Table of Contents

Fx3 Firmware

This page shows how to use the modified Fx3 Firmware, which provides a UART interface and Endpoint access via libusb. In addition, the usage of the software to access the endpoints is described.

Driver verification on Linux

In general, Linux loads the correct drivers automatically, without user intervention. To check if the correct drivers are used, after uploading the firmware the lsusb command can be used:

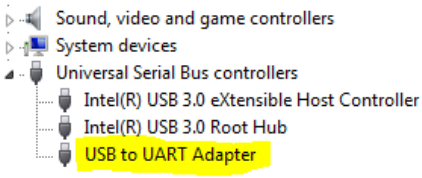
```
$lsusb
...
Bus 004 Device 004: ID 04b4:0008 Cypress Semiconductor Corp.
...
$lsusb -t
/:  Bus 04.Port 1: Dev 1,, Driver=xhci_hcd/6p, 5000M
    |__ Port 1: Dev 4, If 0,, Driver=cdc_acm, 5000M
    |__ Port 1: Dev 4, If 1, Data, Driver=cdc_acm, 5000M
    |__ Port 1: Dev 4, If 2, Specific Class, Driver=, 5000M
```

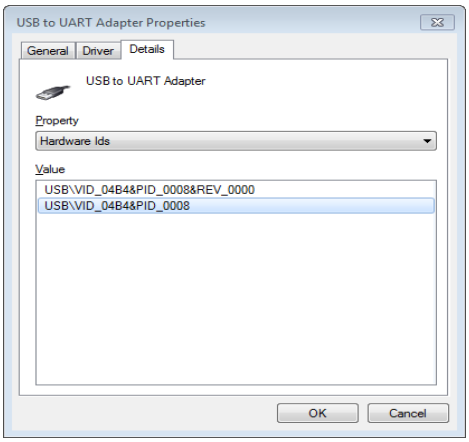
The driver class cdc_acm provides access to the serial interface; for libusb no additional driver is required. The serial port should be registered as "/dev/ttyUSBx" or "/dev/ttyACMx"

Installing Usb Driver on Windows

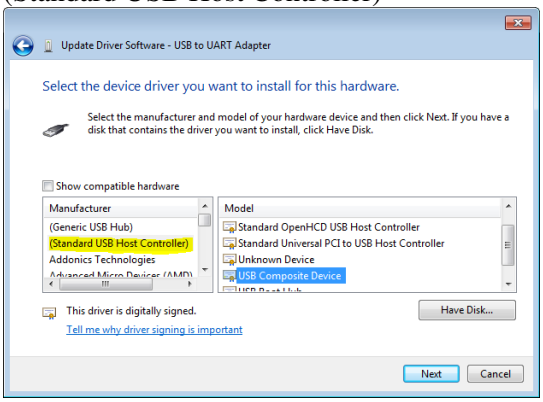
For accessing both serial port and usb bulk interfaces on windows, a composite driver has to be installed. This can be done by executing the following steps in the Windows Device Manager:

1. Upload the firmware image (SlaveFifoSync_uart.img) to the Fx3 chip. Wait for Windows to install default drivers
2. Switch the driver of the Usb controller to 'Composite device'
    1. Select the correct device from the 'Universal Serial Bus controllers' list
    2. This is most likely a "Usb To Uart Adapter" device. The correct hardware id should be "USB\VID_04B4&PID_0008"

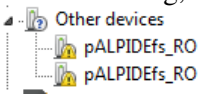




    3. Update the driver for the device
        1. Select Update Driver Software in the Context menu of the device
        2. Select "Browse my computer for driver software"
        3. Select "Let me pick from a List of device drivers on my computer
        4. The correct driver is "USB Composite Device".
        5. If not listed as compatible hardware, uncheck the box and select it in the group (Standard USB Host Controller)

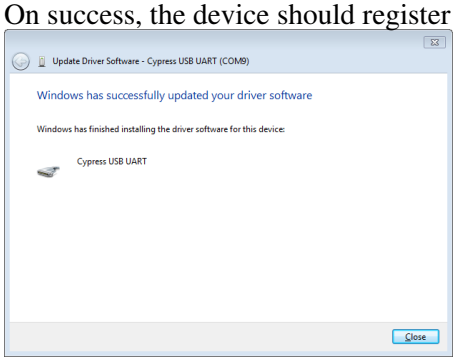


        6. install the selected driver
3. After refreshing, 2 new devices will be found, listed as "pALPIDEfs_RO"

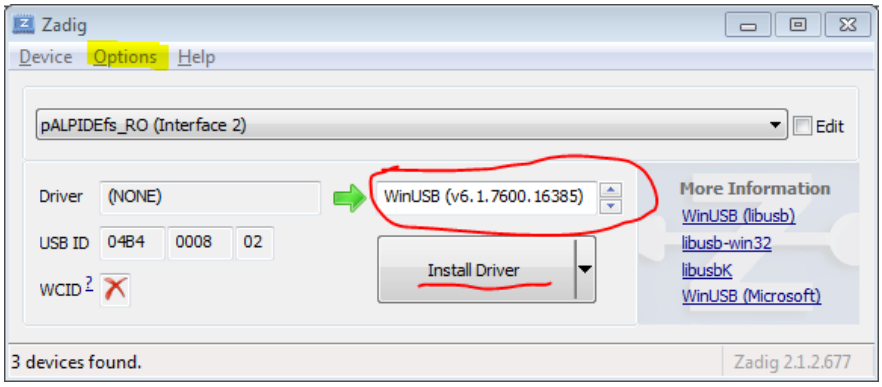


4. Select the device with hardware id "USB\VID_04B4&PID_0008&MI_00", which is interface 0 (the virtual com port)
5. Update the driver for this device
    1. Select "Browse my computer for driver software"
    2. Search for a driver at a specific location
    3. Download the attached driver_win.inf file and browse to the directory where it is saved
    4. Windows will give a warning that the driver is not verified; install it anyway

5. On success, the device should register as "Cypress USB UART" with a virtual COM port



6. Update and install the driver for USB access via usb_comm software
    1. With libusb-1.0 (which is used by the chip interface) a default WinUsb driver is used.
    2. An easy way for installation is using the tool Zadig. The executable can be found on the website or on the WP10 Cernbox (software/Cypress_Fx3/usb_comm_win).
    3. After starting Zadig, select "Options/List all devices".
    4. In the Dropdown menu, the pALPIDEfs_RO device should now appear with 2 interfaces.
    5. Select the pALPIDEfs_RO interface 2 device, check that WinUSB is selected in the driver list and press "Install Driver" or "Replace Driver" (depending if windows installed a default driver).



    6. Back in the device manager, the Device should now be listed in your device manager as "Universal Serial Bus Device"

Device access from software

To access the interfaces from software, either serial connection or libusb is used.

Serial access

The firmware is designed to use the "CDC Serial" interface, which means the settings of the COM port are dynamic and can be set to fit the target device connected to the UART port of the Fx3.

Access via usb_comm

The usb_comm software is used to handle the USB communication between host and Fx3. It handles USB transfers for all endpoints provided. For accessing the endpoints, it provides network sockets, which can be connected to via TCP streams.

Build and installation

The program is written in C++, using libusb for USB access and boost for network code. The source code can be found in the WP10 git repository (software/usb_comm). Build instructions and requirements are found in the folder as well.

For simplification, binaries are provided in the WP10 cernbox, under "Software/Cypress_Fx3/usb_comm". An additional folder for a windows distribution is provided.

Running the usb_comm software

The software runs as console application. Command line parameters are set in unix style:

```
./usb_comm -h
Usb Chip Interface

Basic options:
  -h [ --help ]         Print this help message
  -v [ --verbose ]      Enable Verbose output
  --v arg               Verbose output up to level x (call with --v=x)

Network:
  --port_control arg (=30000) Port for control communication
  --port_data0 arg (=30001)   Port for Data communication on Dataport 0
  --port_data1 arg (=30002)   Port for Data communication on Dataport 1

Usb Device options:
  --vendor_id arg (=04B4)         Vendor Id
  --product_id arg (=00F0)        Product Id
  --interface arg (=0)            Interface number
  --endpoint_control_out arg (=1) Control endpoint OUT (write)
  --endpoint_control_in arg (=1)  Control endpoint IN (read)
  --endpoint_data0_in arg (=2)    Data endpoint 0 IN (read)
  --endpoint_data1_in arg (=3)    Data endpoint 1 IN (read)
```

For debugging output (especially first connection), the Verbose flag ("-v") should be set. In addition, vendor_id, product_id and interface have to be set correctly. For the provided firmware, a software startup would look like the following:

```
$ ./usb_comm --product_id 0008 --interface 2 --endpoint_control_out 3 --endpoint_control_i
2015-12-07 10:44:20,866 INFO  [default] Starting application with following parameters set
2015-12-07 10:44:20,867 INFO  [default] Network Ports: Control(30000), Data0(30001), Data1
```

```
2015-12-07 10:44:20,867 INFO  [default] Usb Device: VID(04B4), PID(0008), Interface(2)
2015-12-07 10:44:20,867 INFO  [default] Endpoints: Control Out(3), Control In(3), Data0 In
2015-12-07 10:44:20,867 INFO  [default] Wait for stop
2015-12-07 10:44:20,867 INFO  [default] Wait for connection on port 30001 (1)
2015-12-07 10:44:20,867 INFO  [default] Wait for connection on port 30000 (1)
2015-12-07 10:44:20,867 INFO  [default] Wait for connection on port 30002 (1)
```

After successful startup, the application waits for client connections on different Ports (default: 30000 for Control, 30001 for Data port 1 and 30002 for data port 2).

Connections are handled one at a time, with no simultaneous client access possible.

Client access

To access the software via clients, a TCP4 socket connection is required. A simple example for access is using the ncat client:

```
#Write all data received from a dataport to a file
ncat -4 localhost 30001 > data0.txt
```

Connection example for python:

```
#!/usr/bin/env python
import socket
import sys
from struct import unpack

PORT_CONTROL = 30000
PORT_DATA0 = 30001
PORT_DATA1 = 30002
HOSTNAME = 'localhost'

class DAQ:
    def __init__(self, ctlOnly = False):
        self.socketControl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socketControl.connect((HOSTNAME,PORT_CONTROL))
        if ctlOnly:
            self.socketData0 = None
            self.socketData1 = None
        else:
            self.socketData0 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socketData0.connect((HOSTNAME,PORT_DATA0))
            self.socketData1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socketData1.connect((HOSTNAME,PORT_DATA1))

    def __del__(self):
        self.closeConnections()

    def closeConnections(self):
        try:
            for sock in (self.socketControl,self.socketData0,self.socketData1):
                if sock is not None:
                    sock.shutdown(socket.SHUT_RDWR)
                    sock.close()
        except:
            print ("Error while shutting down sockets")


    def write_reg(self,mod_addr,sub_addr,data):
        assert mod_addr|0x0F==0xF
        assert sub_addr|0xFF==0xFF
```

```
        write = bytearray([data>>0&0xFF, data>>8&0xFF, sub_addr, mod_addr|0x80])
        assert self.socketControl.sendall(write) is None

    def read_reg(self,mod_addr,sub_addr):
        assert mod_addr|0x0F==0xF
        assert sub_addr|0xFF==0xFF

        write = bytearray([0x00, 0x00, sub_addr, mod_addr])
        assert self.socketControl.sendall(write) is None
        data = self.socketControl.recv(4)
        ret = bytearray(data)

        #assert len(ret)==4
        #assert ret[3]==0 and ret[2]==0
        return ret[0]|ret[1]<<8

    def read_data(self,socket,length):
        remaining = length
        assert length % 4 == 0
        msg = b""
        while remaining > 0:
            chunk = socket.recv(remaining)
            msg += chunk
            remaining -= len(chunk)


        intarr = unpack('i'*int(length/4),msg)

        return intarr

    def read_data0(self,length):
        return self.read_data(self.socketData0,length)

    def read_data1(self,length):
        return self.read_data(self.socketData1,length)


daq=DAQ()

#SLV_ADDR | REG_ADDR | DATA
daq.write_reg(0x1,0x01,0xFAFA)

#SLV_ADDR | REG_ADDR
print "0x%08x" % daq.read_reg(0x1,0x01)
```

This topic: ALICE > SwReadoutUnitUsbConnection
Topic revision: r6 - 2015-12-15 - MatthiasBonora