

D3XX for Android 0.5.21

By Future Technology Devices International

Sep 7 2017

Contents

Introduction	1
History	1
Installation	2
Demo application	2
Demo application for Android	3
Buffered Design	3
Static link	4
D3XX API matrix	4
New Android only APIs	5
Notes	9

Introduction

D3XX for Android is an userspace library implemented based on libusb¹, which provides D3XX compatible APIs² on Android. There are some differences between Windows and Android implementation, most notably is streaming mode (auto repeat) is always enabled and data is buffered.

libftd3xx uses an unmodified version of libusb which is distributed under the terms of the LGPL³.

History

- 0.5.21
 - Bug fix: Ignore failure of update U1 for Revision A device
- 0.5.20
 - Limit maximum URB size to 16KB if Android kernel version ≤ 3.3
 - Move workaround for FT600/FT601 Revision A chip from sample code into library
 - Start IN streaming after `FT_ReadPipe()`/`FT_ReadPipeEx()` has been called
- 0.5.17
 - Compiled with Ubuntu 14.04 (GCC 4.8.4) to support SUSE Android Enterprise Server 11 SP4
 - For static link build, please use Ubuntu 14.04
- 0.5.6
 - Cross compiled with Android NDK R15, targeted to Android 6.0 API level 23
 - Static link to LLVM libc++ runtime
- 0.5.3
 - Compile with GCC 4.9.2, GLIBC 2.19
 - Bug fix: Fix potential session not auto repeated issue

¹<https://github.com/libusb/libusb>

²http://www.ftdichip.com/Support/Documents/ProgramGuides/AN_379%20D3xx%20Programmers%20Guide.pdf

³<https://github.com/libusb/libusb/blob/master/COPYING>

- API: `FT_GetDeviceInfoList()` will not longer open device if `pftHandle` is not `NULL`, which follows Windows Library's implementation
- API: Filling correct value in `puLengthTransferred` of `FT_ControlTransfer()`
- 0.5.0
 - Add Rev B chip support
- 0.4.11
 - Bug fix: `FT_ReadGPIO()` returns wrong value for Rev A chip
 - Bug fix: Change return value to `FT_OK` when `FT_ReadPipe()` and `FT_ReadPipeEx()`'s return value is `FT_TIMEOUT` and at least 1 byte had been read
- 0.4.10
 - API: implemented `FT_EnableGPIO()` `FT_WriteGPIO()` `FT_ReadGPIO()` for Revision A device
 - API: implemented `FT_GetDescriptor()` `FT_GetStringDescriptor()` `FT_GetDeviceDescriptor()` `FT_GetConfigurationDescriptor()` `FT_GetInterfaceDescriptor()` `FT_GetPipeInformation()` `FT_ListDevices()`
- 0.4.9
 - Support Revision B device: request device send ZLP before `FT_ReadPipe()`/`FT_ReadPipeEx()` timeouts
- 0.4.8
 - Bug fix: multiple channel read/write not working correctly
 - Remove `FT_AddVIDPID()` API, stop detecting D3XX devices by using VID/PID anymore
- 0.4.7
 - Allow `NULL` parameters for `FT_GetDeviceInfoDetail()`
- 0.4.6
 - Stop changing chip configuration in `FT_Create()`, please remove `ENABLENOTIFICATIONMESSAGE_INCHALL` and set `DISABLECANCELSESSIONUNDERRUN` in your application
 - Bug fix: read buffer pointer is not increased which cause data not been copied.
 - New demo application: `file_transfer.cpp` which use FT245 loopback to duplicate file.
- 0.4.5
 - Add workaround for FT600/FT601 Rev.A device to prevent sending control requests when streaming is on, which may cause system hang or failed to create device.
- 0.4.4
 - Demo application: add new argument to change FIFO mode.
 - API: implemented `FT_EnableGPIO()` `FT_WriteGPIO()` `FT_ReadGPIO()` `FT_SetGPIOPull()` for Revision B device.
- 0.4.3
 - First beta release

Installation

```
sudo rm /usr/lib/libftd3xx.so
sudo cp libftd3xx.so /usr/lib/
sudo cp libftd3xx.so.0.5.21 /usr/lib/
```

Demo application

1. Extract the release package, compile the demo application.

```
make
```

2. Streamer application usage:

- Arguments: `./streamer <read channel count> <write channel count> [mode]`
 - Mode: 0 = FT245 mode (default), 1 = FT600 mode

- Examples:
 - FT245 loopback FPGA: `./streamer 1 1`
 - FT600 loopback FPGA: `./streamer 4 4 1`
 - FT245 streaming FPGA (read only): `./streamer 1 0`
 - FT245 streaming FPGA (write only): `./streamer 0 1`
 - FT245 streaming FPGA (read and write): `./streamer 1 1`
 - FT600 streaming FPGA (3 channel read 1 channel write): `./streamer 3 1 1`
- Please run with root permission if streamer app failed to detect any device.
- Alternatively you can run the streamer application without installing library:


```
sudo LD_LIBRARY_PATH=. ./streamer 1 0
```

3. File transfer loopback application usage:

- Arguments: `./file_transfer <src file> <dest files> <FIFO mode/channel count> [loop]`
 - FIFO mode/channel count: 0 = FT245 fifo mode, 1 - 4 FT600 channel count
 - loop: 0 = oneshot (default), 1 = loop forever

Demo application for Android

1. Install CMake and NDK from Android SDK Manager
2. Extract the release package, open `android-build` with your preferred text editor, the content of the file looks like below:

```
#!/bin/bash
ASDK=/usr/local/opt/android-sdk
ANDROID_API_LEVEL=android-23
TMP_DIR=/tmp/d3xx-demo
ARCHS=(arm64-v8a armeabi-v7a armeabi x86_64 x86)
# Set to OFF if dynamic link is preferred
STATIC_LINK_TO_D3XX=ON
```

3. Change ASDK to point to the path of your Android SDK
4. Compile the demo application

```
./android-build
```

5. Run the compiled application on your Android target

```
adb push streamer /data/local/tmp
adb push libftd3xx.so /data/local/tmp
adb shell
su
cd /data/local/tmp
LD_LIBRARY_PATH=. ./streamer 1 1
```

Buffered Design

- Read and write will be buffered at library level. Use `FT_SetTransferParams` to change the buffer size.
 - Device to host read will be started in `FT_Create` even before `FT_ReadPipeEx` is called, `FT_FlushPipe` could be used to discard the buffered data.
 - Default is 8 concurrent URBs for both read and write pipe for each FIFO interface, each pipe has 256 URB buffers in the queue, and each URB buffer is 32KBytes.

- Function return:
 - `FT_WritePipe` / `FT_WritePipeEx` will return when all user buffer been copied to URB buffer or no free URB buffer after a timeout, or on an error condition.
 - `FT_ReadPipe` / `FT_ReadPipeEx` will return when user buffer is fully filled or no URB buffer coming after a timeout, or on an error condition.
- Use `FT_GetReadQueueStatus` to get unread buffer length in the queue.
- Use `FT_GetWriteQueueStatus` to get unsent buffer length in the queue.
- On error condition, `FT_GetUnsentBuffer` could be used to retrieve back unsent buffers in library's OUT queue.
 - Calling this function will close the pipe, call `FT_Create` or `FT_FlushPipe` to enable the pipe.

Static link

1. GCC 4.9 or later must be used
2. Link with GCC 4.9.2 C++ static library is required:


```
LIBS = -L . -lftd3xx-static -llibstdc++-static
```

D3XX API matrix

API	Android	Win	Remark for Android
<code>FT_CreateDeviceInfoList</code>	I ⁴	I	
<code>FT_GetDeviceInfoList</code>	I	I	
<code>FT_GetDeviceInfoDetail</code>	I	I	
<code>FT_ListDevices</code>	I	I	
<code>FT_Create</code>	I	I	Multiple URB read requests will be submitted for all the channels in the call
<code>FT_Close</code>	I	I	
<code>FT_WritePipe</code>	I	I	<code>pOverlapped</code> parameter is not supported, internally will call <code>FT_WritePipeEx</code> with timeout option from <code>FT_SetPipeTimeout</code> , default timeout is 1second
<code>FT_ReadPipe</code>	I	I	<code>pOverlapped</code> parameter is not supported, internally will call <code>FT_ReadPipeEx</code> with timeout option from <code>FT_SetPipeTimeout</code> , default timeout is 1second
<code>FT_GetOverlappedResult</code>	X ⁵	I	
<code>FT_InitializeOverlapped</code>	X	I	
<code>FT_ReleaseOverlapped</code>	X	I	
<code>FT_SetStreamPipe</code>	X	I	Use <code>FT_SetTransferParams</code> to change streaming size
<code>FT_ClearStreamPipe</code>	X	I	Streaming mode always enabled in D3XX for Android
<code>FT_AbortPipe</code>	X	I	
<code>FT_FlushPipe</code>	I	I	Android will only flush library's queue buffers
<code>FT_GetDeviceDescriptor</code>	I	I	
<code>FT_GetConfigurationDescriptor</code>	I	I	Active configuration only
<code>FT_GetInterfaceDescriptor</code>	I	I	Active configuration only

⁴Implemented

⁵Non-exists

API	Android	Win	Remark for Android
FT_GetStringDescriptor	I	I	
FT_GetPipeInformation	I	I	
FT_GetDescriptor	I	I	
FT_ControlTransfer	I	I	
FT_GetVIDPID	I	I	
FT_SetGPIO	X	I	Obsoleted
FT_GetGPIO	X	I	Obsoleted
FT_EnableGPIO	I	I	
FT_WriteGPIO	I	I	
FT_ReadGPIO	I	I	
FT_SetGPIOPull	I	X	For Rev. B device only
FT_SetNotificationCallback	X	I	
FT_ClearNotificationCallback	X	I	
FT_GetChipConfiguration	I	I	
FT_SetChipConfiguration	I	I	
FT_IsDevicePath	X	I	Android doesn't support GUID path
FT_GetDriverVersion	I	I	
FT_GetLibraryVersion	I	I	
FT_GetFirmwareVersion	I	I	
FT_ResetDevicePort	I	I	
FT_CycleDevicePort	X	I	
FT_SetPipeTimeout	I	I	Added to keep compatible with D3XX for Windows 1.2.0.5 RC6. Set 0 to read from /write to library buffer only
FT_SetTransferParams	I	X	Must be called before FT_Create is called
FT_ReadPipeEx	I	X	equivalent to FT_SetPipeTimeout + FT_ReadPipe. Use FIFO index instead of endpoint to address pipe
FT_WritePipeEx	I	X	equivalent to FT_SetPipeTimeout + FT_WritePipe. Use FIFO index instead of endpoint to address pipe
FT_GetReadQueueStatus	I	X	Get total unread buffer length in library's queue
FT_GetWriteQueueStatus	I	X	Get total unsent buffer length in library's queue
FT_GetUnsentBuffer	I	X	Read back unsent buffer in library's OUT pipe queue

New Android only APIs

```
enum FT_GPIO_PULL {
    GPIO_PULL_50K_PD,
    GPIO_PULL_HIZ,
    GPIO_PULL_50K_PU,
    GPIO_PULL_DEFAULT = GPIO_PULL_50K_PD
};
```

```
enum FT_PIPE_DIRECTION {
    FT_PIPE_DIR_IN,
    FT_PIPE_DIR_OUT,
    FT_PIPE_DIR_COUNT,
};
```

```

struct FT_PIPE_TRANSFER_CONF {
    /* set to true PIPE is not used, default set to FALSE */
    BOOL fPipeNotUsed;

    /* Enable non thread safe transfer to increase throughput, set this flag
     * if guarantee only single thread access the pipe at a time, default
     * set to FALSE */
    BOOL fNonThreadSafeTransfer;

    /* Concurrent URB request number, 8 by default, set value < 2 to use
     * default value */
    BYTE bURBCount;

    /* 256 by default, set value < 2 to use default value */
    WORD wURBBufferCount;

    /* 32K by default, set value < 512 to use default value */
    DWORD dwURBBufferSize;

    /* 1G by default, used by FT600 and FT601 only, set 0 to use
     * default value */
    DWORD dwStreamingSize;
};

typedef struct _FT_TRANSFER_CONF {
    /* structure size: sizeof(FT_TRANSFER_CONF) */
    WORD wStructSize;

    /* Please refer to struture FT_PIPE_TRANSFER_CONF */
    struct FT_PIPE_TRANSFER_CONF pipe[FT_PIPE_DIR_COUNT];

    /* Stop reading next URB buffer if current buffer is not fully filled,
     * default set to FALSE */
    BOOL fStopReadingOnURBUnderrun;

    /* Reserved, set to 0 */
    BOOL fReserved1;

    /* Do not flush device side residue buffer after reopen the
     * device, default set to FALSE */
    BOOL fKeepDeviceSideBufferAfterReopen;
} FT_TRANSFER_CONF;

/* Set transfer parameters for each FIFO channel
 * Must be called before FT_Create is called. Need to be called again
 * after FT_Close(), otherwise default parameters will be used.
 *
 * Default value will be used for each FIFO channel if this function
 * is not been called. Please refer to structure defines for default
 * value.
 *
 * pConf: Please refer to structure FT_TRANSFER_CONF
 * dwFifoID: FIFO interface ID. Valid values are 0-3 which represents

```

```

*           FIFO channel 1-4 for FT600 and FT601 */
FTD3XX_API FT_STATUS WINAPI FT_SetTransferParams(
    FT_TRANSFER_CONF *pConf,
    DWORD dwFifoID);

/* ReadPipe with timeout
*
* dwFifoID: FIFO interface ID. Valid values are 0-3 which represents
* FIFO channel 1-4 for FT600 and FT601
* dwTimeoutInMs: timeout in milliseconds, 0 means return immediately
* if no data */
FTD3XX_API FT_STATUS WINAPI FT_ReadPipeEx(
    FT_HANDLE ftHandle,
    UCHAR ucFifoID,
    PCHAR pucBuffer,
    ULONG ulBufferLength,
    PULONG pulBytesTransferred,
    DWORD dwTimeoutInMs);

/* WritePipe with timeout
*
* dwFifoID: FIFO interface ID. Valid values are 0-3 which represents
* FIFO channel 1-4 for FT600 and FT601
* dwTimeoutInMs: timeout in milliseconds, 0 means return immediately
* if no data */
FTD3XX_API FT_STATUS WINAPI FT_WritePipeEx(
    FT_HANDLE ftHandle,
    UCHAR ucFifoID,
    PCHAR pucBuffer,
    ULONG ulBufferLength,
    PULONG pulBytesTransferred,
    DWORD dwTimeoutInMs);

/* Get total unread buffer length in library's queue
*
* dwFifoID: FIFO interface ID. Valid values are 0-3 which represents
* FIFO channel 1-4 for FT600 and FT601 */
FTD3XX_API FT_STATUS WINAPI FT_GetReadQueueStatus(
    FT_HANDLE ftHandle,
    UCHAR ucFifoID,
    LPDWORD lpdwAmountInQueue);

/* Get total unsent buffer length in library's queue
*
* dwFifoID: FIFO interface ID. Valid values are 0-3 which represents
* FIFO channel 1-4 for FT600 and FT601 */
FTD3XX_API FT_STATUS WINAPI FT_GetWriteQueueStatus(
    FT_HANDLE ftHandle,
    UCHAR ucFifoID,
    LPDWORD lpdwAmountInQueue);

/* Read unsent buffer for OUT pipe
* Set byBuffer to NULL first to close the pipe to get accurate buffer
* length, allocate buffer with the length, then call this function

```

```

    * again with the allocated buffer to read out all buffers
    *
    * dwFifoID: FIFO interface ID. Valid values are 0-3 which represents
    *           FIFO channel 1-4 for FT600 and FT601
    * byBuffer: User allocated buffer
    * lpdwBufferLength: Pointer to receive the size of buffer if byBuffer
    *                   is NULL. Size of buffer if byBuffer is not NULL. */
FTD3XX_API FT_STATUS WINAPI FT_GetUnsentBuffer(
    FT_HANDLE ftHandle,
    UCHAR ucFifoID,
    BYTE *byBuffer,
    LPDWORD lpdwBufferLength);

/* Enable GPIOs
 * Each bit represents one GPIO setting, GPIO0-GPIO2 from LSB to MSB
 *
 * dwMask: set bit to 0 to skip the GPIO, 1 to enable the GPIO
 * dwDirection: set bit to 0 for input, 1 for output */
FTD3XX_API FT_STATUS WINAPI FT_EnableGPIO(
    FT_HANDLE ftHandle,
    DWORD dwMask,
    DWORD dwDirection
);

/* Set GPIO level
 * Each bit represents one GPIO setting, GPIO0-GPIO2 from LSB to MSB
 *
 * dwMask: set bit to 0 to skip the GPIO, 1 to enable the GPIO
 * dwDirection: set bit to 0 for low, 1 for high */
FTD3XX_API FT_STATUS WINAPI FT_WriteGPIO(
    FT_HANDLE ftHandle,
    DWORD dwMask,
    DWORD dwLevel
);

/* Get level of all GPIOs
 * Each bit represents one GPIO setting, GPIO0-GPIO2, RD_N, OE_N from
 * LSB to MSB */
FTD3XX_API FT_STATUS WINAPI FT_ReadGPIO(
    FT_HANDLE ftHandle,
    DWORD *pdwData
);

/* Set GPIO internal pull resistors
 * dwMask: Each bit represents one GPIO setting, GPIO0-GPIO2 from
 * LSB to MSB
 * dwPull: Each two bits represents one GPIO setting, GPIO0-GPIO2 from
 * LSB to MSB
 *
 * dwMask: set bit to 0 to skip the GPIO, 1 to enable the GPIO
 * dwPull: refer to enum FT_GPIO_PULL */
FTD3XX_API FT_STATUS WINAPI FT_SetGPIOPull(
    FT_HANDLE ftHandle,
    DWORD dwMask,

```



```
DWORD dwPull
);
```

Notes

1. Kernel 3.3 and older limitations

- Maximum URB size cannot be larger than 16384 bytes
- Kernel may not be able to allocate more than 60 of concurrent URB requests, please refer to kernel source⁶

The following code example shows how to cope with the limits,

```
static void old_kernel_workaround(void)
{
    FT_TRANSFER_CONF conf;

    memset(&conf, 0, sizeof(FT_TRANSFER_CONF));
    conf.wStructSize = sizeof(FT_TRANSFER_CONF);
    conf.pipe[FT_PIPE_DIR_IN].bURBCount = 7;
    conf.pipe[FT_PIPE_DIR_OUT].bURBCount = 7;
    conf.pipe[FT_PIPE_DIR_IN].dwURBBufferSize = 16384;
    conf.pipe[FT_PIPE_DIR_OUT].dwURBBufferSize = 16384;
    for (DWORD i = 0; i < 4; i++)
        FT_SetTransferParams(&conf, i);
}
```

2. Please call FT_CreateDeviceInfoList again after FT_ResetDevicePort is been called, because the device is disconnected after port reset.

```
FT_ResetDevicePort(handle);
FT_Close(handle);
FT_CreateDeviceInfoList(&count);
```

3. Rev.A chip failed to get 1K aligned data issue

When FIFO master's last write is 1K aligned data, and master stop writing after this, part of the 1K aligned data may stuck in the host URB buffer, and application will not be able to receive it, until fifo master start to write again.

If your application hits this special case, please set `dwURBBufferSize` of `FT_SetTransferParams` to 1024 for USB 3.0 port, and 512 for USB 2.0 port. This will make sure host URB requests return at the aligned boundary but leads to poor performance.

Throughput is around 165MiB/s for single channel read configuration when set to 1K, 362MiB/s when set to 32K.

4. Endpoint/Pipe ID vs FIFO ID

Endpoint or Pipe ID is used for `FT_ReadPipe()` and `FT_WritePipe()`. FT600/601 has 4 channels, the endpoint number is 0x2-0x5 for OUT pipes, and 0x82-0x85 for IN pipes.

FIFO ID is used for new APIs e.g. `FT_SetTransferParams()`, `FT_ReadPipeEx()`, `FT_WritePipeEx()`, `FT_GetReadQueueStats()` and `FT_GetWriteQueueStatus()`, which is [0, 3] for FT600/FT601.

⁶<http://elixir.free-electrons.com/linux/v3.3/source/drivers/usb/host/xhci-ring.c#L2462>