Interfacing a Field Programmable Gate Array with a 16-bit Adiabatic Microprocessor for Testing

Richard McManus
Electrical Engineering
University of Notre Dame
Notre Dame, IN
rmcmanu2@nd.edu

***Abstract* – This report describes the progress made during the Spring 2022 semester in Dr. Snider's research group and focuses specifically on my contributions. The project described consists of multiple Verilog hardware description language (HDL) modules designed to interface a microprocessor unit under test (UUT) with the Virtex-7 VC707 Field Programmable Gate Array (FPGA) by Xilinx.**

I. INTRODUCTION

This semester I had the opportunity to continue working with Dr. Snider's research group and contribute to their efforts. My contributions consisted of researching FPGAs and programming in hardware description languages and developing multiple modules written in Verilog to offset the needs of the adiabatic microprocessor UUT. In the following sections, I will discuss the processes for these contributions and the results obtained.

II. Programming the FPGA for Microprocessor Testing

A. Background

This project was designed to offset the functionality of the S90 adiabatic microprocessor developed by Dr. Snider's research group. Dr. Snider's research group aims to reduce energy dissipation, one of the main limiting factors in computing, in microprocessors using adiabatic logic. The S90 test microprocessor is unable to interface directly with a personal computer (PC) and requires and external reversable clock. Therefore, the S90, referred to in figure 1 as the microprocessor without interlocked pipelined stages (MIPS), requires external circuitry for efficient testing.

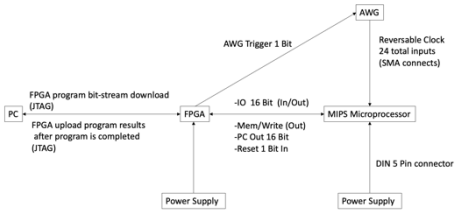

Figure 1. Diagram of Microprocessor Testing Layout

The S90 Microprocessor (MIPS Microprocessor) will be powered by an external power supply and receive clock signals from two external arbitrary waveform generators (AWG). Additional documentation on the generation of these clock signals by the two AWGs can be found in *Arbitrary Waveform Generator Python Programming and Peltier Module Test Environment,* my final report from the Fall 2021 semester. Additionally, the S90 microprocessor will interface with the Virtex-7 VC707 FPGA via a break-out board connected to the FPGA Mezzanine Card (FMC) connector on the FPGA. This break-out board can be seen in figure 2.

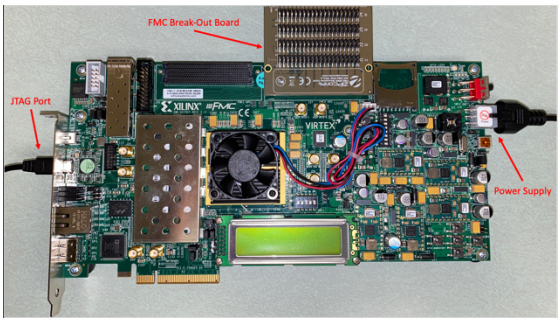

Figure 2. Labeled FPGA Connections

The Virtex-7 FPGA provides numerous functionalities for the microprocessor. Most notably, the FPGA will send the instructions to the microprocessor, wait for execution, and then save the

output for observation. The FPGA interfaces with a PC via the FPGA's joint test action group (JTAG) port using a universal serial bus (USB) cable. This interface allows for programming and observing the outputs of the microprocessor during testing.

B. Familiarization with Virtex-7 VC707

To familiarize myself with the Virtex-7 VC707 FPGA, I began by reading through Xilinx's *Getting Started with the Virtex-7 FPGA VC707 Evaluation Kit* and *VC707 Evaluation Board for the Virtex-7 FPGA User Guide.* Xilinx provides the Virtex-7 with a built-in self test (BIST) pre-loaded on the device. The BIST tests several features of the device and is manipulated within a terminal window on the host computer as shown in figure 3.

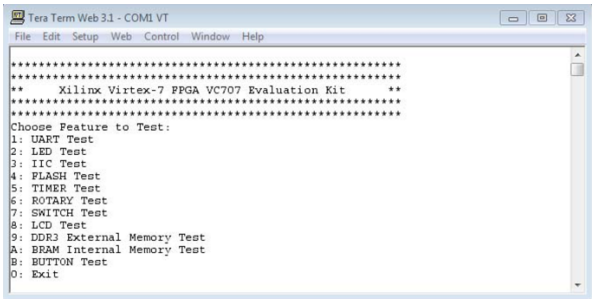

Figure 3. Built-In Self Test (BIST) Terminal Window

After successfully testing several of the BIST functions I moved on to familiarizing myself with Verilog and the Vivado Design Suite to program the FPGA.

C. Familiarization with Verilog HDL, and Vivado Design Suite

To begin exploring the Vivado Design Suite, I researched online for example programs using Vivado to program an FPGA. During this process, I discovered Lance Simms' Example, *Simple Flashing LED Program for the VC707*. This example provided a step-by-step walk through of the Vivado design process from creating a project, designing HDL and constraint files, synthesizing, and ultimately generating a bitstream used to program the VC707. I successfully followed the instructions in this example using the Vivado design suite and programmed the Virtex-7 FPGA with the resulting bitstream. The RTL netlist for this project can be seen in figure 4 and the resulting clocked LEDs can be seen in figure 5.

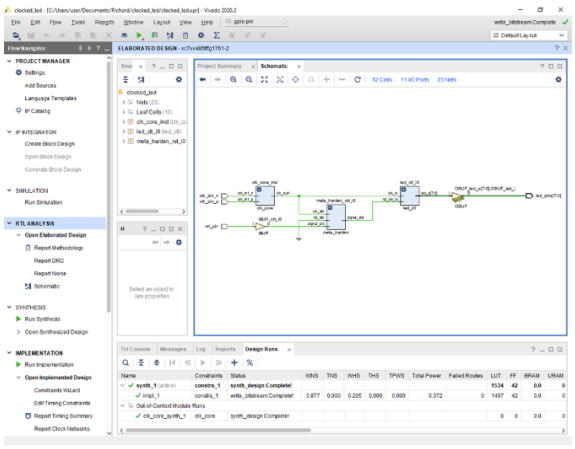
Figure 4. Clocked LED Example RTL Netlist

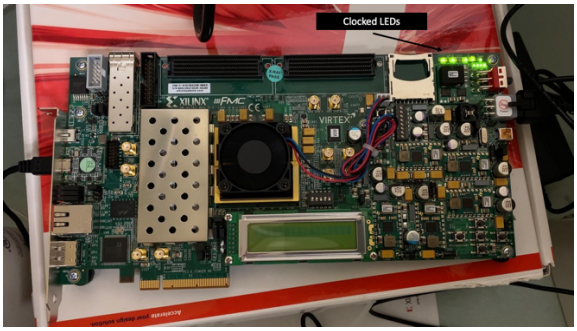

Figure 5. Programmed VC707 Clocked LEDs

As seen in figure 5, this example program resulted in the built in GPIO LEDs flashing in a pre-determined pattern as specified within the project files.

D. Testing The VC707 FMC Logic Levels

Now familiar with verilog and the Vivado Design Suite, Jon Cowart and I set out to test the logic voltage levels of the FMC connectors on the Virtex-7 FPGA. These voltages were not clearly defined in the provided documentation. This testing was necessary as the FMC pins serve as both inputs and outputs during testing. The S90 microprocessor requires signals in the range of -0.7 volts to 0.7 volts so this testing would determine whether a logic level shifting circuit may be required to accommodate both devices (Virtex-7 FPGA and S90 microprocessor).

For this testing, I designed a simple Vivado project, *FMC_TEST*, to manipulate two FMC pins on the Virtex-7 FPGA. The top-level module for this test project can be seen in Figure 6.

```
module fmc_test(
    inout out0,
    inout out1
    );

assign out0 = 1'b1;  //pin high
assign out1 = 1'b0;  //pin low

endmodule
```

Figure 6. Top-Level Module for Testing Logic Levels

The outputs out0 and out1 as shown in Figure 6 were assigned to two FMC pins (C33 and C34) on the FPGA. The bitstream was then generated and successfully uploaded to the Virtex-7 FPGA. The simple RTL netlist for this project with both pins set to high can be seen in figure 7.

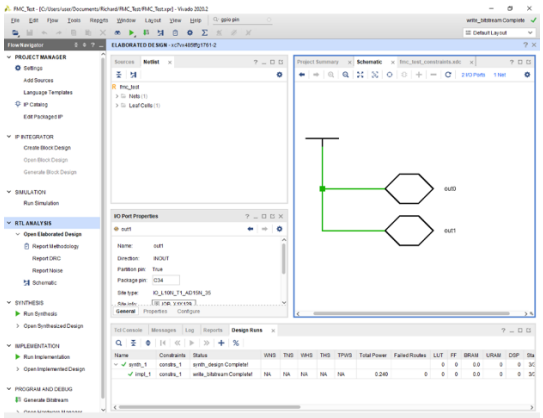

Figure 7. Logic Level Testing Project RTL Netlist

E. Designing Verilog Modules for FPGA – Microprocessor Interface

As requested by Jon Cowart, I began writing four verilog modules to be implemented when interfacing the Virtex-7 FPGA with the S90 microprocessor to be tested. These modules include a reset block, instruction memory, a data input module, and data memory.

The reset block shown in figure 8 was designed to trigger the AWG clock signals and reset the microprocessor upon starting a test program.

```
// Synchronous Reset for AWG and Microprocessor
module reset (clk, rst, rst_out)
    input rst;
    output reg rst_out;

    always @(posedge clk)
    begin
    if(rst)
        rst_out <= 1'b0; // assign low (Reset when low)
    else
        rst_out <= 1'b1; // assign high (Reset when low)
    end
endmodule
```

Figure 8. Verilog Reset Module

As shown in figure 8, the reset function is synchronous, and the output reset signal for the AWG and S90 microprocessor will be set low when a reset is called.

The instruction memory module shown in figure 9 loads in a sequence of 16-bit instructions to be sent to the S90 microprocessor during testing.

```
// 16-bit Instruction Memory for input to S90
module im (clk, pc, instruction);
    input clk;
    input [15:0] pc;
    reg [15:0] instructionMem [500:0];
    output reg [15:0] instruction;

    initial begin
        // Read Instructions from File
        //$readmemb("./test/test/prog",instructionMem,0,14);

        // Hard Code Instructions
        //instructionMem[100] = 16'b1000110000100010;
        //instructionMem[104] = 16'b1000110000100011;
        //instructionMem[108] = 16'b1000110000100100;
        //instructionMem[112] = 16'b1000110000100101;
        //instructionMem[116] = 16'b0000000001001010;
    end

    always @(posedge clk) begin
            instruction <= instructionMem[pc]; // Load Instruction to Output to S90
endmodule
```

Figure 9. Verilog Instruction Memory Module

The sequence of instructions can either be loaded in from an external file or hardcoded directly into the program. These instructions will be generated using a 16-bit MIPS compiler. At each rising clock edge, a new instruction will be loaded before being sent to the S90 microprocessor by the data input module.

The data input module serves as the data line to the S90 microprocessor. This module can be seen in figure 10.

```
module data_in (clk, instruction, dataIn);
    input clk;
    input[15:0] instruction;
    output reg [15:0] dataIn;
    always @(posedge clk) begin
        dataIn <= data; // Input Data
    end

endmodule
```

Figure 10. Verilog Data Input Module

As shown in figure 10, the data input module receives the current instruction and sends it via the dataIn output register to the microprocessor under test.

The data memory module shown in figure 11 is used to store the outputs from the S90 microprocessor.

```
// 16-bit Data Memory to save Outputs from S90
module data_mem (clk, memAddr, dataIn, memWriteEn, memRead, memOut);
    input clk;
    input [15:0] memAddr;
    input [15:0] dataIn;
    input memWriteEn;
    input memRead;
    reg [15:0] ram [127:0]; // Memory


    initial begin
        for(i = 0 ; i < 128 ; i = i + 1)
            ram[i] <= 16'd0; // Clear Memory
    end

    always @(posedge clk) begin
        if (memWriteEn) begin
            ram[memAddr] = dataIn; // Write Data to Memory
    end
endmodule
```

Figure 11. Multiline String Containing .csv Files

As shown in figure 11, the data memory module begins by clearing the FPGA memory that will be used to store outputs from the microprocessor. Then, on every rising edge of the clock the module will check if a write enable line is high. If high, the data memory module will input the data from the S90 microprocessor and store it for observation upon completion of testing.

III. Next Steps

Now familiar with programming the Virtex-7 FPGA and having developed the foundational modules for testing an adiabatic microprocessor, the next step is to integrate these modules into a top-level entity. Before complete, additional modules such as a read/write function will need to be designed and implemented into this top-level module. After this, the pins of the FPGA should be mapped to the program for physical implementation. At this point, additional circuitry for logic level shifting could be connected via the FPGA's FMC connections and the project would be ready to test a microprocessor.

REFERENCES

https://www.xilinx.com/support/documents/boards_and_kits/vc707/ug848-VC707-getting-started-guide.pdf

https://www.xilinx.com/support/documents/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf

https://lancesimms.com/Xilinx/VC707_Simple_LED_Example_Part1.html

https://www.fpga4student.com/2017/01/verilog-code-for-single-cycle-MIPS-processor.html

https://www.fpga4student.com/2017/04/verilog-code-for-16-bit-risc-processor.html

https://stackoverflow.com/questions/58540462/how-to-identify-synchronous-resets-in-verilog

https://books.google.com/books?id=ZyrjlPlpoFEC&printsec=frontcover#v=onepage&q&f=false