FPGA Interfacing

Wadhwani Electronics Lab - IIT Bombay

Kalpesh Krishna - 140070017
Abhin Shah - 140070013

Objectives

There have been two major aims of this project and we made some progress in each of them. This document primarily describes the progress in each of the two objectives along with the list of useful tutorials and guides.

- The first motive is to design and develop a circuit which makes use of an Altera Max V to control and divide computations between two Altera FPGA Integrated Circuits with the aim of maximizing productivity and minimizing the cost.
- The second motive is to program an Altera Max V using a microcontroller, which can be used with Quartus II, Altera's proprietary software to communicate with FPGAs. Having done this, we intend to make a circuit board which will have such a CPLD and a microcontroller circuit.
  This would effectively eliminate the need for any external dependencies, be it Altera's USB Blaster, FTDI's FT2232H, or UrJTAG.

The entire project has been hosted on Github, under https://github.com/Abhin02/FPGA

Objective #1

The project is a continuation of the work by Kamal Galrani and Manmohan Mandhana, who had designed a development board for the first objective as a part of their EDL-2 project.
The documentation of the old project has been added in the Github repository under docs/edl2_project.pdf.
The circuit data (Eagle file) for this circuit can be found in main_circuit.sch

Progress

This section elaborates the progress we've made in the above objective.

1 Altera Max V

The old project had a few Printed Circuit Boards on which we could start working immediately.
The first part of the circuit involved interfacing the Altera Max V integrated circuit soldered on our PCBs with the peripherals present on the Krypton Board (main_brd.sch), primarily the USB to JTAG converter FTDI device, FT2232H.
The Krypton Board seemed to have a GND loop connection which was completed through the daughter card (daughter_brd.sch). As we were just using the peripherals, without any daugther card, we needed to make an external connection between two disconnected GND pins on the Krypton Board to successfully generate 3.3V and 1.8V outputs.
Wires were taken from the daughter card slots to our CPLD chip. The four JTAG wires, 3.3V, 1.8V and GND. Our circuit had pins to host wires from JTAG and 3.3V. To power the 1.8V line, we were forced to use the connectors between our main CPLD controller circuit and FPGA daughter circuit.

Programming was carried out using UrJTAG which would burn SVF files to the CPLD via FT2232H hosted on the Krypton Board.
Effectively, we made our circuit play the role of the daughter board and successfully burnt a working VHDL schematic to blink an LED.

2 FT2232H

FT2232H is a multipurpose Integrated circuit that is developed by FTDI. In our case, we are using it to burn Serial Vector Files to the CPLD using UrJTAG. Hence FT2232H effectively acts like a USB to JTAG bridge. FT2232H has been used in the Krypton Board for this very purpose. The circuit we've used is identical to the Krypton Board.
Here is the datasheet.

The first step involved understanding the requirements of the FT2232H circuit. Since we were emulating the Krypton design, we primarily understood the voltage requirements of the integrated circuit.
3.3V was supplied to the circuit using a regulator developed at WEL, which used USB's 5V as an input supply.
1.8V was an output of the FT2232H circuit which was used to power the 1.8V supply line to the CPLD circuit using an external connection. (This will be replaced once the buck converter is in place).
After having supplied correct voltages, we succeeded in programming the CPLD completely using out circuit, without any external help from the Krypton Board. This was verified using the same LED Blink VHDL program.

3 TPS65400

TPS65400 is the powerhouse of our circuit. It's a Texas Instruments programmable Buck Converter, which gives four output voltages which can be decided based on external resistor values and I2C programmed internal registers.
Here is its datasheet.

Since we didn't have surface mount inductors, (needed for this integrated circuit), we decided to put this on hold until we procure those inductors.
We spent a considerable time understanding this circuit and the way it works. Most of the design has been taken from a schematic well explained in the datasheet. The values of the resistors have been defined and need to be soldered. Since the circuit involved I^2C, we decided to learn about the I^2C protocol.

4 I^2C (Learning only)

Inter Integrated Circuit is a popular communication protocol between devices. The first steps involved understanding the protocol. We tried a few test codes on Arduino's Wire library, and were successful in carry out communications between a Master Arduino and a Slave Arduino. The project files used are under "Master Reader" and "Master Writer" on Github.

5 CDCE925

CDCE925 is a clock synthesizer integrated circuit developed by Texas Instruments which is used to amplify the frequencies of an input clock connection. As input, we've attached an input XTAL 27 MHz crystal. The output clocks have been used at different parts of the circuit as per the requirement.
Here is the datasheet.
Our circuit has two special pins allotted specifically for I^2C communication with CDCE925 and TPS65400. CDCE925 heavily relies on I^2C and is locked by default. To program this circuit, we used Arduino's Wire library. We had an option to write our specification in VHDL, but we voted against it.

Since Arduino outputs 5V by default, we used a MOSFET based Logic level converter, (details) to step down the voltage to the CDCE925 compatible 3.3 Volts. Connections were straight forward, with a high impedance assigned to the Altera Max V pins connected to the I^2C pins of the CDCE925 (via VHDL), I^2C wires going from the Arduino to our I^2C pins via the voltage converter, and voltage lines powered up through FT2232H and the USB.

We started by running an I^2C identifier code in Arduino, to find our device. This code is from the Arduino Playground.
Due to an excellent documentation given by Texas Instruments, we were successful in executing a "Block Read" and "Block Write" I^2C protocol and we could successfully read and write to the registers in CDCE925.
To permanently store data, this had to be written to the EEPROM register and CDCE925 has a specific procedure to write a value to EEPROM.

Unfortunately, we were unable to produce a clock output from the integrated circuit after unlocking it. We suspect a bad input circuit, or a particular faulty connection in our circuit which has damaged the IC. We hope to test

this chip further in a breakout board, or buy Texas Instrument's circuit to test CDCE925.

Future Work

This project has a lot of components and will take substantial time to complete. We still need to work on the following:

- Figure out the mistakes in the CDCE925 circuit.
- Solder, program and test TPS65400, the buck converter.
- Solder and test the individual FPGAs on the daughter card.
- Write a VHDL specification to control the two FPGAs using the Altera CPLD controller.
- Make use of the Audio / Video components on the circuit board to develop a useful application of the project.
- (Optional) Add Ethernet connectivity using Silicon Devices Ethernet controller present on our circuit board.
- (Optional) Make use of the Flash memory on the circuit board.

Objective #2

This is the part of the project where we hoped to use a microcontroller to program the CPLD circuit on our board. This has been done by Xilinx for their FPGAs and we hope to emulate a USB Blaster code in a microcontroller and successfully program the FPGA.

Progress

We have tried to achieve this using two microcontrollers and have had limited success. We will describe our endeavours with each of them.

Arduino

We found a library on Github, which uses an Arduino to burn XSVF files onto a any CPLD or FPGA. The library included an Arduino sketch to burn the XSVF player to Arduino, and external Python scripts which use *pyusb* to send those XSVF files to Arduino which sends them to a CPLD's JTAG pins.
This library can be found on Github, at https://github.com/mrjimenez/JTAG.

This library did not work successfully with an Arduino Leonardo connected to our working CPLD circuit through JTAG wires. We suspect this is because the library is very specific to XSVF files and our CPLD will not support this file format.

Raspberry Pi

We spent considerable time understanding and installing Raspberry Pi, since it was our first time with this magical device. Once this was setup, we started off by installing OpenOCD, a library which has JTAG programming support for a number of microcontrollers.
Unfortunately, due to the lack of literature online for Altera CPLDs with OpenOCD, we could not use this.
We then installed UrJTAG on the Raspberry Pi and succeeded in communicating with the Altera CPLD chip with the GPIO pins acting as JTAG pins. The tutorial followed to achieve this is here.