XMOS

AN02019: Using Device Firmware Upgrade (DFU) in USB Audio

Publication Date: 2024/12/13
Document Number: XM-015226-AN v1.0.0

IN THIS DOCUMENT

1 Introduction

This application note describes the Device Firmware Upgrade (DFU) process for a USB Audio application based on the *XMOS USB Audio Reference Design*. DFU is a protocol defined by the USB Implementers Forum that allows for the upgrading of firmware of a USB device without the need for specialised programming hardware. This ensures that USB devices can receive updates and improvements after their initial deployment.

The DFU implementation in the *XMOS USB Audio Reference Design* is compliant with version 1.1 of Universal Serial Bus Device Class Specification for Device Firmware Upgrade.

Note: This application note replaces the document *DFU loader for XMOS USB AUDIO devices (XM000524A)*

1.1 Host options

Since the device DFU implementation is compliant to the USB DFU Specification, any host implementing a compliant host-side protocol can be used. There are, however, three recommended and tested implementations:

- *TL-USBDFU* by *Thesycon*
- *xmosdfu*
- *dfu-util*

TL-USBDFU

Provided by Thesycon, *TL-USBDFU* is a cross-platform DFU solution which works on *Windows* and *macOS*.

Thesycon is an official driver partner of *XMOS*. Due to their experience and the quality level of their product and support their solutions form the recommended approach.

When licensing the Thesycon TUSBAudio driver for Windows then *TL-USBDFU* is the natural choice.

Note: The DFU instructions using the Thesycon TL-USBDFU solution are only valid for Thesycon driver package version 5.70.0 and above.

xmosdfu

XMOS provides a simple C++ program providing an example of how to interact with the DFU interface. It uses *libusb* for low level communications with the USB device. It is not intended that this should be used "as is" in a production environment, but rather intended that developers use this code as starting point for integrating into custom host control panel GUI applications.

A build flow is provided for macOS (ARM64, x86, x64), Linux, Windows & Raspberry Pi.

xmosdfu is licensed under the very permissive XMOS Public Licence, developers are free to produce and distribute derivative works without restriction.

dfu-util

dfu-util is an open source host side implementation of the USB DFU 1.0 and DFU 1.1 specifications and has been tested with many different devices.

dfu-util is licensed under the GPL version 2.

dfu-util is available via various software package managers. Whilst every effort is exerted to ensure correct operation, *dfu-util* is an open-source project that *XMOS* has no influence on. A product developer must weigh up the risks of using this 3rd party application without taking appropriate precautions regarding stability, availability and GPL licence obligations.

1.2 General operation

Fig. 1 depicts the general flash format as supported by *xflash* and associated flash libraries bundled in the XMOS XTC tools.

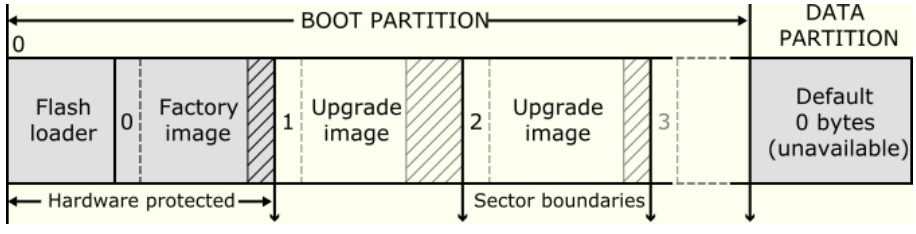


Fig. 1: *xcore* flash format

DFU support in the *XMOS USB Audio Reference Design* is limited to a single upgrade image - in practice this is adequate for almost all designs.

Separate images are provided for "factory" and "upgrade" in order to guard against corrupted or interrupted upgrade via unexpected power outages etc.

If a valid "upgrade" image is present, it is booted, if not the "factory" image is booted. Image integrity is assured via CRC based checks.

It is intended that a product be shipped from a manufacturing facility with a valid "factory" image only. The DFU functionality intentionally does not include the facility for updating the "factory" image to protect from accidental or malicious damage.

Initial in-field upgrade writes to the "upgrade" image slot. Subsequent upgrades overwrite this upgrade image.

XMOS supplied libraries (`libquadflash` and `libflash`) facilitate reading and writing image data.

Currently a solution for upgrading items in the optional "data partition" is not provided.

1.3 Supported DFU operations

The USB Audio DFU implementation supports the standard DFU Download and Upload operations as described in the DFU version 1.1 specification. Additionally, it also supports a proprietary method of reverting to the factory image.

DFU Download

DFU download is the process through which the host downloads an upgrade image into the device. To do this, the host application detaches the device in DFU mode, issues `DFU_DNLOAD` commands to send the upgrade image to the device and detaches the device back into runtime mode, causing it to boot from the upgrade image.

DFU Upload

DFU upload is the process through which the host reads back the upgrade image from the device. To do this, the host application detaches the device in DFU mode, reads the upgrade image by sending `DFU_UPLOAD` commands to the device, and detaches the device back to runtime mode. The upgrade image read from the device is saved as a binary file on the host.

Revert to factory

When the device boots, the upgrade image is always loaded if valid and present. Only when the upgrade image is invalid (or missing), the factory image is loaded. This means that even after updating the factory image by re-flashing the device using xflash, the device would still boot from the upgrade and not the factory image.

There are two options for reverting to the factory image via the DFU system:

- Download an invalid upgrade image to the device. For example, DFU download a binary file containing the word 0xFFFFFFFF to the device.
- The XMOS DFU implementation supports a custom DFU command `XMOS_DFU_REVERTFACTORY`. `XMOS_DFU_REVERTFACTORY`, the value for which is defined as `0xf1`, is expected as a USB Host to Device Vendor request on the DFU interface by the device, with the `bRequest` for the request being `XMOS_DFU_REVERTFACTORY`. The device, on receiving this request erases the upgrade image from flash so that it boots from the factory image on subsequent reboots.

  Note: The support for reverting to factory using the `XMOS_DFU_REVERTFACTORY` command is only available in the TL-USBDFU and xmosdfu host applications.

Alternatively, if the *xTag* interface is accessible, the entire flash could be erased before flashing a new factory image. i This would ensure that the device boots from the newly flashed factory image. For example:

```
xflash --erase-all --target-file=./app_an02019/src/core/xk-audio-316-mc.xn
```

2 Application examples

The sample application provided alongside this application note is called `app_an02019`. It has two build configurations for building two application binaries, a factory firmware application and an upgrade firmware application. Both the factory and upgrade applications are USB audio applications capable of performing a DFU over USB. The device is flashed with the factory firmware to begin with, and the upgrade image generated from the upgrade firmware is used for the DFU operation.

Both the applications are based on the *XMOS USB Audio reference design* which uses `lib_xua` and associated `XK-AUDIO-316-MC` hardware. The applications are the identical except for the *bcdDevice* version number that they enumerate with. Having different versions enables the user to verify the success of the upgrade process by checking the *bcdDevice* version number of the device enumerated post DFU.

The table Table 1 describes the application builds.

Table 1: Example Application Builds

Build	bcdDevice version	Vendor ID	Runtime Product ID	DFU mode Product ID
factory	0x1000	0x20b1	0x0016	0xd016
upgrade	0x9901	0x20b1	0x0016	0xd016

The DFU device interface is enabled by default in the *XMOS USB Audio Reference Design* software (See the `XUA_DFU_EN` define in lib_xua)

2.1 Building the examples

The following section assumes you have downloaded and installed the XMOS XTC tools (see *README* for required version). Installation instructions can be found here. Be sure to pay attention to the section Installation of required third-party tools.

The application uses the xcommon-cmake build system as bundled with the XTC tools.

The `an02019` software zip-file should be downloaded and unzipped into a chosen directory.

The file *CMakeLists.txt* contains build configurations named `factory` and `upgrade`.

To configure the build run the following from an XTC command prompt:

```
cd an02019
cd app_an02019
cmake -G "Unix Makefiles" -B build
```

All required dependencies are included in the software download, however, if any are missing it is at this configure step that they will be downloaded by the build system.

Finally, the application binaries can be built using `xmake`:

```
xmake -j -C build
```

This will create the two application binaries, one for each build configuration: `bin\factory\app_an02019_factory.xe` and `bin\upgrade\app_an02019_upgrade.xe`.

2.2 Installing the factory image to the device

Before verifying the DFU operation, the device needs to be flashed with the factory firmware. Before doing this, ensure that there are USB cables connecting both the **USB** and **DEBUG** ports on the XK-AUDIO-316-MC to the host computer. To flash the factory firmware, from the `app_an02019` directory, run:

```
xflash --factory bin\factory\app_an02019_factory.xe
```

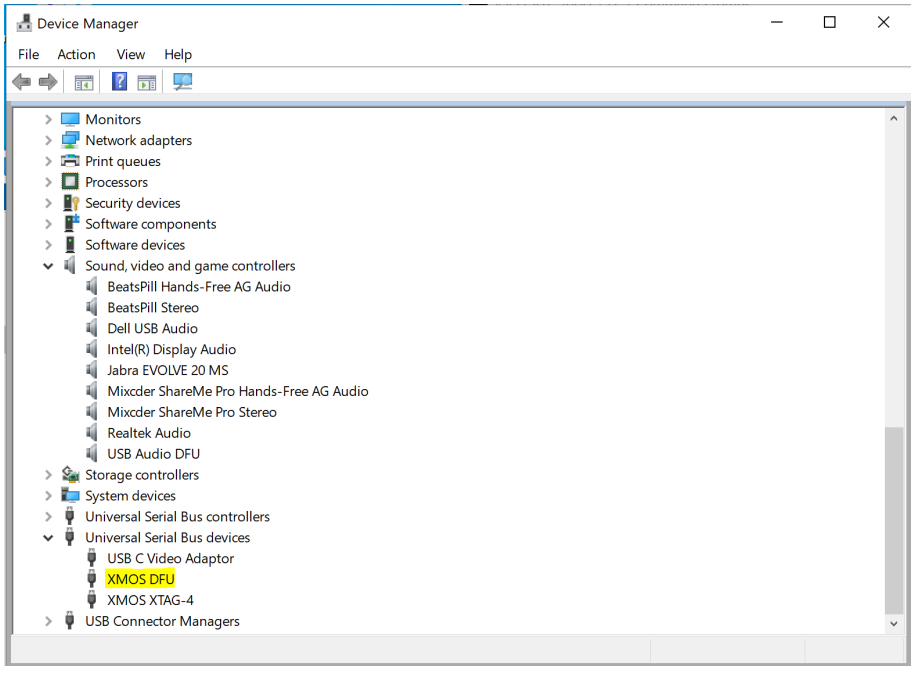
This programs the factory firmware image into the flash device over the xTag debug adapter. The device is now ready to receive firmware updates via the DFU mechanism. The USB cable connecting the **DEBUG** port of the XK-AUDIO-316-MC to the host can now

be disconnected. Updates to the device can now happen over USB. Make sure that the **USB** port on the XK-AUDIO-316-MC remains connected to the host computer.

Note: Using the xflash command as described above will allocate the entire flash for the bootloader, factory image and any upgrade images. If the flash is to be used for other purposes, for example, storing data, a portion of the flash can be allocated to boot images using the switch `--boot-partition-size`.

Enumerating as a WinUSB device on Windows

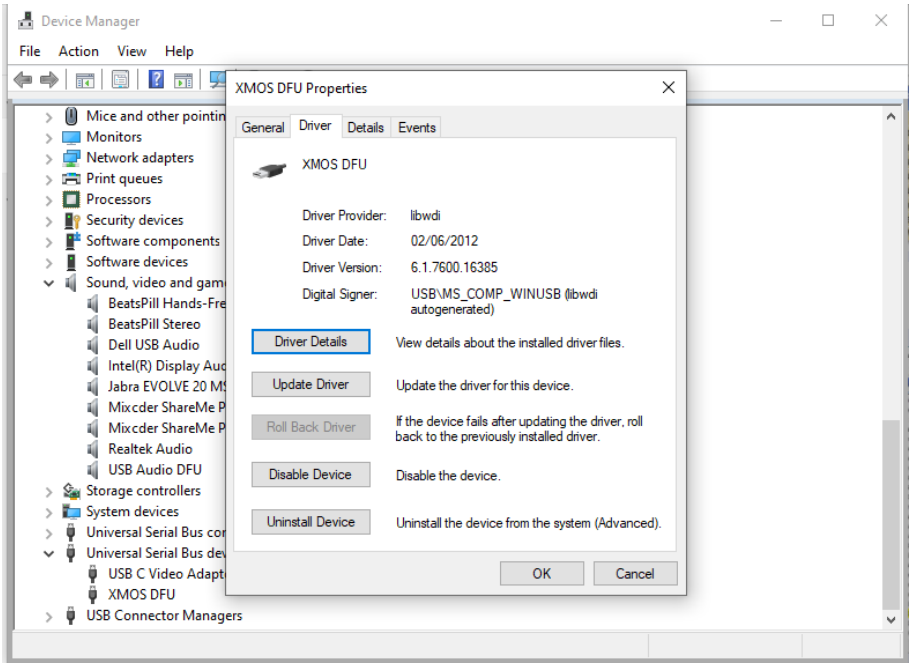
The application firmware supports extra Microsoft operating system (MSOS) descriptors that allow the device to enumerate as a WinUSB device on Windows. The MSOS descriptors report the compatible ID as *WINUSB* which enables Windows to load Winusb.sys as the device's function driver without a custom INF file. This means that when the device is connected, the DFU interface shows up as WinUSB compatible automatically, without requiring the user to manually load a driver for it using a utility like Zadig. This can be verified by opening the Device Manager and verifying that **XMOS DFU** shows up under **Universal Serial Bus Devices**.



On double clicking the **XMOS DFU** and navigating to the driver tab, the driver can be seen as `USB\MS_COMP_WINUSB` which is the generic WinUSB driver on Windows.

The MSOS descriptors, in addition to specifying WinUSB compatibility, also specify the device interface GUID. The GUID is required to access the device's DFU interface from a user application such as `dfu-util` or the Thesycon DFU driver running on Windows.

Note: The MSOS descriptors for reporting WinUSB compatibility are only relevant for Windows. Other OSs can access the interface without the need to install a driver.



2.3 Creating the Upgrade Image

Create an upgrade image from the `bin\upgrade\app_an02019_upgrade1.xe` firmware executable. To do this, from the `app_an02019` directory run:

```
xflash --factory-version 15.3 --upgrade 1 bin\upgrade\app_an02019_upgrade1.xe -o bin\upgrade\app_an02019_
↪upgrade1.bin
```

This will create the upgrade image `bin\upgrade\app_an02019_upgrade1.bin` that can be downloaded into the device using the DFU download operation.

3 DFU using the Thesycon TL-USBDFU loader

To perform DFU using the TL-USBDFU loader, install the evaluation version of the Thesycon DFU solution v5.70.0 from the xmos website. Run the `Thesycon Evaluation Driver DSK.exe`. This will extract the Thesycon Evaluation Driver DSK on the host machine. `<Path to DSK>\DfuCons\x64\tlusbdfucons.exe` is the command line DFU host application.

After the device is flashed with the factory firmware, check that it is seen as a valid DFU device by running the `tlusbdfucons` devinfo command:

```
tlusbdfucons devinfo VendorID:RuntimeProducID,VendorID:DfuProductID

tlusbdfucons devinfo 0x20b1:0x0016,0x20b1:0xd016
```

This should result in an output containing:

```
Enumerate available devices...
Found 1 device(s).
Open device...
Device information:
Vendor ID:          0x20B1
Product ID:         0x0016
BcdDevice:          0x1000
Manufacturer:       'XMOS'
Product:            'USB Audio DFU'
Serial number:      ''
Device Instance ID: 'USB\VID_20B1&PID_0016&MI_03\6&35B1C28E&3D&0003'
Physical Device ID: '{8B088D61-691D-11EF-86E5-9CB6D08CA822}'
Current run mode:   APP (application)
```

3.1 Download upgrade image into the device

The upgrade image can be downloaded into the device by running the `tlusbdfucons upgrade` command. The `upgrade` command is of the form:

```
tlusbdfucons upgrade VendorID:RuntimeProducID,VendorID:DfuProductID <upgrade image>
```

To do DFU download, run the following from the directory containing tlusbdfucons.exe:

```
tlusbdfucons.exe upgrade 0x20b1:0x0016,0x20b1:0xd016 <path to the app_an02019_upgrade.bin file>
```

Running this would generate a console output similar to:

```
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode APP (application).
Rebooting device to switch to run mode DFU (bootloader).
Waiting until device is gone...
The device has disconnected itself from USB.
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode DFU (bootloader).
Downloading to target 0, press any key to abort...
State = Finished : 69632 bytes of 69632 bytes transferred (100 %)
Upgrade successfully finished (took 7.875 seconds).
Rebooting device to switch to run mode APP (application).
Firmware upgrade finished. The device should start in application mode now.
```

Run the `tlusbdfucons devinfo` command to verify that the device is running the upgrade image

```
tlusbdfucons devinfo 0x20b1:0x0016,0x20b1:0xd016
```

```
Enumerate available devices...
Found 1 device(s).
Open device...
Device information:
Vendor ID:          0x20B1
Product ID:         0x0016
BcdDevice:          0x9901
Manufacturer:       'XMOS'
Product:            'USB Audio DFU'
```

(continues on next page)

(continued from previous page)

```
Serial number:      ''
Device Instance ID: 'USB\VID_20B1&PID_0016&MI_03\6&35B1C28E&3D&0003'
Physical Device ID: '{8B088D61-691D-11EF-86E5-9CB6D08CA822}'
Current run mode:   APP (application)

Found Runtime: [20b1:0016] ver=9901, devnum=33, cfg=1, intf=3, path="1-1", alt=0, name="XMOS DFU", serial=
↪"UNKNOWN"
```

Note how `BcdDevice:  0x9901` now shows the `bcdDevice` version 0x9901 of the upgrade image. This indicates a successful DFU download operation.

3.2 Upload upgrade image from the device

The upgrade image that is downloaded in the device can be read back and saved on the host by using the `tlusbdfucons readout` command. The `readout` command is of the form:

```
tlusbdfucons readout VendorID:RuntimeProducID,VendorID:DfuProductID <upload bin file name>
```

Run the following from the directory containing `tlusbdfucons.exe`:

```
tlusbdfucons.exe readout 0x20b1:0x0016,0x20b1:0xd016 app.bin
```

Running this should generate a console output similar to:

```
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode APP (application).
Rebooting device to switch to run mode DFU (bootloader).
Waiting until device is gone...
The device has disconnected itself from USB.
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode DFU (bootloader).
Uploading from target 0, press any key to abort...
State = Finished : 69120 bytes transferred
Readout successfully finished (took 0.156 seconds).
Firmware image successfully stored in app.bin. Image type is RawBinary.
Rebooting device to switch to run mode APP (application).
Firmware readout finished. The device should start in application mode now.
```

The upgrade image read from the device is saved in the `app.bin` file. This file can be used as the input file in a subsequent download operation.

3.3 Revert to the factory image

`tlusbdfucons` supports the *proprietary XMOS revert to factory* method. To revert to factory, run the `xmosrevertfactory` command:

```
tlusbdfucons.exe xmosrevertfactory 0x20b1:0x0016,0x20b1:0xd016
```

Running this should generate a console output similar to:

```
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode APP (application).
Rebooting device to switch to run mode DFU (bootloader).
Waiting until device is gone...
The device has disconnected itself from USB.
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode DFU (bootloader).
Reverting to factory image...
Enumerate available devices...
Found 1 device(s).
Open device...
The device is operating in run mode DFU (bootloader).
Rebooting device to switch to run mode APP (application).
Waiting until device is gone...
The device has disconnected itself from USB.
Operation finished. The device should start in application run mode now.
```

Run the **devinfo** command to verify that the device is now running the factory image:

```
tlusbdfucons devinfo 0x20b1:0x0016,0x20b1:0xd016
```

This should result in an output containing:

```
Enumerate available devices...
Found 1 device(s).
Open device...
Device information:
Vendor ID:          0x20B1
Product ID:         0x0016
BcdDevice:          0x1000
Manufacturer:       'XMOS'
Product:            'USB Audio DFU'
Serial number:      ''
Device Instance ID: 'USB\VID_20B1&PID_0016&MI_03\6&35B1C28E&3D&0003'
Physical Device ID: '{8B088D61-691D-11EF-86E5-9CB6D08CA822}'
Current run mode:   APP (application)
```

Note the **BcdDevice: 0x1000** now shows the **bcdDevice** version of the factory image, indicating the success of the revert to factory operation.

4 DFU using the xmosdfu loader

The `xmosdfu` loader is provided as source as part of the USB Audio framework, located in lib_xua/host/xmosdfu. The loader is compiled using libusb. The code for the loader is contained in the file `xmosdfu.cpp` It has support for CMake based compilation.

4.1 Building the app

To compile, clone lib_xua and run the build commands from the `lib_xua/host/xmosdfu` directory.

MacOS or Linux

To compile the `xmosdfu` application, from the `lib_xua/host/xmosdfu`, run:

```
cmake -B build
make -C build
```

The `xmosdfu` application is created in the `lib_xua/host/xmosdfu/build` directory.

Windows

When compiling on Windows, ensure that Visual Studio Build Tools with C++ support are installed on the machine and the commands for building the `xmosdfu` application are run from a Developer Command Prompt.

To compile the `xmosdfu` application, from `lib_xua/host/xmosdfu`, run:

```
cmake -B build -G "NMake Makefiles"
cd build
nmake
```

The `xmosdfu` application is created in the `lib_xua/host/xmosdfu/build` directory.

Once the device is flashed with the factory firmware, to check if it is seen as a valid DFU capable device, run the following command from the `lib_xua/host/xmosdfu/build` directory where `xmosdfu` is located:

```
./xmosdfu --listdevices
```

The device should show up in the list of DFU capable devices.

```
Found Runtime: [20b1:0016] ver=1000
```

Note the VendorID, ProductID and bcdVersion of the factory image are displayed.

4.2 Download upgrade image into the device (xmosdfu)

The upgrade image can be downloaded into the device by running the `xmosdfu download` command. The `download` command is of the form:

```
./xmosdfu VendorID:RuntimeProducID,VendorID:DfuProductID --download <upgrade image>
```

To do DFU download, run the following from the directory containing `xmosdfu`:

```
./xmosdfu 0x20b1:0x0016,0x20b1:0xd016 --download <path to the app_an02019_upgrade.bin file>
```

Running this would generate a console output similar to:

```
Found Runtime: [20b1:0016] ver=1000
Opening DFU capable USB device, [20b1:0016], Runtime mode.
XMOS DFU application started - Interface 3 claimed
Detaching device from application mode.
Waiting for device to restart and enter DFU mode...
Found DFU: [20b1:d016] ver=1000
```

(continues on next page)

(continued from previous page)

```
Opening DFU capable USB device, [20b1:0016], DFU mode.
... DFU firmware upgrade device opened
... Downloading image (/Users/shuchitak/sandboxes/an02019_sandbox/an02019/app_an02019/bin/upgrade/app_an02019_
↪upgrade.bin) to device
... Download complete
... Returning device to application mode
```

Run the `xmosdfu listdevices` command to verify that the device is running the upgrade image

```
./xmosdfu --listdevices
```

```
Found Runtime: [20b1:0016] ver=9901
```

Note the `ver=9901` now shows the `bcdDevice` version 0x9901 of the upgrade image, indicating success of the DFU Download operation.

4.3 Upload upgrade image from the device (xmosdfu)

The upgrade image that is downloaded in the device can be read back and saved on the host by using the `xmosdfu upload` command. The `upload` command is of the form:

```
./xmosdfu VendorID:RuntimeProducID,VendorID:DfuProductID --upload <upload bin file name>
```

Run the following from the directory containing `xmosdfu`:

```
./xmosdfu 0x20b1:0x0016,0x20b1:0xd016 --upload app.bin
```

Running this should generate a console output similar to:

```
Found Runtime: [20b1:0016] ver=9901
Opening DFU capable USB device, [20b1:0016], Runtime mode.
XMOS DFU application started - Interface 3 claimed
Detaching device from application mode.
Waiting for device to restart and enter DFU mode...
Found DFU: [20b1:d016] ver=9901
Opening DFU capable USB device, [20b1:d016], DFU mode.
... DFU firmware upgrade device opened
... Uploading image (app.bin) from device
... Returning device to application mode
```

The upgrade image read from the device is saved in the `app.bin` file. This file can be used as the input file in a subsequent download operation.

4.4 Revert to the factory image (xmosdfu)

`xmosdfu` supports the *custom XMOS revert to factory* method. To revert to the factory image, run:

```
./xmosdfu 0x20b1:0x0016,0x20b1:0xd016 --revertfactory
```

Running this should generate a console output similar to:

```
Found Runtime: [20b1:0016] ver=9901
Opening DFU capable USB device, [20b1:0016], Runtime mode.
XMOS DFU application started - Interface 3 claimed
Detaching device from application mode.
Waiting for device to restart and enter DFU mode...
Found DFU: [20b1:d016] ver=9901
Opening DFU capable USB device, [20b1:d016], DFU mode.
... DFU firmware upgrade device opened
... Reverting device to factory image
... Returning device to application mode
```

Following this up with `./xmosdfu --listdevices` will confirm this:

```
Found Runtime: [20b1:0016] ver=1000
```

Note the `ver=1000` now shows the `bcdDevice` version of the factory image, indicating the success of the revert to factory operation.

5 DFU using the dfu-util loader

DFU can be performed using the *dfu-util* utility that implements the DFU host side implementation of the DFU 1.1 specification. *dfu-util* installation instructions are available for various operating systems.

> **Warning:** `dfu-util` will not work on a Windows machine if the machine also has the Thesycon USB Audio driver, any version prior to v5.70.0, installed on it. This is because, prior to v5.70.0, the Thesycon audio driver would take over the entire device, leaving the device's DFU interface unavailable for `dfu-util` to claim. From v5.70.0 onwards, the Thesycon driver installs only on the audio interface leaving the DFU interface available for `dfu-util`.

After the device is flashed with the factory firmware, check that it is seen as a valid DFU device by doing:

```
dfu-util -l
```

This should result in an output containing:

```
Found Runtime: [20b1:0016] ver=1000, devnum=30, cfg=1, intf=3, path="1-1", alt=0, name="XMOS DFU", serial=
↪"UNKNOWN"
```

Note that 20b1:0016 are the Vendor and Product IDs respectively of the factory firmware, 1000 is the bcdDevice version and *intf=3* implies that the interface with *bInterfaceNumber* 3 is seen as DFU compatible and *XMOS DFU* is the string describing the interface as seen in the DFU interface's string descriptor.

5.1 Download Upgrade Image into the Device (dfu-util)

The upgrade image can be downloaded into the device by running the `dfu-util` download command. The download command is of the form:

```
dfu-util -d VendorID:RuntimeProducID,VendorID:DfuProductID -D <upgrade image> -R
```

To do DFU download, run the following from `app_an02019` directory:

```
dfu-util -d 20b1:0016,20b1:d016 -D bin\upgrade\app_an02019_upgrade1.bin -R
```

Running this would generate a console output similar to:

```
Opening DFU capable USB device...
Device ID 20b1:0016
Run-Time device DFU version 0110
Claiming USB DFU (Run-Time) Interface...
Setting Alternate Interface zero...
Determining device status...
DFU state(0) = appIDLE, status(0) = No error condition is present
Device really in Run-Time Mode, send DFU detach request...
Device will detach and reattach...
Opening DFU USB Device...
Claiming USB DFU Interface...
Setting Alternate Interface #0 ...
Determining device status...
DFU state(2) = dfuIDLE, status(0) = No error condition is present
DFU mode device DFU version 0110
Device returned transfer size 64
Copying data from PC to DFU device
Download        [=========================] 100%        69632 bytes
Download done.
DFU state(2) = dfuIDLE, status(0) = No error condition is present
Done!
Resetting USB to switch back to Run-Time mode
```

Run `dfu-util -l` post download to verify that the device is running the upgrade image:

```
Found Runtime: [20b1:0016] ver=9901, devnum=33, cfg=1, intf=3, path="1-1", alt=0, name="XMOS DFU", serial=
↪"UNKNOWN"
```

Note how ver=9901 now shows the bcdDevice version 0x9901 of the upgrade image. This indicates a successful DFU download operation.

5.2 Upload upgrade image from the device (dfu-util)

The upgrade image that is downloaded in the device can be read back and saved on the host by using the `dfu-util` upload command. The upload command is of the form:

```
dfu-util -d VendorID:RuntimeProducID,VendorID:DfuProductID -U <upload bin file name> -R
```

Run the following:

```
dfu-util -d 20b1:0016,20b1:d016 -U app.bin -R
```

Running this should generate a console output similar to:

```
Opening DFU capable USB device...
Device ID 20b1:0016
Run-Time device DFU version 0110
Claiming USB DFU (Run-Time) Interface...
Setting Alternate Interface zero...
Determining device status...
DFU state(0) = appIDLE, status(0) = No error condition is present
Device really in Run-Time Mode, send DFU detach request...
Device will detach and reattach...
Opening DFU USB Device...
Claiming USB DFU Interface...
Setting Alternate Interface #0 ...
Determining device status...
DFU state(2) = dfuIDLE, status(0) = No error condition is present
DFU mode device DFU version 0110
Device returned transfer size 64
Copying data from DFU device to PC
Upload  [=========================] 100%        69120 bytes
Upload done.
Received a total of 69120 bytes
Resetting USB to switch back to Run-Time mode
```

The upgrade image read from the device is saved in the `app.bin` file. This file can be used as the input file in a subsequent download operation.

6 Frequently Asked Questions

The flashed device when connected to Windows host does not appear as a DFU capable device when running tlusbdfucons devinfo or dfu-util -l. How do I fix this?

Windows caches the USB device details by storing information about the USB devices in its registry. This can cause driver changes or other changes made in the device interfaces to not take effect till the registry is cleared. Effect of the cached registry can show up as the DFU interface not being reported as WinUSB compatible if there was previously a driver installed manually for it. Caching would also lead to a change in the device interface GUID to not take effect. To clear the registry cache, use a third party tools like USBDeview. Uninstall the USB device drivers by using *USBDeview* GUI, or by typing from a command line with Administrator rights the lines below:

```
USBDeview.exe /RunAsAdmin /remove_by_pid 20b1;0016
USBDeview.exe /RunAsAdmin /remove_by_pid 20b1;d016
```

where 20b1 is the USB VendorID, 0016 is the USB Product ID in runtime mode and d016 is the USB product ID in DFU mode as specified in the application built as part of this app note.

How do I change the Product IDs used in the application?

The application has two Product IDs, the runtime product ID and the DFU mode product ID. The runtime Product ID can be changed by changing the `PID_AUDIO_2` define and the DFU mode Product ID can be changed by changing the `DFU_PID` define in xua_conf.h.

```
#define PID_AUDIO_2 (0x0016)
#define DFU_PID     (0xd000 + PID_AUDIO_2)
```

If changing the Product IDs, make sure to modify the DFU commands described in the previous sections to use the new Product IDs.

Warning: If using the Thesycon Evaluation Driver `tlusbdfucons` host app, the supported Product IDs are hardcoded within the driver's `tlusbdfuapi.dll.license.ini` file. The driver uses this list to identify devices for DFU purposes. Changing the Product IDs to something not present in the supported Product IDs list will make DFU operations fail when using the `tlusbdfucons` host application.

How do I change the device interface GUID?

The default GUID is specified in the `WINUSB_DEVICE_INTERFACE_GUID` define in xua_conf_default.h This can be overriden by redefining `WINUSB_DEVICE_INTERFACE_GUID` in the application's `xua_conf.h`. A utility like guidgenerator can be used for generating a GUID.

Warning: If using the Thesycon Evaluation Driver `tlusbdfucons` host app, the device interface GUID that the `tlusbdfucons` applications looks for is hardcoded in the driver's `tlusbdfuapi.dll.config.ini` file and is the same as the GUID defined in xua_conf_default.h. If the device runs with a different GUID, the `tlusbdfucons` driver would not be able to locate the WinUSB interface created by the XMOS device and DFU operations will fail.

Why might I need to create a new device interface GUID?

On Windows, the device interface GUID is the link between the product and product specific apps. When using default apps/utilities, it's okay to stick to the predefined GUID.

However, when developing custom applications, it's better to assign a fresh GUID so it can be ensured that the apps talk to their products only.