PyCONFMC: CON-FMC Python Binding

Version 0 Revision 1

Feb. 7, 2019 (Feb. 7, 2019)

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

Abstract

This document addresses Python binding, which allows the user level API (Application Programming Interface) of CON-FMC to be called from Python.

Table of Contents

1 Overview

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

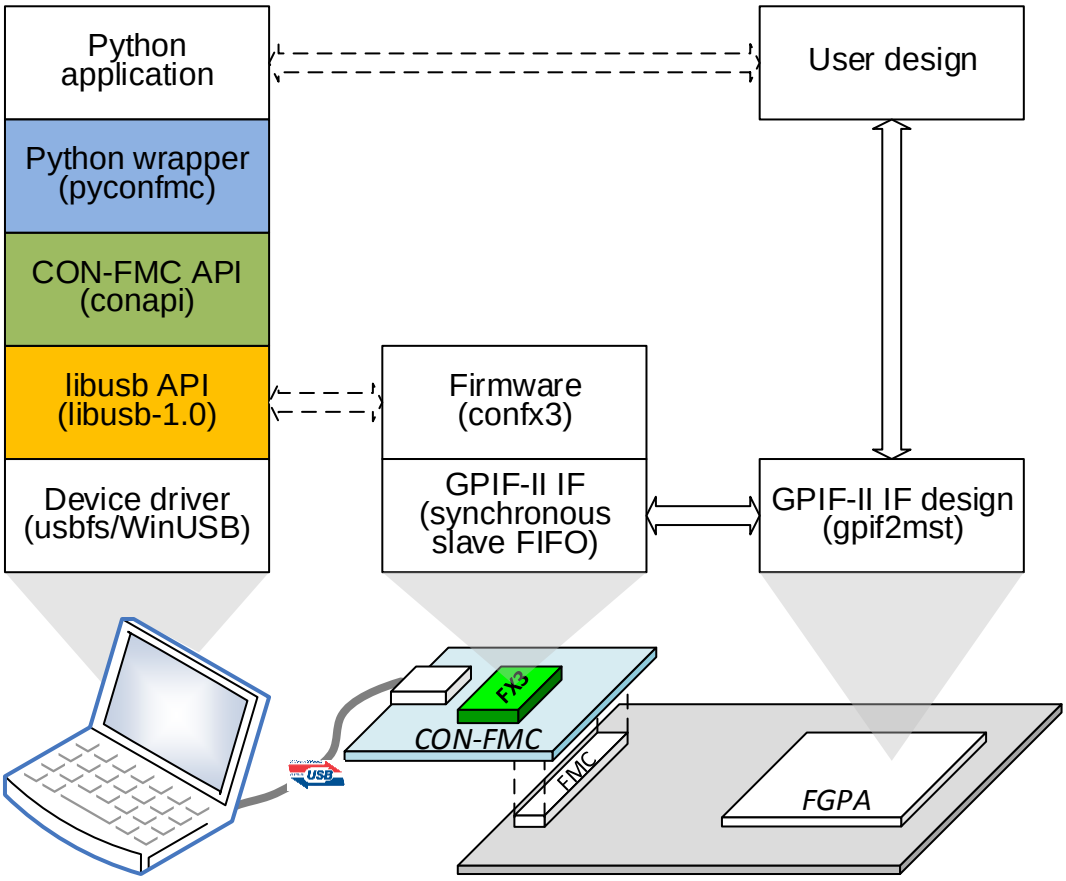

Figure 1: Overview

Licensing issues:

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

Python bindings provide support for importing CON-FMC API libraries as Python modules. Coverage of most of CON-FMC C API is provided. The intent has been to allow the programmer to write complete manipulation scripts in Python, to allow integration of CON-FMC with other Python tools and workflows.

2 Getting started

As shown in code below, ‘confmc.pyconfmc’ should be imported to use PyCONFMC and its basic steps are as follows.

1. import module, i.e., ‘confmc.pyconfmc’.
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().

```
import sys
import confmc.pyconfmc as confmc

hdl = confmc.conInit()
if not hdl: sys.exit(1)

cid = confmc.conGetCid(hdl)
if cid<0: sys.exit(1)

print(“CON-FMC:” + str(cid) + “found.”)

confmc.conRelease(hdl)
```

To run example Python program say ‘test.py’, do as follows and do not forget to be ready of CON-FMC HW with card id (CID) 0.

```
$ python test.py
```

3 Python API

3.1 Initialize and release

3.1.1 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’.

Function prototype:

```
conInit(con_cid=0, con_mode=0, conapi_log_level=0)
```

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

3.1.2 Release: conRelease()

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

Function prototype:

```
conRelease ( con_handle )
```

Arguments:
✧ con_handle: CON-FMC handler.

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

It sets operation mode to 'CON_MODE_CMD' and calls 'conResetEp()' for input endpoint, when operation mode is 'CON_MODE_SU2F'.

3.2 Utility

3.2.1 Read CID: conGetCID()

'conGetCID()' reads CID.

Function prototype:

```
conGetCid ( con_handle )
```

Arguments:
✧ con_handle: CON-FMC handle

Return value:
✧ 0~7 on success
✧ <0 on failure

3.2.2 Reset: conReset()

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

Function prototype:

```
conReset ( con_handle, 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

3.2.3 Mode: conSetMode()

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

Function prototype:

```
conSetMode ( con_handle, con_mode )
```

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.

3.2.4 Master information: conGetMasterInfo()

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

Function prototype:

```
conGetMasterInfo ( con_handle, pInfo )
```

Arguments:

- ✧ con_handle: CON-FMC handler.

- ✧ pInfo: pointer to structure to store command FIFO depth.

Return value:

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

3.2.5 Board information: conGetBoardInfo()

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

Function prototype:

```
conGetBoardInfo( con_handle, pInfo
                 , length=ctypes.sizeof(con_BoardInfo)
                 , crc_check=0 )
```

Arguments:

- ✧ con_handle: CON-FMC handler.
- ✧ pInfo: pointer to the board information data structure.
- ✧ length: the number of byte of the structure pointed by pInfo
- ✧ 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.

3.2.6 Get API version: conGetVersionApi()

'conGetVersionApi()' returns CONAPI version.

Function prototype:

```
conGetVersionApi( )
```

Arguments:

Return value:

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

3.2.7 FX3 information: conGetFx3Info()

'conGetFx3Info()' returns FX3 firmware version.

Function prototype:

```
conGetFx3Info ( con_handle, pInfo )
```

Arguments:

- ✧ con_handle: CON-FMC handle

- ✧ pInfo: pointer to the structure

Return value:

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

3.3 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.

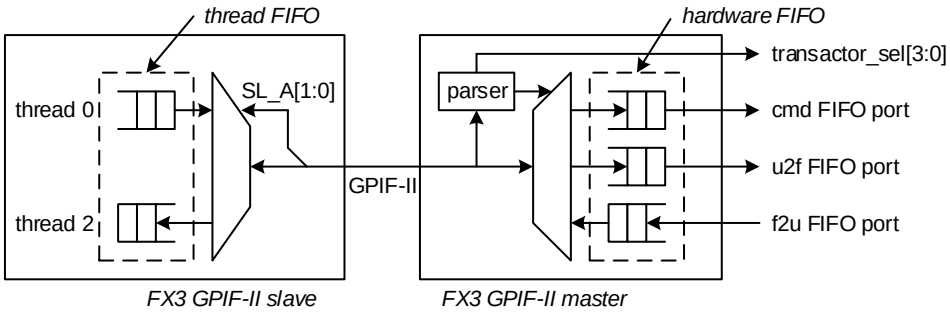

Figure 2: Pseudo-DMA mode

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)
- 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.

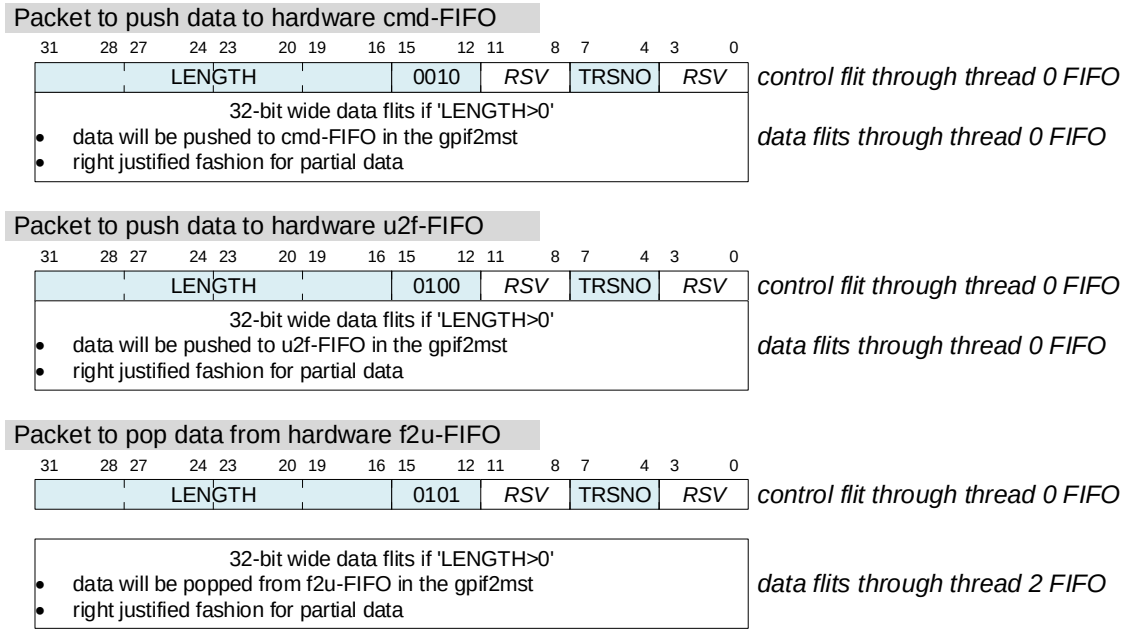

Figure 3: Communication packet formats

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

3.3.1 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

3.3.2 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:

```
conCmdWrite( con_handle, pBuffer
                       , nNumberOfItemsToWrite
                       , pNumberofItemsWritten
                       , 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.

3.3.3 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:

```
conDataWrite( con_handle, pBuffer
                       , nNumberOfItemsToWrite
                       , pNumberofItemsWritten
```

```
                    , 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.

3.3.4 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:

```
conDataRead( con_handle, pBuffer
                    , nNumberOfItemsToWrite
                    , pNumberofItemsWritten
                    , 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

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

4 Troubleshooting

4.1 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:
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
```

4.2 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`
```

4.3 Python virtualen

It is highly recommended to use virtual environment for different versions of Python due to version incompatibility problem.

4.3.1 for Ubuntu

Install Python Virtualenv and then create virtual environments for different versions of Python. If 'pip' is not available, then install it first '$ sudo apt-get install python-pip'.

```
$ sudo apt-get update
$ sudo pip install virtualenv
```

```
$ virtualenv -p python2 <dir for python2 virtual environment>
$ virtualenv -p python3 <dir for python3 virtual environment>
```

Do not forget to activate Python virtual environment before using it.

```
$ source <dir for python? virtual environment>/bin/activate
```

Do not forget to deactivate Python virtual environment before leaving it.

```
$ deactivate
```

4.3.2 for CentOS

Install Python Virtualenv and then create virtual environments for different versions of Python. If 'pip' is not available, then install it first '$ sudo apt-get install python-pip'.

```
$ sudo pip install virtualenv
```

```
$ virtualenv -p python2 <dir for python2 virtual environment>
$ virtualenv -p python3 <dir for python3 virtual environment>
```

Do not forget to activate Python virtual environment before using it.

```
$ source <dir for python? virtual environment>/bin/activate
```

Do not forget to deactivate Python virtual environment before leaving it.

```
$ deactivate
```

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

Wish list

☐

Index

Revision history

- 2018.03.10: Started by Ando Ki (adki@future-ds.com)

– End of document –