ECE532S Digital Systems Design

Lab Test and Warm-up Demo

Last Updated: Jan, 2023

The purpose of this test is to show that you have mastered some of the basics of using Vivado. In advance of your demonstration in the lab, you should have all of the functionality described in the two tasks below complete and ready to demonstrate. The first task must be completed and demonstrated **individually**, without aide from your groupmates. The second task should be completed together with your **group** and can be demonstrated to the TA as a group. Individual grades will be assigned for this test, based on the TA's assessment of work distribution.

This is worth 10% of your final grade. If you do not have this exercise prepared in advance, you will be assigned a grade of 0%. Please ensure that you have been assigned a demo time slot prior to the lab session. Contact your TA if you do not have one or do not remember your time.

1 Individual Task – Simple Module and Simulation [4 marks]

Build a module, using the HDL of your choice, that inputs a stream of 4-bit numbers and outputs the largest of the numbers currently input. Assume all numbers are unsigned positive integers. The block has the following ports:

`clk` **(input)** – the clock
`reset` **(input)** – an active high reset signal
`in_data[3:0]` **(input)** – the data integer input, only valid when `in_valid` signal is high
`in_valid` **(input)** – indicates (active high) that the value on the `in_data` bus is a valid input
`out_largest[3:0]` **(output)** – the largest of the valid input values so far

The block operates as follows:

1. The `reset` signal is set to one for a few cycles to reset the module.
2. On a positive `clk` edge, if `in_valid` is high, then `in_data[3:0]` has a valid input.
3. The output `out_largest[3:0]` is 0 after reset and then shows the largest of the numbers that have been input so far. This output changes on the next positive `clk` edge after the positive `clk` edge on which the valid input arrived.
4. The circuit continues to operate until a reset.

Your task is to:

1. Create the module described above in the HDL of your choice.

2. Develop a testbench to be used to simulate the developed module. This testbench should test enough cases to convince someone that your circuit works correctly. This can be an HDL testbench, like the one provided in Tutorial 1, or some other testbench method (e.g. TCL).

3. Simulate this block using the Vivado simulator and show the results of some of the tests to the grading TA in the waveform viewer.

4. Create a Vivado project with a new top-level HDL module that instantiates your module and connects the ports of your module to pins of the Nexys 4 DDR FPGA (any arbitrary pins).

5. Use Vivado to synthesize, implement, and generate a bitstream for the project and show that it can synthesize and implement without errors.

    - You will not need to actually load the completed project onto the FPGA and show that it is working in hardware. This individual test focuses on simulation.
    - While we will not be testing the module's behaviour on the FPGA board, you may still want to connect all of your ports to relevant pins so that you may test it on your own to confirm it's behaviour (e.g. switches for the inputs, LEDs for the outputs, one of the available clocks for `clk`, and a push-button for the reset)

2 Group Task – Networking Warm-up Demo [6 marks]

In Tutorial 5, you were directed to a Digilent tutorial that instructed you on how to build a TCP echo server on the Nexys DDR board. Tutorial 5 later described how to modify the code of that tutorial to turn the system into a TCP client instead. In this warm-up demo, you will be required to build the TCP client system again, with modifications. In particular, you will need to add additional peripherals to your system, and interface with them in software, to determine the contents and timing of your sent packets. In addition, this warm-up demo will have you include an ILA hardware debugger module to demonstrate your knowledge and ability to perform that level of system debugging. The system has the configuration shown in Figure 1:

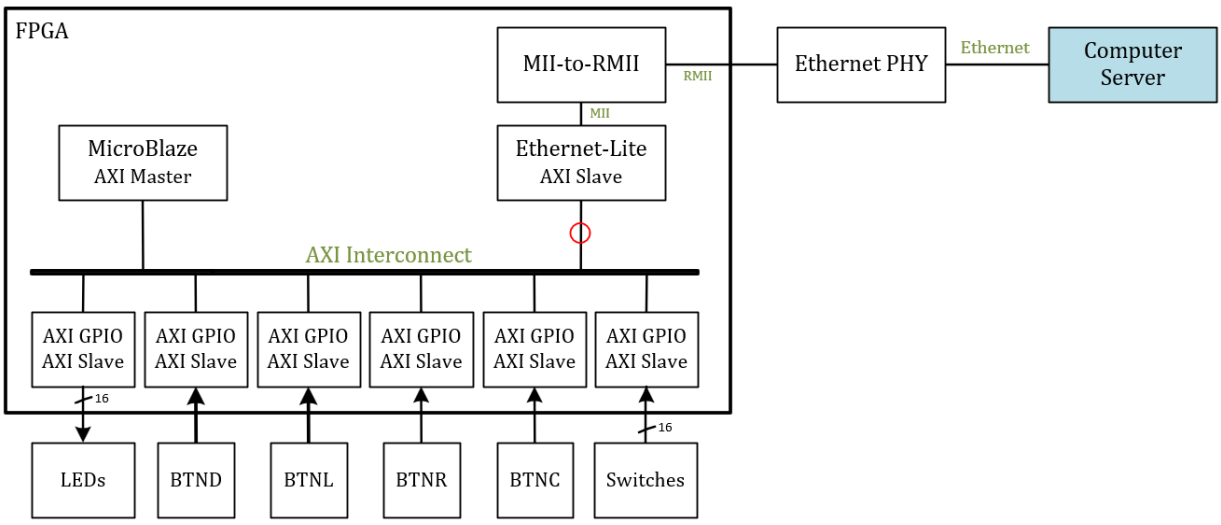

Figure 1: Block diagram of the project. Red circles indicate probe points for ILAs

Your task is to:

1. Create the TCP client project described in tutorial 5 for the Nexys DDR FPGA board.
   (a) Modify the block diagram of that project to connect the 16 LEDs, the 16 switches, and four of the push-buttons (BTND, BTNL, BTNR, and BTNC) to AXI GPIO blocks; these AXI GPIO blocks should in turn be connected to the AXI Interconnect of the MicroBlaze processor system.
      - Note - The diagram in Figure 1 shows six separate AXI GPIO blocks instantiated, though you may instantiate as few or as many as you like, so long as all of the requisite buttons and LEDs are connected and accessible from the AXI Interconnect.
   (b) Add an Integrated Logic Analyzer (ILA), as described in Tutorial 4, to the AXI connection indicated by the red-circle in Figure 1 (The AXI input to the EthernetLite core).
2. Modify the code for the TCP client project such that it has the following behaviour:
   (a) The program should store two *separate* 32-bit integer values, one for the value to be sent to the Computer Server (`value_send`), and the other for the most recent value received from the Computer Server (`value_recv`), both initialized to zero.
   (b) Upon a press of `BTNL`, the following two things should occur:
      i. The most-significant 16-bits of `value_send` should be updated with the values present on the switches.
      ii. The most-significant 16 bits of `value_recv` should be displayed on the LEDs.
   (c) Upon a press of `BTNR`, the following two things should occur:
      i. The least-significant 16-bits of `value_send` should be updated with the values present on the switches.
      ii. The least-significant 16 bits of `value_recv` should be displayed on the LEDs.
   (d) Upon a press of `BTNC`, a TCP packet should be sent to the Computer Server with the word 'POST' followed by the 32-bit value stored in `value_send` as its contents.
      - E.g., If the integer `value_send` currently has the value `0xBAADF00D`, then the contents of the sent packet should be `0x50, 0x4F, 0x53, 0x54, 0xBA, 0xAD, 0xF0, 0x0D`.
      - `0x50, 0x4F, 0x53, 0x54` correspond to the ASCII codes for the letters 'P', 'O', 'S', and 'T', and the next 32-bits is the value itself.
      - The 'POST' message instructs the Computer Server to update its stored value to the value that follows 'POST' in the packet
   (e) Upon a press of `BTND`, a TCP packet should be sent to the Computer Server with the word 'GET' (ASCII encoded, `0x47, 0x45, 0x54`) as its contents.
      - The 'GET' message instructs the Computer Server to send its stored value back to the TCP client in a response packet.
   (f) Upon receiving a packet from the Computer Server, `value_recv` should be updated to store the contents of the newly received packet (should be a 32-bit response packet from the Computer Server).

3. During the test in the lab:

   (a) Connect the FPGA board to an Ethernet cable and then connect the other end of the Ethernet cable to a computer system running the *echoserver.py* TCP server script.

   (b) Program the bitstream for your project onto the FPGA board and start the modified TCP client program.

   (c) Demonstrate the `BTNL`, `BTNR`, and `BTNC` functionality by using the buttons to change the value of `value_send` and sending a 'POST' packet to the Computer Server.

   (d) Demonstrate the `BTND` functionality by using it to send a 'GET' message to the Computer Server in order to retrieve the Computer Server's integer value and storing that retreived value in `value_recv`.

   (e) Open the *Hardware Manager* in Vivado and setup the ILA to trigger on data received on the read-data channel (the channel with the `rvalid` and `rready` signals).

   (f) Send another 'POST' message and observe the ILA's captured waveform.

   - What packet(s) did the ILAs waveform viewer capture? Was it the data packet containing the 'POST' message followed by the 32-bit value? If not, should it have been? Why or why not?

2.1 Python Server Specifications

When a connection is made to the server, one of two commands is expected. A `GET` command will cause the server to send a 32-bit value corresponding to the value stored in the server. On the other hand, a `POST` command will update the 32-bit value. The `POST` message must be accompanied immediately with the new value to set. The Python3 files `echoserver.py` and `echoclient.py` demonstrate simple versions of the server and client. You may use the server implementation but the client will have to be implemented in the Nexys 4 DDR board.

2.2 Some Useful Hints

1. When each of the buttons is pressed, the corresponding processes initiated by that button press should happen only once. If the code were written in a way as to simply check the current value of the button, in an infinite loop perhaps, then each time the button's value is read from the AXI GPIO the action would occur. Taking `BTNC` as an example, if `BTNC` is pressed, then each time the button's value is checked in the infinite loop another 'POST' packet would be sent. Since infinite loops can be processed very many times in a single press of a button, many packets would be sent. Instead of reading the current value of the button, your code should look for the rising edge of that button press, i.e, when the button changes from unpressed to pressed. Note, some debouncing may also be necessary.

2. In the LWIP code, once the network connection is established, the main function proceeds in an infinite loop called the *Event Loop*. It is important for the proper operation of the LWIP library that this Event Loop be allowed to have an iteration frequently, on the order of at least every 250 milliseconds. If you are polling for AXI GPIO values in this Event Loop, make sure your polling structures are non-blocking, as any blocking code that waits for the user to press the button will almost certainly be waiting for longer than 250 milliseconds.