Lab 2: Communicate with your hardware

Charlie Coleman
29 October, 2018

Pre-Lab: N/A

Objective: The objective of this lab is to familiarize ourselves with the connection between our hardware and software, and create our first design to utilize it. This will allow us to transfer data between the halves and test the hardware.

Circuit Diagrams: N/A

Outcome Predictions: At the end of this lab, we should have a settable counter implemented in hardware. When the divisor is set, this counter should increment a register. The software should be able to read this value out of the memory. Then the value of the register will be written to the LEDs on-board.

Equipment:
- Computer
- Digilent Zybo 7000
- Xilinx Vivado 2018

Procedure:
- Part 1.
  1. Choose how the software and hardware halves will communicate. For the Zybo 7000, the processor and FPGA are on the same board, so this communication is built in.
  2. Setup your simulation environment for the board
  3. Create a new AXI IP package with 16 registers
  4. Test these registers using the Vivado SDK & `xil_io.h`
- Part 2.
  1. Using the same IP block as part one, use one register as a divisor
  2. Create a counter using the system clock with a frequency divisor
  3. Using the Vivado SDK, write a value into your divisor register
  4. In a while loop, read the value out of the counter register and output it on the LEDs

Recalculations and Predictions: N/A

Data and Observations: Using a 100,000,000 divisor, we are able to obtain a count rate of 1/second. We are able to time this. We have 4 LEDs, so the cycle from 0000 to 1111 took 16 seconds.

Analysis and Discussion: All clock rates set by the SDK were accurately represented on the LEDs. Initially, I assumed the clock rate for the Zybo was 50MHz, so all of the tests were off by a factor of 2. Correcting this in the calculations showed that the divisor was working as intended.

Results: The final result of part two worked exactly as intended. The SDK was able to read and write data out of the hardware half of the design. We encountered issues when the Xilinx SDK would cache the results of the memory read, meaning that the value of the register was never updated. Using some tools from `xil_cache.h` we were able to remedy this issue. We also had a problem where the counter was not updating the register. This was fixed by creating a new signal and replacing the reference to the old register in the output to this new register and adding it to the sensitivity list. After all of these errors, the lab was working as intended.

Conclusions: This lab provided me with a refresher on using Xilinx, VHDL, and C. It gave me experience with connecting hardware and software in useful ways. These skills are vital when working on embedded systems and other systems involving both hardware and software components. This lab can be considered a success as it familiarized me with the board I'm using and how to make the two halves of the design communicate efficiently.