Reference firmware: preloaded USB Firmware (FX2 USB microcontroller's EEPROM)

The reference firmware is used to:

- initialize VID and PID with the Productor (Trenz Electronic) Values;
- setup interrupt table and setup/install handlers (aka functions) called to manage the interrupt set in interrupt table
- initialize the USB FX2 microcontroller register, memory, etc..;
- enter in infinite loop (aka super-loop) that manage incoming USB setup data packet and conect the TE USB FX2 module (both FX2 microcontroller and FPGA's MicroBlaze) with host computer's software though USB;
- control the FPGA status (power on/off, etc) from host computer's software though USB;
- read/write the EEPROM (change FX2 microcontroller's firmware aka Firmware Layer) from host computer's software though USB;
- read/write the SPI Flash (change FPGA image aka Logic Architecture Layer) from host computer's software though USB;
- send SPI Flash commands to SPI Flash from host computer's software though USB;
- read/write the device connected to I2C bus from host computer's software though USB;

This activity are (with the exception of first four point) realized by TE API Commands (FW APIs) and can be controlled by SW API Layer's functions.

USB FX2 microcontroller reference firmware does NOT support Slave Parallel (SelectMAP) Mode and/or Slave Serial Mode; if the user needs to use these configuration modes, he/she should write a custom firmware that load the configuration data from a source (SPI Flash, USB connection or B2B connection) and write the retrieved configuration data in the FPGA.

Reference firmware description

In the following table is described the tasks executed by the reference firmware (Trenz Electronic v3.02).

step	pseudocode function	description
1	interrupt_disable(); //EA=0	Disable interrupts
2	setup_autovectors ();	Setup interrupt table
3	usb_install_handlers ();	Setup/install handlers (aka functions) called to manage the interrupt set in interrupt table
4	interrupt_enable(); //EA=1	Enable interrupts
5	fx2_renumerate();	Simulates disconnection and then reconnection of TE USB FX2 module USB cable with host computer
6	task0(); //system_init();	Initialization phase. The reference firmware is used to: • initialize VID and PID with the Productor (Trenz Electronic) Values; • initialize the USB FX2 microcontroller register, memory, etc.. to (possibly) establish a connection of the TE USB FX2 module (FX2 microcontroller) with host computer's though USB. Step 1,2,3,4,5 can be called pre_task0() or pre_system_init().
7	while(1) {	Enter the **superloop**
8	task1();	Manage incoming USB setup data packet. See here. It is used in connection and disconnection (of TE USB FX2 module) phases from USB host computer: it is used to establish/remove a connection of the TE USB FX2 module (FX2 microcontroller) with host computer's though USB .

9	task2();	Manage incoming USB data packet conforming to TE (FW) API aka USB "command/reply" packet. It is used: • to control the FPGA status (power on/off, etc) • to read/write the EEPROM (change of FX2 microcontroller's firmware aka Firmware Layer) • to read/write the SPI Flash (change of FPGA image aka Logic Architecture Layer) • to send SPI Flash commands to SPI Flash • to read/write the device connected to I2C bus • to read/write the TE USB FX2 module's DRAM from host computer's software though USB. The previous activities are realized by TE API Commands (FW APIs) and can be controlled by SW API Layer's functions.
10	task3();	Manage interrupt request (INT0 pin) incoming from FPGA chip (normally, FPGA's MicroBlaze soft-processor): used with (some) MB Command
11	}	While running the FX2 microcontroller should never exit from the superloop and repeat (until a reset) step 8,9 and 10

Reference firmware (Trenz Electronic v3.02 description): pseudocode

Super loop

In this super loop ("while(1)") reference firmware is used to:

1. manage incoming USB setup data packet (used for device registration by OS of USB host);
2. manage incoming USB data packet (this data packet should normally contains TE API Commands (FW APIs));
3. manage interruption (INT0 pin) incoming from FPGA chip (normally FPGA's MicroBlaze soft-processor).

Task 1 is implemented (carried out) by

fw.c, Trenz Electronic v3.02

```
if(usb_setup_packet_avail()) usb_handle_setup_packet();
```

Task 2 is implemented (carried out) by

te_api.c, Trenz Electronic v3.02

```
void ep1_pool(void)
```

Task 3 is implemented (carried out) by

te_api.c, Trenz Electronic v3.02

```
void int_pin_pool(void)
```

Task 2 and task 3 are grouped in a single function called activity() running in the superloop of fw.c;

te_api.c, Trenz Electronic v3.02

```
void activity(void){
        ep1_pool();
        int_pin_pool();
}
```

Manage incoming USB setup data packet (task 1)

Handles the setup package and the basic device requests like reading descriptors, get/set configuration etc. It is used in phase of connection /disconnection of the TE USB FX2 module from host computer's USB port. It is also used by CyControl, CyConsole (and other similar programs) to retrieve information about the configuration of the USB device (TE USB FX2 module view as USB device).

See this link for further explanations.

At this moment (Trenz Electronic v3.02), this part of code does not calls the app_class_cmd or app_vendor_cmd functions when needed, because CLASS command and VENDOR commands are not supported.

Even if user/devloper's firmware sets RENUM bit to 1, meaning that it wants to handle vendor commands by itself, the user/developer can still replace the current running firmware with another one.

Manage incoming USB data packet (task2)

This process is implemented (carried out) by function ep1_pool().

This function pull 64 bytes from EP1OUTBUF FIFO; in this FIFO are stored possible TE API Commands (FW APIs) sent by host computer's SW through USB connection.

EP1OUTBUF[0] and EP1OUTBUF[1:63] are written by host computer's software C++ TE_USB_FX2_SendCommand(...,command,...) or C# TE_USB_FX2_SendCommand(...,command,...) or libusb(x) C libusb_bulk_transfer(usbDeviceHandle, LIBUSB_ENDPOINT_OUT | 1, command, x, &actual_length, 1000) used with command[0] = USB FX2 API Command.

ep1_pool() function uses a switch construct (EP1OUTBUF[0] decoded as USB FX2 API Command) to select the proper code (calling other functions,if any) to execute the task requested by the decoded USB FX2 API Command .

TE API decoder pseudocode, Trenz Electronic v3.02

```
Test data for internal test;
if (in the EP1 OUT FIFO exist at least one USB command packet not yet readed)
{
        Increment command count;
        Fill output buffer EP1INBUF with 0xFF value;
        switch(EPOUTBUF[0])         //Decode USB FX2 API Command correponding to received EPOUTBUF[0]
            default:                // EPOUTBUF[0] different from any USB FX2 API encoded
value
                        Code to execute if EPOUTBUF[0] does not equal the value of any of the
cases
                        break;
       Â ...
                case USB_FX2_API_0xYZ:  // EPOUTBUF[0] = USB_FX2_API_0xYZ , USB_FX2_API encoded value
hexadecimal YZ
                        Run the correspoding code snippet/function with auxiliary functions (if any)
                        Write the reply (if any) in EP1INBUF
                        Set a flag_new_data to indicate that a USB reply packet could be delivered to host
computer's USB
                        break;
                ...

        Free input buffer;
    if (flag_new_data = 1)    //Check the flag to know if a USB reply packet should be delivered to host
computer's USB
        {
                if(it is possible to send the USB reply packet using EP1)
                {
                        Send the USB reply packet using EP1
                        Reset the flag flag_new_data to indicate that a USB reply packet has been delivered to
host computer's USB
                        //the buffer is free for use
                }
        }
}
```

Manage interruption (INT0 pin) incoming from FPGA chip (task 3: Pull INT pool)

Mainly used with XPS_I2C_SLAVE custom IP block and the FW APIs SET_INTERRUPT command, GET_INTERRUPT command.

Pull INT pool. Brief description.

If

- an autoresponse interrupt is preconfigured (in the ep1_pool()) by host computer's SW and
- an FPGA_INT0 "interrupt" rised by FPGA chip should be managed (FPGA_INT0=1),

the FX2 microcontroller pull (using I2C in int_pin_pool()) x (x defined by EP1OUTBUF[2]) number of bytes from y I2C address (y defined by EP1OUTBUF [1]).

The host computer's SW should use a polling procedure to retrieve the I2C bytes read (and stored) by FX2 microcontroller (the pull response to host computer's pull is implemented (carried out) by task2: ep1_pool())

Pull INT pool. Longer description true for every Logical Architecture Layer

If

- an autoresponse interrupt is preconfigured (by host computer's SW using SET_INTERRUPT => CMD_SET_AUTORESPONSE => sts_int_auto_configured = 1) and
- an FPGA_INT0 "interrupt" risen by FPGA chip should be managed (FPGA_INT0=1),

the FX2 microcontroller firmware reads (using I2C) a maximum of 32 byte (in the firmware a maximum of 12 is preconfigured but it could be overwritten by a host software FX2_API_Command SET_INTERRUPT => CMD_SET_AUTORESPONSE => iar_adress = EP1OUTBUF[1]; iar_count = EP1OUTBUF[2];) from an I2C address.

The I2C bytes data are copied in the byte array auto_response_data by the int_pin_pool() firmware function.

This byte array coul be pulled out by host computer'SW using SET_INTERRUPT (=> CMD_GET_AUTORESPONSE => for(i = 0; i < 32; i++) EP1INBUF [i+1] = auto_response_data[i];).

Pull INT pool. Longer description true for Reference Design: Logical Archirecture Layer = Reference Architecture Layer

It is usually used with XPS_I2C_SLAVE custom IP block (LINK) for command, settings and status communication. When MicroBlaze write data to MB2FX2_REG0, the interrupt pin INT0 (aka FPGA_INT0 in firmware files) is rised. This pin is connected to PA0/INT0 pin of FX2 microcontroller. When the FX2 microcontroller's firmware read the rise of pin INT0 (=1 because MicroBlaze writes data to MB2FX2_REG0) it set the firmware variable FPGA_INT0 to 1.

If

- an autoresponse interrupt is preconfigured (sts_int_auto_configured = 1) and
- an FPGA_INT0 "interrupt" should be managed (FPGA_INT0=1),

the FX2 microcontroller firmware reads (using I2C) all MB2FX2 registers (12 bytes).

The registers value are copied in the byte array auto_response_data by the int_pin_pool() firmware function.

This byte array could be pulled out by host computer's SW using SET_INTERRUPT (=> CMD_GET_AUTORESPONSE => for(i = 0; i < 32; i++) EP1INBUF [i+1] = auto_response_data[i];).