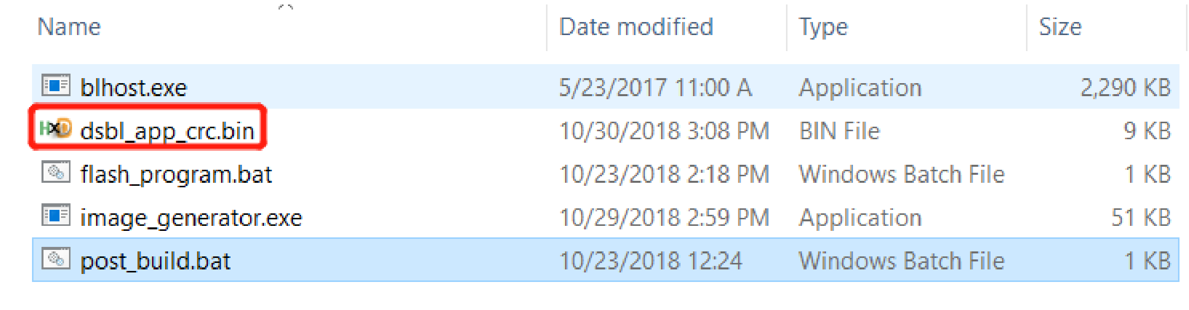
1. Connect USB with **Debug and UART USB connector (1)** to power up board and establish debug and UART connection.
2. Open, compile, and download the **lpc55xx_dsbl** project. Open your serial terminal with 115200-N-8-N-1.
3. Hold **Wake-up button (7)**, and then press the **RESET** button. This forces DSBL to enter boot loader mode. In this mode, DSBL does not boot any application, but waits for UART connection.
4. By default, the `lpc55xx_dsbl` enables debug log. The terminal provides information on Figure 2, which indicates that the DSBL is successful running and enters the boot loader mode.
5. Open and compile project: **lpc55xx_dsbl_app**. Do not use IDE to download the `lpc55xx_dsbl_app` project. Otherwise, it is meaningless to demonstrate the boot loader feature.



```
CroreClock:96MHz
enter lpc5500 dual bootloader demo...
```

Figure 8. DSBL enter boot loader mode

6. Open the *\boards\<board_name>\dual_sbl\lpc55xx_dsbl_app\cm33_core0\tools* folder and double click **post_build.bat**. This generates **dsbl_app_crc.bin** , which adds CRC and image length information to **image_generator.exe**.



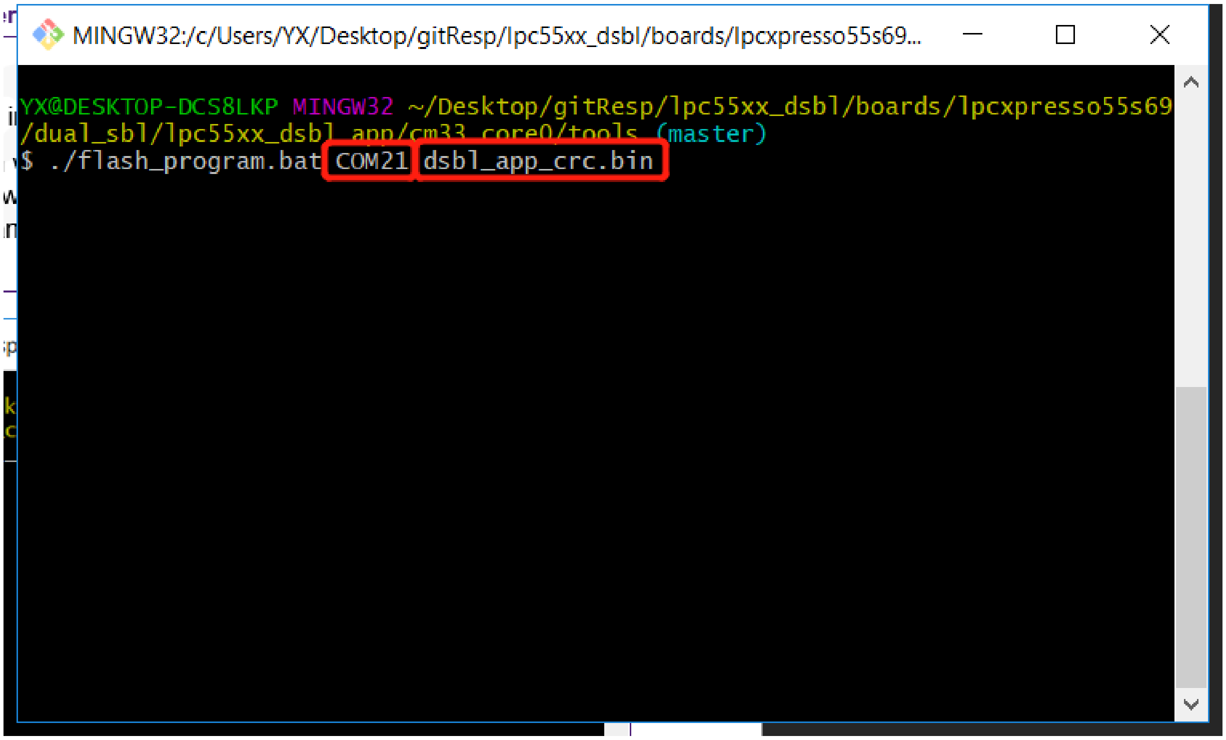
Name	Date modified	Type	Size
blhost.exe	5/23/2017 11:00 A	Application	2,290 KB
dsbl_app_crc.bin	10/30/2018 3:08 PM	BIN File	9 KB
flash_program.bat	10/23/2018 2:18 PM	Windows Batch File	1 KB
image_generator.exe	10/29/2018 2:59 PM	Application	51 KB
post_build.bat	10/23/2018 12:24	Windows Batch File	1 KB

Figure 9. Using post_build.bat to generate dsbl_app_crc.bin

The **dsbl_app_crc.bin** is the binary image to be downloaded in the Receive region.

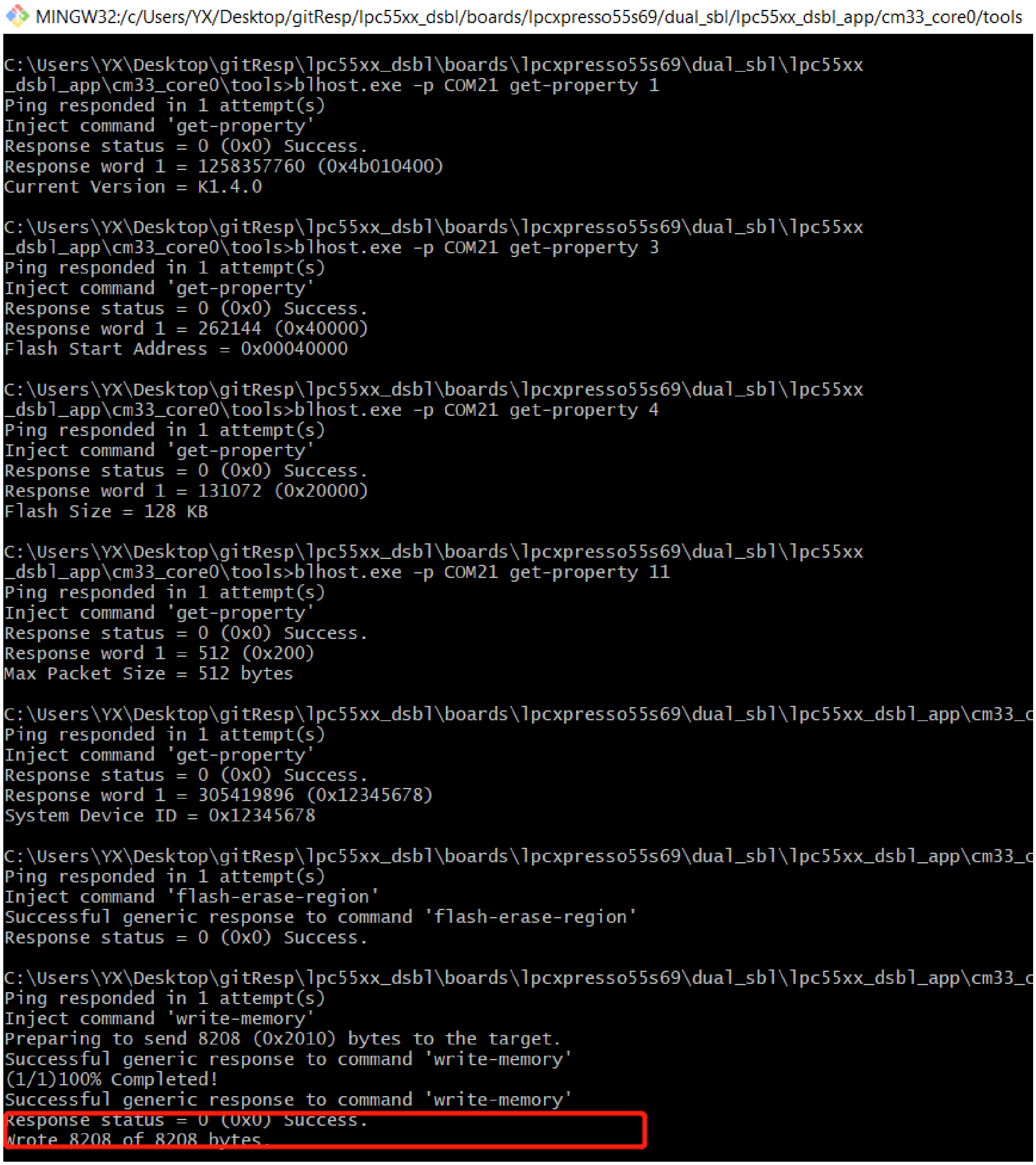
7. Close the serial terminal. Open bash window or command window and execute **flash_program.bat.** This script calls blhost.exe and downloads **dsbl_app_crc.bin** in the **Receive** region. Running **flash_program.bat** need two parameters: UART COM index and the full name of the app image.

8. As the script executes, the new image is downloaded in the **Receive** region.


MINGW32:/c/Users/YX/Desktop/gitResp/lpc55xx_dsbl/boards/lpcxpresso55s69...

```
YX@DESKTOP-DCS8LKP MINGW32 ~/Desktop/gitResp/lpc55xx_dsbl/boards/lpcxpresso55s69
/dual_sbl/lpc55xx_dsbl_app/cm33_core0/tools (master)
$ ./flash_program.bat COM21 dsbl_app_crc.bin
```

9. Reopen UART terminal and press the **RESET** button.

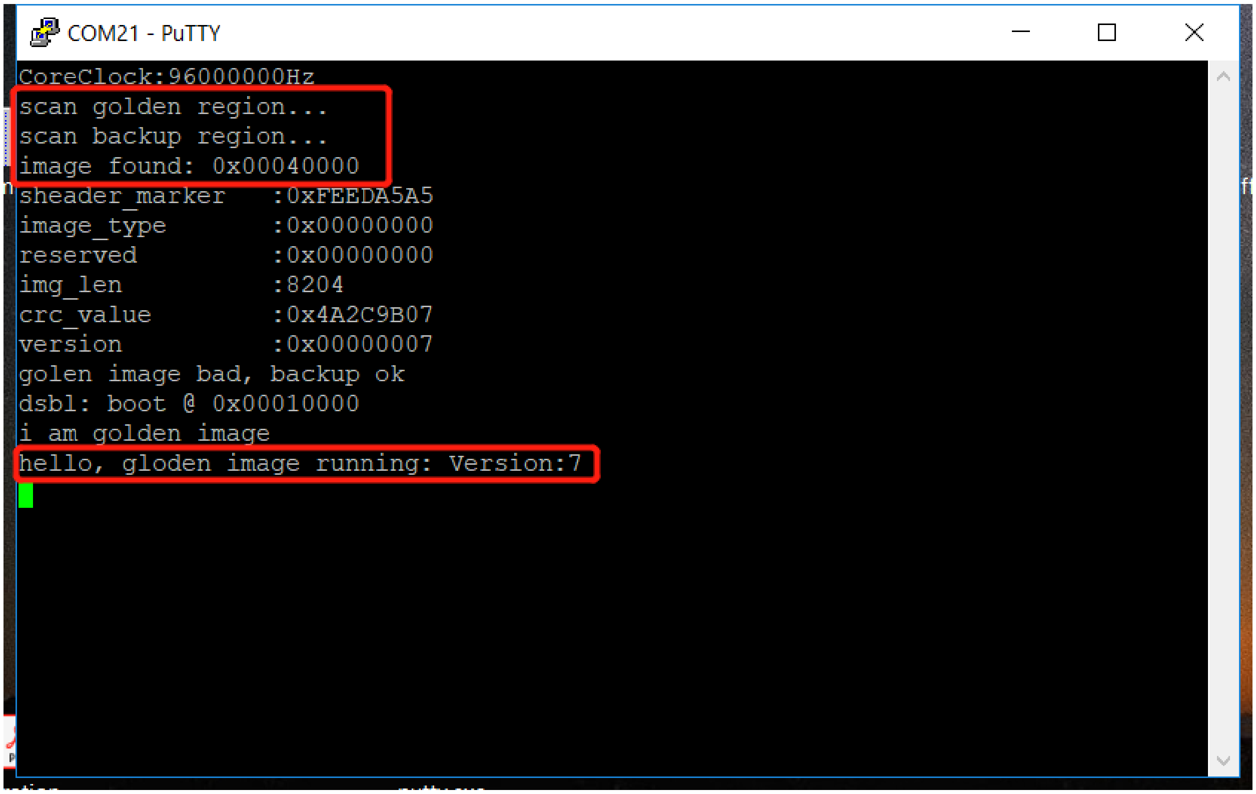


MINGW32:/c/Users/YX/Desktop/gitResp/lpc55xx_dsbl/boards/lpcxpresso55s69/dual_sbl/lpc55xx_dsbl_app/cm33_core0/tools

```
C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx
_dsbl_app\cm33_core0\tools>blhost.exe -p COM21 get-property 1
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 1258357760 (0x4b010400)
Current Version = K1.4.0

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx
_dsbl_app\cm33_core0\tools>blhost.exe -p COM21 get-property 3
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 262144 (0x40000)
Flash Start Address = 0x00040000

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx
_dsbl_app\cm33_core0\tools>blhost.exe -p COM21 get-property 4
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 131072 (0x20000)
Flash Size = 128 KB

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx
_dsbl_app\cm33_core0\tools>blhost.exe -p COM21 get-property 11
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 512 (0x200)
Max Packet Size = 512 bytes

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx_dsbl_app\cm33_c
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 305419896 (0x12345678)
System Device ID = 0x12345678

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx_dsbl_app\cm33_c
Ping responded in 1 attempt(s)
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.

C:\Users\YX\Desktop\gitResp\lpc55xx_dsbl\boards\lpcxpresso55s69\dual_sbl\lpc55xx_dsbl_app\cm33_c
Ping responded in 1 attempt(s)
Inject command 'write-memory'
Preparing to send 8208 (0x2010) bytes to the target.
Successful generic response to command 'write-memory'
(1/1)100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.
Wrote 8208 of 8208 bytes.
```

Figure 10. Download log for flash_program.bat

The log **image found: 0x0004_0000** indicates that DSBL has detected there is an image resided in **Receive** region. Since the main region does not have any valid image, the DSBL copes receives image in the main region and boots it.



```
CoreClock:96000000Hz
scan golden region...
scan backup region...
image found: 0x00040000
sheader_marker    :0xFEEDA5A5
image_type        :0x00000000
reserved          :0x00000000
img_len           :8204
crc_value         :0x4A2C9B07
version           :0x00000007
golen image bad, backup ok
dsbl: boot @ 0x00010000
i am golden image
hello, gloden image running: Version:7
```

Figure 11. Image copied to main region and booted

The log **hello, main image running, Version: 7** indicates thatthe main image has already run.

3.3 Methods to reenter DSBL

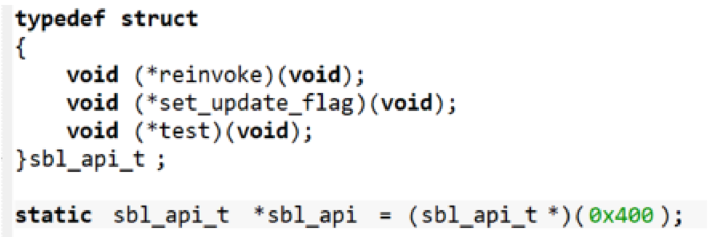
Besides using Wake-up button to enter DSBL, there are two more methods to enter DSBL for an application update.

3.3.1 Reinvoke

Define the `sbl_api` structure in your application, as shown in like Figure 12. Then, call `re_invoke`.

```
sbl_api->reinvoke();
```

Calling re-invoke forces the CPU to jump to DSBL immediately just like `ROM_API` reinvokes in the legacy LPC parts.



```
typedef struct
{
    void (*reinvoke)(void);
    void (*set_update_flag)(void);
    void (*test)(void);
}sbl_api_t ;

static sbl_api_t  *sbl_api  = (sbl_api_t *)(0x400);
```

Figure 12. DSBL API structure

3.3.2 Set_update_flag

Different from reinvoke, this API does not enter the DSBL immediately. It lets DSBL enter the update mode at next power cycle. A non-volatile update flag is set. DSBL counts the update the failure times. If the image update in the Receive region fails more than three times, the DSBL clears the `update_flag` and boots the main image. Otherwise, on every power cycle, DSBL does not boot main image and waits for a successful download. Calling this API is same as reinvoke.

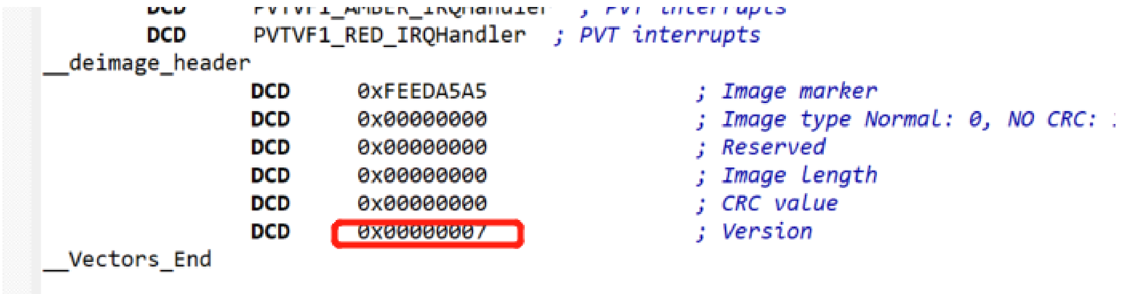
```
sbl_api->set_update_flag();
```

3.4 Modify application image

Update application image version information is simple. You just need to modify the version word in the image header.

NOTE

DSBL copies the received image to the main region only if the received image has higher version number than the main image.



```
        DCD     PVTVF1_AMBER_IRQHandler  ; PVT interrupts
        DCD     PVTVF1_RED_IRQHandler  ; PVT interrupts
__deimage_header
                DCD     0xFEEDA5A5              ; Image marker
                DCD     0x00000000              ; Image type Normal: 0, NO CRC: :
                DCD     0x00000000              ; Reserved
                DCD     0x00000000              ; Image length
                DCD     0x00000000              ; CRC value
                DCD     0x00000007              ; Version
__Vectors_End
```

Figure 13. Update DSBL_APP image version information

4 Consideration and limitation

This section provides information on the flash read operation, UART multiplex, and the steps to enable or disable debug log.

4.1 Flash read operation

In most cases, AHB bus reads Flash directly. However, in LPC55xx, any attempt to directly read an erased flash (erased but not written) leads to Hard Fault due to Flash ECC mechanism. The fault is inconvenient for boot loader development. To tackle this problem, implement a **Non-AHB method to read flash data API** to replace AHB bus directly read. The code for **Non-AHB method to read Flash data API** is in **memory.c**. Check the code for details.

4.2 UART multiplex

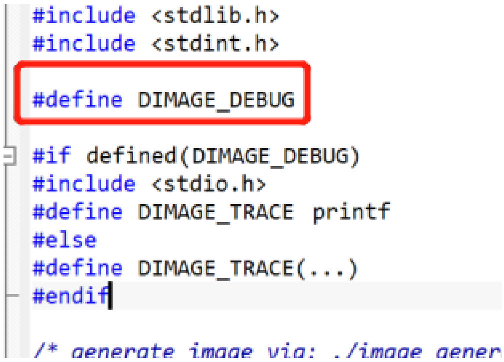
In this demo, same UART is used by three functions:

1. DSBL debug log output
2. Application demo log output
3. The communication interface for DSBL to download image

Consequently, there is a UART multiplex conflict issue. Whenever for using the `blhost` to download images, UART terminal must be closed to release PC COM port resource for `blhost` use.

4.3 Enable / disable debug log

DSBL debug log is enabled and disabled by macro. Commenting macro `DIMAGE_DEBUG` in *dimage.h* disables all debug output. See Figure 14.



```
#include <stdlib.h>
#include <stdint.h>

#define DIMAGE_DEBUG

#if defined(DIMAGE_DEBUG)
#include <stdio.h>
#define DIMAGE_TRACE printf
#else
#define DIMAGE_TRACE(...)
#endif

/* generate image via: ./image gener
```

Figure 14. Enable/Disable DSBL debug log

5 Revision history

Table 4 summarizes the changes since the initial release.

Table 4. Revision history

Revision number	Date	Substantive changes
0	23 January 2019	Initial release
1	26 February 2020	Updated Steps to run the demo and other general changes
2	21 May 2020	Updated Figure 8
3	30 October 2020	Replaced LPC55S16 with LPC5500 series

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: 30 Octorber 2020
Document identifier: AN12327