FPGA4U: FX2 Firmware Implementation

Jérôme Maye

March 11, 2007

Contents

Chapter 1

Introduction

This document describes the implementation of the FX2 firmware, as it was developed for the FPGA4U project during an internship in the summer 2006. The main goal of the FX2 chip is to provide an efficient replacement of the USB-Blaster cable from Altera. This cable allows the connection between an USB port of a PC and the JTAG port of an FPGA. It can serve for the programming of the FPGA, for the downloading of an object code that will be run by a softcore processor, for the debugging of this code, or for providing a serial interface to the FPGA.

Besides its primary intent, the FX2 can be employed to provide a fast USB interface between the FPGA and a host computer. It is particulary interesting in the case the frames of a camera, connected to the FPGA, have to be acquired by the PC. So-called FIFO interface is used for this purpose. The FX2 can also serve as a stand-alone microprocessor for the teaching of embedded systems.

In the rest of this document, the FX2 is firstly briefly introduced. The details of the implementation of the firmware are then clearly outlined.

Chapter 2

Short Description of the FX2

The Cypress Semiconductor EZ-USB FX2 is a single-chip solution that provides an USB 2.0 interface. It is composed of a combination of 8051-based CPU and an USB interface (see Figure 2). The Serial Interface Engine (SIE) is able to perform a full USB enumeration independently of the microprocessor. This allows the software of the 8051 to be downloaded via USB. After the code is running, the chip can re-enumerate under another identity.

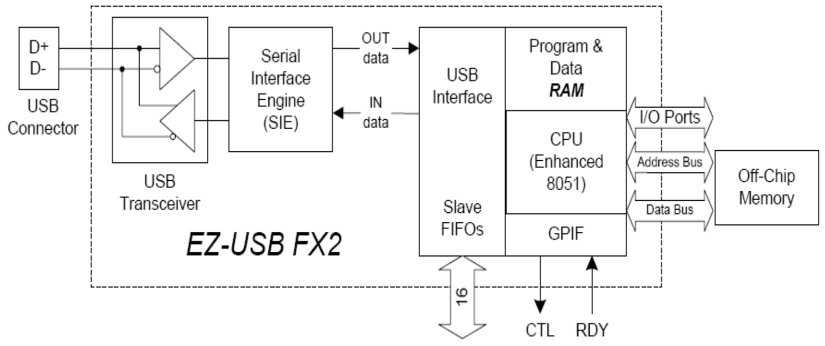

Figure 2.1: FX2 architecture.

An optional EEPROM can be attached to the I^{2}C lines of the microcontroller. Depending on the first byte of the EEPROM at power-up, the chip can either enumerate as a standard Cypress device, enumerate as a custom device whose characteristics follow in the EEPROM, or load a firmare from the EEPROM into the microcontroller.

To benefit from the high speed of the USB 2.0 standard, it is necessary to bypass the CPU of the FX2 when transferring data from the host. For this reason, the FX2 contains four slave FIFOs. The reading/writing of the FIFOs is done through a bus interface on the side of the device. The FIFOs transfers are automatically performed by the FX2.

For more details, we refer to the exhaustive FX2 Technical Reference Manual[1].

[1] `http://www.keil.com/dd/docs/datashts/cypress/fx2_trm.pdf`

Chapter 3

Firmware

The FX2 mainly aims at replacing the expensive USB-Blaster cable from Altera. Starting from an ingenious framework found on the Internet[1], the FX2 firmware has been enhanced with the features that suit our hardware platform. Shortly, the firmware firstly initializes the USB interface. USB events are then handled by specific functions in the file `usbjtag.c`.

On the FPGA4U board, the JTAG port of the FPGA is accessible by the FX2 chip or by an external connector. To avoid shortcuts, the FX2 has to enable passers before using the JTAG port. In the file `usbjtag.c`, the declarations `sbit JTAG_EN = 0xA0+7` and `#define bmJTAG_EN bmBIT7` are added for this purpose. To allow an output on the JTAG enable pin, we use `OEC = bmJTAG_EN` in the function `TD_Init`. The JTAG is enabled by default with the command `JTAG_EN = 1` in `TD_Init`. To dynamically change this setting, the vendor command 0x92 is used and this is implemented in the function `DR_VendorCmnd`.

FIFO 6 is used as an OUT endpoint, while FIFO 8 as an IN endpoint. The FIFOs are configured as follows:

```
EP1OUTCFG = 0xA0;
EP1INCFG = 0xA0;
SYNCDELAY;                   // see TRM section 15.14
EP2CFG = 0xA2;
SYNCDELAY;                   //
EP4CFG = 0xA0;
SYNCDELAY;                   //
EP6CFG = 0xA2;               // OUT Endpoint for FIFO mode
SYNCDELAY;
EP8CFG = 0xE0;               // IN  Endpoint for FIFO mode
// out endpoints do not come up armed
// since the defaults are double buffered we must write dummy byte counts twice
SYNCDELAY;                   //
EP2BCL = 0x80;               // arm EP2OUT by writing byte count w/skip.
```

[1] http://www.ixo.de/info/usb_jtag/

```
SYNCDELAY;                    //
EP4BCL = 0x80;
SYNCDELAY;                    //
EP2BCL = 0x80;                // arm EP4OUT by writing byte count w/skip.
SYNCDELAY;                    //
EP4BCL = 0x80;
SYNCDELAY;

FIFORESET = 0x80; // set NAKALL bit to NAK all transfers from host
SYNCDELAY;
FIFORESET = 0x06; // reset EP6 FIFO
SYNCDELAY;
FIFORESET = 0x08; // reset EP8 FIFO
SYNCDELAY;
FIFORESET = 0x00; // clear NAKALL bit to resume normal operation
SYNCDELAY;

EP6FIFOCFG = 0x00;                   // firmware has to see a rising edge on auto bit
SYNCDELAY;
EP6FIFOCFG = bmAUTOOUT | bmWORDWIDE; // auto commitment, 16 bits data bus
SYNCDELAY;
EP8FIFOCFG = 0x00;                   // firmware has to see a rising edge on auto bit
SYNCDELAY;
EP8FIFOCFG = bmAUTOIN  | bmWORDWIDE; // auto commitment, 16 bits data bus
SYNCDELAY;

// size in bytes of the IN data automatically commited (512 bytes here)
// can use signal PKTEND if you want to commit a shorter packet
EP8AUTOINLENH = 0x02;
SYNCDELAY;
EP8AUTOINLENL = 0x00;
SYNCDELAY;

// put the system in FIFO mode by default
// internal clock source at 48Mhz, drive output pin, synchronous mode
// NOTE: Altera USB-Blaster does not work in another mode
IFCONFIG = bmIFCLKSRC | bm3048MHZ | bmIFCLKOE | bmIFCFG1 | bmIFCFG0;
```

In the enumeration process, the devices cannot draw more than 100 mA. Therefore, the FPGA has to be disabled in the `TD_Init` function with the code:

```
// set port E output enable
OEE = (1 << 6);

// disconnect the 1.2V converter, so the FPGA do not start
IOE = (1 << 6);
```

When the host has allowed the device to enumerate, the FPGA can be started. The event is catched by the function `DR_SetConfiguration`. In this latter, the command `IOE &= (̃1 << 6)` is added.

When an external power is connected, the FX2 asks 0 mA to the host. In `TD_Init`, this is done with the following code:

```
// check external power connector
// change power requirement accordingly
if (IOE & (1 << 7))  // external power is connected
{
  // put the system in self-powered mode
  pHighSpeedConfigDscr = (WORD)&HighSpeedLPConfigDscr;
  pFullSpeedConfigDscr = (WORD)&FullSpeedLPConfigDscr;
}
else // external power is not connected
{
  pHighSpeedConfigDscr = (WORD)&HighSpeedConfigDscr;
  pFullSpeedConfigDscr = (WORD)&FullSpeedConfigDscr;
}
```

The configurations `HighSpeedLPConfigDscr` and `FullSpeedLPConfigDscr` are defined in `dscr.a51`.

For proper operation, the command `Rwuen = TRUE` has been added in `TD_Init`. It seems that the USB-Blaster emulation do not work correctly without it. In the configuration of the processor (`CPUCS`), the output of the clock has been disabled. Finally, a hidden register allows to switch between high-speed and full-speed mode. By default, the FX2 is set to high-speed with the command `CT1 &= 0̃x02`. With the vendor command 0x91, the mode can be dynamically changed.

The Altera tools sometimes request to read the EPROM of the USB-Blaster. It is emulated in software with the file `eeprom.c` that contains the content of the EPROM. The reading is done via a vendor command with the following code:

```
if(SETUPDAT[1] == 0x90) // READ EEPROM
{
  BYTE addr = (SETUPDAT[4]<<1) & 0x7F;
  EP0BUF[0] = PROM[addr];
  EP0BUF[1] = PROM[addr+1];
}
else
{
  EP0BUF[0] = 0x31;
  EP0BUF[1] = 0x60;
}
EP0BCH = 0;
```

```
EP0BCL = 2; // Arm endpoint with # bytes to transfer
EP0CS |= bmHSNAK; // Acknowledge handshake phase of device request
```

The last step in this implementation is to modify the USB descriptors found in the file `dscr.a51`, so as to match those of the Altera USB-Blaster.