AN12037

LPC11U6x USB DFU Secondary Bootloader

Rev. 1 — 4 September 2017 Application note

Document information

Info	Content
Keywords	LPC11U68, secondary boot loader, dfu, image creator tool, firmware update
Abstract	This application note introduces the image creator tool and DFU utility programs to facilitate the use of a DFU SBL with an LPC11U6x application to enable firmware updates in the field. This program or document is based on AN11732 and support dual image.

Revision history

Rev	Date	Description
1.0	20170904	Initial release

Contact information

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

1. Introduction

The LPC11U6x provides the user a convenient way to update the flash content in the field for bug fixes or product updates. It can be achieved using the following two methods:

- ISP: In-System Programming mode can be used to program or re-program the on-chip flash memory using the internal bootloader and UART0 serial port. It can be done when the part resides on the end-user board.
- IAP: In-Application Programming mode performs erase and write operations on the on-chip flash memory, as directed by the end user application code.

A Secondary BootLoader (SBL) is a piece of code that allows the user application code to be downloaded using alternative channels other than the standard UART0 used by the internal bootloader. The primary bootloader is the firmware that resides in the boot ROM block of the microcontroller and is executed on power-up and resets. After the execution of boot ROM, the secondary bootloader would be executed, which will then execute the end-user application.

The purpose of this document is to explain how to use the two tools(FlashMagic or SWD debugger) provided by NXP to easily incorporate a Device Firmware Update (DFU) SBL with any given LPC11U6x application binary.

The Secondary BootLoader will not initialize any peripherals except the USB port.

If user requires to configure the dedicated pin function, initialize the pin function in application project or if needed, configure the pin function in bootloader project.

2. Package Contents

The extracted contents of the package is shown in Fig 1.

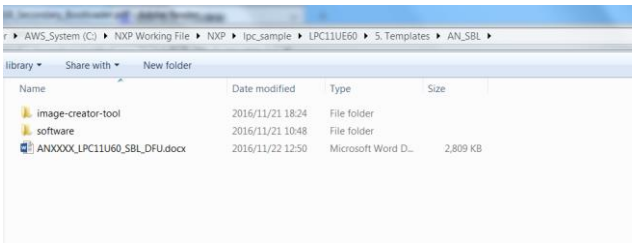


Fig 1. Package Content

Following is a brief description for each of the folder:

1. **dfu-util:** This folder contains the dfu-util tool with the source available from http://dfu-util.gnumonks.org/. It is used to interface with the SBL through DFU, windows, OS X and Linux operating system, each have a dfu-util too.

2. **Drivers:** This folder contains the “lpcdevice” driver (including USB descriptors) that must be installed for a Windows7 machine to successfully detect the DFU mode used on the LPC11U6x.

3. **Image-creator-too**l: This folder contains the lpc11xx_secimgcr.exe program. It is used to create encryption keys, generating and inserting a valid CRC, encrypting firmware images and generating factory images. The DFU SBL is embedded inside this tool.

4. **Software binaries:** This folder contains sample binaries of all the files that can be generated with the image creator tool.

   a) lpc11u68_1st_blinky.bin – Sample application binary which is the first image

   b) lpc11u68_1st_burn.bin – The first sample application binary created by lpc11xx_secimgcr.exe

   c) lpc11u68_2nd_blinky.bin – Sample application binary which is the second image

   d) lpc11u68_2nd_burn.bin – The second sample application binary is created by lpc11xx_secimgcr.exe

   e) sbl_dfu_lpc11u60.bin – Secondary bootloader image, which can program into flash directly by flashmagic or other programming tools

3. Hardware environment

This application was tested using LPCXpresso board for LPC11U68 (OM13058).

For more information on this board, visit the following link:

http://www.nxp.com/pages/lpcxpresso-board-for-lpc11u68:OM13058?lang_cd=en

Fig 2. OM13058 LPCXpresso Board for LPC11U68

Schematic: LPCXpresso board for LPC11U68

Note: Download the right schematic based on your board mark

(the mark is located next to the NXP logo)

This DFU SBL is tested and developed in real product.

The sample test application is only a test application.

A **key note** for on-board USB ports.

LPCXpresso board has two USB connectors. One is LPC11U68 serial port for debugging LPC11U68 and log print The other port is USB port of LPC11U68 for DFU communication with PC.

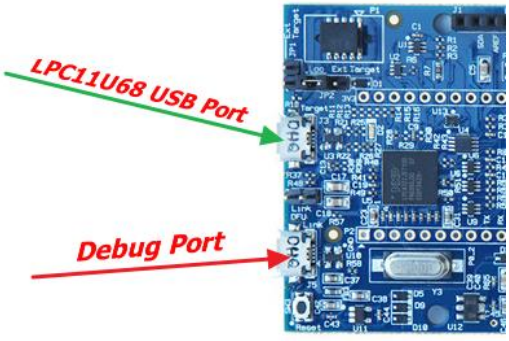


Fig 3. USB port on LPCXpresso11U68 board

4. Software environment

Following is the list of software download links that can be used in the next steps:

The MCU part software is developed under LPCXPresso IDE or MCUXpresso IDE.

If the customer wants to modify the SBL, download the right IDE:

LPCXpresso Download (not preferred; only for classic customer)

http://www.nxp.com/products/software-and-tools/software-development-tools/software-tools/lpc-microcontroller-utilities/lpcxpresso-ide-v8.2.2:LPCXPRESSO?fsrch=1&sr=5&pageNum=1

MCUXpresso Download(preferred)

http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-arm7-arm9-mcus/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE

Software: DFU source

https://sourceforge.net/p/dfu-util/dfu-util/ci/master/tree/src/

dfu-util website

http://dfu-util.sourceforge.net/

FlashMagic

FlashMagic is a PC tool for programming flash-based microcontrollers from NXP through a serial port.

http://www.flashmagictool.com/

5. Development flow

Following steps should be followed to enable DFU updates for a given application:

1. Modify the existing application to support the DFU SBL. It includes reserving flash and SRAM for the SBL, adding an image header to the application binary, enumerating the MCU as a DFU device, and supporting the DFU detach command to context switch the application to SBL. See Section 6 for information on the test application, which can be used as a base for a USB application that is compatible with the DFU SBL. See Section 7, for the information on how to integrate the requirements for the DFU SBL in an existing application.

2. Use the image creator tool to modify the application binary. While the DFU SBL is not provided in the form of a project or binary, it is embedded in the image creator tool. To add the SBL into an application, use the image creator tool to combine the DFU and application binary into one binary, which is called a factory image. The image creator tool is also used to generate and insert a CRC checksum into the image header region of the application binary. Optionally, the image creator tool can generate encryption keys and use these keys to encrypt application images. See Section 8.1, for more information on the image creator tool.

3. Use the dfu-util executable or provided DFU scripts to perform a DFU update by issuing a detach command, and then sending updated application binary.

After these steps, the field updates with the DFU SBL follow the flow chart shown in Fig 4.

If the image is valid, the SBL jumps to apps. The timeout is defined in main_dfu.c, line595

5. SBL does not initialize any pin except the USB pins.

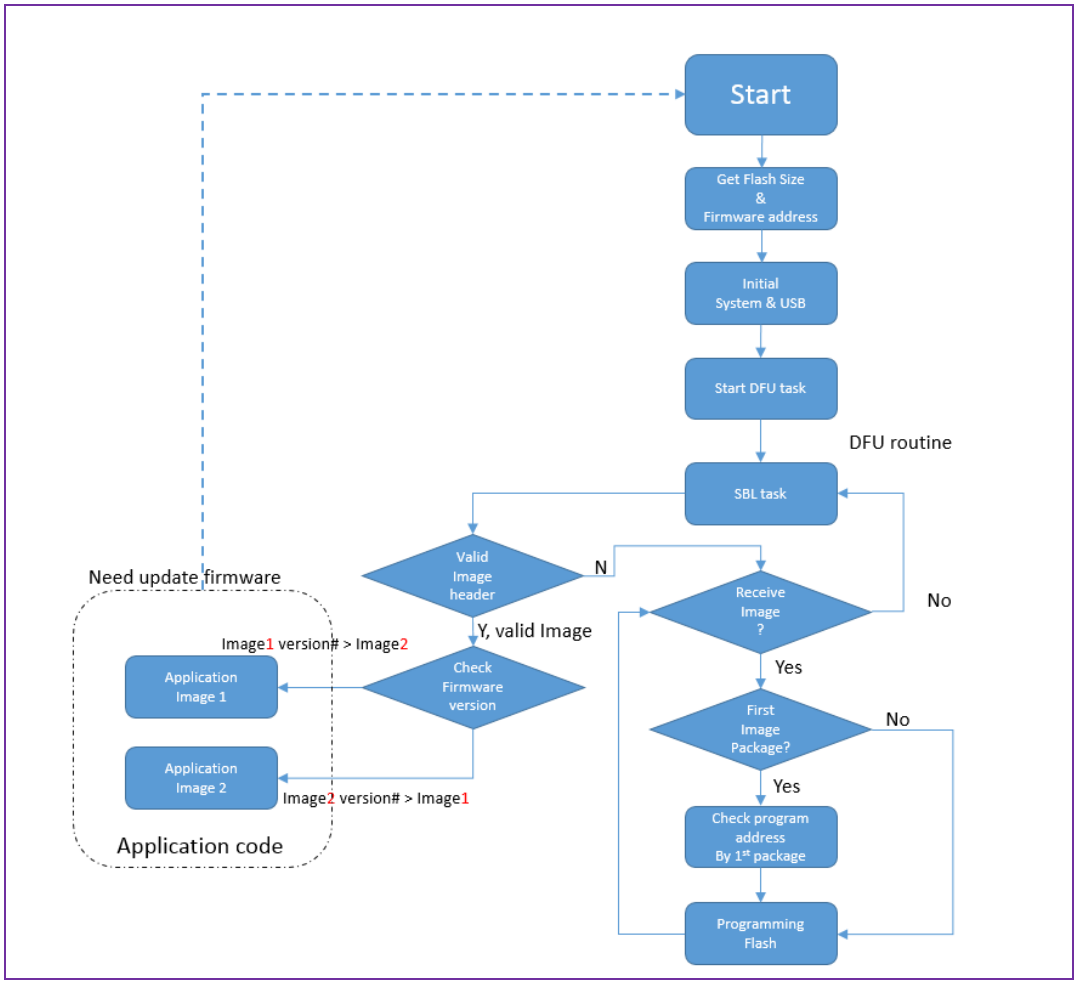


Fig 4. Filed update flow

5.1 Programming flow

With a DFU SBL capable application binary, the image creator tool should be used for application binaries with valid CRC checksums. The factory images and DFU capable application codes are essentially binaries.The factory images will end with a '.bin' extension,while the DFU capable files end in a '.dfu'. This difference in the extension is not required while using either the image creator tool or dfu-util tool. It helps to minimize the confusion.

See Fig 5 for field firmware update flow.

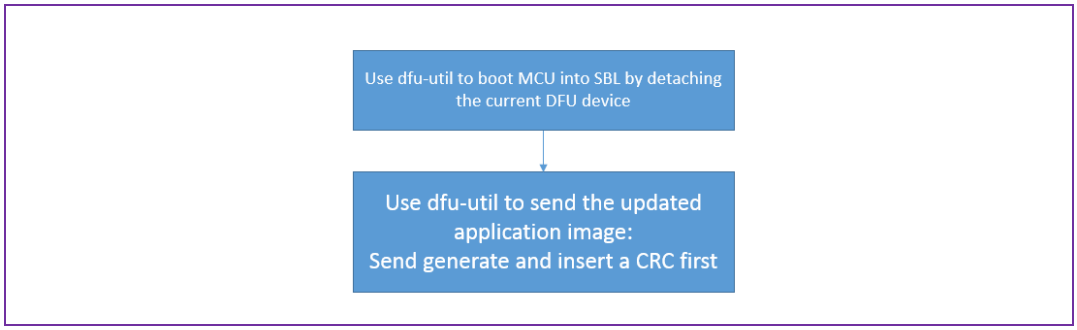


Fig 5. Field firmware update flow

5.2 Flash assignment

Fig 6 shows the flash assignment. Image1 and Image2 regions are dual image storage.

LPC_SBL_CalFlashSize() function (in source code file sbl_lpc.c) can calculate flash size and determine the secondary image boundary address automatically by MCU's product id.

Image1 and Image2 are active images. SBL(secondary bootloader) compares the version numbers of the two images.The image with new version number (bigger number) will be selected as the active image and SBL will jump to the newest image after the firmware is updated.

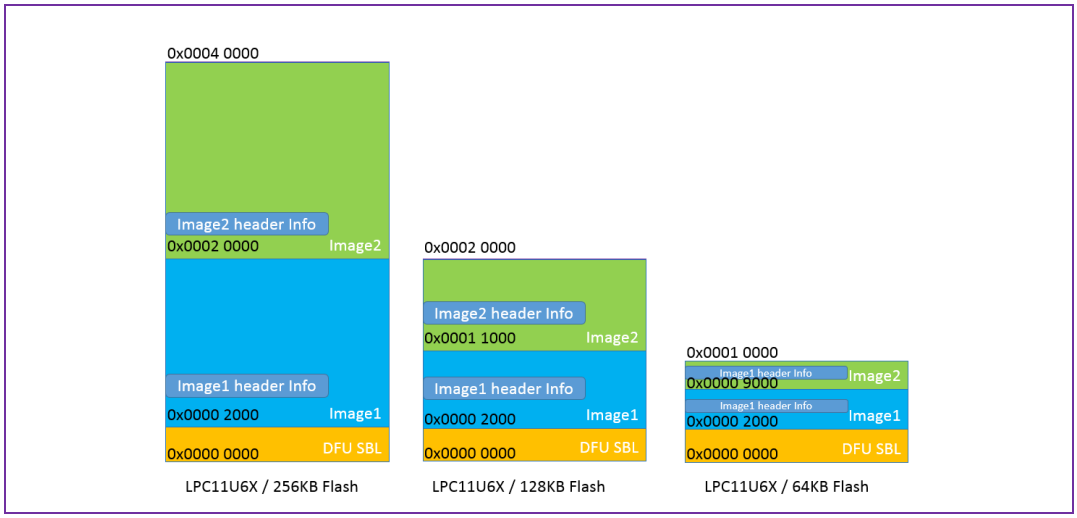


Fig 6. SBL and dual image flash region assignment

6. Introduction to Test Application

A simple LED blinky project that has all the settings configured to enable DFU updates is provided. It toggles LED_D0 and LED_D2 on the LPC11U68 LPCXpresso board. Image1 toggles LED_D0 and Image2 toggles LED_D2.The DFU SBL occupies the first two sectors of the user flash memory which means the test application is present at an offset 0x2000. Fig 7 shows the MCUXpresso options for the “Properties” window. The SBL uses the first 512 bytes of RAM from address 0x10000000, that is, from 0x10000000 to 0x10000200. This RAM space is used only when the DFU SBL is invoked from the application. This space can be used by the application but will be corrupted once the DFU SBL is invoked.

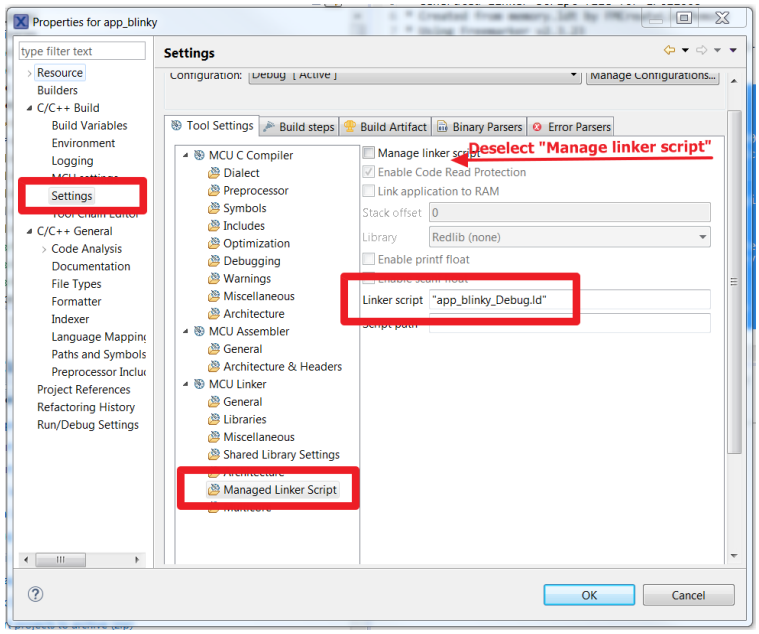


Fig 7. MCUXpresso properties setting

Following program is used for the memory setting in “app_blinky_debug_memory.ld”. We may set the flash size and RAM size here. This link descriptor file is applied for image1.

```
MEMORY
{
  /* Define each memory region */
  MFlash256 (rx) : ORIGIN = 0x2000, LENGTH = 0x1E000 /* 120K bytes (alias Flash) */
  Ram0_32 (rwx) : ORIGIN = 0x10000200, LENGTH = 0x1c00 /* 7K bytes (alias RAM) */
}

  /* Define a symbol for the top of each memory region */
  __base_MFlash256 = 0x2000; /* MFlash256 */
  __base_Flash = 0x2000; /* Flash */
  __top_MFlash256 = 0x2000 + 0x1E000; /* 120K bytes */
  __top_Flash = 0x2000 + 0x1E000; /* 120K bytes */
  __base_Ram0_32 = 0x10000200; /* Ram0_32 */
  __base_RAM = 0x10000200; /* RAM */
  __top_Ram0_32 = 0x10000200 + 0x1C00; /* 7K bytes */
  __top_RAM = 0x10000200 + 0x1C00; /* 7K bytes */
```

"app_blinky_debug.ld" is the linker script file. User may modify this file to relocate the application start address and allocate memory. This link descriptor file is dedicated for image1.

Note: The last line of the link descriptor file is the firmware version number of the image and can be changed by the user.

```
INCLUDE "app_blinky_Debug_library.ld"
INCLUDE "app_blinky_Debug_memory.ld"

ENTRY(ResetISR)

SECTIONS
{
    /* MAIN TEXT SECTION */
    .text : ALIGN(4)
    {
        FILL(0xff)
        __vectors_start__ = ABSOLUTE(.) ;
        KEEP(*(.isr_vector))
        KEEP(*(.sbl_sect))
        /* Global Section Table */
……
……
……

/* ## Create image header (used in startup) ## */

    PROVIDE(__image_header = 0xFEEDA5A5);

    /* ## Create SBL Config Info1 (used in startup) ## */
    PROVIDE(__sbl_config1 = (0x00000000);
    PROVIDE(__sbl_config2 = (0x00000000);

    /* ## Create crc32 length(used in startup) ## */
    PROVIDE(__crc32_len = 0x00000114 );
    PROVIDE(__crc32_val = 0xf15f814e );


    PROVIDE(__firmware_version = 0x00000002 );


}
```

The image1 's version number should be changed here.

6.1 Image header

The image header is stored in the memory region 0x100 – 0x117. It is used to indicate that it is a DFU capable application binary. The image header contains the necessary information for the SBL, such as the pin configuration table, CRC length, and CRC checksum. These parameters can be updated by the user.

Fig 8. shows the image header located in the MCUXPresso linker script file:

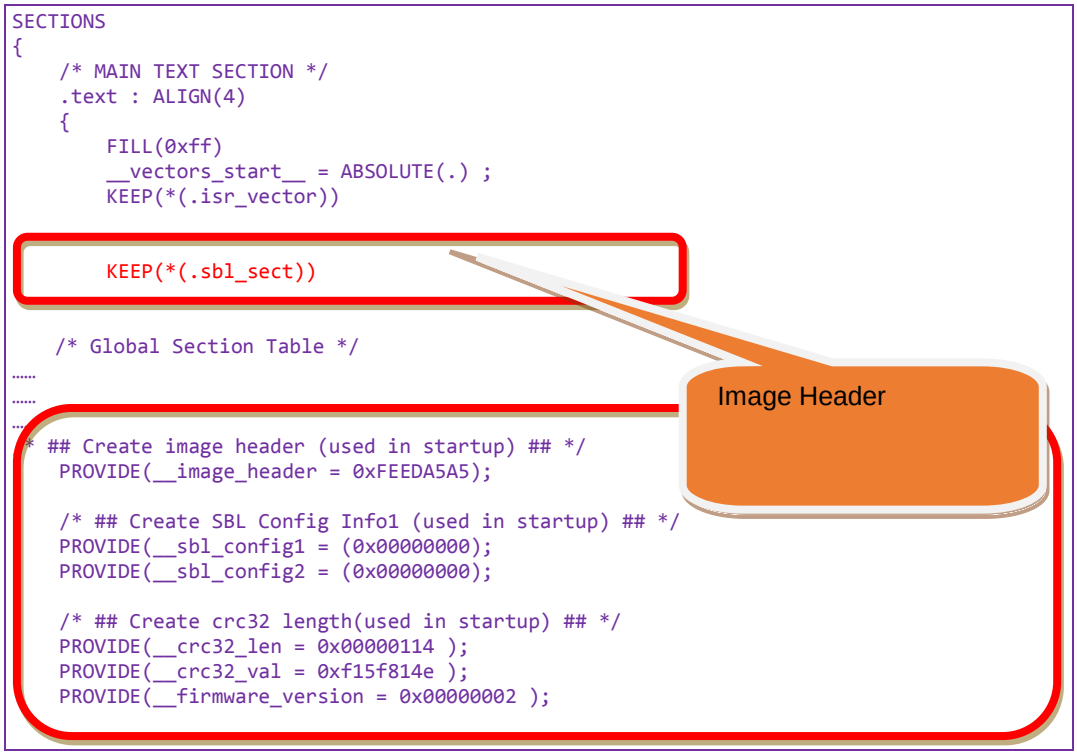


Fig 8. Image header located in the MCUXPresso linker script file

The USB Vendor ID (VID) passed is 0x1FC9 and USB Product ID (PID) is 0x5002. The USB string descriptors that are passed are defined in 'usbd_desc.c' file of the test application.

The USB string descriptors in the test application contains Index 0 to 5. Index 0x00 indicates the length of the descriptor, descriptor type and the language id.

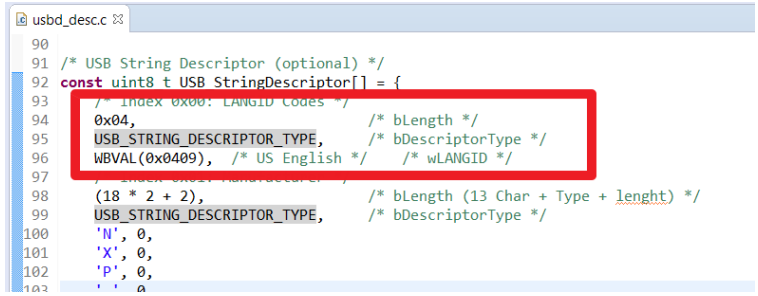


Fig 9. Index 0x00 of string descriptors

Index 0x01 indicates manufacturer details.

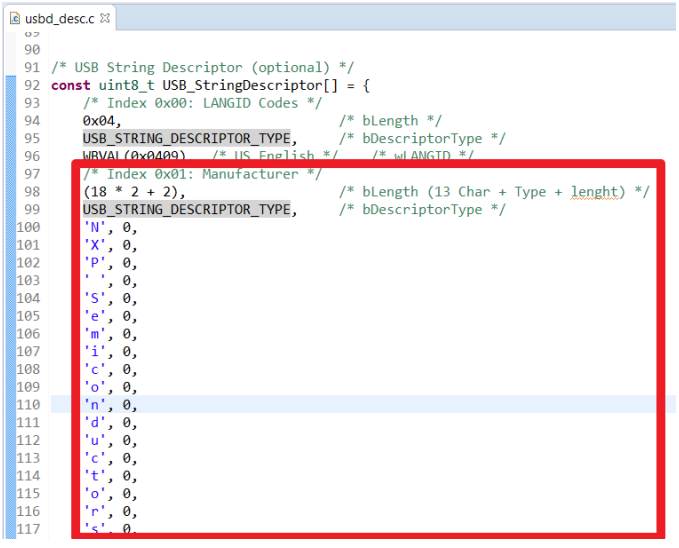


Fig 10. Index 0x01 of string descriptors

Index 0x02 contains the product name. For the example in Fig 11, it is LPC.

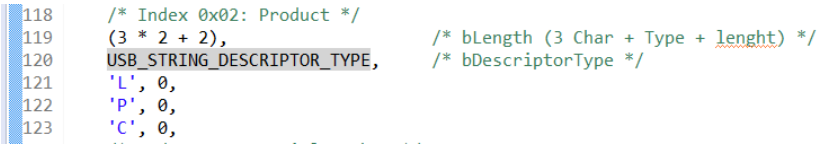


Fig 11. Index 0x02 of string descriptors

Index 0x03 contains the serial number of the device.

```
124     /* Index 0x03: Serial Number */
125     (13 * 2 + 2),               /* bLength (13 Char + Type + lenght) */
126     USB_STRING_DESCRIPTOR_TYPE,     /* bDescriptorType */
127     'A', 0,
128     'B', 0,
129     'C', 0,
130     'D', 0,
131     '1', 0,
132     '2', 0,
133     '3', 0,
134     '4', 0,
135     '5', 0,
136     '6', 0,
137     '7', 0,
138     '8', 0,
139     '9', 0,
```

Fig 12. Index 0x03 of string descriptors

Index 0x04 indicates the name of the interface 0. The test application uses DFU as interface 0.

```
140     /* Index 0x04: Interface 0, Alternate Setting 0 */
141     (3 * 2 + 2),                    /* bLength (3 Char + Type + lenght) */
142     USB_STRING_DESCRIPTOR_TYPE,     /* bDescriptorType */
143     'D', 0,
144     'F', 0,
145     'U', 0,
```

Fig 13. Index 0x04 of string descriptors

Index 0x05 indicates the name of interface 1. The test application uses a billboard class device ‘BILLBOARD’ as interface 1.

```
/* Index 0x05: Interface 1, Alternate Setting 0 */
(9 * 2 + 2),          /* bLength (9 Char + Type + lenght) */
USB_STRING_DESCRIPTOR_TYPE,   /* bDescriptorType */
'B', 0,
'I', 0,
'L', 0,
'L', 0,
'B', 0,
'O', 0,
'A', 0,
'R', 0,
'D', 0,
```

Fig 14. Index 0x05 of string descriptors

User can update all the string descriptors.

USB device descriptors, billboard class descriptors, and WCID handlers are present in the test application ‘**usbd_desc.c**’ file. The WCID handler enables the MCU to enumerate on Windows 8 and later without a driver. For Windows 7, the lpcdevice drivers are provided with the package in the *Drivers* folder.

6.2 Dual Image

In the application project (“app_blinky”), as the two images should have different flash allocation settings, there are two different *Run/Debug settings*. *app_blinky_Debug* is the settings for image1 and *app_blinky_Debug_SecondImage* is the settings for image2; see Fig 15.

The differences of the two configuration settings are the link descriptor content which are introduced in Section6.1

If the user creates the project, they are required to modify the flash assignment address values in “xxxx_memory.ld”. Refer Fig 16.

Note: Do not forget to change the version number of image2 in “app_blinky_Debug_SecondImage.ld”.

Note: If the user modifies any of the linker descriptor file, clean and refresh the project.

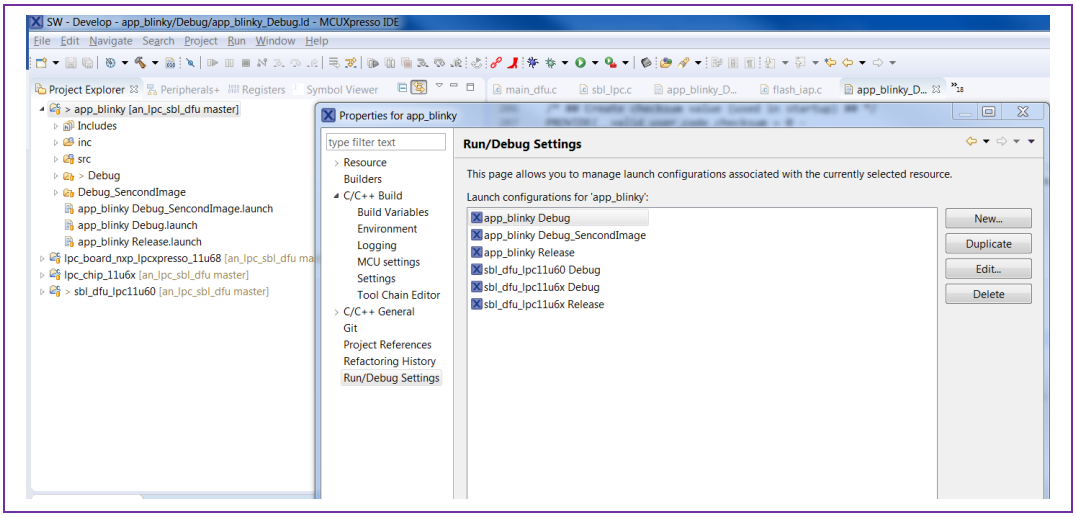


Fig 15. Dual Image Run/Debug Settings

```
MEMORY
{
  /* Define each memory region */
  MFlash256 (rx) : ORIGIN = 0x20000, LENGTH = 0x20000   /* 128K bytes (alias Flash) */
  Ram0_32 (rwx) : ORIGIN = 0x10000200, LENGTH = 0x1c00 /* 7K bytes (alias RAM) */
  Ram1_2 (rwx) : ORIGIN = 0x20000000, LENGTH = 0x800 /* 2K bytes (alias RAM2) */
  Ram2USB_2 (rwx) : ORIGIN = 0x20004000, LENGTH = 0x800 /* 2K bytes (alias RAM3) */
}

  /* Define a symbol for the top of each memory region */
  __base_MFlash256 = 0x20000  ; /* MFlash256 */
  __base_Flash = 0x20000; /* Flash */
  __top_MFlash256 = 0x20000 + 0x20000 ; /* 128K bytes */
  __top_Flash = 0x20000 + 0x20000 ; /* 128K bytes */
  __base_Ram0_32 = 0x10000200  ; /* Ram0_32 */
  __base_RAM = 0x10000200 ; /* RAM */
```

Fig 16. Linker descriptor file for Image2

7. Enabling DFU on LPC11U68 application projects

The section shows how to add in-field firmware update capability using Device Firmware Upgrade (DFU) class interface to an existing USB application. To enable DFU capability, a Secondary BootLoader (SBL) is implemented for LPC11U6x which performs the image integrity check during the booting and DFU method to update the application image.

7.1 Secondary BootLoader

The Secondary BootLoader (SBL) described and implemented in this application note provides a solution for the in-field update of USB application implemented on LPC11U6x. It utilizes the USB of the boot ROM and IAP API functionalities to program LPC11U6x flash.

The SBL occupies the first two sectors of user flash and routines to perform the following functionalities:

- Application image CRC checking: During the boot time, the SBL computes the CRC32 of the application image and checks it with the value stored in the image header. SBL executes the application image only if the CRC check passes. The check avoids booting partially programmed or corrupted images. Partial or corrupted images are formed due to the power failures during the firmware update.
- Vector redirection: LPC11U6x is an ARM Cortex-M0+ based microcontroller, which expects the vector table to be at address 0x0. The first sector of the flash resides at the address 0x0. The SBL implements a vector redirector to redirect the exception and interrupt handling to the application image.
- DFU class handler: USB org has defined DFU class specification as a firmware update method for USB applications. SBL handles all the USB control messages to do firmware update.
- DFU API: To eliminate the overhead of implementing DFU protocol in the application code, the SBL provides access to the DFU API. Applications can use this API to invoke DFU mode.
- SBL update: SBL implements a mechanism to update its firmware using re-invoke USB ISP method of ROM
- Dual Image: SBL manages two image areas. Both the images are active for SBL. SBL checks the header and version numbers. If both the information is valid for SBL, SBL selects the newest version image as the active application. User may refer the code in function *LPC_SBL_BootImageCheck()* located in sbl_lpc.c source file.

7.2 Image header

For the application to be a valid binary for a DFU SBL, a proprietary image header must be in the memory region 0x100 to 0x117.

If the user is familiar with Keil, the startup file in AN11732 should be referred.

In this application note, we use MCUXpresso and refer the .ld file.

Fig 8. shows a snapshot of the image header.

7.3 Program flow

The DFU can be enabled after the USB is normally initialized, making it easy to integrate into an existing project. To enable the DFU, a WCID handler should be registered to allow the automatic driver installation on Windows 8 and DFU initialization function should be executed. After receiving a detach command, the MCU should wait for a couple of milliseconds to ensure that the device is successfully disconnected from the host. Another function should be called to redirect the MCU back to the SBL; see Fig 17.

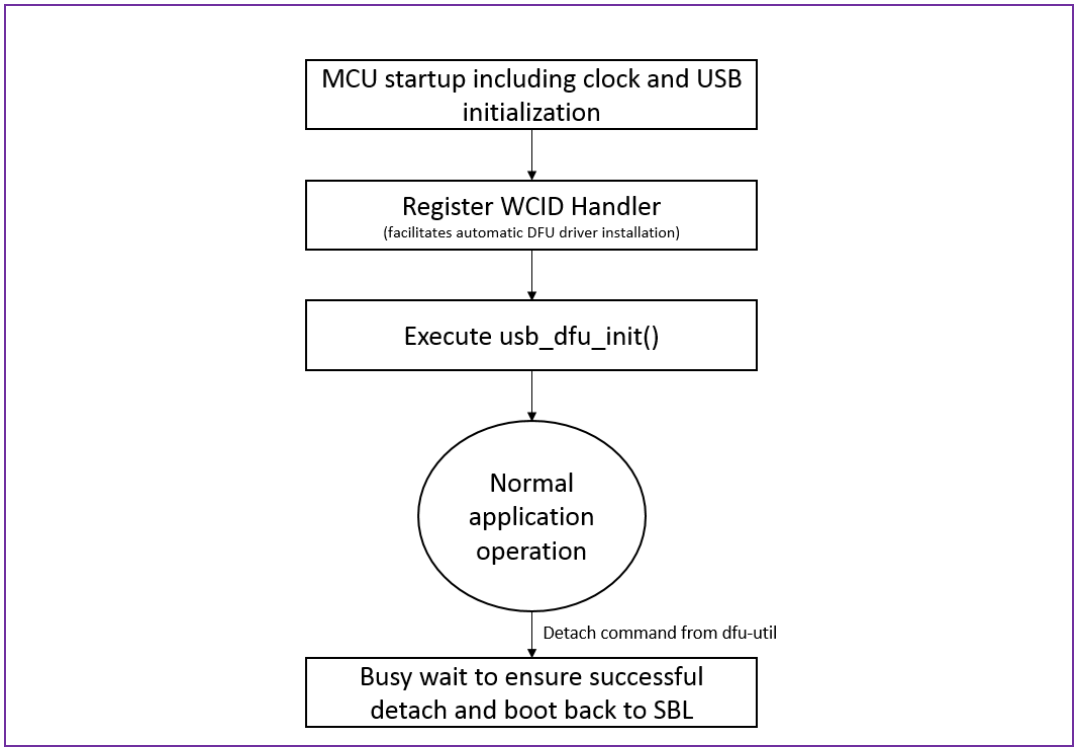
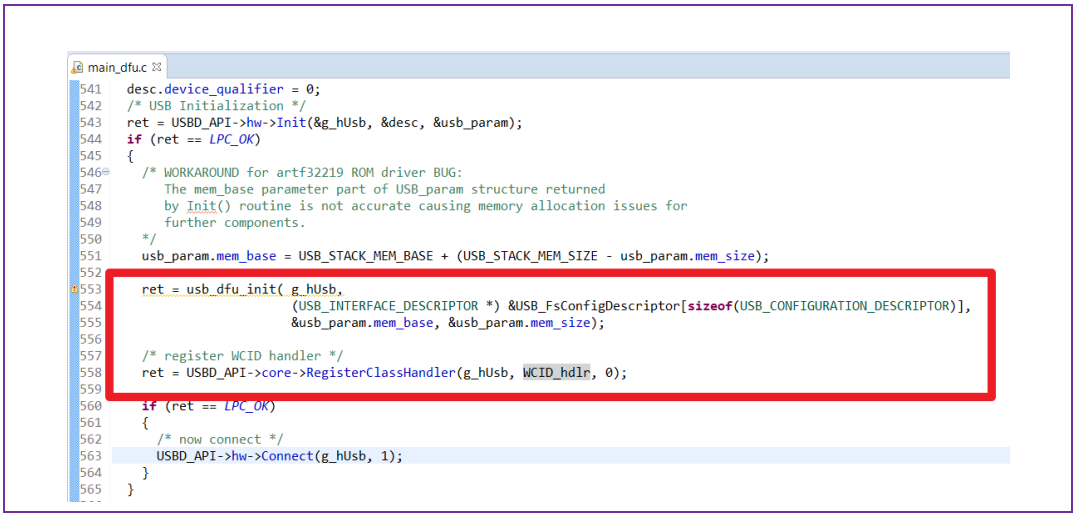


Fig 17. Program flow with DFU enabled

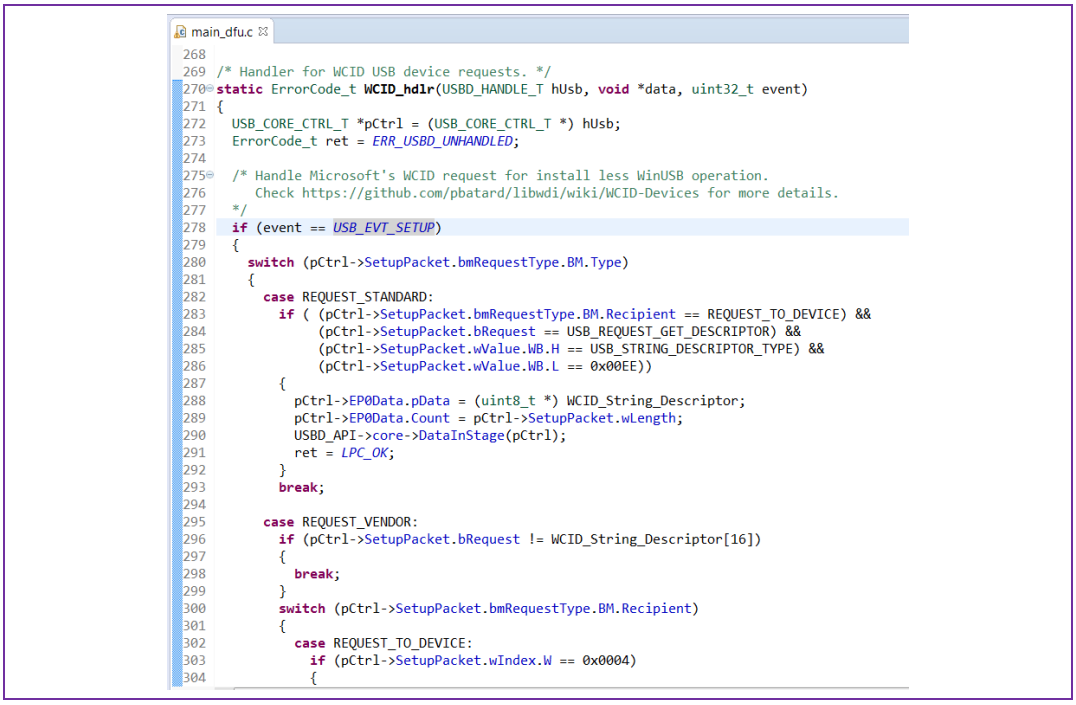
There are three source files from the test application project that have the required source code: main.c, usbd_desc.c, and usbd_dfu.c. In *main.c*, after initializing the USB, the WCID handler should be registered and the usb_dfu_init() function should be executed; see Fig 18.



```
main_dfu.c
541  desc.device_qualifier = 0;
542  /* USB Initialization */
543  ret = USBD_API->hw->Init(&g_hUsb, &desc, &usb_param);
544  if (ret == LPC_OK)
545  {
546    /* WORKAROUND for artf32219 ROM driver BUG:
547       The mem_base parameter part of USB_param structure returned
548       by Init() routine is not accurate causing memory allocation issues for
549       further components.
550    */
551    usb_param.mem_base = USB_STACK_MEM_BASE + (USB_STACK_MEM_SIZE - usb_param.mem_size);
552
553    ret = usb_dfu_init( g_hUsb,
554                        (USB_INTERFACE_DESCRIPTOR *) &USB_FsConfigDescriptor[sizeof(USB_CONFIGURATION_DESCRIPTOR)],
555                        &usb_param.mem_base, &usb_param.mem_size);
556
557    /* register WCID handler */
558    ret = USBD_API->core->RegisterClassHandler(g_hUsb, WCID_hdlr, 0);
559
560    if (ret == LPC_OK)
561    {
562      /* now connect */
563      USBD_API->hw->Connect(g_hUsb, 1);
564    }
565  }
```

Fig 18. WCID handler and usb_dfu_init( )

Add the definition of WCID event handler; see Fig 19.



```
main_dfu.c
268
269 /* Handler for WCID USB device requests. */
270 static ErrorCode_t WCID_hdlr(USBD_HANDLE_T hUsb, void *data, uint32_t event)
271 {
272   USB_CORE_CTRL_T *pCtrl = (USB_CORE_CTRL_T *) hUsb;
273   ErrorCode_t ret = ERR_USBD_UNHANDLED;
274
275   /* Handle Microsoft's WCID request for install less WinUSB operation.
276      Check https://github.com/pbatard/libwdi/wiki/WCID-Devices for more details.
277   */
278   if (event == USB_EVT_SETUP)
279   {
280     switch (pCtrl->SetupPacket.bmRequestType.BM.Type)
281     {
282       case REQUEST_STANDARD:
283         if ( (pCtrl->SetupPacket.bmRequestType.BM.Recipient == REQUEST_TO_DEVICE) &&
284              (pCtrl->SetupPacket.bRequest == USB_REQUEST_GET_DESCRIPTOR) &&
285              (pCtrl->SetupPacket.wValue.WB.H == USB_STRING_DESCRIPTOR_TYPE) &&
286              (pCtrl->SetupPacket.wValue.WB.L == 0x00EE))
287         {
288           pCtrl->EP0Data.pData = (uint8_t *) WCID_String_Descriptor;
289           pCtrl->EP0Data.Count = pCtrl->SetupPacket.wLength;
290           USBD_API->core->DataInStage(pCtrl);
291           ret = LPC_OK;
292         }
293         break;
294
295       case REQUEST_VENDOR:
296         if (pCtrl->SetupPacket.bRequest != WCID_String_Descriptor[16])
297         {
298           break;
299         }
300         switch (pCtrl->SetupPacket.bmRequestType.BM.Recipient)
301         {
302           case REQUEST_TO_DEVICE:
303             if (pCtrl->SetupPacket.wIndex.W == 0x0004)
304             {
```

Fig 19. WCID event handler

To handle a detach, a 'dfu_detach_sig' variable is used as a flag. It is declared as a global variable in 'usbd_dfu.c' file. It should also be declared as an external variable in the *app_usbd_cfg.h* file, along with other DFU related variables and functions; see Fig 20.

app_usbd_cfg.h

```
 Since these are the values used for compiling USB stack.
 */
#define USB_MAX_IF_NUM          8       /*!< Max interface number used for building USBDL_Lib. DON'T CHANGE. */
#define USB_MAX_EP_NUM          5       /*!< Max number of EP used for building USBD ROM. DON'T CHANGE. */
#define USB_MAX_PACKET0         64      /*!< Max EP0 packet size used for building USBD ROM. DON'T CHANGE. */
#define USB_FS_MAX_BULK_PACKET  64      /*!< MAXP for FS bulk EPs used for building USBD ROM. DON'T CHANGE. */
#define USB_HS_MAX_BULK_PACKET  512     /*!< MAXP for HS bulk EPs used for building USBD ROM. DON'T CHANGE. */
#define USB_DFU_XFER_SIZE       USB_MAX_PACKET0

#define MAX_USB_DESC_SZ 255

/* USB descriptor arrays defined *_desc.c file */
extern USB_DEVICE_DESCRIPTOR USB_DeviceDescriptor;
extern uint8_t USB_HsConfigDescriptor[];
extern const uint8_t USB_FsConfigDescriptor[];
extern const uint8_t USB_StringDescriptor[];
extern const uint8_t USB_DeviceQualifier[];
extern  uint8_t WCID_String_Descriptor[];
extern const uint8_t WCID_CompatID_Descriptor[];
extern const uint8_t WCID_ExtProp_Descriptor[];

extern ErrorCode_t AppDFU_Ep0_Hdlr(USBD_HANDLE_T hUsb, void* data, uint32_t event);
extern void DFU_ProcessData(void);
extern void decode_data(uint32_t *v, int n, const uint32_t key[4]);
extern void ReinvokeISP(void);
```

Fig 20. Defining all the DFU variables as external

7.4 Code enters Common_Handler( )

The SBL code executes the Common_Handler( ) when there is a system fault. In case of a system fault, the user should check if the application code is valid.

```
/* Function to common handler in the vector table */
void Common_Handler(void)
{
  typedef  void (*pFunction)(void);

  pFunction Jump_To_Application;
  volatile uint32_t JumpAddress;
  volatile uint32_t i;

  /* Getting interrupt status from Interrupt Program and Status register */
  i = __get_IPSR();
  /* Based on the value of 'i' the corresponding vector is redirected to Application  Image vector table */
  JumpAddress = *(__IO uint32_t*) (gLPCImageOffset + (4*i));
  Jump_To_Application = (pFunction) JumpAddress;
  Jump_To_Application();
  /* Should not come back here */
}
```

Fig 21. Common handler

8. Creating factory images and DFU update capable images

The image creator is a command line tool that can be used for the **Windows platform**. The tool has preloaded DFU SBL and inserts the intended application binaries at the beginning of the SBL when using the factory command. It is also used to calculate and add a CRC checksum into the application binary to make it a DFU update capable application image.

8.1 Image creator tool

The image creator tool is available in the 'image-creator-tool' folder. Open the command prompt and navigate to the directory where the executable file is located; see Fig 22.

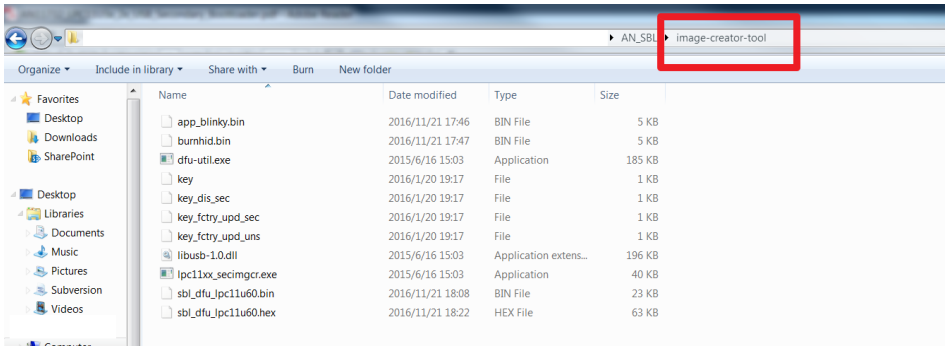


Fig 22. Image creator tool file location

8.1.1 Inserting a CRC checksum in the application image

To make an application image acceptable to the DFU SBL, add a CRC checksum to the application binary. See Section 9.2 for more information on how to perform DFU updates with a DFU capable application image.

The syntax to invoke the tool to create an output binary file with image header from an input binary file is:

lpc11xx_secimgcr.exe <input file name.bin> <output file name.dfu>

The syntax in Fig 23. generates the CRC for the input application binary file 'app_blinky.bin' and creates an output file 'burnhid.dfu'.

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

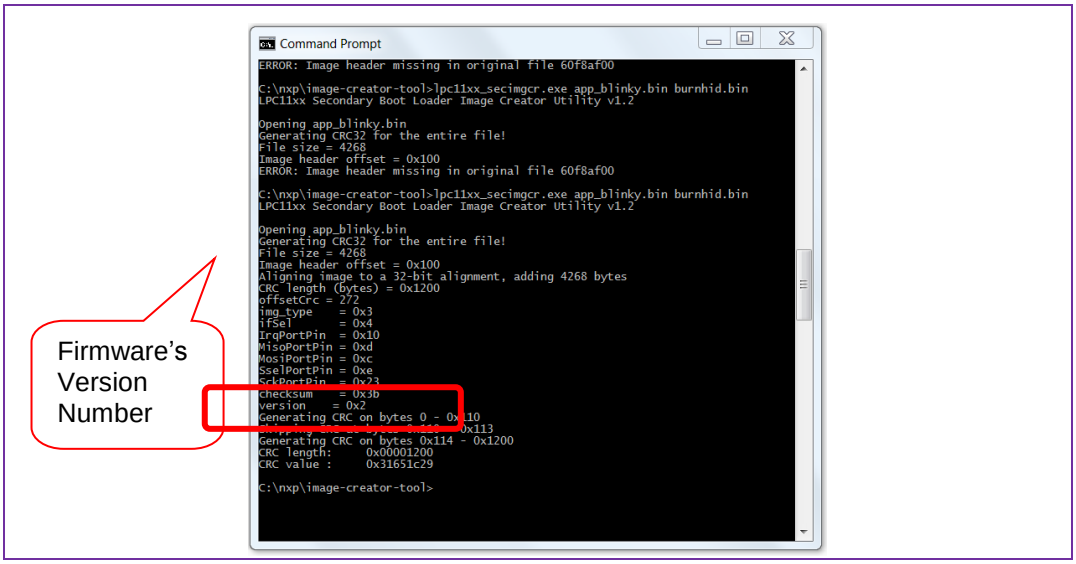


Fig 23. Image with CRC header

9. Downloading files using DFU-UTIL

This application note provides a pre-compiled dfu-util executable file on Windows, OS X, and Ubuntu. It also provides the script files to automate the DFU update process.

For more information on dfu-util and commands, visit: http://dfu-util.sourceforge.net.

Note: The dfu-util tools included in the package is tested and verified with DFU SBL. The dfu-util tools provided in the package contains a patch from NXP Semiconductors.

9.1 Dfu-util tool platform dependencies

The dfu-util tool uses libusb to interface DFU USB devices on the host machine. For certain platforms, it requires some extra steps for the dfu-util tool to function correctly.

9.1.1 Windows 7 requirements

Windows 7 does not natively support the DFU device class and requires a driver installation before the dfu-util can be used. Note that for Windows 8 and later, this step is the necessary support added by Microsoft.

When executing the test application, Windows 7 device manager should show an unknown LPC device; see Fig 24.

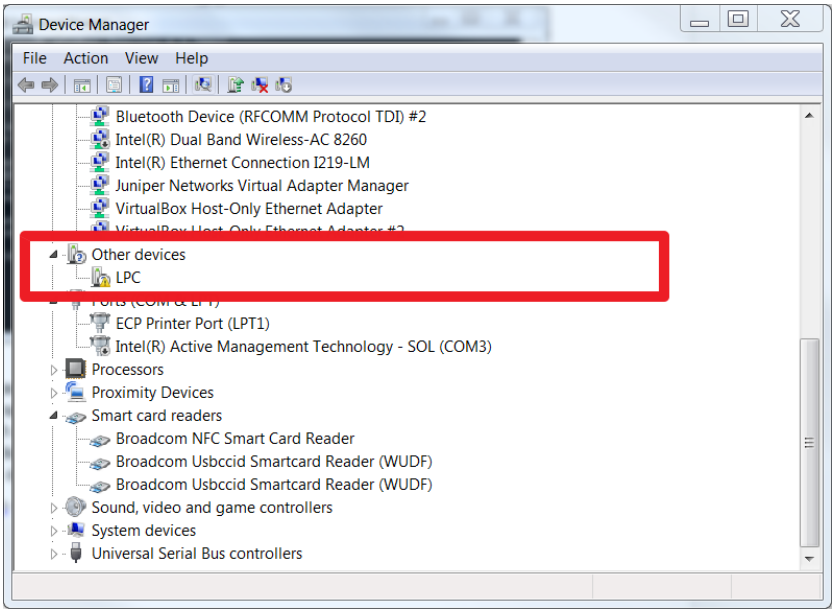


Fig 24. Device manager

Right click on the LPC device and click update driver software. Choose the option *browse my computer* for driver software. Type the path of the location where the folder *lpcdevice* is present and click next. If the drivers are successfully installed, a DFU device will appear under LpcDevice field in the device manager.

Note: AN package includes the usb driver in file name *drivers*; see Fig 25.

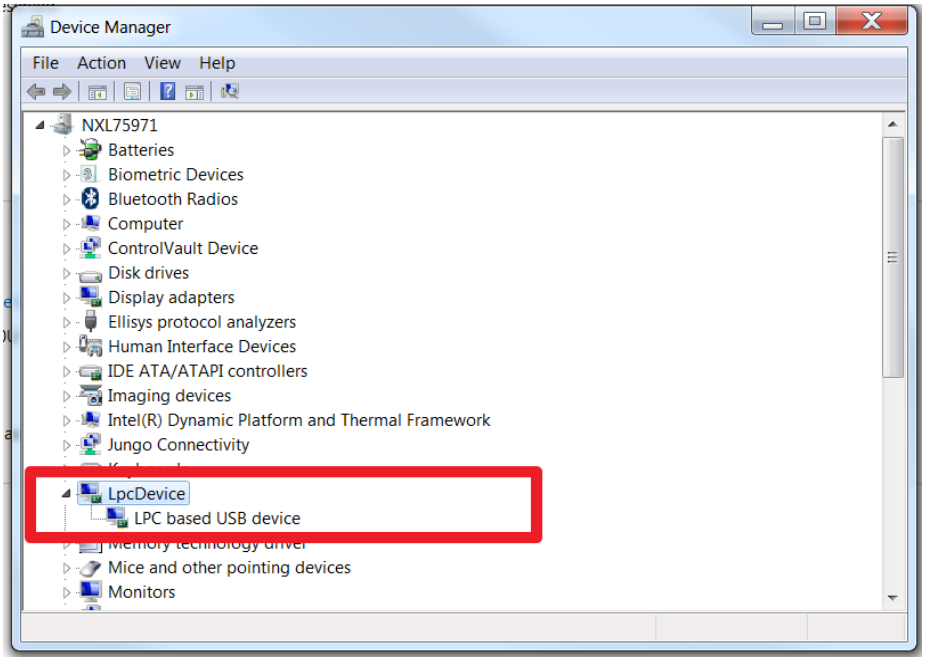


Fig 25. DFU Device

9.1.2 OS X requirements

The libusb library is not provided in this package for OS X. It should be installed manually. Open terminal and copy-and-paste the following command. Follow the on-screen installation directions:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Next, install libusb using brew with the following command:

```
brew install libusb
```

With libusb installed, the dfu-util tool can now be used.

9.1.3 Ubuntu requirements

The libusb library is not provided in this package for Ubuntu and must be installed manually. Open terminal and copy-and-paste the following commands. Follow the on-screen installation directions:

sudo apt-get install libusb-1.0-0:i386

9.2 Performing DFU updates manually

To manually update the FW through dfu-util, open command prompt or terminal and navigate to the directory where the dfu-util executable file is located. All the relevant dfu-util commands are introduced in this section.

To detect if a valid DFU device is present, type the command:

dfu-util.exe -l

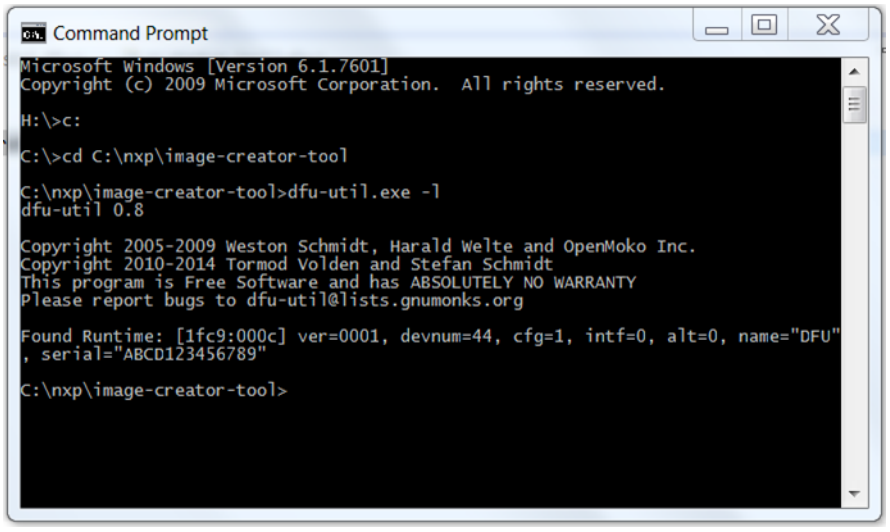


Fig 26. List of dfu devices

After a detach command is sent, the MCU is in the SBL context. Firmware updates and DFU commands can now be sent via dfu-util. For firmware updates, the process to update with an encrypted binary or unencrypted binary is the same. If the secure mode is enabled, the SBL handles the decryption of the encrypted binary. Send the new application image or DFU commands with the following command:

dfu-util.exe –t 64 –D <DFU file>

Fig 27 shows the expected output when a new application image with a valid CRC checksum is sent.

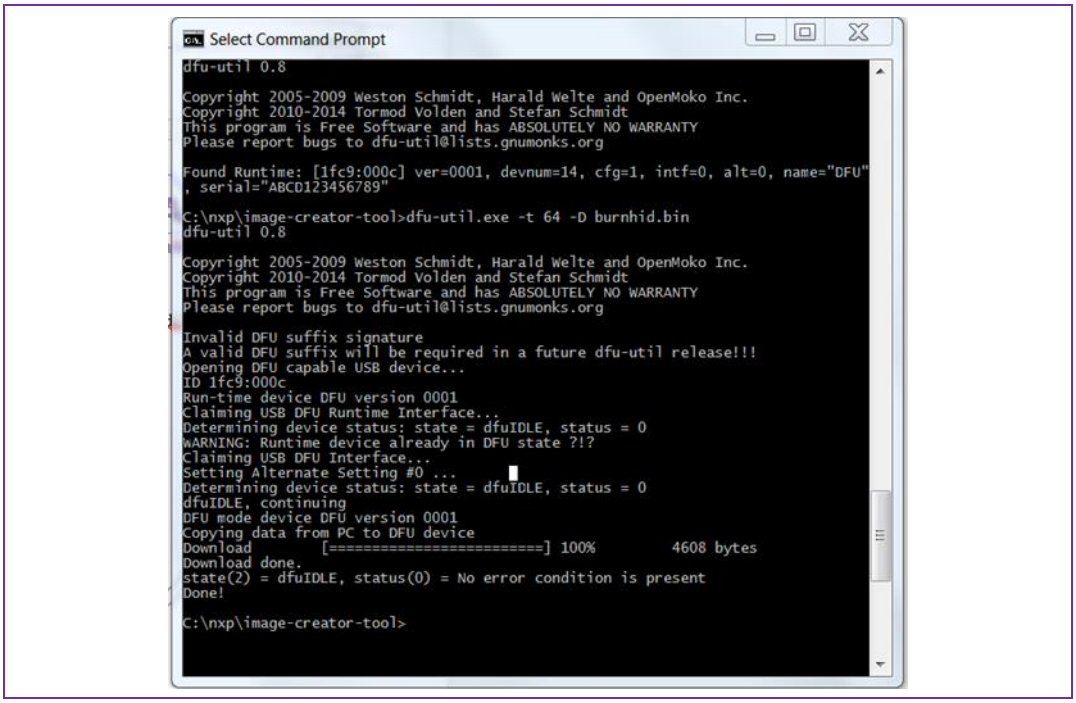


Fig 27. Field firmware update

In addition to sending a new FW, the keys that are generated by the image creator tool can be sent as DFU commands. It is the easiest way to perform factory image updates to update the SBL or to enable secure mode in the field.

10. Using SBL on different OS platforms

If the chip or board is not programmed with SBL binary, we may use the FlashMagic tool to program the SBL binary into LPC11U68 internal flash.

Program “sbl_dfu_lpc11u60.hex” by FlashMagic is shown in Fig 28

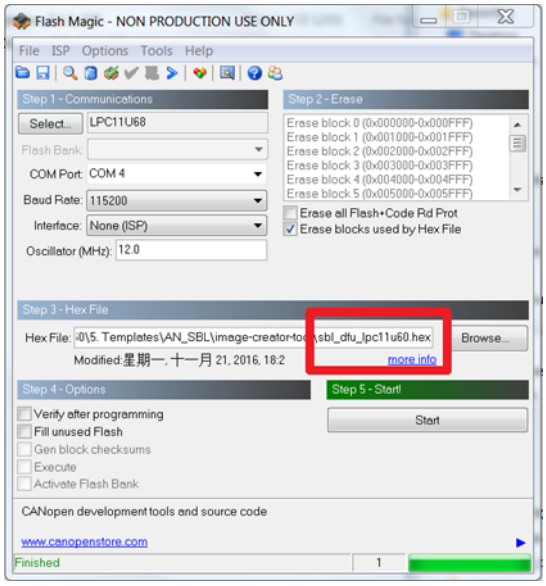


Fig 28. FlashMagic configuration

Enter the chip, COM port, baud rate, interface, oscillator and hex file path details and click on the start button to program the SBL program into LPC11U68; see Fig 28. on Windows Platform

Step 1: Copy necessary binary files (hid.bin) into the *image-creator-tool*; see Fig 29.

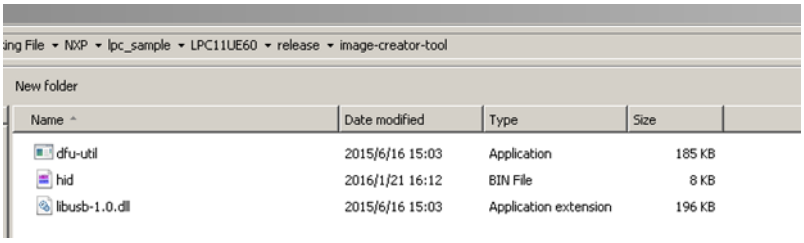


Fig 29. Copy target binary file in to the image-creator-tool

Step 2: Type lpc11xx_secimgcr.exe hid.bin burnhid.bin in the command shell.

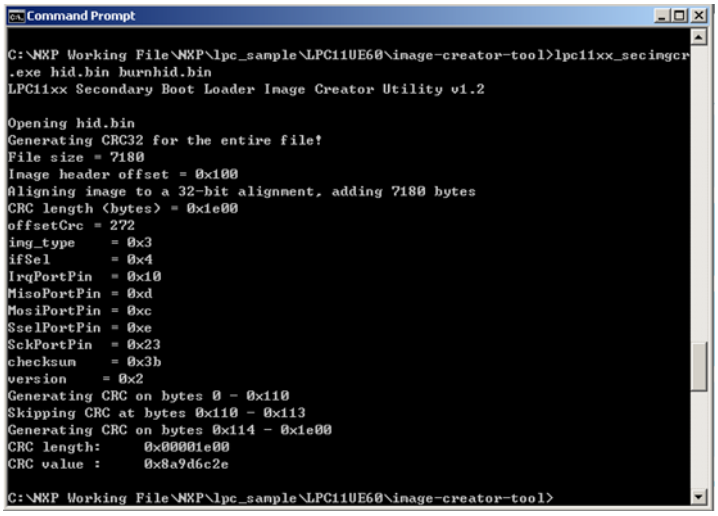


Fig 30. Generate the programming binary file

Step 3: Connect lpc11u60 usb port with PC, then press SW2(WAKE) when reset to enter DFU mode.

Input '*dfu-util.exe -l*' to discovery device.

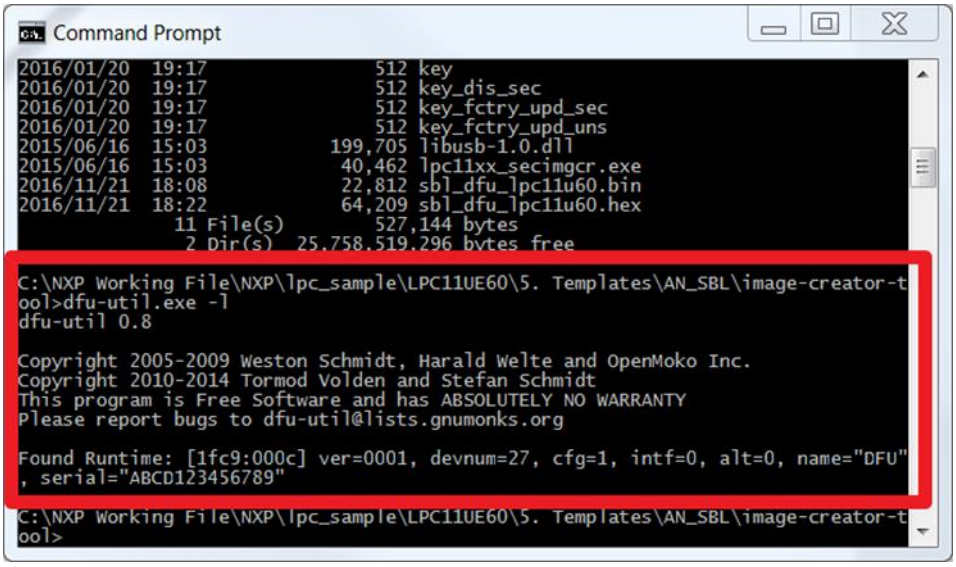


Fig 31. Check the DFU device is connect with PC

Step 4: If discovery device, input '*dfu-util.exe –t 64 –D burnhid.bin*':

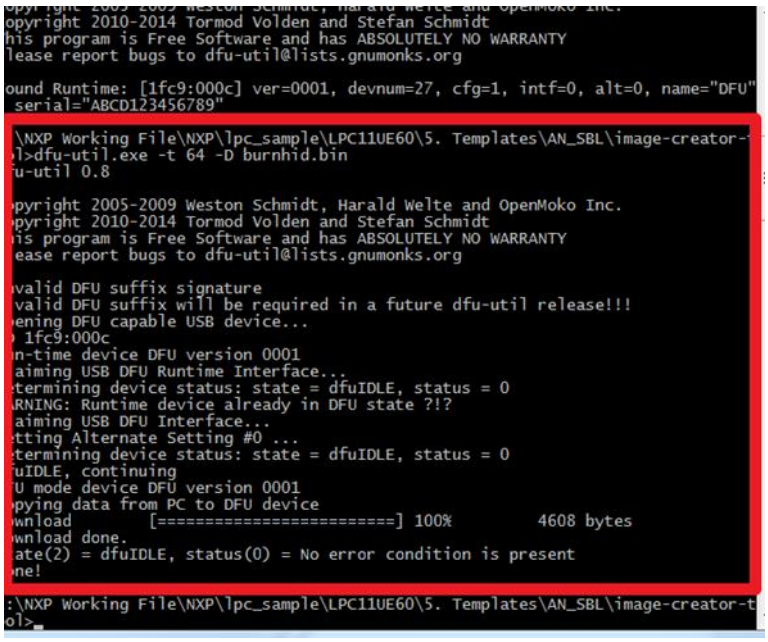


Fig 32. Programming binary by DFU command

After programming is finished, the blue and red LEDs starts blinking together.

10.1 On MAC OS (OSX) Platform

Step 1: If the internet needs proxy, copy the following commands in to the shell:

Export http_proxy=xxx.xxx.com:0000

Export https_proxy=xxx.xxx.com:0000

Or, set the proxy in network.

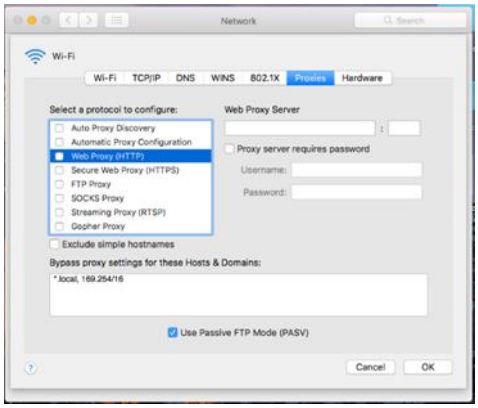


Fig 33. Proxy setting in network for OSX

Step 2: Install brew first, and copy the following command in to the shell:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Fig 34. Install brew on OSX

For more details, visit the link: http://brew.sh/

Step 3: Install libusb by input "*brew install libusb*" in shell

brew install libusb

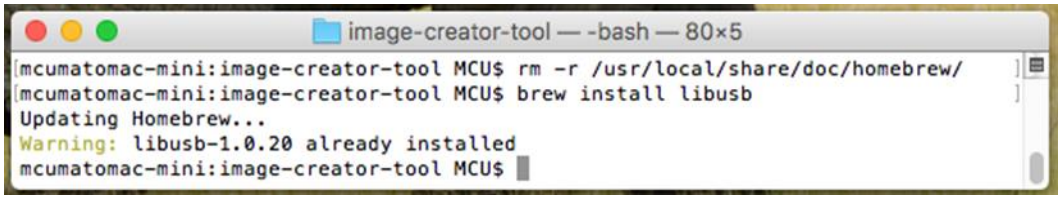


Fig 35. Install libusb on OSX

Step 4: Enter the image-creator-tool path and input

./dfu-util_mac –d 1fc9:000c –t 64 –D burnhid.bin

For old dfu-util version try the following command (if *-d* is not supported):

./dfu-util_mac –device 1fc9:000c –t 64 –D burnhid.bin

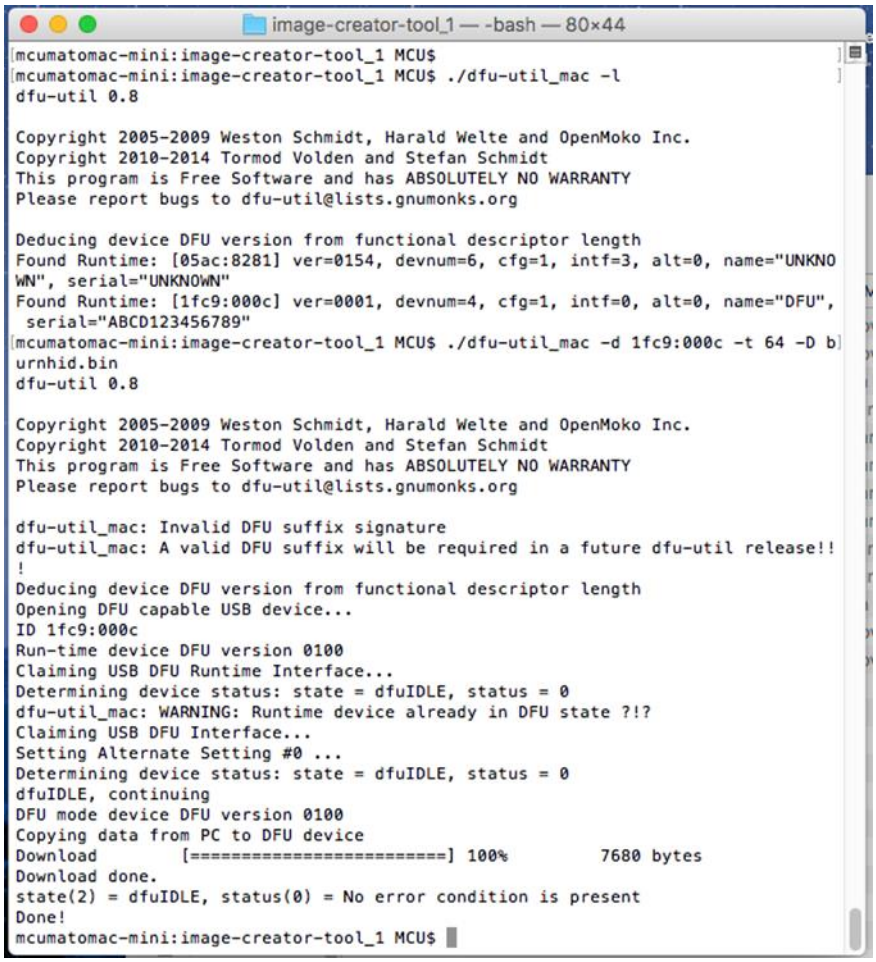


Fig 36. Update program from DFU on OSX

11. References

[1] UM10850 LPC54100 User manual

[2] AN11732 LPC11U3x/2x USB Secondary Bootloader

12. Legal information

12.1 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.

12.2 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.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

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 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.

12.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

12.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

12.5 Trademarks

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

<Name> — is a trademark of NXP Semiconductors N.V.

13. Index

No index entries found.

14. List of figures

15. List of tables

No table of figures entries found.

16. Contents

Date of release: 4 September 2017
Document identifier: AN12037