AN11732

LPC11U3x/2x USB Secondary Bootloader

Rev. 1.2 — 14 November 2016 Application note

Document information

Info	Content
Keywords	LPC11U3x,LPC11U2x, secondary bootloader, image creator tool, DFU utility, firmware update, field update
Abstract	This application note introduces the image creator tool and DFU utility programs to help facilitate the use of a DFU SBL with an LPC11U3x/2x application to enable firmware updates in the field.

Revision history

Rev	Date	Description
1.2	20161114	Updated the dfu-util tool
1.1	20160404	The application note was restructured and rewritten, dfu-util support for OS X and Linux was added in addition to DFU update scripts for Windows, OS X and Linux.
1.0	20150828	Initial revision.

Contact information

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

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

1. Introduction

The LPC11U3x/2x provides the user a convenient way to update the flash content in the field for bug fixes or product updates. This 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. This can be done when the part resides on the end-user board.
- IAP: In-Application programming 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 a 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 microcontroller's boot ROM block and is executed on power-up and resets. After the boot ROM's execution, 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 provided by NXP to easily incorporate a Device Firmware Update (DFU) SBL with any given LPC11U2x/LPC11U3x application binary.

2. Package contents

The extracted contents of the package should look like the following:

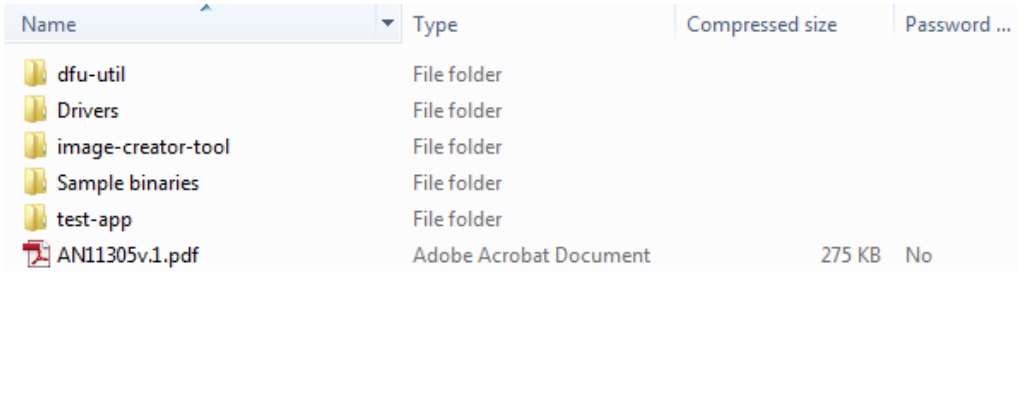

Fig 1. Package contents

A brief description for each of the folder is explained here:

1. **dfu-util** – This folder contains the dfu-util tool with 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 tool.
2. **Drivers** – This folder contains the "lpcdevice" drivers (including USB descriptors) that must be installed in order for a Windows 7 machine to successfully detect the DFU mode used on the LPC11U3x/2x.

3. **image-creator-tool** – This folder contains the lpc11xx_secimgcr.exe program that 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. **Sample binaries** – This folder contains sample binaries of all of the files that can be generated with the image creator tool.
    a. tst_11uxx_dfu.bin – Sample application binary that was used to create all the sample firmware images in this folder.
    b. secure_fac_img.bin – Secure factory image, which is a merged image of the DFU SBL with CRP2 activated + application binary that has been encrypted with the 'key' file in this folder. CRC is generated and inserted into application binary. Key file is placed in SBL region of memory so it can decrypt the application code.
    c. plain_fac_img.bin – Plain factory image, which is a merged image of the DFU SBL with no CRP enabled + unencrypted application binary. CRC is generated and inserted into application binary.
    d. test_app_crc.dfu – Unencrypted application binary with CRC generated and inserted.
    e. encrypted_app_img.dfu – Application binary with CRC generated and inserted, encrypted with the 'key' file in this folder
    f. key – Key file, which is used to encrypt and decrypt the application binary. The 'key' file used to encrypt any given application image must also be used by the SBL to decrypt.
    g. key_dis_sec – This file is used by the dfu-util.exe. Sending this file via dfu-util will turn off CRP2 of a secure factory image making the factory image unsecure. This file will be rejected by unsecure factory images.
    h. key_fctry_upd_sec – This file is used by the dfu-util.exe. Sending this file via dfu-util to a secure factory image will re-invoke USB ISP mode so that the MCU will enumerate itself as a MSC (mass storage class) device. This file will be rejected by unsecure factory images.
    i. key_fctry_upd_uns – This file is used by the dfu-util.exe. Sending this file via dfu-util to an unsecure factory image will re-invoke USB ISP mode so that the MCU will enumerate itself as a MSC device.
5. **test-app** – This folder contains a Keil project for the test application.

3. Hardware environment

The sample test application was tested using Keil MDK IDE v.5.14.0.0 and the NGX LPC11U24/301 board (#OM13033) and LPC11U37 LPCXpresso board (#OM13074).

For more information on these boards, visit the following links:

http://www.nxp.com/board/OM13033.html

http://www.nxp.com/board/OM13074.html

The DFU SBL has been tested and deployed in real solutions such as NXP's USB Type-C 1 – 3 Multiport Adapter. The sample test application has not been tested on LPC11U1x.

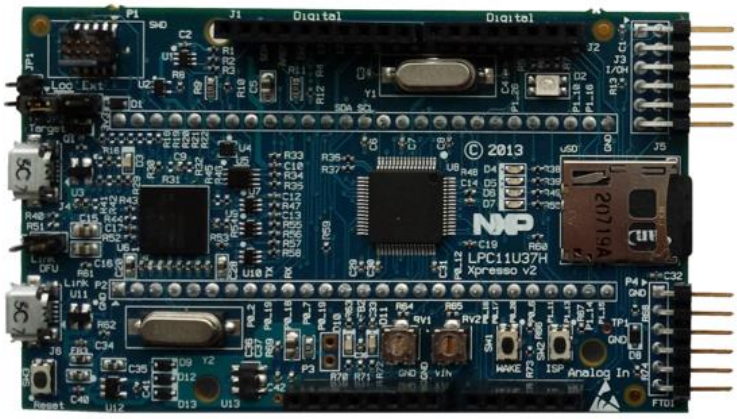

Fig 2. LPCXpresso board for LPC11U37H

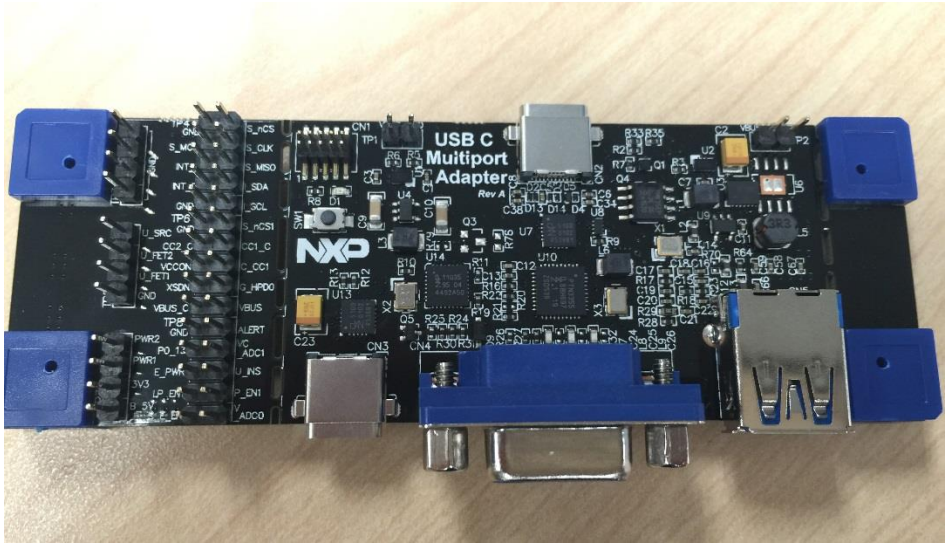

Fig 3. NXP USB Type-C 1 - 3 multiport adapter

4. Development flow

To enable DFU updates for a given application:

1) Modify the existing application to support the DFU SBL. This 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 from the application to SBL. See Section 5 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 6 for 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 7 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. See Section 8 for more information.

After all of these steps have been completed, the field updates with the DFU SBL follow the flow chart shown in Fig 4.

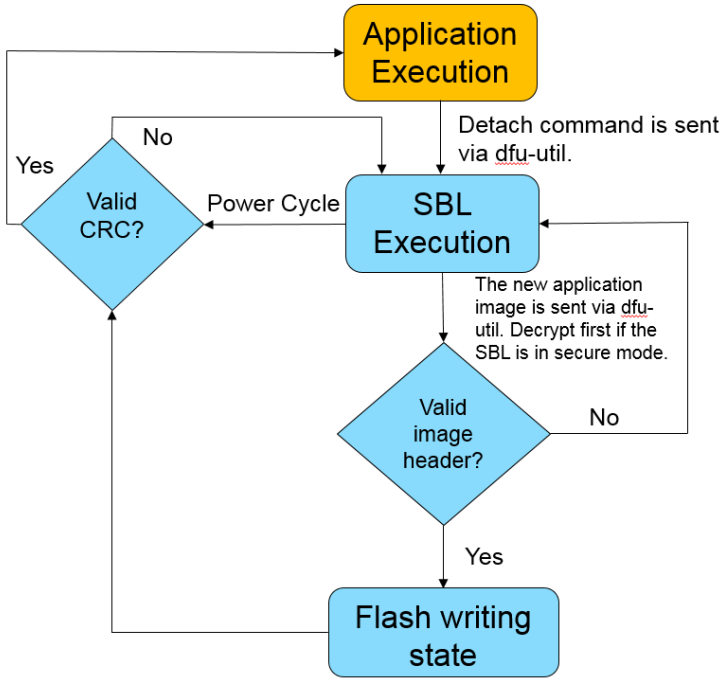

Fig 4. Field update flow

4.1 Programming flow

With a DFU SBL capable application binary, the image creator tool should be used to create factory image and application binaries with valid CRC checksums. While both factory images and DFU capable application codes are essentially binaries, factory images will end with a .bin extension while DFU capable files end in a .dfu. This differentiation is not needed when using either the image creator tool or dfu-util tool but helps minimize confusion.

Fig 5 shows an example of the process of creating a secure factory image.

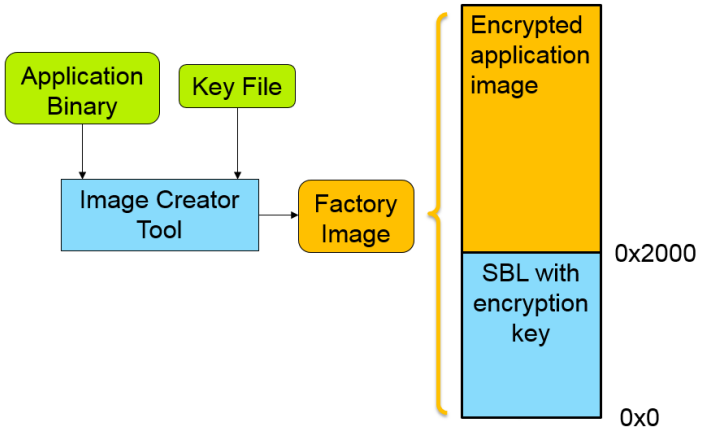

Fig 5. Creating a secure factory image

Fig 6, Fig 7, and Fig 8 illustrate the programming flow at different stages:

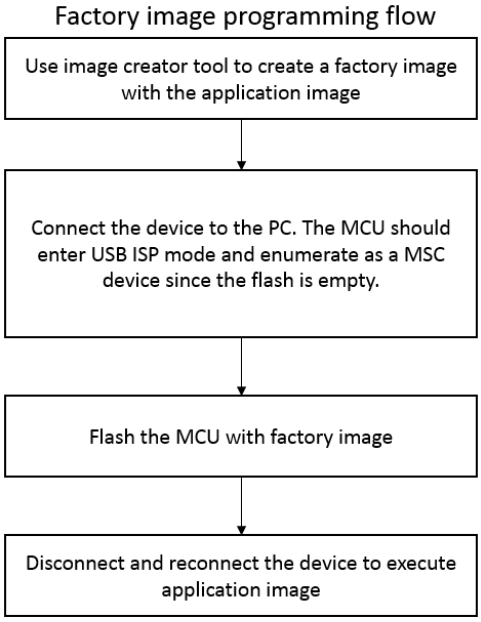

Fig 6. Factory programming flow

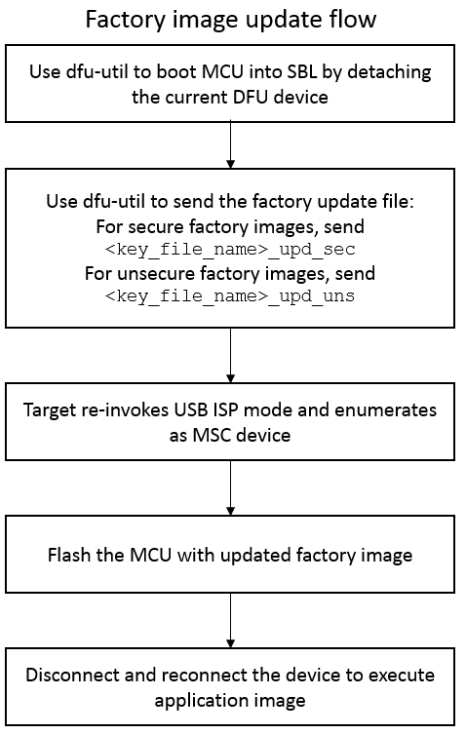

Fig 7. Field factory image update flow

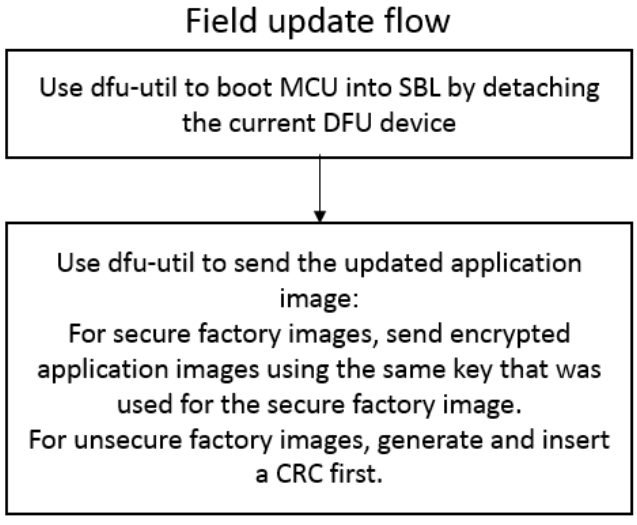

Fig 8. Field firmware update flow

5. Test application

A simple blinky project that has all the settings configured to enable DFU updates is provided. It toggles PIO1_26, which is LED D2 on the LPC11U37 LPCXpresso board.

To see the necessary code that needs to be added to an existing USB project, please see Section 6.

The DFU SBL occupies the first two sectors of the user flash which means the test application is present at an offset 0x2000. Fig 9 shows the Keil Options for the target window. The SBL uses 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.

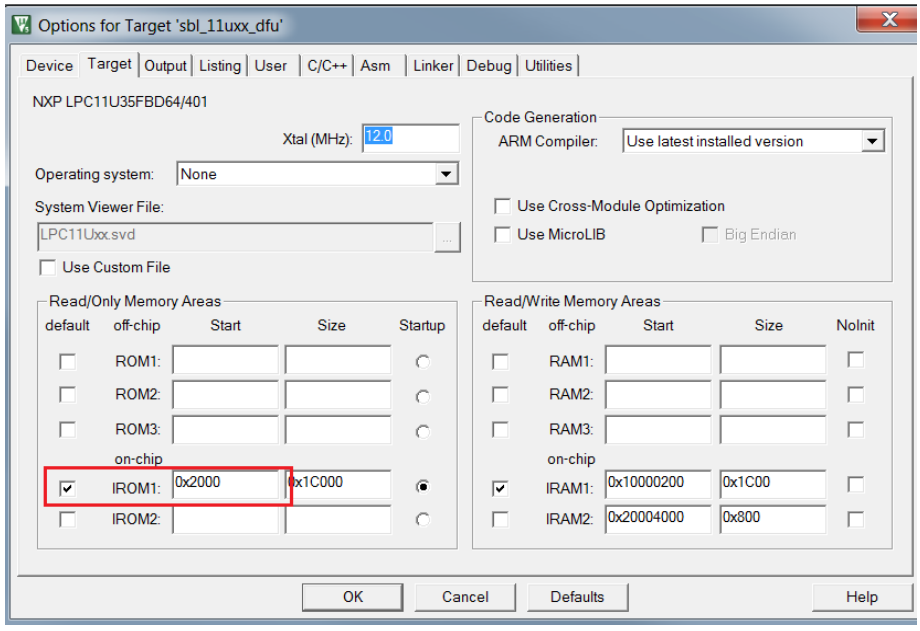

Fig 9. Target settings

Fig 10 shows the *.ini file of the test application.

```
FUNC void Setup (void) {
  SP = _RDWORD(0x00002000);
  PC = _RDWORD(0x00002004);
  _WDWORD(0x40048000, 0x00000001);    // MEMMAP = 1
}

LOAD %L INCREMENTAL                  // Download

Setup();                                    // Setup for Running
g, main
```

Fig 10. *.ini file of test application

Build the project by clicking Project->Build. After the build is completed, a binary file ‘tst_11uxx_dfu.bin’ is generated in the ‘keil output’ folder.

See Fig 11 for the linker script generated by the test application.

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x00002000 0x0001C000  {    ; load region size_region
  ER_IROM1 0x00002000 0x0001C000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x10000200 0x00001C00  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

Fig 11. Linker script test application

5.1 Image header

The image header is stored in the memory region 0x100 – 0x117. This image header is used to indicate that it is a DFU capable application binary. This header contains necessary information for the SBL, such as the pin configuration table, CRC length, and CRC checksum. When the DFU SBL is re-invoked from the test application, the test application sends a pin configuration table ‘PINONLYCFGTABLEFLASH’ and USB descriptors of the test application to the SBL. See the SBL invoke function ‘bootSecondaryLoader ()’ in the ‘usbd_dfu.c’ file. These parameters can be updated by the user.

Fig 12 shows the image header located in the Keil startup file.

```
IMG_HEADER      DCD     0xFEEDA5A5              ; Image header marker
              ; DCB       0                       ; img_type:         See img_type values above
            ;   DCB     0, 0, 0                 ; Reserved
                ;DCD       0xFFFFFFFF              ; Reserved

                EXPORT PINONLYCFGTABLEFLASH
PINONLYCFGTABLEFLASH
                DCB     0       ; img_type: See img_type values above
                DCB     4       ; ifSel:    Interface selection for host (0=AUTODETECT, 1=I2C0, 2=I2C1, 3=I2C2, 4=S
                DCB     ((0 << 5) + 8); hostIrqPortPin: Host IRQ port (bits 7:5) and pins (bits 4:0)
                DCB     ((0 << 5) + 13); hostMisoPortPin: SPI MISO port (bits 7:5) and pins (bits 4:0)
                DCB     ((0 << 5) + 12); hostMosiPortPin: SPI MOSI port (bits 7:5) and pins (bits 4:0)
                DCB     ((0 << 5) + 14); hostSselPortPin: SPI SEL port (bits 7:5) and pins (bits 4:0)
                DCB     ((1 << 5) + 3); hostSckPortPin: SPI SCK port (bits 7:5) and pins (bits 4:0)
                DCB     0^4^((0 << 5) + 8)^((0 << 5) + 13)^((0 << 5) + 12)^((0 << 5) + 14)^((1 << 5) + 3)
                EXPORT CRC32_LEN
                EXPORT CRC32_VAL
CRC32_LEN       DCD     0x00000e00              ; Length for CRC32 check starting at offset 0, in 32-bit words
CRC32_VAL       DCD     0xdd99a277            ; CRC32 value
FW_VERSION      DCD     0x00000001              ; Firmware version number
```

Fig 12. Image header

When SBL is invoked, the USB descriptors are also passed as another parameter from the test application to the SBL. The USB Vendor ID (VID) passed is 0x1FC9 and USB Product ID (PID) is 0x5002. See 'enter_DFU_SL()' in 'usbd_dfu.c' file. 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 language id.

```
/* Index 0x00: LANGID Codes */
0x04,              /* bLength */
USB_STRING_DESCRIPTOR_TYPE,    /* bDescriptorType */
WBVAL(0x0409),  /* US English */     /* wLANGID */
```

Fig 13. Index 0x00 of string descriptors

Index 0x01 indicates Manufacturer details.

```
/* Index 0x01: Manufacturer */
(18 * 2 + 2),         /* bLength (13 Char + Type + lenght) */
USB_STRING_DESCRIPTOR_TYPE,    /* bDescriptorType */
'N', 0,
'X', 0,
'P', 0,
' ', 0,
'S', 0,
'e', 0,
'm', 0,
'i', 0,
'c', 0,
'o', 0,
'n', 0,
'd', 0,
'u', 0,
'c', 0,
't', 0,
'o', 0,
'r', 0,
's', 0,
```

Fig 14. Index 0x01 of string descriptors

Index 0x02 contains the product name. Here it is 'LPK'.

```
/* Index 0x02: Product */
(3 * 2 + 2),          /* bLength (3 Char + Type + lenght) */
USB_STRING_DESCRIPTOR_TYPE,    /* bDescriptorType */
'L', 0,
'P', 0,
'K', 0,
```

Fig 15. Index 0x02 of string descriptors

Index 0x03 contains the serial number of the device.

```
/* Index 0x03: Serial Number */
(13 * 2 + 2),         /* bLength (13 Char + Type + lenght) */
USB_STRING_DESCRIPTOR_TYPE,   /* bDescriptorType */
'A', 0,
'B', 0,
'C', 0,
'D', 0,
'1', 0,
'2', 0,
'3', 0,
'4', 0,
'5', 0,
'6', 0,
'7', 0,
'8', 0,
'9', 0,
```

Fig 16. Index 0x03 of string descriptors

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

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

Fig 17. 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 18. Index 0x05 of string descriptors

The above string descriptors can be updated by the user.

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, lpcdevice drivers have been provided with the package in the 'Drivers' folder.

6. Enabling DFU on LPC11U2x/3x application projects

This 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 Boot Loader (SBL) is implemented for LPC11U3x/2x which does image integrity check during booting and DFU method to update the application image.

6.1 Secondary Boot Loader

The Secondary Boot Loader (SBL) described and implemented in this application note provides a solution for in-field update of USB application implemented on LPC11U3x/2x. It utilizes the boot ROM's USB and IAP API functionalities to program LPC11U3x/2x flash. The SBL occupies the first two sectors of user flash and contains routines to perform the following functionalities:

- Application image CRC checking: During boot time, the SBL computes the CRC32 of the application image and checks it against the value stored in the image header. SBL will execute the application image only if CRC check passes. This check is required to avoid booting partially programmed or corrupted images. Partial or corrupted images could be formed due to power failures during firmware update.
- Vector redirection: LPC11U3x/2x is an ARM Cortex-M0 based microcontroller, which expects the vector table to be at address 0x0. Since the first sector of the flash resides at that location, the SBL implements a vector redirector to redirect the exception and interrupt handling to 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.
- Security: The SBL supports download of encrypted images. When a secure key is programmed in the device the SBL sets the CRP level of the part to CRP2, preventing debug access and ISP capabilities. Any firmware update or flash

programming is only possible through the application which is already programmed in flash.

- DFU API: To eliminate the overhead of implementing DFU protocol in application code, the SBL exposes DFU API. Applications can use this API to invoke DFU mode.
- SBL update: SBL also implements a mechanism to update its firmware using ROM's re-invoke USB ISP method.

6.1.1 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 – 0x117. Refer to the Keil start up file used in the test application provided in this application note. Fig 7 shows a snapshot of the image header.

6.2 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 for automatic driver installation on Windows 8 and DFU initialization function should be executed. Upon receiving a detach command, the MCU should wait a couple milliseconds to ensure the device successfully detached from the host. Another function should be called in order to redirect the MCU back to the SBL. This is shown in Fig 19.

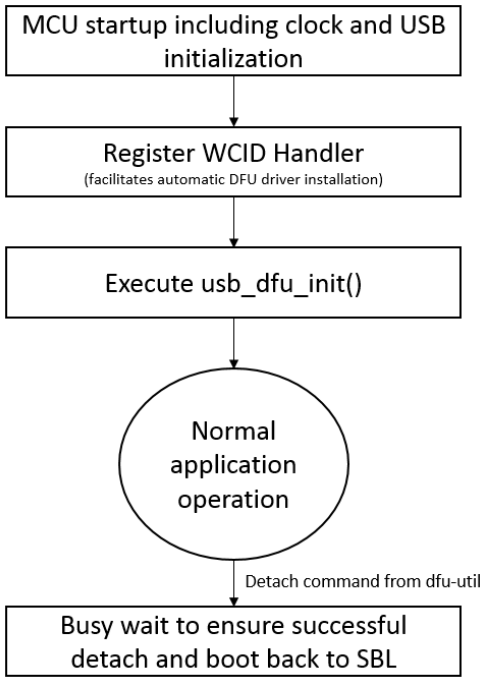

Fig 19. Program flow with DFU enabled

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 20.

usbd_dfu.c | usbd_desc.c | main.c | app_usbd_cfg.h

```
276   // usb_param.USB_Configure_Event = USB_Configure_Event;
277
278   /* Set the USB descriptors */
279   desc.device_desc = (uint8_t *) &USB_DeviceDescriptor;
280   desc.string_desc = (uint8_t *) &USB_StringDescriptor[0];
281   /* Note, to pass USBCV test full-speed only devices should have both
282    * descriptor arrays point to same location and device_qualifier set
283    * to 0.
284    */
285   desc.high_speed_desc = (uint8_t *) &USB_FsConfigDescriptor[0];
286   desc.full_speed_desc = (uint8_t *) &USB_FsConfigDescriptor[0];
287   desc.device_qualifier = 0;
288
289   /* USB Initialization */
290   ret = USBD_API->hw->Init(&g_hUsb, &desc, &usb_param);
291   if (ret == LPC_OK) {
292
293     /* Link Power Management is supported. */
294     LPC_USB->DEVCMDSTAT |= (0x1<<11);
295     LPC_USB->LPM |= (0x2<<4);/* RESUME duration. */
296
297     /*  WORKAROUND for artf32219 ROM driver BUG:
298         The mem_base parameter part of USB_param structure returned
299         by Init() routine is not accurate causing memory allocation issues for
300         further components.
301      */
302     usb_param.mem_base = USB_STACK_MEM_BASE + (USB_STACK_MEM_SIZE - usb_param.mem_size);
303
304     /* register WCID handler */
305     ret = USBD_API->core->RegisterClassHandler(g_hUsb, WCID_hdlr, 0);
306
307     ret = usb_dfu_init(g_hUsb,
308                (USB_INTERFACE_DESCRIPTOR *) &USB_FsConfigDescriptor[sizeof(USB_CONFIGURATION_DESCRIPTOR)],
309                &usb_param.mem_base, &usb_param.mem_size);
310
311     if (ret == LPC_OK) {
312         /* enable USB interrupts */
313   NVIC_EnableIRQ(USB0_IRQn);
314       /* now connect */
315       USBD_API->hw->Connect(g_hUsb, 1);
```

Fig 20. WCID handler and usb_dfu_init()

Add the definition of WCID event handler. See Fig 21.

```
/* Handler for WCID USB device requests. */
static ErrorCode_t WCID_hdlr(USBD_HANDLE_T hUsb, void *data, uint32_t event)
{
  USB_CORE_CTRL_T *pCtrl = (USB_CORE_CTRL_T *) hUsb;
  ErrorCode_t ret = ERR_USBD_UNHANDLED;

  /* Handle Microsoft's WCID request for install less WinUSB operation.
     Check https://github.com/pbatard/libwdi/wiki/WCID-Devices for more details.
   */
  if (event == USB_EVT_SETUP) {
  switch (pCtrl->SetupPacket.bmRequestType.BM.Type) {
  case REQUEST_STANDARD:
    if ((pCtrl->SetupPacket.bmRequestType.BM.Recipient == REQUEST_TO_DEVICE) &&
       (pCtrl->SetupPacket.bRequest == USB_REQUEST_GET_DESCRIPTOR) &&
       (pCtrl->SetupPacket.wValue.WB.H == USB_STRING_DESCRIPTOR_TYPE) &&
       (pCtrl->SetupPacket.wValue.WB.L == 0x00EE)) {
      pCtrl->EP0Data.pData = (uint8_t *) WCID_String_Descriptor;
      pCtrl->EP0Data.Count = pCtrl->SetupPacket.wLength;
      USBD_API->core->DataInStage(pCtrl);
      ret = LPC_OK;
    }
    break;

  case REQUEST_VENDOR:
    if (pCtrl->SetupPacket.bRequest != WCID_VENDOR_CODE) {
      break;
    }
    switch (pCtrl->SetupPacket.bmRequestType.BM.Recipient) {
    case REQUEST_TO_DEVICE:
      if (pCtrl->SetupPacket.wIndex.W == 0x0004) {
        pCtrl->EP0Data.pData = (uint8_t *) WCID_CompatID_Descriptor;
        pCtrl->EP0Data.Count = pCtrl->SetupPacket.wLength;
        USBD_API->core->DataInStage(pCtrl);
        ret = LPC_OK;
      }
    /* Fall-through. Check note1 of
       https://github.com/pbatard/libwdi/wiki/WCID-Devices#wiki-Defining_a_Device_Interface_GUID_or_other_device_specific_properties
       break;
     */

    case REQUEST_TO_INTERFACE:
      if (pCtrl->SetupPacket.wIndex.W == 0x0005) {
        pCtrl->EP0Data.pData = (uint8_t *) WCID_ExtProp_Descriptor;
        pCtrl->EP0Data.Count = pCtrl->SetupPacket.wLength;
        USBD_API->core->DataInStage(pCtrl);
        ret = LPC_OK;
      }
      break;
    }
    break;
  }
}
  return ret;
}
```

Fig 21. WCID event handler

To handle a detach, a ‘dfu_detach_sig’ variable is used as a flag. This is declared as a global variable in ‘usbd_dfu.c’ file and also should be declared as an external variable in the ‘app_usbd_cfg.h’ file, along with other DFU related variables and functions. See Fig 22.

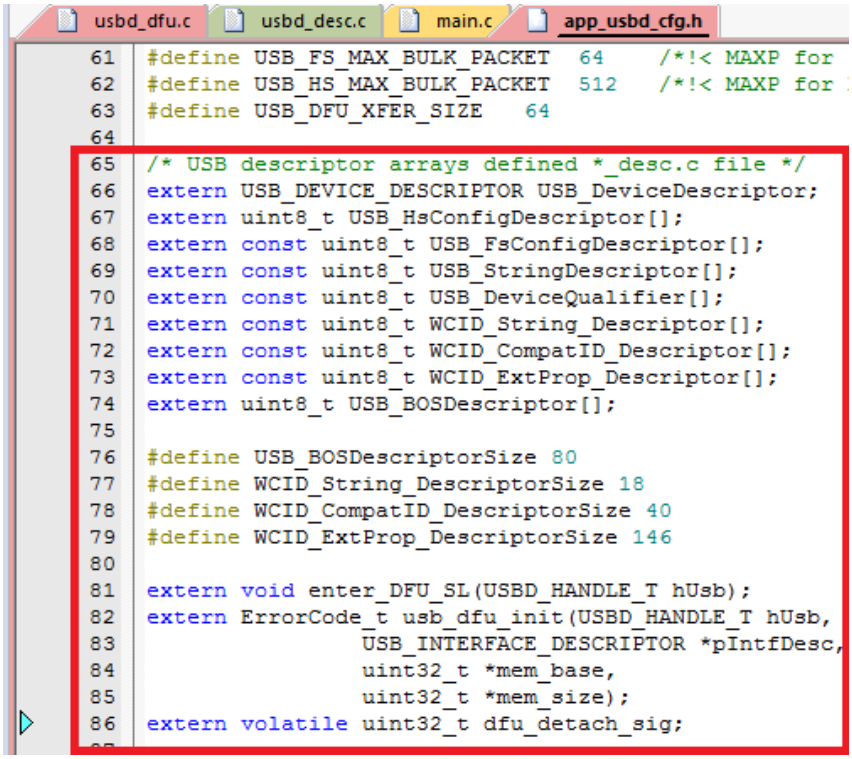

Fig 22. Defining all the DFU variables as external

To handle detach event and boot the SBL, simply polling the ‘dfu_detach_sig’ flag until it becomes a non-zero value is sufficient. Calling the function enter_DFU_SL() will turn off the USB and may cause the detach event to not be successful between host and device. Therefore, a delay is recommended before calling enter_DFU_SL() function. See Fig 23.

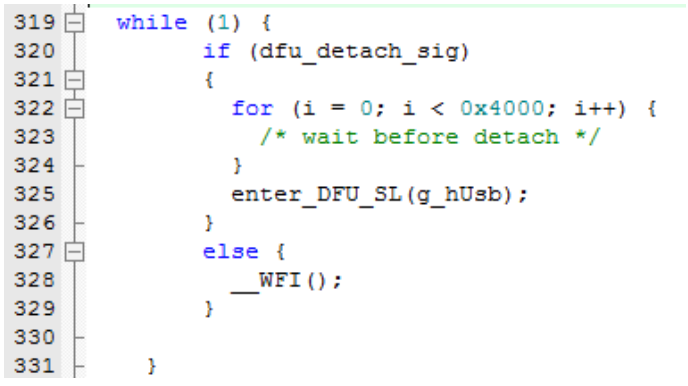

Fig 23. Polling the detach signal

7. Creating factory images and DFU update capable images

The image creator tool is a command line only tool that can be used for the Windows platform. This tool has the DFU SBL embedded inside and can prepend inputted application binaries with the SBL when using the factory command. The image creator tool is also used to calculate and add a CRC checksum into the application binary to make it a DFU update capable application image. The image creator tool can also generate encryption keys. These encryption keys are used in conjunction with the secure mode that the image creator tool and SBL support. When using this secure mode while generating a factory image, the inputted encryption key is used to encrypt the application binary with the XXTEA algorithm while storing the encryption key in the SBL. To provide additional security, the Code Read Protection (CRP) level 2 of the MCU will also be enabled.

7.1 Image creator tool

The image creator tool is provided in the ‘image-creator-tool’ folder. Open the command prompt and navigate to the directory where the executable is located.

7.1.1 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 8.4 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 24 generates the CRC for the input application binary file ‘tst_11uxx_dfu.bin’ and creates an output file ‘test_app_crc.dfu’.

```
C:\Testing>lpc11xx_secimgcr.exe tst_11uxx_dfu.bin test_app_crc.dfu
LPC11xx Secondary Boot Loader Image Creator Utility v1.2

Opening tst_11uxx_dfu.bin
Generating CRC32 for the entire file!
File size = 4096
Image header offset = 0x100
Aligning image to a 32-bit alignment, adding 4096 bytes
CRC length (bytes) = 0x1000
offsetCrc = 272
img_type    = 0x0
ifSel       = 0x4
IrqPortPin  = 0x8
MisoPortPin = 0xd
MosiPortPin = 0xc
SselPortPin = 0xe
SckPortPin  = 0x23
checksum    = 0x20
version     = 0x2
Generating CRC on bytes 0 - 0x110
Skipping CRC at bytes 0x110 - 0x113
Generating CRC on bytes 0x114 - 0x1000
CRC length:     0x00001000
CRC value :     0xf57adc1b
```

Fig 24. Image with CRC header

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

The syntax is:
lpc11xx_secimgcr.exe –n[1,2] <input file name.bin> <output file name.dfu>

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

7.1.2 Creating a plain factory image

The DFU SBL is integrated inside the image creator tool. When using the factory image option of the image creator tool, it will automatically use the integrated DFU SBL and combine it with the input application image. The syntax to create a plain factory image is:

lpc11xx_secimgr.exe –n[1,2] –f <input file name.bin> <output file name.bin>

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

Fig 25 shows the generation of a plain factory image 'plain_fac_img.bin' from the input application image 'tst_11uxx_dfu.bin'.

```
C:\Testing>lpc11xx_secimgcr.exe -n1 -f tst_11uxx_dfu.bin plain_fac_img.bin
LPC11xx Secondary Boot Loader Image Creator Utility v1.2
SBL binary size: 8192, application binary size: 4096
SBL Version: v0.1
Generating CRC32 for the entire file!
Image header offset = 0x100
Aligning image to a 32-bit alignment, adding 4096 bytes
CRC length (bytes) = 0x1000
offsetCrc = 272
img_type    = 0x0
ifSel       = 0x4
IrqPortPin  = 0x8
MisoPortPin = 0xd
MosiPortPin = 0xc
SselPortPin = 0xe
SckPortPin  = 0x23
checksum    = 0x20
version    = 0x2
Generating CRC on bytes 0 - 0x110
Skipping CRC at bytes 0x110 - 0x113
Generating CRC on bytes 0x114 - 0x1000
CRC length:     0x00001000
CRC value :     0xf57adc1b
```

Fig 25. Plain factory image generation

7.1.3 Generating key files for encryption

The image creator tool can automatically generate a random 128-bit encryption key. The syntax to generate a key file is:

lpc11xx_secimgr.exe –g <key file name>

The syntax in Fig 26 generates a key file named 'key' which containing the encryption key. This encryption key can be used to encrypt inputted application binaries when generating and inserting a CRC or creating a factory image. The encryption algorithm is XXTEA.

```
C:\Testing>lpc11xx_secimgcr.exe -g key
LPC11xx Secondary Boot Loader Image Creator Utility v1.2
```

Fig 26. Key generation

In addition to generating a key file this option produces the following three more files:

1. <key file name>_dis_sec – This file can be used to disable secure mode (CRP2) on an already secure device (CRP2 enabled). This is useful in Return Merchandise Authorization (RMA) analysis.
2. <key file name>_fctry_upd_sec – This file can be used to invoke USB ISP Mass storage mode to reprogram a factory image in secure device (CRP2 enabled).
3. <key file name>_fctry_upd_uns- This file can be used to invoke USB ISP Mass storage mode to reprogram a factory image in unsecure device (CRP disabled).

These files are sent to the device via the dfu-util when it is in SBL context. See Section 8.5 for information on how to send these files over the dfu-util.

NOTE: Store the key file in a safe location as the key file will be used in the future for field firmware updates.

7.1.3.1 Generating an encrypted application image for field updates

The syntax to create an encrypted image with CRC header from plain image for field update is:

lpc11xx_secimgr.exe –e <key file name> <input file name.bin> <output file name.dfu>

The image is encrypted with the 128-bit key file using the XXTEA encryption algorithm.

The syntax in Fig 27 generates the CRC for the input application binary file 'tst_11uxx_dfu.bin', encrypts the entire application binary and creates an output file 'encrypted_app_img.dfu'.

```
C:\Testing>lpc11xx_secimgcr.exe -e key tst_11uxx_dfu.bin encrypted_app_img.dfu
LPC11xx Secondary Boot Loader Image Creator Utility v1.2

Opening tst_11uxx_dfu.bin
Generating CRC32 for the entire file!
File size = 4096
Image header offset = 0x100
Aligning image to a 32-bit alignment, adding 4096 bytes
CRC length (bytes) = 0x1000
offsetCrc = 272
img_type    = 0x0
ifSel       = 0x4
IrqPortPin  = 0x8
MisoPortPin = 0xd
MosiPortPin = 0xc
SselPortPin = 0xe
SckPortPin  = 0x23
checksum    = 0x20
version    = 0x2
Generating CRC on bytes 0 - 0x110
Skipping CRC at bytes 0x110 - 0x113
Generating CRC on bytes 0x114 - 0x1000
CRC length:     0x00001000
CRC value :     0xf57adc1b
```

Fig 27. Encrypted Image for Field Updates

7.1.3.2 Generating a secure factory image

The DFU SBL is integrated inside the image creator tool. When using the factory image option of the image creator tool, it will automatically use the integrated DFU SBL and combine it with the input application image. Supplying a key with the factory option will automatically embed the key inside the SBL region of memory and enable CRP2. The syntax to create a secure factory image is:

lpc11xx_secimgr.exe –f <key file name> <input file name.bin> <output file name.bin>

In secure mode, CRC is mandatory to be calculated over the entire image length. The syntax in Fig 28 generates a secure factory image 'secure_fac_img.bin' from the input application image 'tst_11uxx_dfu.bin'.

```
C:\Testing>lpc11xx_secimgcr.exe -f key tst_11uxx_dfu.bin secure_fac_img.bin
LPC11xx Secondary Boot Loader Image Creator Utility v1.2
SBL binary size: 8192, application binary size: 4096
SBL Version: v0.1
Generating CRC32 for the entire file!
Image header offset = 0x100
Aligning image to a 32-bit alignment, adding 4096 bytes
CRC length (bytes) = 0x1000
offsetCrc = 272
img_type    = 0x0
ifSel       = 0x4
IrqPortPin  = 0x8
MisoPortPin = 0xd
MosiPortPin = 0xc
SselPortPin = 0xe
SckPortPin  = 0x23
checksum    = 0x20
version    = 0x2
Generating CRC on bytes 0 - 0x110
Skipping CRC at bytes 0x110 - 0x113
Generating CRC on bytes 0x114 - 0x1000
CRC length:     0x00001000
CRC value :     0xf57adc1b
```

Fig 28. Secure factory image generation

8. Downloading files using DFU-UTIL

This application note provides a pre-compiled dfu-util executable on Windows, OS X, and Ubuntu as well as 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 has been tested and verified with DFU SBL. The dfu-util tools provided in the package contains a patch from NXP Semiconductors.

8.1 Dfu-util tool platform dependencies

The dfu-util tool uses libusb in order to interface to DFU USB devices on the host machine. For certain platforms, this will require some extra steps in order for the dfu-util tool to function correctly.

8.1.1 Windows 7 requirements

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

When executing the test application, Windows 7 Device Manager should show an unknown LPK device. See Fig 29.

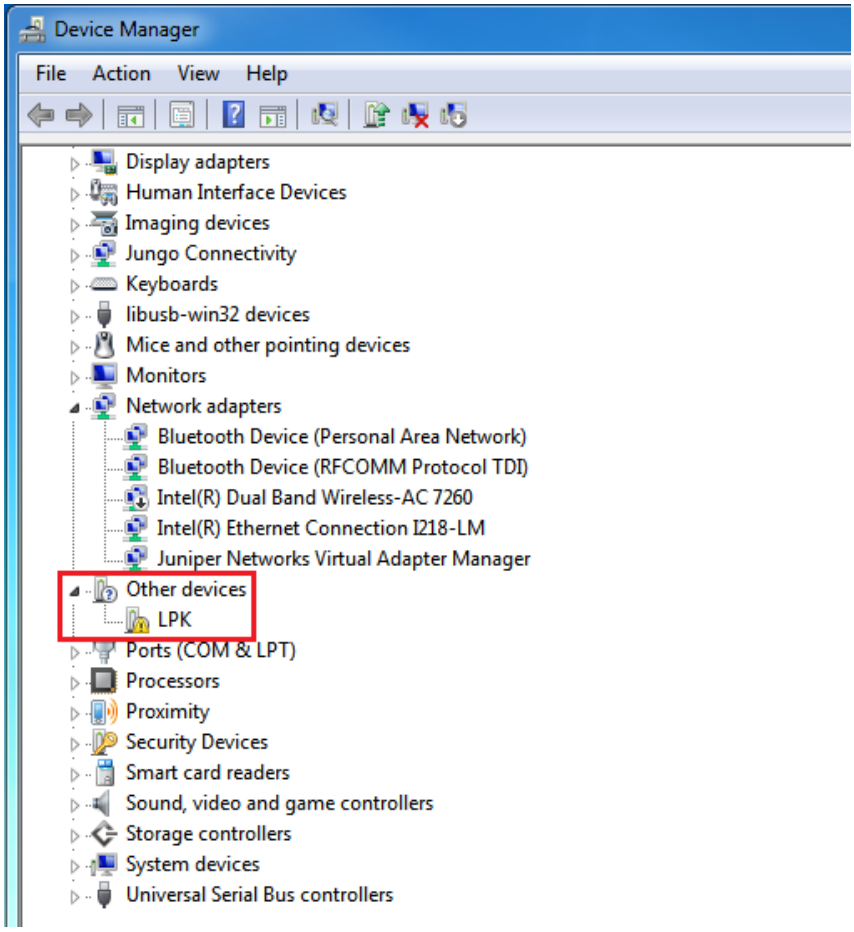

Fig 29. Device manager

Right click on the LPK device and click Update driver software. Choose the option 'Browse my computer for driver software'. Input the path of the location where the folder 'lpcdevice' is present and click next. If the drivers are successfully installed, a TypeC DFU device will appear under LpcDevice field in device manager.

See Fig 30.

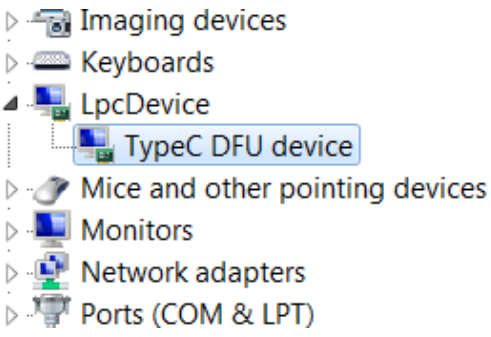

Fig 30. TypeC DFU device

8.2 OS X requirements

The libusb library is not provided in this package for OS X and must be installed manually. Out of the box, OS X is very sandboxed, and so you must install a command line command called “brew” to enable most of the typical Linux commands. To do so, 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.

8.3 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
```

Note: without setting up a udev rule, administrative privileges will be needed in order for dfu-util to access USB devices. The script will prompt the computer’s password since the `sudo` command is used so the dfu-util can perform the DFU update.

8.4 DFU update script

A script file for each of the three major x86 platforms is included: Windows, OS X, and Linux. The DFU update script is implemented slightly different for each platform but behaves the same. The script can be double clicked to execute in order to easily perform the FW update. The script automatically searches for any .dfu files in the working directory and will use the dfu-util tool to communicate with the MCU and perform a DFU update. Only one .dfu can be present in the same directory as the script and will not execute if it finds more than one.

Fig 31 shows the expected output if the script does not find only one .dfu in the working directory.

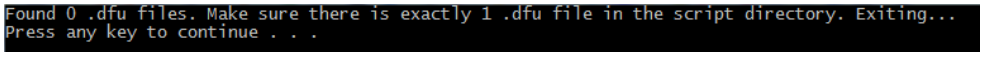

Fig 31. Expected output if the script does not find one .dfu

Fig 32 shows the expected output when it successfully finds one .dfu and performs the DFU detach to switch the MCU context to the DFU SBL, and then sends the new application binary.

```
Found one .dfu: test_app_crc.dfu
Make sure the device is connected and ready to receive the update.
Press any key to continue . . .
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Opening DFU capable USB device...
ID 1fc9:5002
Run-time device DFU version 0100
Claiming USB DFU Runtime Interface...
Determining device status: state = appIDLE, status = 0
Device really in Runtime Mode, send DFU detach request...
Device will detach and reattach...

Waiting for 0 seconds, press a key to continue ...
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Opening DFU capable USB device...
ID 1fc9:5002
Run-time device DFU version 0100
Claiming USB DFU Runtime Interface...
Determining device status: state = dfuIDLE, status = 0
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0100
Copying data from PC to DFU device
Download        [=========================] 100%         4096 bytes
Download done.
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
FW update complete. Exiting...

Waiting for 4 seconds, press a key to continue ...
```

Fig 32. Successful DFU update with the DFU script

8.5 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 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_windows.exe -l

```
C:\Testing>dfu-util_windows.exe -l
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Found Runtime: [1fc9:5002] ver=0639, devnum=16, cfg=1, intf=0, alt=0, name="DFU", serial="000003cec3
95"
```

Fig 33. List of dfu devices

To perform a field firmware update, the secondary bootloader needs to be invoked from with the application context. Send a detach command from dfu-util:

dfu-util_windows.exe -e

```
C:\Testing>dfu-util_windows.exe -e
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Opening DFU capable USB device...
ID 1fc9:5002
Run-time device DFU version 0100
Claiming USB DFU Runtime Interface...
Determining device status: state = appIDLE, status = 0
Device really in Runtime Mode, send DFU detach request...
Device will detach and reattach...
```

Fig 34. DFU detach

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. The SBL will handle the decryption of the encrypted binary if secure mode is enabled. Send the new application image or DFU commands with the following command:

dfu-util_windows.exe –t 64 –D <DFU file>

The syntax shown in Fig 35 is the expected output when sending a new application image with a valid CRC checksum.

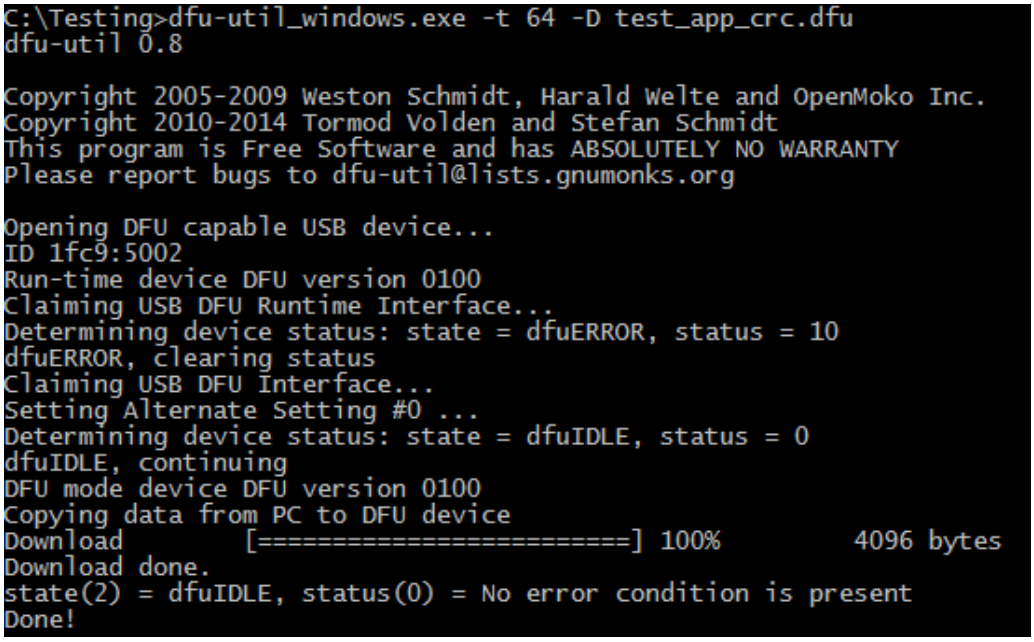

Fig 35. Field firmware update

In addition to sending new FW, the key that are generated by the image creator tool in Section 7.1.3 can be sent as DFU commands. This the easiest way to perform factory image updates in order to update the SBL or enable secure mode in the field.

9. Legal information

9.1 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.

9.2 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 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.

9.3 Trademarks

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

10. Contents

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

Date of release: 14 November 2016
Document identifier: AN11732