Laboratory Exercise #3
Creating a Custom Hardware IP and Interfacing it with Software

ECEN 449-511

Seth Pregler

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

Introduction

The purpose of this lab was to familiarize us with the process of creating, importing and interfacing a custom IP module for the Zynq Processing System (PS) that performs integer multiplication.

Procedure

The following is an overview of the procedure of implementing and testing the design for this lab.

1. Create Block Diagram and add Zynq PS IP
2. Customize the PS IP to only use UART for peripheral I/O
3. Create a custom IP with 4 32-bit registers and interface it with the PS
4. Edit the multiply_v1_o_S00_AXI.v file so that there are no writes to register 2
5. Add user functionality for integer multiplication
6. Package the hardware and create HDL wrapper
7. Export hardware and open project in SDK
8. Create new application for testing multiplication module
9. Program FPGA and interconnect via serial link
10. Observe output, make adjustments if necessary

Results

My implementation of the integer multiplication module was pretty straightforward. Instead of using the MicroBlaze, I interfaced the peripheral multiplication module with the ARM Cortex-A9 processor using the AXI Interconnect. I used UART to connect the board to my computer. Running a linux terminal I was able to display the results of the unit test that I wrote to test the functionality of the multiplication peripheral. The test was a short C program that wrote integer values to registers 0 and 1, read the result from register 2 and printed it to the console. See *Appendix A* for the output of the terminal after running the test

Conclusion

After completing this lab, I experienced the process of creating and interfacing a custom IP module with the ARM Cortex-A9. It was very simple and was less involved as compared to the last lab where we interfaced GPIO with the MicroBlaze.

Questions

(a) What is the purpose of the tmp reg from the Verilog code provided in lab, and what happens if this register is removed from the code?

temp_reg is meant to synchronize the register multiplication with the clock. If this was removed, there would be a greater chance of running into race conditions and an incorrect result.

(b) What values of 'slv reg0' and 'slv reg1' would produce incorrect results from the multiplication block? What is the name commonly assigned to this type of computation error, and how would you correct this? Provide a Verilog example and explain what you would change during the creation of the corrected peripheral

If the values of slv_reg0 and slv_reg1 were very large 32-bit integers, this would cause the result to overflow and an incorrect value would be displayed to the terminal. This could be corrected by either increasing the bit width for all 3 registers or adding more registers and doing several smaller multiplications and adding the results from each output register.

In addition, if we multiply a signed integer by an unsigned one, the result will be incorrect because the signed integer will be implicitly typecast to an unsigned integer.

We can fix both of these issues with implementing the following verilog code:

```
// Registers are now 64-bit so they will be able to handle much larger values
reg signed[0 : 63] tmp_reg, slv_reg2;
// Declare inputs as signed
input signed[0 : 63] slv_reg0, slv_reg1;

always@(posedge S_AXI_ACLK) begin
        if (S_AXI_ARESETN == 1'b0) begin
                slv_reg2 <= 0;
                tmp_reg <= 0;
        end

        else begin
                tmp_reg <= slv_reg0 * slv_reg1;
                slv_reg2 <= tmp_reg;
        end
end
```

Appendices

List of Contents

Appendix A
Output of the Terminal (Top Right)