CON-FMC API based on LIBUSB

Version 0 Revision 4

May 3, 2019 (April 18, 2018)

Future Design Systems, Inc.
www.future-ds.com / contact@future-ds.com

Copyright © 2018-2019 Future Design Systems, Inc.

Abstract

This document addresses user level API (Application Programming Interface) of CON-FMC. This API utilized LIBUSB.

Table of Contents

Copyright © 2018-2019 Future Design Systems, Inc.

1 Overview

As shown in Figure 1, this is about CON-FMC API (**conapi**) on top of LIBUSB. The LIBUSB is a C library providing generic access to USB device through usbfs or WinUSB.

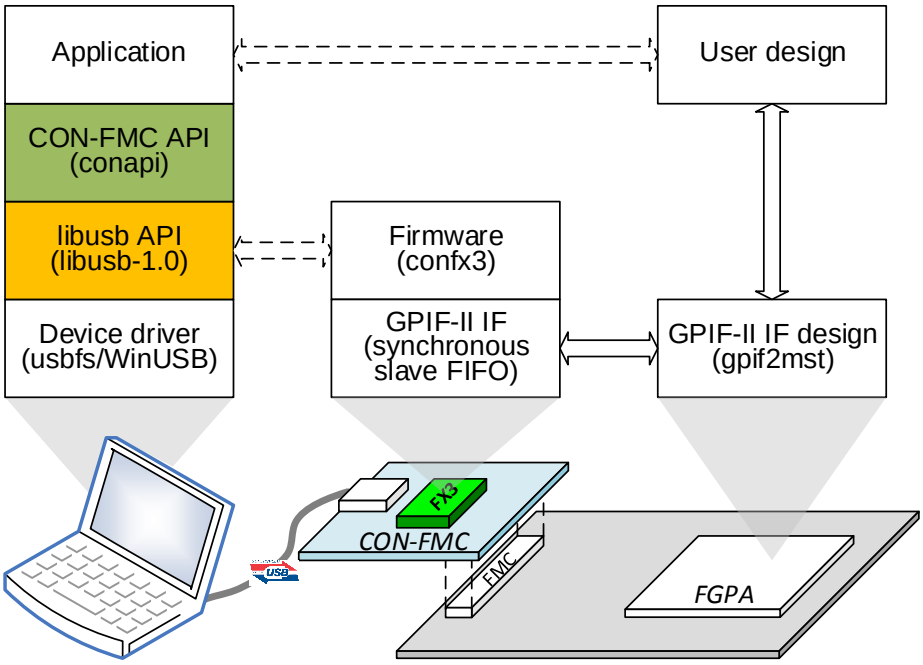

Figure 1: Overview

Licensing issues:

- 'libusb' is released under version 2.1 of the GNU Lesser General Public License (LGPL).

2 Getting started

As shown in code below, 'conapi.h' should be included to use CON-FMC API and its basic steps are as follows.

1. Include API header, i.e., 'conapi.h'.
2. Get handler by calling 'conInit' with proper arguments, which include card-id, operation-mode, and log-level.
3. Do some operation using the handler such as assert reset, get CID, and so on.
4. Release all resource by calling 'conRelease().

Copyright © 2018-2019 Future Design Systems, Inc.

```
#include "conapi.h"

int main(int argc, char *argv[])
{
    unsigned int cid=0;
    unsigned int mode=CON_MODE_CMD;
    unsigned int loglevel=CONAPI_LOG_LEVEL_INFO;

    con_Handle_t handle=conInit(cid, mode, loglevel);
    conReset(handle, 1);
    cid=conGetCID(handle);
    conRelease(handle);

    return 0;
}
```

3 API convention

Most API returns zero or non-NULL pointer on successful completion. Otherwise, it returns non-zero number and error number is stored in the internal variable. The error code is returned by ‘conGetErrorConapi()’ and the error message can be got by ‘conGetErrorMsg()’.

```
// pointer retuning case
if ((handle=conInit(cid, mode, loglevel))==NULL) {
     printf("%d %s\n", conGetErrorConapi(),
                       conErrorMsgConapi(conGetErrorConapi()));
     return 0;
}

// zero retuning case
if (conReset(handle, 1)) {
    printf("%d %s\n", conGetErrorConapi(),
                      conErrorMsgConapi(conGetErrorConapi()));
}
```

4 Macros, data types, and enumerations

4.1 API version

```
#define CONAPI_VERSION 0x20180420
```

4.2 CON-FMC handler

Copyright © 2018-2019 Future Design Systems, Inc.

```
struct _con_Handle {
        unsigned int  mode;
        unsigned int  cid;
        struct _usb   usb;
};
typedef struct _con_Handle *con_Handle_t;
```

4.3 USB related

```
struct _usb {
      struct libusb_context       *ctx;
      struct libusb_device        *dev;
      struct libusb_device_handle *handle;
      int                          interface;
      int                          kernelDriverDetached;
                                  // 0: initial value
                                  // 1: dettached
      int    speed; // x0.1-Mbps
      int    bulk_max_pkt_size_out; // in bytes
      int    bulk_max_pkt_size_in ; // in bytes
      int    iso_max_pkt_size_out ; // in bytes
      int    iso_max_pkt_size_in  ; // in bytes
};
typedef struct _usb usb_t;
```

'speed' reflects the device speed:

- 15: 1.5Mbps for USB1.0
- 120: 12Mbps for USB1.1
- 4,800: 480Mbps for USB2.0
- 50,000: 5Gbps for USB3.0
- 100,000: 10Gbps for USB3.2

'bulk_max_pkt_size_out/in' reflects the number of bytes for bulk operation.

- 512: for USB2.0
- 1024: for USB 3.0

4.4 USB vendor requests

These codes are used to request vendor specific functions to the firmware (i.e, confxe) through USB control transfer conUsbControlTransfer().

```
enum conapi_vendor_req {
    CON_CID_READ  =0xC0
  , CON_RST_DRIVE =0xC1
  , CON_MODE_DRIVE=0xC2
  , CON_FX3_INFO_READ=0xC3
  , CON_LED_DRIVE =0xC4
  , CON_I2C_WRITE =0xBA
  , CON_I2C_READ  =0xBB
  , CON_FX3_RESET =0xE2
```

Copyright © 2018-2019 Future Design Systems, Inc.

```
   , CON_EP_RESET = 0xE1
};
```

4.5 USB vendor ID and Product ID

It is fixed value to identify USB device.

```
enum conapi_vid_pid {
      CON_USB_VID=0x04B4
    , CON_USB_PID=0x00F3
};
```

4.6 USB endpoint address

It is fixed value to identify USB endpoint. For CON-FMC, ENDPOINT 1 is used for read and write.

```
enum conapi_ep_num {
     CON_EP_OUT_DOWN=0x01 // ENDPOINT 1 DownStream (i.e, OUT, write)
   , CON_EP_IN_UP   =0x81 // ENDPOINT 1 UpStream (i.e., IN, read)
};
```

4.7 CON-FMC operation mode

CON-FMC operation mode.

```
enum conapi_mode {
     CON_MODE_CMD  =0x0 // command mode
   , CON_MODE_SU2F =0x1 // stream-out (USB-to-FPGA) mode
   , CON_MODE_SF2U =0x2 // stream-in (USB-from-FPGA) mode
   , CON_MODE_SLOOP=0x3 // CON_MODE_SU2F|CON_MODE_SF2U // stream-
in/out mode
};
```

4.8 FX3 information

FX3 firmware information.

```
struct _con_Fx3Info {
        uint32_t version;
};
typedef struct _con_Fx3Info con_Fx3Info_t;
```

4.9 Board information

CON-FMC board information

Copyright © 2018-2019 Future Design Systems, Inc.

```
struct __attribute__((__packed__)) _con_BoardInfo {
	char	MagicID[4];
	uint32_t FormatVersion;
	uint32_t Length;
	char	Name[32];
	uint32_t PcbVersion;
	uint32_t PcbSerial;
	char	MajorInfo[32];
	char	MajorPart[32];
	uint32_t CRC;
};
typedef struct _con_BoardInfo con_BoardInfo_t;
```

- MagicID[4]: null terminated string for "FPI"
- FormatVersion: Little-endian BCD (binary coded decimal), e.g., 0x2018_04_25
- Length: num of bytes from the beginning to the end (CRC included)
- Name[32]: null terminated string for board name, e.g., "CON-FMC-FX3"
- PcbVersion: Little-endian BCD, e.g., 0x1810_0101
- PcbSerial: Little-endian binary (not BCD)
- MajorInfo[32]: null terminated string for major part, e.g., "Cypress FX3"
- MajorPart[32]: null terminated string for board name, e.g., "CYUSB3014-BZXI
- CRC: formula 0x04C11DB7 (0xED888320/reversed) use not inverted

4.10 Gpif2mst information

Gpif2mst RTL information.

```
struct _con_MasterInfo {
	uint32_t version;
	uint16_t depth_cmd; // num of entries (i.e., words)
	uint16_t depth_u2f; // num of entries (i.e., words)
	uint16_t depth_f2u; // num of entries (i.e., words)
	uint8_t  clk_mhz; // frequency in Mhz
	uint8_t  clk_inv; // inverted when 1
};
typedef struct _con_MasterInfo con_MasterInfo_t;
```

5 API

5.1 Initialize and release

5.1.1 Initialize: conInit()

'conInit()' prepares CON-FMC USB device through libusb APIs and returns a handler when there is a CON-FMC with the given 'con_cid'.

Copyright © 2018-2019 Future Design Systems, Inc.

Function prototype:

```
con_Handle_t conInit ( unsigned int con_cid
                     , unsigned int con_mode
                     , unsigned int conapi_log_level );
```

Arguments:

- ✧ con_cid: card identification between 0 ~ 7.
- ✧ con_mode: gpif2mst operation mode and bitwise or of followings
  - ➢ CON_MODE_CMD: pseudo-DMA mode
  - ➢ CON_MODE_SU2F: stream output, i.e., stream to FPGA through FX3
  - ➢ CON_MODE_SF2U: stream input, i.e., stream from FPGA through FX3
  - ➢ CON_MODE_SLOOP: both CON_MODE_SU2F and CON_MODE_SF2U
- ✧ conapi_log_level: verbosity level and one of followings
  - ➢ CONAPI_LOG_LEVEL_NONE,
  - ➢ CONAPI_LOG_LEVEL_ERROR,
  - ➢ CONAPI_LOG_LEVEL_WARNING,
  - ➢ CONAPI_LOG_LEVEL_INFO,
  - ➢ CONAPI_LOG_LEVEL_DEBUG

Return value:

- ✧ non-NULL pointer to con_Handle_t on success
- ✧ NULL pointer on failure

5.1.2 Release: conRelease()

‘conRelease()’ releases CON-FMC USB device for the given ‘con_cid’.

Function prototype:

```
int conRelease ( con_Handle_t con_handle );
```

Arguments:

- ✧ con_handle: CON-FMC handler.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.2 Core functions

5.2.1 USB device initialize: conUsbInit()

‘conUsbInit()’ finds a device with a particular CON-FMC card id by using ‘conOpenDevWithVidPidCid()’

Function prototype:

Copyright © 2018-2019 Future Design Systems, Inc.

```
struct libusb_device_handle *conUsbInit (
                        , unsigned int card_id
                        , unsigned int conapi_log_level
                        , struct _usb *pusb );
```

Arguments:

- ✧ card_id: Card ID
- ✧ conapi_log_level: verbosity level and one of followings
  - ➢ CONAPI_LOG_LEVEL_NONE,
  - ➢ CONAPI_LOG_LEVEL_ERROR,
  - ➢ CONAPI_LOG_LEVEL_WARNING,
  - ➢ CONAPI_LOG_LEVEL_INFO,
  - ➢ CONAPI_LOG_LEVEL_DEBUG
- ✧ pusb: pointer to ‘struct _usb’; it can be NULL

Return value:

- ✧ Pointer of a USB device handler, i.e., not NULL on success
- ✧ NULL on failure

5.2.2 Release: conUsbRelease()

‘conUsbRelease()’ releases USB device.

Function prototype:

```
int conUsbRelease ( usb_t *pusb );
```

Arguments:

- ✧ pusb: pointer to ‘struct _usb’

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.2.3 USB device open: conOpenUsbWithVidPidCid()

‘conOpenUsbWithVidPidCid()’ finds a device with a particular Vendor ID (0x04B4), Product ID (0x00F3), and Card ID.

Function prototype:

```
struct libusb_device_handle *conOpenUsbWithVidPidCid (
                          struct libusb_context *ctx
                        , uint16_t              vendor_id
                        , uint16_t              product_id
                        , uint8_t               card_id );
```

Arguments:

- ✧ ctx: context
- ✧ vendor_id: USB vendor ID

Copyright © 2018-2019 Future Design Systems, Inc.

- ✧ produt_id: USB product ID
- ✧ card_id: Card ID

Return value:

- ✧ Pointer of a USB device handler, i.e., not NULL on success
- ✧ NULL on failure

It first gathers USB device list and tries to get devices with the given vendor ID and product ID. And then it checks if there is a device with the given card ID. If a device exists, it opens the device.

5.2.4 USB control transfer: conUsbControlTransfer()

'conUsbControlTransfer()' generates USB control transfer to read and write resources in the USB controller, i.e., FX3.

Function prototype:

```
int conUsbControlTransfer( struct libusb_device_handle *dev_handle
                         , uint8_t   type
                         , uint16_t  value
                         , uint8_t   cmd
                         , uint8_t  *data
                         , uint16_t  length );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ type: direction of the transfer
  - ➢ LIBUSB_ENDPOINT_IN for read
  - ➢ LIBUSB_ENDPOINT_OUT for write
- ✧ value: fills wValue and its meaning depends on 'cmd'
- ✧ cmd: Vendor Specific Requests and see 'enum conapi_vendor_req'.
  - ➢ CON_CID_READ
  - ➢ CON_RST_DRIVE
  - ➢ CON_MODE_DRIVE
  - ➢ CON_LED_DRIVE
  - ➢ CON_I2C_WRITE
  - ➢ CON_I2C_READ
- ✧ data: pointer to the buffer carrying data if any depending on 'cmd'
- ✧ length: the number of bytes to carry through 'data' buffer

Return value:

- ✧ the number of bytes actually transferred (>=0) on success
- ✧ <0 on failure, which will be

5.2.5 USB bulk transfer: conUsbBulkTransfer()

'conUsbBulkTransfer()' generates USB data transfer to push or pop data from FPGA through FX3.

Function prototype:

Copyright © 2018-2019 Future Design Systems, Inc.

```
int conUsbBulkTransfer( struct libusb_device_handle *dev_handle
                      , uint8_t                      ep_addr
                      , uint8_t                     *pData
                      , int                          toTransfer
                      , int                         *pTransferred );
```

Arguments:

- ✧ dev_handle: USB device handler.
- ✧ ep_addr: USB ENDPOINT address for downstream or upstream
  - ➢ CON_EP_OUT_DOWN
  - ➢ CON_EP_IN_UP
- ✧ pData: pointer to the buffer for data if any
- ✧ toTransfer: the number of bytes to be transferred
- ✧ pTransferred: the actual number of bytes have been transferred

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.2.6 Board I2C transfer: conUsbFx3I2cTransfer()

'conUsbFx3I2cTransfer()' generates I2C transactions on FX3. Note that I2C device address is fixed and dealt with the firmware (confx3).

Function prototype:

```
int conUsbFx3I2cTransfer( struct libusb_device_handle *dev_handle
                        , uint8_t   type
                        , uint16_t  byte_addr
                        , uint8_t  *buffer
                        , uint16_t  byte_count );
```

Arguments:

- ✧ dev_handle: USB handler.
- ✧ type: direction of the transfer
  - ➢ LIBUSB_ENDPOINT_IN for read
  - ➢ LIBUSB_ENDPOINT_OUT for write
- ✧ byte_addr: I2C byte address
- ✧ buffer: pointer to the data to be used
- ✧ byte_count: the number of bytes to be transferred

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.3 Utility

5.3.1 Read CID: conGetCID()

'conGetCID()' reads CID.

Copyright © 2018-2019 Future Design Systems, Inc.

Function prototype:

```
int conGetCID( con_Handle_t  con_handle );
```

Arguments:

- ✧ con_handle: CON-FMC handle

Return value:

- ✧ 0~7 on success
- ✧ <0 on failure

5.3.2 Read CID: conUsbGetCID()

'conUsbGetCID()' reads CID.

Function prototype:

```
int conUsbGetCID( struct libusb_device_handle *dev_handle );
```

Arguments:

- ✧ dev_handle: USB handle

Return value:

- ✧ 0~7 on success
- ✧ <0 on failure

5.3.3 Reset: conReset()

'conReset()' asserts reset signals (i.e., SL_RST_N) and then de-asserts it.

Function prototype:

```
int conReset ( con_Handle_t con_handle
        , unsigned int duration );
```

Arguments:

- ✧ con_handle: CON-FMC handler.
- ✧ duration: the number of 'nop()' instruction between driving 0 and 1.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.3.4 Mode: conSetMode()

'conSetMode()' sets CON-FMC operation mode. It asserts reset, drives mode, and then de-assert reset.

Function prototype:

```
int conSetMode ( con_Handle_t  con_handle
        , unsigned int  con_mode );
```

Copyright © 2018-2019 Future Design Systems, Inc.

Arguments:
- ✧ con_handle: CON-FMC handler.
- ✧ con_mode:
  - ➢ CON_MODE_CMD
  - ➢ CON_MODE_SU2F
  - ➢ CON_MODE_SF2U
  - ➢ CON_MODE_SLOOP

Return value:
- ✧ 0 on success
- ✧ !=0 on failure, which will be

It should be noted that the upstream buffers may be full when CON_MODE_SF2U or CON_MODE_SLOOP is set. As a result, access in CON_MODE_CMD may not work after CON_MODE_SF2U or CON_MODE_SLOOP.

5.3.5 Master information: conGetMasterInfo()

'conGetMasterInfo()' reads gpif2mst related information. It can be used when the operation mode is CON_MODE_CMD.

Function prototype:

```
int conGetMasterInfo ( con_Handle_t      con_handle
                     , con_MasterInfo_t *pInfo );
```

Arguments:
- ✧ con_handle: CON-FMC handler.
- ✧ pInfo: pointer to structure to store command FIFO depth; its structure is given in section 4.10 Gpif2mst information.

Return value:
- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.3.6 Board information: conGetBoardInfo()

'conGetBoardInfo()' reads board information form I2C EEPROM.

Function prototype:

```
int conGetBoardInfo  ( con_Handle_t     con_handle
                     , con_BoardInfo_t *pInfo
                     , unsigned int     length
                     , unsigned int     crc_check );
```

Arguments:
- ✧ con_handle: CON-FMC handler.
- ✧ pInfo: pointer to the board information data structure; its structure is given in section 4.9 Board information.
- ✧ length: the number of byte of the structure pointed by pInfo

Copyright © 2018-2019 Future Design Systems, Inc.

- ✧ crc_check: ignore CRC if 0

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

There is ‘conSetBoardInfo()’ that writes board information to the I2C EEPROM.

5.3.7 Get API version: conGetVersionApi()

‘conGetVersionApi()’ returns CONAPI version.

Function prototype:

```
unsigned int conGetVersionApi( void );
```

Arguments:

Return value:

- ✧ 4-byte version on success
- ✧ <0 on failure, which will be

5.3.8 FX3 information: conGetFx3Info()

‘conGetFx3Info()’ returns FX3 firmware version.

Function prototype:

```
int conGetFx3Info( con_Handle_t   con_handle
                                  , con_Fx3Info_t *pInfo )
```

Arguments:

- ✧ con_handle: CON-FMC handle
- ✧ pInfo: pointer to the structure

Return value:

- ✧ 0 on success
- ✧ <0 on failure, which will be

5.3.9 FX3 information: conUsbGetFx3Info()

‘conUsbGetFx3Info()’ returns FX3 firmware version.

Function prototype:

```
int conGetFx3Info( struct libusb_device_handle *dev_handle
                  ,int *pVersion );
```

Arguments:

- ✧ dev_handle: USB handle
- ✧ pVersion: pointer to the variable to return version

Return value:

Copyright © 2018-2019 Future Design Systems, Inc.

- ✧ 0 on success
- ✧ <0 on failure, which will be

5.3.10 Get GPIF2MST version: conGetVersionMst()

'conGetVersionMst()' returns GPIF2MST RTL version.

Function prototype:

```
int conGetVersionMst( con_Handle_t con_handle
                    , unsigned int *pVersion );
```

Arguments:

- ✧ con_handle: CON-FMC handle
- ✧ pVersion: pointer to the variable to return version

Return value:

- ✧ 0 on success
- ✧ <0 on failure, which will be

5.3.11 Reset FX3: conResetFx3()

'conResetFx3()' resets USB device.

Function prototype:

```
int conResetFx3 ( struct libusb_device_handle *dev_handle
          , unsigned int warm_reset );
```

Arguments:

- ✧ dev_handle: USB handler.
- ✧ warm_reset: make USB device reset
  - ➢ 0: cold-reset
  - ➢ 1: warm-reset

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

This requests to reset the FX3 device and the device will disappears and establishes the USB connection again without loading the firmware.

5.3.12 Reset Endpoint: conResetEp()

'conResetEp()' requests to flush and reset the endpoint.

Function prototype:

```
int conResetEp ( struct libusb_device_handle *dev_handle
          , unsigned int ep_flag );
```

Arguments:

Copyright © 2018-2019 Future Design Systems, Inc.

- ✧ dev_handle: USB handler.
- ✧ ep_flag: which endpoint
  - ➢ 1: OUT/down-stream endpoint
  - ➢ 2: IN/up-stream endpoint
  - ➢ 3: Both endpoints.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

5.3.13 Timeout: conUsbTimeout()

'conUsbTimeout()' requests to flush and reset the endpoint.

Function prototype:

```
int conUsbTimeout (int millisecond );
```

Arguments:

- ✧ millisecond: timeout in millisecond for bulk and control
  - ➢ positive value: timeout value to set
  - ➢ negative value: returns current timeout value without modification
  - ➢ o: non-blocking access

Return value:

- ✧ >0 on success
- ✧ <0 on failure

5.4 Pseudo-DMA functions

This functions are used under CON_MODE_CMD mode, which uses command to control gpif2mst, in which the command is fed through thread 0 FIFO to cmd-FIFO.

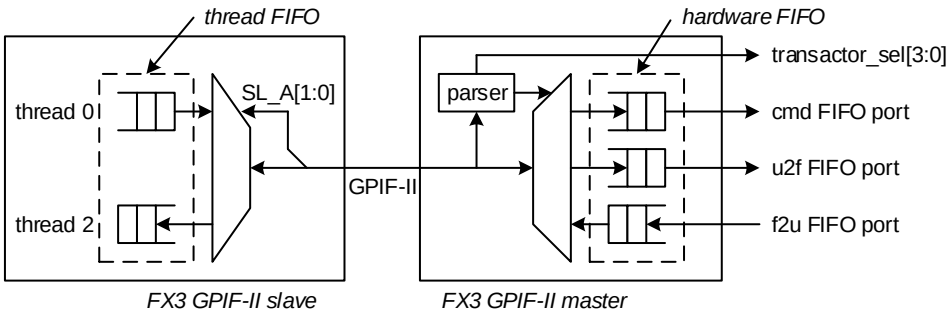

Figure 2: Pseudo-DMA mode

There are three types of communication as follows.

- Pushing cmd FIFO in the GPIF-II master (Command FIFO)
- Pushing u2f FIFO in the GPIF-II master (USB-to-FPGA data FIFO)

Copyright © 2018-2019 Future Design Systems, Inc.

- Popping f2u FIFO in the GPIF-II mater (FPGA-to-USB data FIFO)

As shown in
Figure 3, all communication packet starts with ‘control flit’, which is fed through thread 0 FIFO and then followed by data flits if required. Especially, the communication packet to get data from F2U FIFO starts with ‘control flit’ and then gets ‘data flits’ through thread 2 FIFO instead of thread 0 FIFO. It should be noted that control flits never shown to beyond gpif2mst.

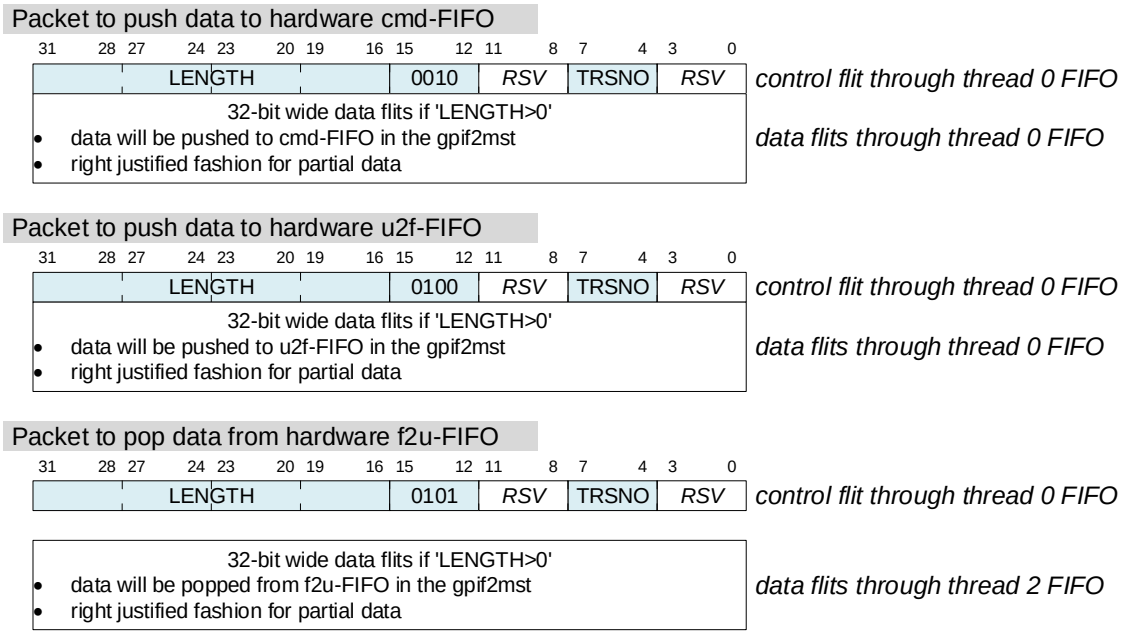

Figure 3: Communication packet formats

The meaning of field shown in
Figure 3 is as follows.

- control command (bit 15-12 of control flit) specifies type of control
  - ✧ 0x2: command packet
  - ✧ 0x4: u2f packet (USB-to-FPGA)
  - ✧ 0x5: f2u packet (FPGA-to-USB)
- ‘TRSNO[3:0]’ (bit 7-4 of control flit) specifies transactor
  - ✧ It simply driven to ‘transactor_sel’ pins of gpif2mst.
  - ✧ This can be used to select further interface beyond gpif2mst.
- ‘LENGTH[15:0]’ (bit 31-16) specifies the number of data flits to be followed
  - ✧ It reflects the number of payload in 32-bit units (i.e., word length).
- data flits
  - ✧ This is user payload.

When GPIF-II interface only uses 16-bit SL_DT[15:0], the lower 16-bit of packet is transferred first as little-endian fashion. This is why bit 15-12 is used for

Copyright © 2018-2019 Future Design Systems, Inc.

control command and as a result meaning of control flit can be parsed using first arriving information.

5.4.1 Communication scenario

Here is a rough scenario to send or receive data in terms of software.

In order to put data into hardware cmd-FIFO

1. make and send a control flit through thread 0 FIFO
   - ✧ With 0x2' control command along with corresponding data flit length
2. make and send data flits through thread 0 FIFO

In order to put data into hardware u2f-FIFO.

1. make and send a control flit through thread 0 FIFO
   - ✧ With 0x4' control command along with corresponding data flit length
2. make and send data flits through thread 0 FIFO

In order to get data from hardware f2u-FIFO.

1. make and send a control flit through thread 0 FIFO
   - ✧ With 0x5' control command along with corresponding data flit length
2. get data flits through thread 2 FIFO

5.4.2 Command write: conCmdWrite()

'conCmdWrite()' pushes command FIFO. It can be used when the operation mode is CON_MODE_CMD. It adds 32-bit control flit at the beginning of data.

Function prototype:

```
int conCmdWrite ( con_Handle_t   con_handle
                , void         *pBuffer
                , unsigned int   nNumberOfItemsToWrite
                , unsigned int  *pNumberOfItemsWritten
                , unsigned int   transactor );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ pBuffer: pointer to the buffer carrying command data
- ✧ nNumberOfItemsToWrite: the number of words (4-byte unit) in 'pBuffer'
- ✧ pNumberOfItemsWritten: the number of words actually written.
- ✧ transactor: 4-bit transactor id.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

It calls 'conUsbBulkTransfer()' twice internally.

Copyright © 2018-2019 Future Design Systems, Inc.

5.4.3 Data write: conDataWrite()

‘conDataWrite()’ pushes data FIFO. It can be used when the operation mode is CON_MODE_CMD. It sends control flit first and the sends data.

Function prototype:

```
int conDataWrite( con_Handle_t  con_handle
                , void        *pBuffer
                , unsigned int  nNumberOfItemsToWrite
                , unsigned int *pNumberOfItemsWritten
                , unsigned int  transactor );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ pBuffer: pointer to the buffer carrying data
- ✧ nNumberOfItemsToWrite: the number of words (4-byte unit) in ‘pBuffer’
- ✧ pNumberOfItemsWritten: the number of words actually written
- ✧ transactor: 4-bit transactor id

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

It calls ‘conUsbBulkTransfer()’ twice internally.

5.4.4 Data read: conDataRead()

‘conDataRead()’ pops data FIFO. It can be used when the operation mode is CON_MODE_CMD. It sends control flit first and then receives data.

Function prototype:

```
int conDataRead ( con_Handle_t  con_handle
                , void        *pBuffer
                , unsigned int  nNumberOfItemsToRead
                , unsigned int *pNumberOfItemsRead
                , unsigned int  transactor );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ pBuffer: pointer to the buffer to be filled
- ✧ nNumberOfItemsToRead: the number of words (4-byte unit) to read
- ✧ pNumberOfItemsRead: the number of words actually read
- ✧ transactor: 4-bit transactor id.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

Copyright © 2018-2019 Future Design Systems, Inc.

It calls ‘conUsbBulkTransfer()’ twice internally; one for write and the other for read.

5.5 Stream functions

This mode does not use cmd-FIFO. It simply pushes or pops data in stream fashion.

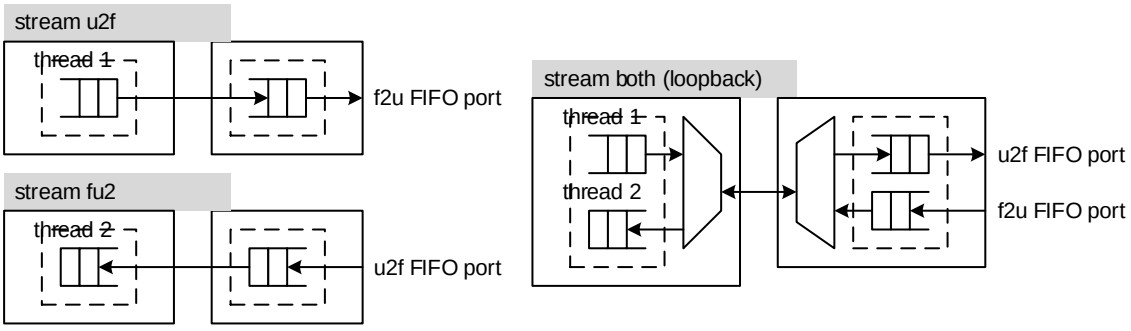

Figure 4: Stream modes

5.5.1 Stream write: conStreamWrite()

‘conStreamWrite()’ pushes data FIFO. It can be used when the operation mode is CON_MODE_SU2F or CON_MODE_SLOOP.

Function prototype:

```
int conStreamWrite( con_Handle_t  con_handle
                  , void         *pBuffer
                  , unsigned int  nNumberOfItemsToWrite
                  , unsigned int *pNumberOfItemsWritten
                  , unsigned int  zlp );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ pBuffer: pointer to the buffer carrying data
- ✧ nNumberOfItemsToWrite: the number of words (4-byte unit) in ‘pBuffer’
- ✧ pNumberOfItemsWritten: the number of words actually written
- ✧ zlp: append ZLP when 1.

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

It calls ‘conUsbBulkTransfer()’ once internally.

A multiple of wMaxPacketSize (Endpoint Descriptor) byte may not appear at one, so a ZLP (zero length packet) may be required to deal with this. The ZLP

Copyright © 2018-2019 Future Design Systems, Inc.

packet can be issued by submitting a transfer of zero length. Following is a good practice to deal with ZLP.

```
unsigned int zlp = (nNumberOfItemsToWrite%BulkMaxPktSizeOut) ? 0 : 1;
conStreamWrite(con_handle, pBuffer, nNumberOfItemsToWrite
                                    , pNumberOfItemsWritten, zlp);
```

5.5.2 Stream read: conStreamRead()

'conStreamRead()' pops data FIFO. It can be used when the operation mode is CON_MODE_SF2U or CON_MODE_SLOOP.

Function prototype:

```
int conStreamRead ( con_Handle_t  con_handle
                  , void         *pBuffer
                  , unsigned int  nNumberOfItemsToRead
                  , unsigned int *pNumberOfItemsRead );
```

Arguments:

- ✧ dev_handle: USB device hander
- ✧ pBuffer: pointer to the buffer to be filled
- ✧ nNumberOfItemsToRead: the number of words (4-byte unit) to read; it should be a multiple of BulkMaxPktSizeIn/4 words.
- ✧ pNumberOfItemsRead: the number of words actually read

Return value:

- ✧ 0 on success
- ✧ !=0 on failure, which will be

It calls 'conUsbBulkTransfer()' once internally.

It should be noted that this function reports packet overflow when there is more data than the requested in the buffer. So this function should be called with wMaxPacketSize at least.

6 Troubleshooting

6.1 Permission problem

Symptom:
When following message appears, check you udev file in '/etc/udev/rules.d' directory.

```
libusb: 0.000000 error [op_open] libusb couldn't open USB device
/dev/bus/usb/006/002: Permission denied.
libusb: 0.000020 error [op_open] libusb requires write access to USB device nodes.
cannot initialize CON-FMC
```

Solution:

Copyright © 2018-2019 Future Design Systems, Inc.

Make a file named ’51-fds-rule.rules’ in ‘/etc/udev/rules.d’ directory. The file looks like below, which consists of two lines.

```
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="04b4", ATTRS{idProduct}=="00f3", MODE=="0666"
SUBSYSTEM=="usb_device", ATTRS{idVendor}=="04b4", ATTRS{idProduct}=="00f3", MODE=="0666"
```

Do not forget to run followings or reboot your system.

```
$ sudo udevadm control --reload-rules
$ sudo service udev restart
$ sudo udevadm trigger
```

6.2 LibUsb problem

Symptom:
When LibUsb header is not found by the compiler.

```

```

Solution:
Use ‘pkg-config --cflags libusg-1.0’ to figure out ‘-I/path-to-libusb.h’

```
$ gcc source.c `pkg-config --cflags libub-1.0`
```

Symptom:
When LibUsb library is not found by the compiler.

```

```

Solution:
Use ‘pkg-config --libs libusg-1.0’ to figure out ‘-lusb-1.0’

```
$ gcc source.c `pkg-config --libs libub-1.0`
```

Symptom:
When LibUsb header and library are not found by the compiler.

```

```

Solution:
Use ‘pkg-config --libs --cflags libusg-1.0’ to figure out ‘-I/path-to-libusb.h’ and ‘-lusb-1.0

```
$ gcc source.c `pkg-config --libs --cflags libub-1.0`
```

6.3 libconapi problem

Symptom:
When ‘conapi.h’ header is not found by the compiler.

```

```

Solution 1:

Copyright © 2018-2019 Future Design Systems, Inc.

Add the directory ‘conapi.h’ residing to ‘C_INCLUDE_PATH’ and ‘CPLUS_INCLUDE_PATH’ environment variable or simply run ‘settings.sh’ in the CON-FMC installation directory

```
$ source /opt/confmc/2019.02/settings.sh
```

where ‘/opt/confmc/2019.02’ is an example and it should be the directory where CON-FMC package is installed.

Solution 2:

Specify the directory ‘conapi.h’ residing with ‘-I’ option of compiler

```
$ gcc … -I /opt/confmc/2019.02/include …
```

where ‘/opt/confmc/2019.02’ is an example and it should be the directory where CON-FMC package is installed.

Symptom:

When ‘libconapi’ library is not found by the compiler.

```
error while loading shared libraries: libconapi.so: cannot open shared object file: No such file or directory
```

Solution 1:

Add the directory ‘libconapi.so’ residing to ‘LD_LIBRARY_PATH’ environment variable or simply run ‘settings.sh’ in the CON-FMC installation directory

```
$ source /opt/confmc/2019.02/settings.sh
```

where ‘/opt/confmc/2019.02’ is an example and it should be the directory where CON-FMC package is installed.

Solution 2::

Specify the directory ‘libconapi.so’ residing using ‘-L’ and ‘-l’ options of compiler whiling linking stage

```
$ gcc … -L /opt/confmc/2019.02/lib/linux_x86_64 -lconapi
```

where ‘/opt/confmc/2019.02/lib/linux_x86_64’ is an example and it should be the directory where CON-FMC package is installed.

6.4 Problems while connection or using CON-FMC

Refer to ‘Trouble shooting’ section in the reference [3].

7 References

[1] Future Design Systems, Board Information Format for I2C PROM, FDS-TD-2018-04-001, 2018.

[2] Future Design Systems, Cypress EZ-USB FX3 GPIF-II Master Controller, FDS-TD-2018-04-002, 2018.

[3] Futue Design Systens, Cypress EZ-USB FX3 GPIF-II Firmware for CON-FMC, FDS-TD-2018-04-003, 2018.

[4] Future Design Systems, CON-FMC User Manual, FDS-TD-2018-03-001, 2018.

Copyright © 2018-2019 Future Design Systems, Inc.

[5] libusb-1.0: A cross-platform user library to access USB device (http://libusb.sourceforge.net/api-1.0/libusb_api.html), libusb.info.
[6] usbfs, USB device filesystem.
[7] WinUSB, a generic USB driver for Windows provided by Microsoft.
[8] Cypress Semiconductor, EZ-USB FX3 SuperSpeed USB Controller, CYUSB301X, 2012.
[9] Cypress Semiconductor, EZ-USB FX3 SDK Firmware API Guide, Version 1.3.3.
[10] Cypress Semiconductor, Cypress CyAPI Programmer's Reference, 2014.
[11] Cypress Seminconductor, Cypress CyUsb3.sys Programmer's Reference, 2012.
[12] Cypress Seminconductor, FX3 Programmers Manual, Doc. #001-64707 Rev. H, 2014.
[13] Cypress Seminconductor, GPIF II Designer 1.0, Doc. No 001-75664 Rev. C, 2013.
[14] Cypress Semincoductor, Designing with the EZ-USB FX3 Slave FIFO Interface, AN65974.
[15] Cypress Seminconductor, Designing a GPIF II Master Interface, AN87216.

Wish list

□

Index

Copyright © 2018-2019 Future Design Systems, Inc.

Revision history

- ☐ 2019.04.10: Adding more cases in the ‘Trouble shooting’ section.
- ☐ 2019.02.09: Updated
- ☐ 2018.03.10: Started by Ando Ki (adki@future-ds.com)

– End of document –

Copyright © 2018-2019 Future Design Systems, Inc.