Embedded Systems (CSCE 4114)

Lab 5

Zack Fravel

11/4/16

zpfravel@uark.edu

Abstract

The purpose of lab 5 was to design and implement a wishbone-bus architecture to be used by the UART transmitter/receiver. Task 1 had us write the VHDL code and simulate a wishbone slave; Task 2 involved implementing the design from Task 1 into a larger system to be used on the DEB2 board in conjunction with the Termite app.

Introduction

The bus protocol we're using, Wishbone, is a design where the bus includes Address, WriteData, ReadData, Instruction, and Control signals in order to process transactions between different hardware modules. Below is a diagram of the bus architecture we wish to implement.

Wishbone Architecture

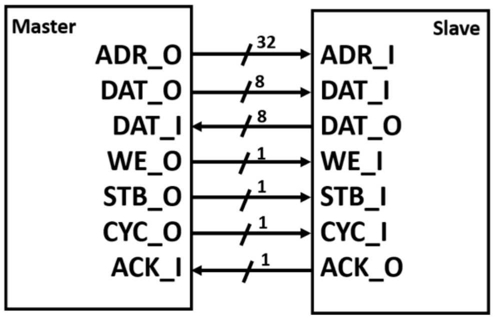


Figure 1

An example transaction between a master and slave would be as follows. If the master wants to read the data stored at a certain address, the master sends STB_O and CYC_O to the slave to indicate a transaction is about to begin. With WE_O = '0', the slave knows the master wishes to read the data stored in the registers associated with ADR_I's address. Once the slave has this information, it sends the ACK_O signal to the master along with the requested DAT_O. The only difference with a "write" transaction is WE_O = '1' and DAT_I contains some value.

Design and Implementation

Task 1 of the lab had us design the wishbone slave module and use the testbench in our simulations to simulate the behavior of the master interface. Within the wishbone slave there are registers, for which the master sends 32 addressing bits to access; we're only required to implement eight of these registers. Below is the diagram for the wishbone slave design, as well as the register specification we are implementing.

Wishbone Slave Top Level Diagram

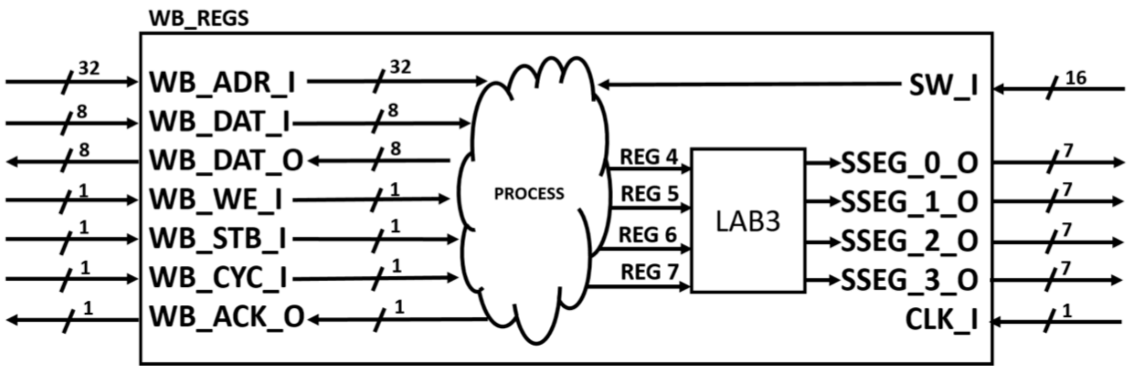


Figure 2

Wishbone Slave Register Specification

Address (32 bits)	Type	Description.
0x00000000	Read-only	Content should be "0000" & SW(3 downto 0).
0x00000001	Read-only	Content should be "0000" & SW(7 downto 4).
0x00000002	Read-only	Content should be "0000" & SW(11 downto 8).
0x00000003	Read-only	Content should be "0000" & SW(15 downto 12).
0x00000004	Read/Write	Bits 3 downto 0 control the 1st 7-seg display.
0x00000005	Read/Write	Bits 3 downto 0 control the 2nd 7-seg display.
0x00000006	Read/Write	Bits 3 downto 0 control the 3rd 7-seg display.
0x00000007	Read/Write	Bits 3 downto 0 control the 4th 7-seg display.

Figure 3

To reiterate, our wishbone slave contains eight registers (0x00000000 - 0x00000007), with 0 - 3 containing the status of the switch signals represented on the DEB2 board. Registers 4 - 7 are not only able to be read, but also are able to be written to and their content's bits (3 downto 0) control the four seven segment displays on the board. The implementation of this in VHDL is fairly straight forward. First, I declare five signals to be used throughout the design, four to be used for the SSD's and one used later on for the acknowledgment signal. The majority of the design lies within one process sensitive to the CLK_i, on each clock cycle the process sets the tempAck = '0' and checks whether WB_STB = '1' and WB_CYC = '1' to indicate a transaction. This is broken up into two separate if statements, one for writing (and WB_WE = '1') and one for reading (and WB_WE = '0'). Within the "Read" if block, I set the tempAck = '1' and have a case statement for the eight cases of register addresses we wish to account for (on previous page). For example, the second register's declaration is "when x"00000001" => WB_DAT_o <= "0000" & SW_i (7 downto 4) (second set of switches).

The "write" if block is very easy to impliment. For the first four cases, I just have null since they're read only registers. For cases 0x"00000004" - 0x"00000007" I set the corresponding internal SSD signal to that register to weaver the WB_DAT_i signal is at the time. I also set the output to the data being written in. For all other cases it is also null. Below the process I have the WB_ACK_o signal set to "and" the tempAck and WB_CYC and WB_STB signals so there is no overlap with the acknowledgement signal whenever a transaction is finished. Other than that, the only other thing included in the wishbone slave design is the connection between the SSD signals and the Lab 3 module used to control the displays.

Once Task 1 was completed, Task 2 was fairly straight forward in terms of designing and implementing. Task 2 had us implement a larger system that we could use in conjunction with the DEB2 development board. Provided to us was a UART to Wishbone controller module that allows us to send "read" and "write" commands to our wishbone bus using the Termite app in windows. Below is the specification for the commands we're able to send through the app.

UART to Wishbone commands

First character	Second character	Characters 3 to 6	Last character
'R' or 'r'	Space	Address to read (hex), for example: '0005'	End of line (enter)

First char.	Second char.	Characters 3 and 4	Fifth Character	Character 6 to 9	Last char.
'W' or 'w'	Space	Data to write (hex), for example: '0A'	Space	Address to write (hex), for example: '0007'	End of line (enter)

Figure 4

In order to get the design to work as we'd like to on the board we need to design a top level entity and connect all the pieces together. Below is the schematic for the top level design.

Top Level Circuit Diagram

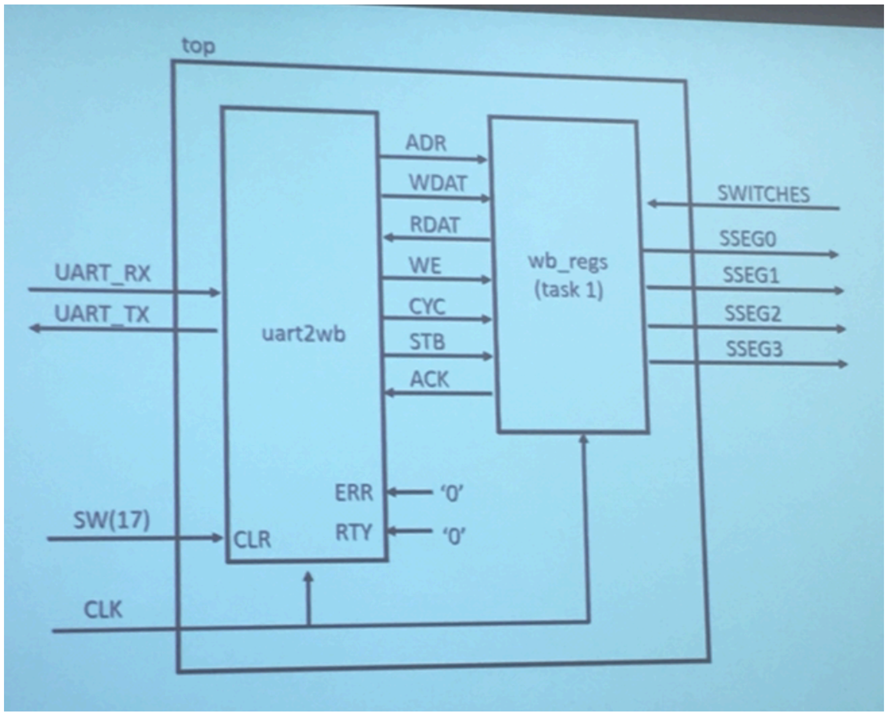


Figure 5

In VHDL, the top level design is fairly easy. I declare my internal signals between the uart2wb and the Wishbone slave and then instantiate the two modules below. The main thing to be careful about when connecting the modules is to make sure and connect outputs to inputs and not get confused with the data_in signals. All VHDL source files are attached to the report.

Results

For the results on Task 1, I designed a testbench that mimicked the behavior of a wishbone master requesting to read each of the four read-only registers and writing A, B, C, D to the four SSD registers. Below is the simulation waveform for the testbench I designed.

Lab 5 Behavioral Waveform

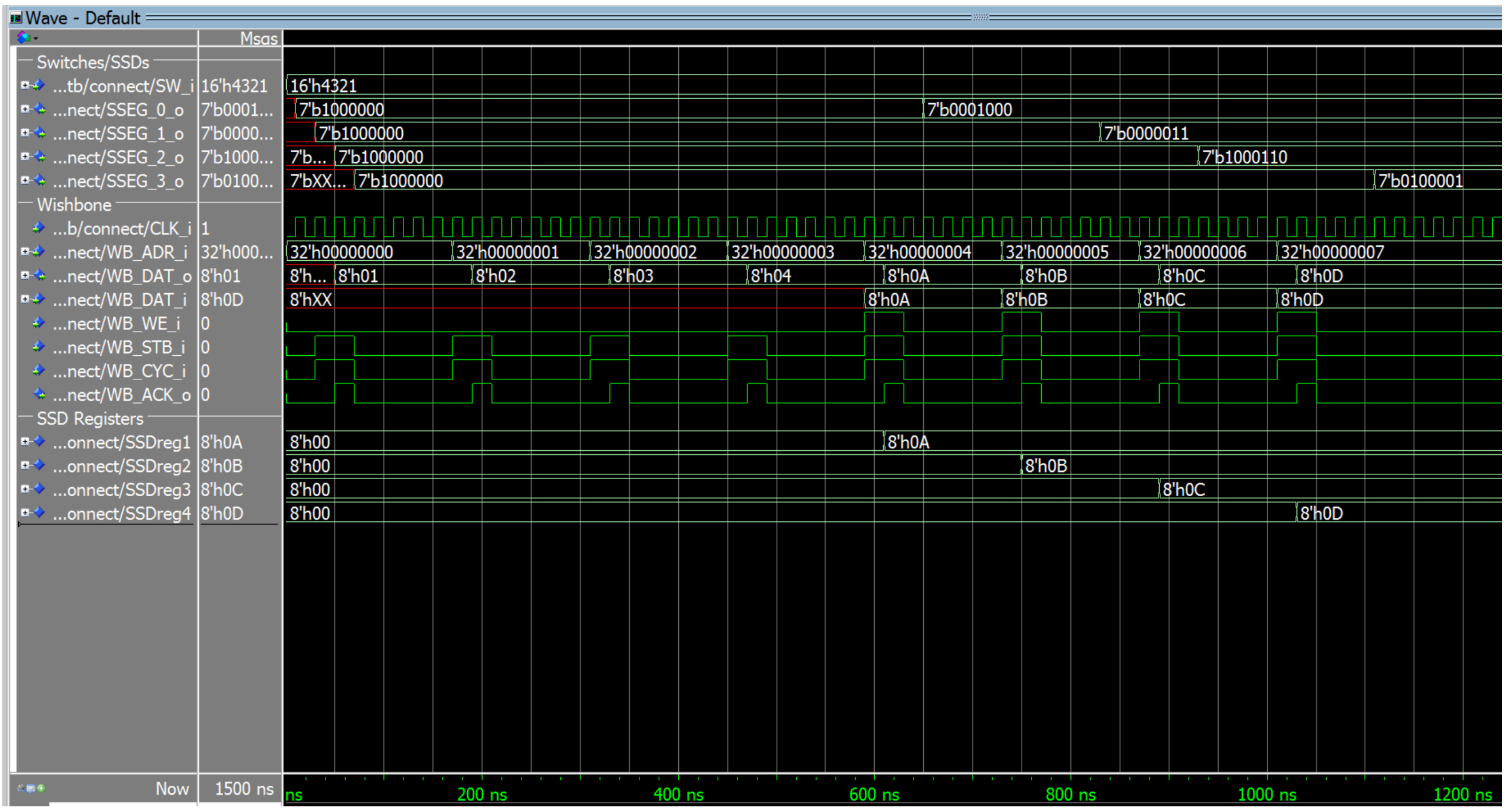
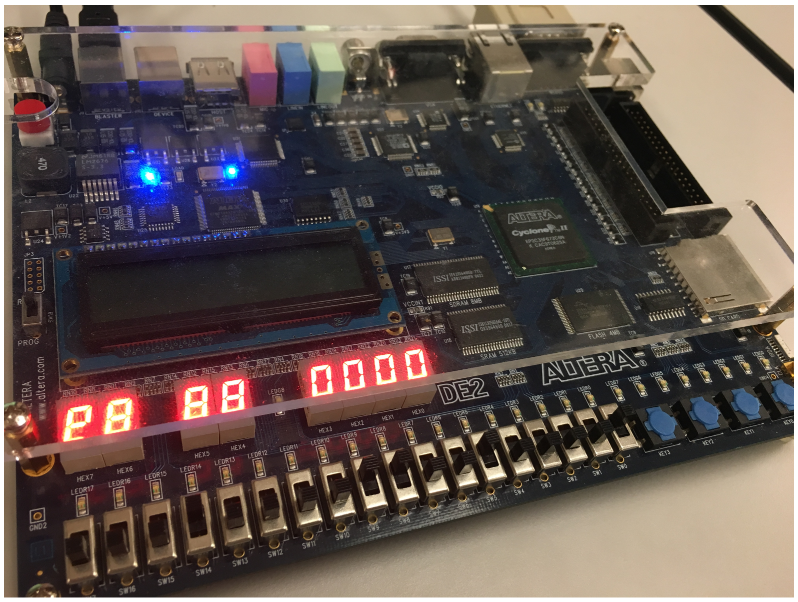


Figure 6

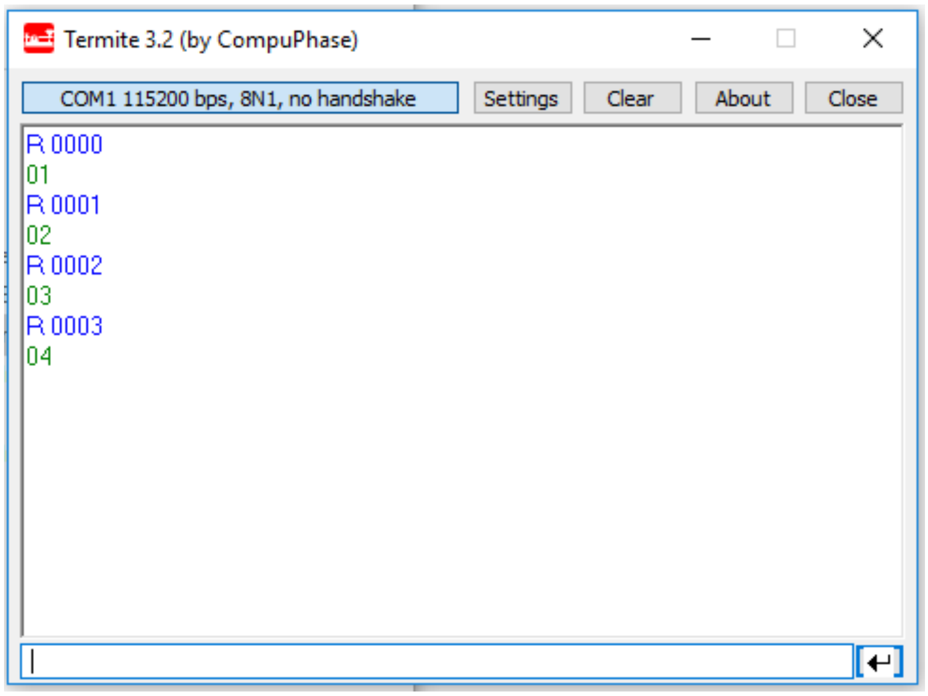
It can be seen above that the Wishbone slave I designed correctly responds to the master's requests as laid out by the specification I described earlier in the report. The acknowledgement signal is only high during the times the master is requesting the data, and has no overhang. It can be seen on each write operation that the module correctly outputs the written in data on the display corresponding to the register written to. I also chose to have my module send the written data back out, although there isn't much purpose to this other than maybe verification with the master. The testbench successfully tests for all cases we have to account for.

In order to test Task 2, I configured my DEB2 board's switches to have the values x"4321" or "0100 0011 0010 0001," compiled my designs, loaded them into the board, and launched termite. With the termite app set to the correct baud rate of 115,200 bits/s I was able to start verifying that my design works as intended. The following screenshots show, in order, me reading from the four read-only registers, which correspond to the status of each set of four switches. Following the reads, I change the orientation of the second set of switches and read again to show the values correctly updated. After the read operations I then send four write operations to each of the SSD registers to display "ABCD" on the displays.

Initial Switch State

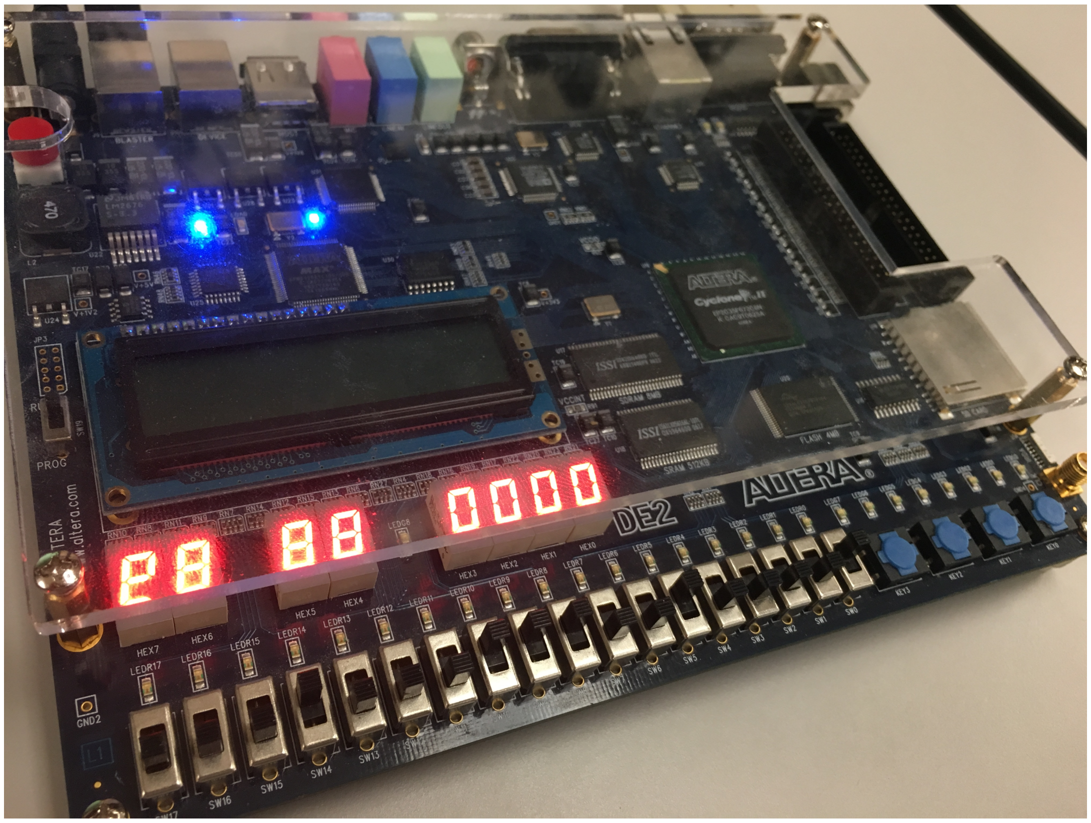


Initial Read

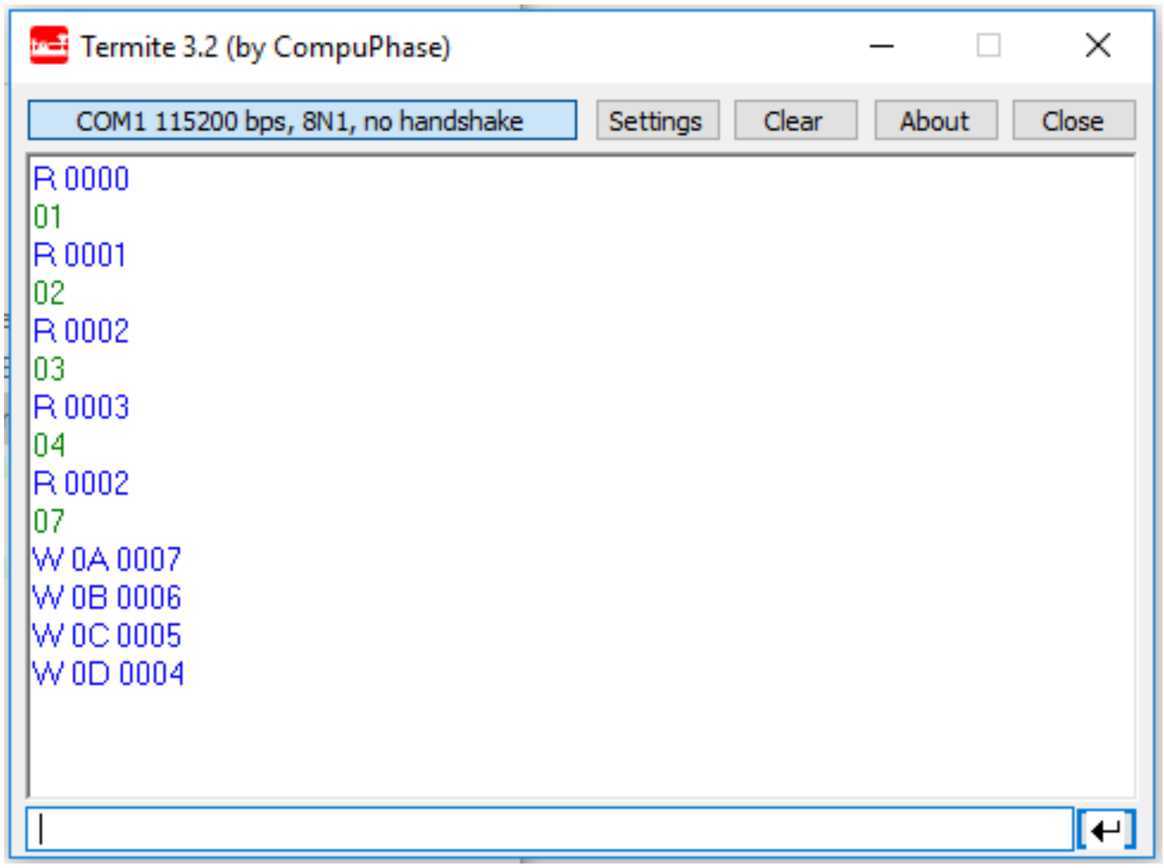

Termite 3.2 (by CompuPhase)
COM1 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
R 0000
01
R 0001
02
R 0002
03
R 0003
04
```

Change Contents of Register 2



Second Read and ABCD Write Commands


Termite 3.2 (by CompuPhase)
COM1 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
R 0000
01
R 0001
02
R 0002
03
R 0003
04
R 0002
07
W 0A 0007
W 0B 0006
W 0C 0005
W 0D 0004
```

Final Output on Board

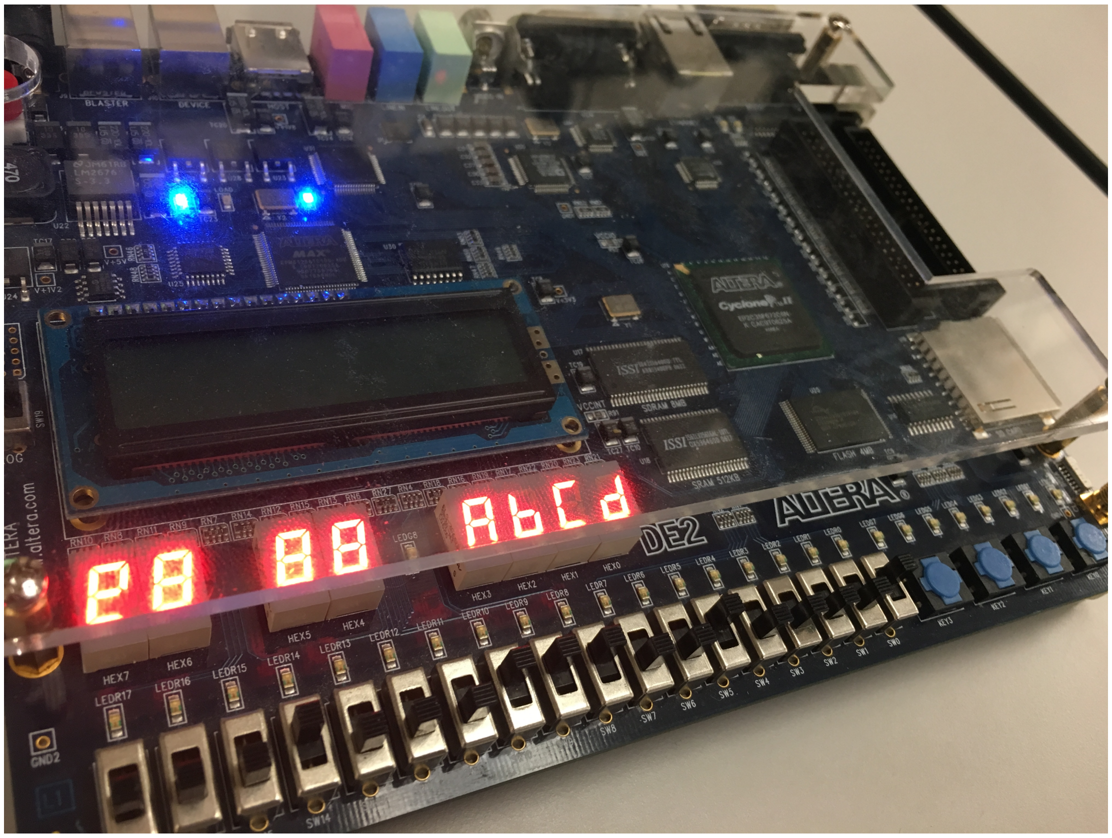


Figure 7

It can be seen from the pictures and screenshots above that my design works precisely as described in the lab description. In total, I did about five hours or so of actual work and designing to complete the lab. In conclusion, the lab did a good job of exposure to the ideas around bus architectures and what goes into designing a total system that is able to have inter communicating pieces to it. All VHDL files are included with the report, as well as the testbench.