ECE 501 Digital Systems Laboratory
Experiment 13 – FPGA Circuit Realization

Seth So
Lab Partner: Martin Klena
Station 2

Date Performed: 4-12-18 to 4-18-18
Date Written: 4-20-18

. . . .

Introduction:

The Field Programmable Gate Array (FPGA), is an integrated circuit chip that can be programmed to implement entire circuits by itself. The FPGA board is an Altera peripheral that compiled schematics can be downloaded to for implementation after pin assignments. In addition to Quartus schematics, the FPGA can also read VHDL (VHSIC [Very High Speed Integrated Circuit] Hardware Description Language) code.

Figure 1: Altera DE2 FPGA Board

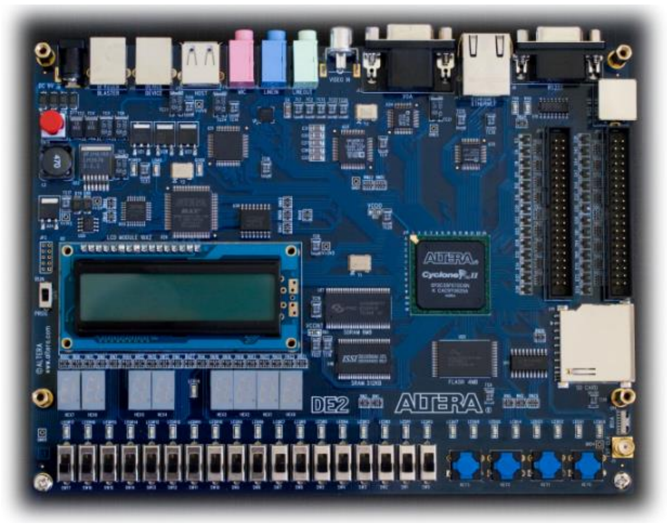


Figure 1 above shows the board that will be used. It features many different I/O ports for high general compatibility, peripheral compatibility for other circuit boards (e.g. Arduino), an onboard LCD display, eight 7-segment LED displays, 19 red/green LEDs, 17 switches, and four push buttons.

In Laboratory 13, the 4-bit ALU made in the previous lab was implemented on the FPGA, rather than the protoboard, using the schematic made before. Various schematic components were sequentially replaced with self-written VHDL-coded components to demonstrate the boards VHDL compatibility.

Part I: Quartus II Circuit Design

Purpose:

The purpose of part 1 was to repurpose the ALU schematic designed in the previous lab for programming onto the FPGA.

Procedure:

1. The first step to repurpose the schematic was altering the components. The 4-bit latch made from D flip-flops was replaced with a new design, which utilized four individual "latch" components in one symbol file. *Figure 2* below in the results section shows this

schematic. Afterwards, the old 7-segment decoders were removed from the schematic. The file DispBin.vhd was also added for the VHDL coding later.

2. The setting and parameters of the circuit were altered so that the schematic could be downloaded successfully to the FPGA board. Specifically, the device family was set to Cyclone II, the specific device was set to EP2C35F672C6, the optimization technique was set to AREA, and the global clock was deselected. The schematic was compiled with these new settings.

Results:

Figure 2: New Latch Design

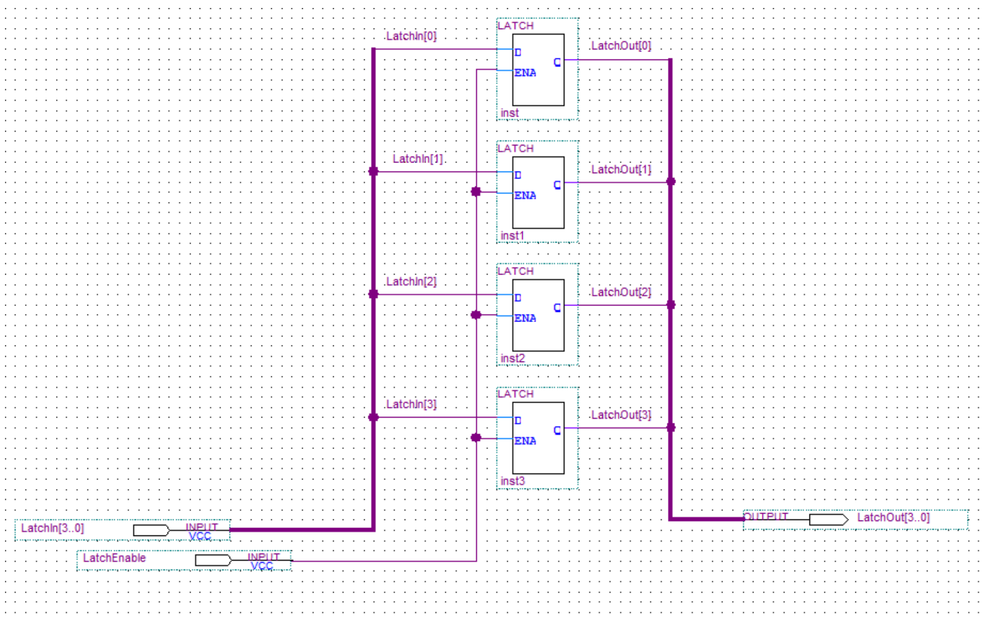


Figure 2 above shows the new 4-bit latch design. Similar to the D flip-flops, the design uses four individual latches in tandem with one another, one for each bit, sending a bus of data and enable input to the latch system. Just as easily, a bus of outputs can be read as a 4-bit output from the design.

Conclusion:
The settings for schematic from lab 12 were prepared for use on the FPGA board. Next, pin assignments would be made to replace the deleted 7-segment displays.

Part II: Pin Assignments for 7 Segment LEDs

Purpose:

The purpose of part 2 was to replace the schematic 7-segment LED displays with the FPGA onboard 7-segment displays via pin assignment. Doing this introduced the idea of pin assignment for input/output-pin assignments later in part III.

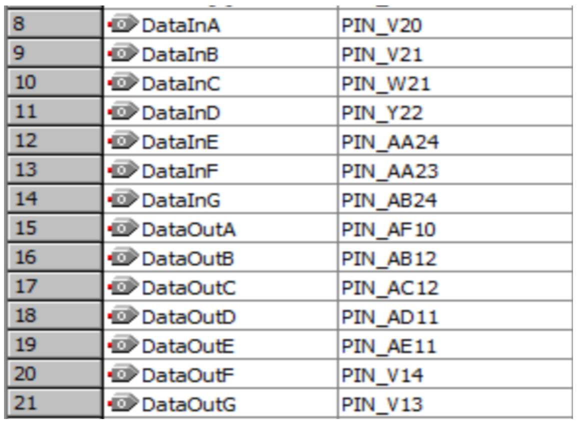
Procedure:

1. Using the Assignment Editor, the outputs of the decoders were assigned to the 7-segment displays Hex0 and Hex1 on the FPGA board. *Figure 3* below in the results section shows the table used for assignment and the corresponding assignment in Quartus II.

Results:

Figure 3: Decoder Pin Inputs

Display Segment	Pin for Hex1	Pin for Hex0
A	V20	AF10
B	V21	AB12
C	W21	AC12
D	Y22	AD11
E	AA24	AE11
F	AA23	V14
G	AB24	V13



8	DataInA	PIN_V20
9	DataInB	PIN_V21
10	DataInC	PIN_W21
11	DataInD	PIN_Y22
12	DataInE	PIN_AA24
13	DataInF	PIN_AA23
14	DataInG	PIN_AB24
15	DataOutA	PIN_AF10
16	DataOutB	PIN_AB12
17	DataOutC	PIN_AC12
18	DataOutD	PIN_AD11
19	DataOutE	PIN_AE11
20	DataOutF	PIN_V14
21	DataOutG	PIN_V13

Figure 3 above shows the decoder pin assignments. On the left is a list of connections from the Altera DE-2 board manual (provided in the lab instruction) for Hex0 and Hex1. On the right is the assignment of the decoder outputs, DataOut (ALU output) and DataIn (Register Read output).

Conclusion:

The basic procedure and references for pin assignments were taught. All pin locations and corresponding parts on the FPGA board were found in the Altera DE-2 manual. Importantly, no pins were double-assigned, and no reserved pins were used.

Part III: Pin Assignments for Inputs and Outputs

Purpose:

The purpose of part 3 was to decide and assign pin assignments for the rest of the circuit, using the same process as part 2.

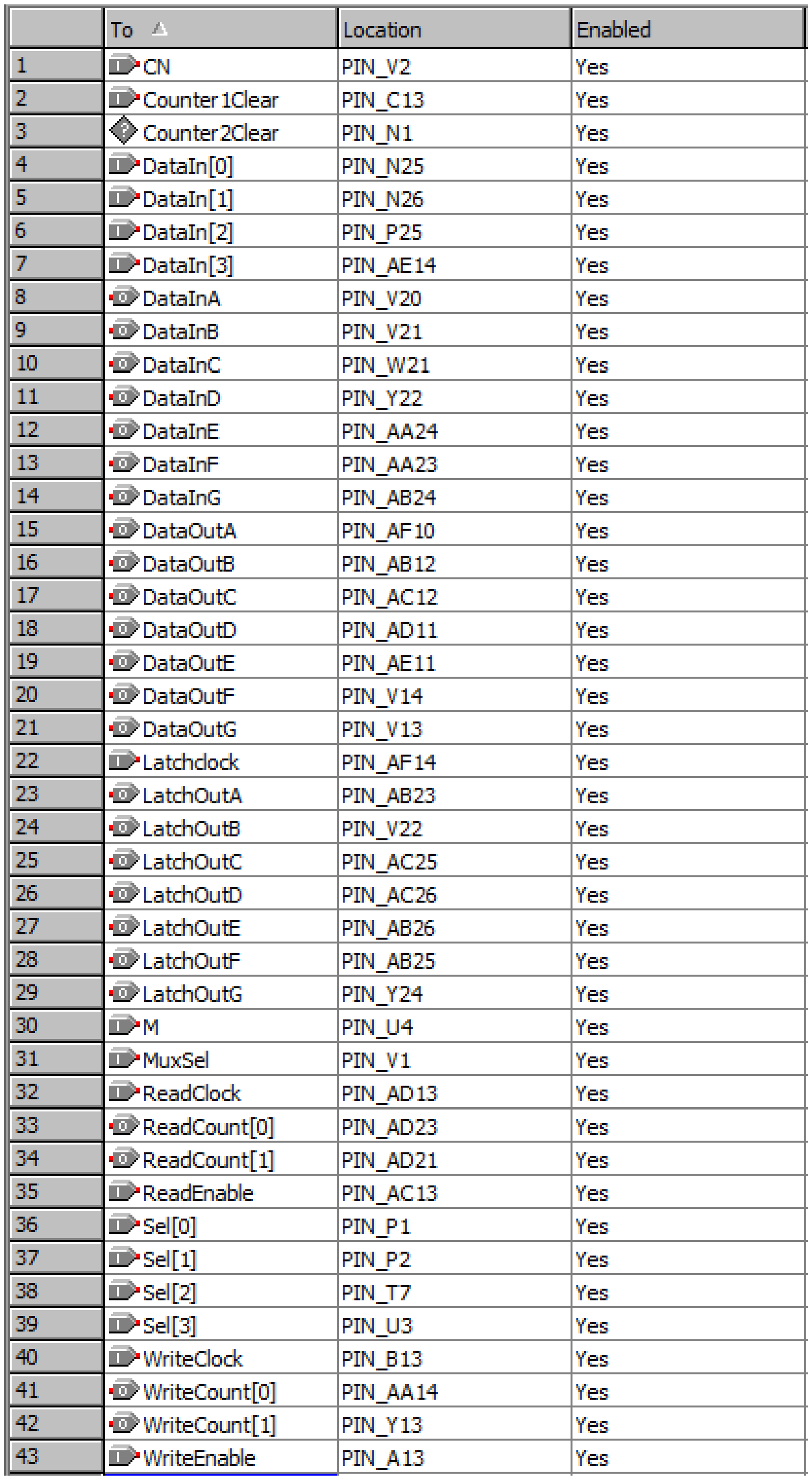
Procedure:

1. Using the Altera DE-2 manual for a full list of pin locations, all the inputs were assigned to switches on the FPGA board. No push buttons were used, as the button seemed more prone to input error (i.e. sticky presses and jamming).

2. In addition to the inputs, extra outputs were added to make the circuit more user-friendly. This included an extra display for the latch output, since it was one of the operands, and the read and write addresses as red LEDs. *Figure 4* below shows the all pin assignments, and Figure 5 shows the location of all pin assignments on the actual FPGA board:

Results:

Figure 4: All FPG Board Pin Assignments



	To	Location	Enabled
1	CN	PIN_V2	Yes
2	Counter1Clear	PIN_C13	Yes
3	Counter2Clear	PIN_N1	Yes
4	DataIn[0]	PIN_N25	Yes
5	DataIn[1]	PIN_N26	Yes
6	DataIn[2]	PIN_P25	Yes
7	DataIn[3]	PIN_AE14	Yes
8	DataInA	PIN_V20	Yes
9	DataInB	PIN_V21	Yes
10	DataInC	PIN_W21	Yes
11	DataInD	PIN_Y22	Yes
12	DataInE	PIN_AA24	Yes
13	DataInF	PIN_AA23	Yes
14	DataInG	PIN_AB24	Yes
15	DataOutA	PIN_AF10	Yes
16	DataOutB	PIN_AB12	Yes
17	DataOutC	PIN_AC12	Yes
18	DataOutD	PIN_AD11	Yes
19	DataOutE	PIN_AE11	Yes
20	DataOutF	PIN_V14	Yes
21	DataOutG	PIN_V13	Yes
22	Latchclock	PIN_AF14	Yes
23	LatchOutA	PIN_AB23	Yes
24	LatchOutB	PIN_V22	Yes
25	LatchOutC	PIN_AC25	Yes
26	LatchOutD	PIN_AC26	Yes
27	LatchOutE	PIN_AB26	Yes
28	LatchOutF	PIN_AB25	Yes
29	LatchOutG	PIN_Y24	Yes
30	M	PIN_U4	Yes
31	MuxSel	PIN_V1	Yes
32	ReadClock	PIN_AD13	Yes
33	ReadCount[0]	PIN_AD23	Yes
34	ReadCount[1]	PIN_AD21	Yes
35	ReadEnable	PIN_AC13	Yes
36	Sel[0]	PIN_P1	Yes
37	Sel[1]	PIN_P2	Yes
38	Sel[2]	PIN_T7	Yes
39	Sel[3]	PIN_U3	Yes
40	WriteClock	PIN_B13	Yes
41	WriteCount[0]	PIN_AA14	Yes
42	WriteCount[1]	PIN_Y13	Yes
43	WriteEnable	PIN_A13	Yes

Figure 4 above: All the pin assignments above are shown on the board on the next page in *Figure 5* (see corresponding row number in chart above for labelling key):

Figure 5: FPG Board Pin Locations

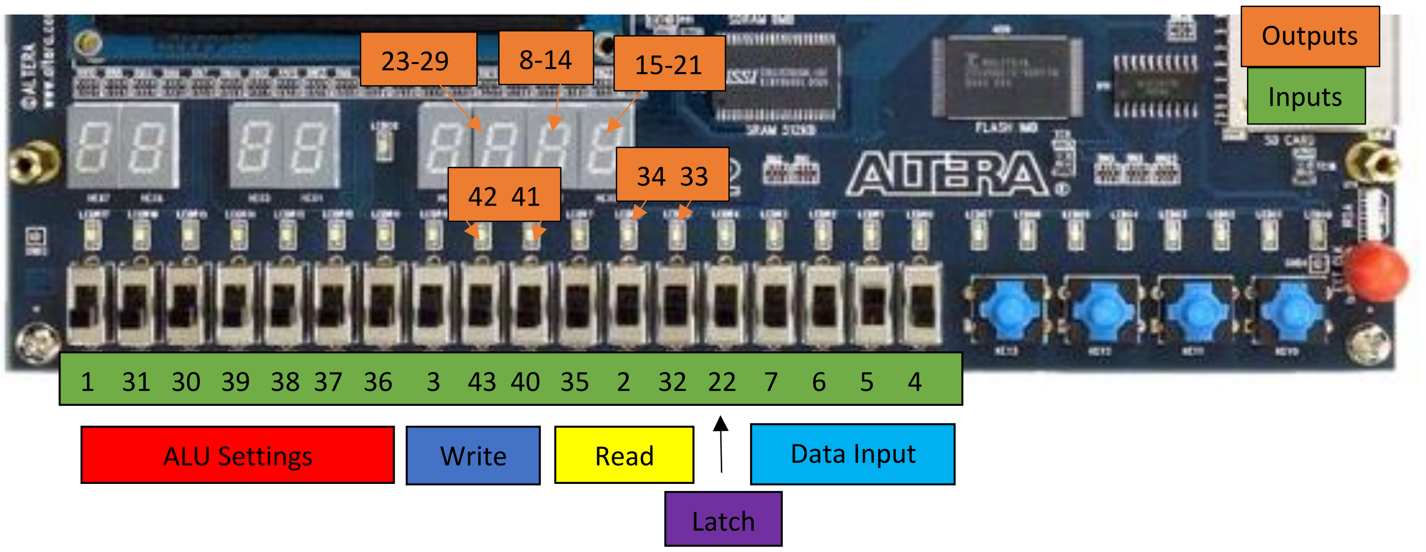


Figure 5 above shows the inputs (green) and outputs (orange) of the pins on the actual FPGA board. The organization of the inputs is also displayed underneath the inputs bar. Note that all numbers refer to the row in *Figure 4* for their pin numbers and uses.

Conclusion:
All pins were successfully assigned and organized in a user-friendly manner. This made operating the FPGA board much easier in the next part, so that functionlity could be tested quickly and accurately.

Part IV: Programming the FPGA

Purpose:
The purpose of part 4 was to download the schematic and pin assignments to the FPGA board and begin testing for functionality.

Procedure:
1. After powering the board with a 9-V supply and connecting it to the desktop via a USB-A cable, the schematic was configured for downloading onto the board by adding the schematic (as top level entry) .sof file to the “Programmer” window, setting mode to JTAG, and setting hardware to USB Blaster.

2. Once the code was downloaded, several operations were tested from the previous lab, now on the board. Slips of paper were added to help demarcate the different switch uses. *Figures 6 and 7* on the following page shows the physical board and outputs.

Results:

Figure 6: Simple Addition Functionality Test

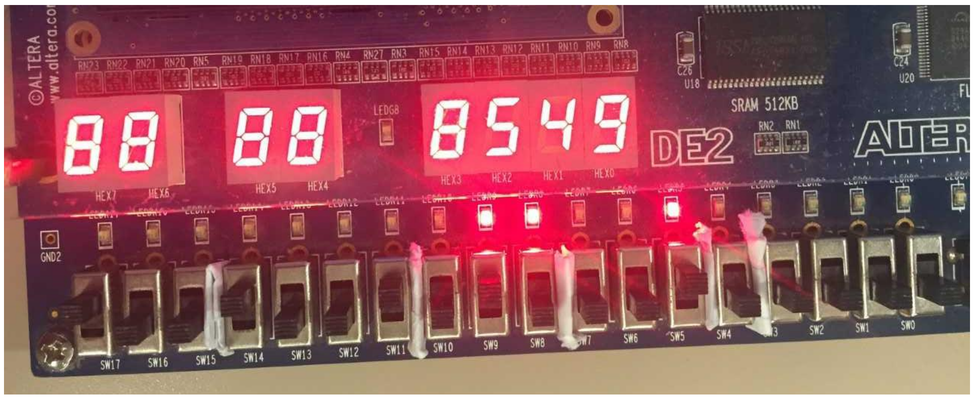


Figure 6 above shows a simple addition operation to test the boards functionality. “8”s are null values, so outputs are only being written to the three right-most 7-segment displays. From left to right, a 5 is latched, a 4 is being read from address 1, and the ALU is adding these two together to output a 9 because of the A plus B settings (switches 11-17). Reference *Figure 5* for what each specific switch does.

Figure 7: Simple Logic Functionality Test

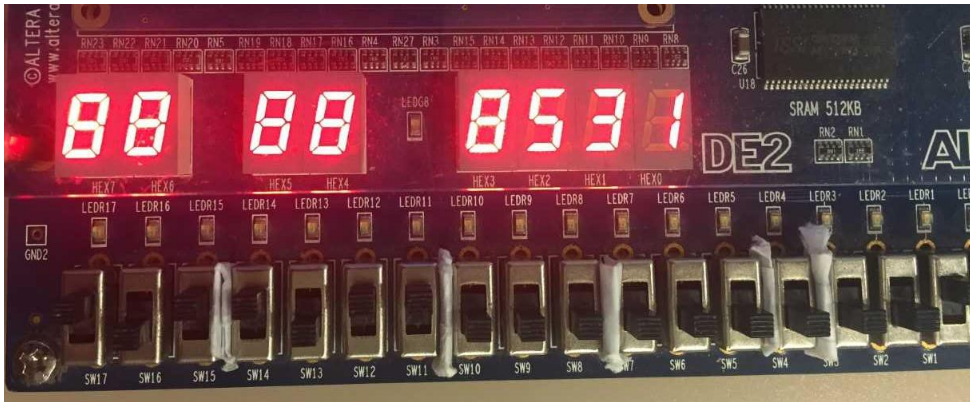


Figure 7 above shows a simple logic operation to test the boards functionality. Again “8”s are null values, so outputs are only being written to the three right-most 7-segment displays. From left to right, a 5 is latched, a 3 is being read from address 0, and the ALU is ANDing these two together to output a 1 because of the A AND B settings (switches 11-17). Reference Figure 5 for what each specific switch does.

Both operations were successful, and other multi-step operations were as well. A multi-number addition was also tested utilizing the MUX select function to read in the ALU output and was also successful.

Conclusion:
All results were as expected. Th Quartus II schematic and pin assignments were successfully downloaded to the the FPGA board, which was then able operated correctly with all of the operations tested, just as the protoboard circuit.

Part V: Recoding Components in VHDL

Purpose:

The purpose of part 5 was to learn introductory VHDL coding to implement self-designed VHDL parts to replace the pre-made parts in the lab. Functionality was tested both through waveform simulation and on the FPGA

Procedure:

1. The DispBin.vhd file added earlier to the project was then opened to reveal a full VHDL code for the 7-segment LED displays that were removed earlier. A symbol file was created directly from this VHDL code to create a replacement part that was then connected to the segment decoders, keeping in mind to change inputs and outputs to busses. After replacement, the schematic was compiled and simulated using the waveform from the previous lab to test for proper functionality of the VHDL components. *Figure 8* below in the results section shows the output waveform of the schematic using the 4-operations file used in the previous lab.

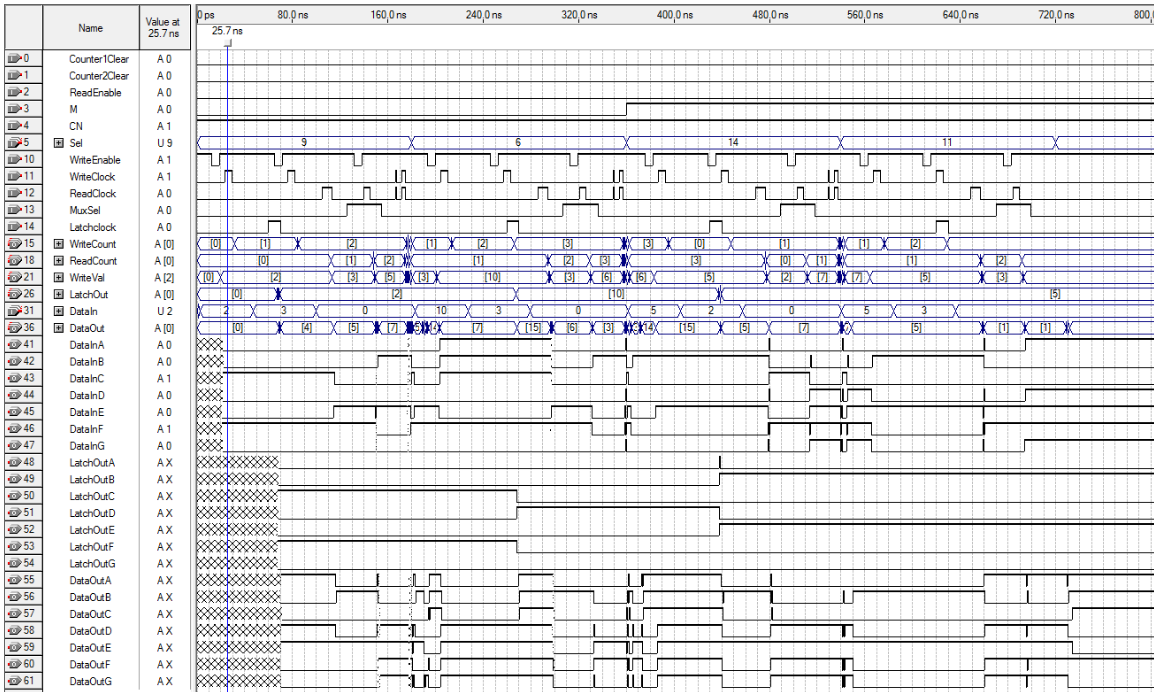
2. Once functionality of the single decoder was confirmed, the other two VHDL 7-segment displays were added. Additionally, the VHDL file itself was edited to display A, b, C, d, E, and F in addition to 0-9, for full hexadecimal representation. Below, *Figure 9* shows the waveform simulation with no VHDL for reference, *Figure 10* shows the edited code, and *Figure 11* shows the output waveform with all three VHDL displays implemented in the circuit.

3. In the same way as the 7-segment displays, the 4x2:1 MUX, 4-bit Latch, and Counters were replaced with self-written VHDL code, checking for functionality on both the FPGA and with the waveform simulator. *Figures 12 and 13* show the 4x2:1 MUX VHDL code and waveform simulation report, *Figures 14 and 15* show the 4-bit latch VHDL code and waveform simulation report, and *Figures 16 and 17* show the Counter VHDL code and waveform simulation report. Finally, *Figure 18* shows the entire circuit with all VHDL components.

(Results begin on next page)

Results:

Figure 8: Waveform Simulation with added Single VHDL LED Display



The waveform in *figure 8* above is completely identical to the waveform below in *figure 9*. This is good, because it confirms identical functionality of the VHDL component and old display.

Figure 9: Waveform Simulation for No VHDL Components

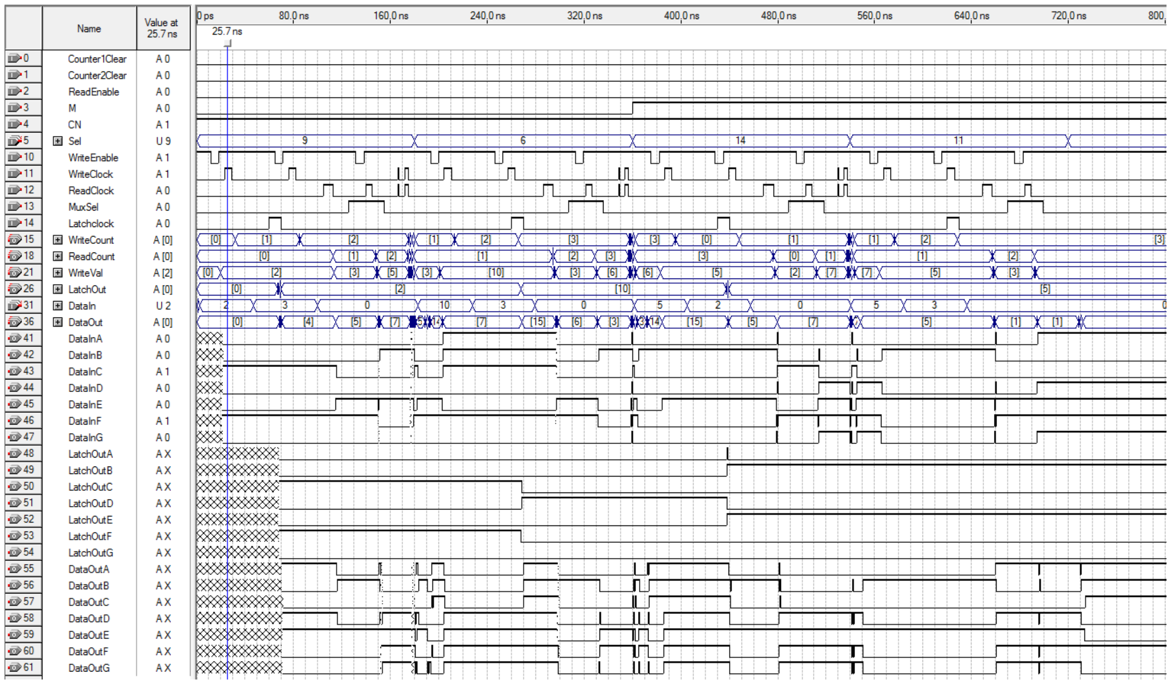
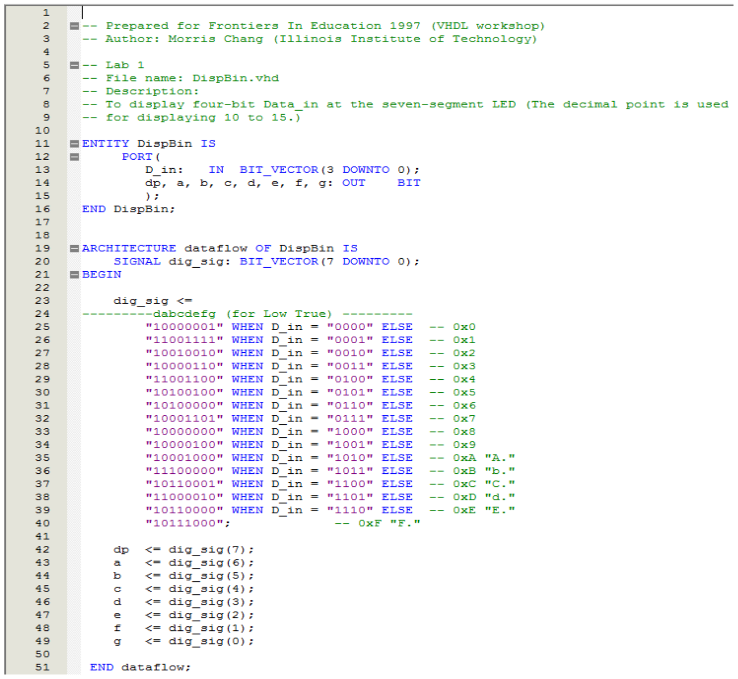


Figure 10: Edited Full Hex Display VHDL Code



```vhdl

-- Prepared for Frontiers In Education 1997 (VHDL workshop)
-- Author: Morris Chang (Illinois Institute of Technology)

-- Lab 1
-- File name: DispBin.vhd
-- Description:
-- To display four-bit Data_in at the seven-segment LED (The decimal point is used
-- for displaying 10 to 15.)

ENTITY DispBin IS
     PORT(
         D_in:   IN  BIT_VECTOR(3 DOWNTO 0);
         dp, a, b, c, d, e, f, g: OUT    BIT
         );
END DispBin;


ARCHITECTURE dataflow OF DispBin IS
    SIGNAL dig_sig: BIT_VECTOR(7 DOWNTO 0);
BEGIN

    dig_sig <=
---------dabcdefg (for Low True) ---------
        "10000001" WHEN D_in = "0000" ELSE  -- 0x0
        "11001111" WHEN D_in = "0001" ELSE  -- 0x1
        "10010010" WHEN D_in = "0010" ELSE  -- 0x2
        "10000110" WHEN D_in = "0011" ELSE  -- 0x3
        "11001100" WHEN D_in = "0100" ELSE  -- 0x4
        "10100100" WHEN D_in = "0101" ELSE  -- 0x5
        "10100000" WHEN D_in = "0110" ELSE  -- 0x6
        "10001101" WHEN D_in = "0111" ELSE  -- 0x7
        "10000000" WHEN D_in = "1000" ELSE  -- 0x8
        "10000100" WHEN D_in = "1001" ELSE  -- 0x9
        "10001000" WHEN D_in = "1010" ELSE  -- 0xA "A."
        "11100000" WHEN D_in = "1011" ELSE  -- 0xB "b."
        "10110001" WHEN D_in = "1100" ELSE  -- 0xC "C."
        "11000010" WHEN D_in = "1101" ELSE  -- 0xD "d."
        "10110000" WHEN D_in = "1110" ELSE  -- 0xE "E."
        "10111000";                   -- 0xF "F."

    dp  <= dig_sig(7);
    a   <= dig_sig(6);
    b   <= dig_sig(5);
    c   <= dig_sig(4);
    d   <= dig_sig(3);
    e   <= dig_sig(2);
    f   <= dig_sig(1);
    g   <= dig_sig(0);

 END dataflow;
```

Figure 10 above shows the edited VHDL code for the 7-segment display. Changes were only made in the "dabcdefg" column of the main, editing the outputted segments for 10-15 to display their respective hex characters A-F rather than non-standard symbols.

Figure 11: Waveform Simulation for three VHDL LED Displays

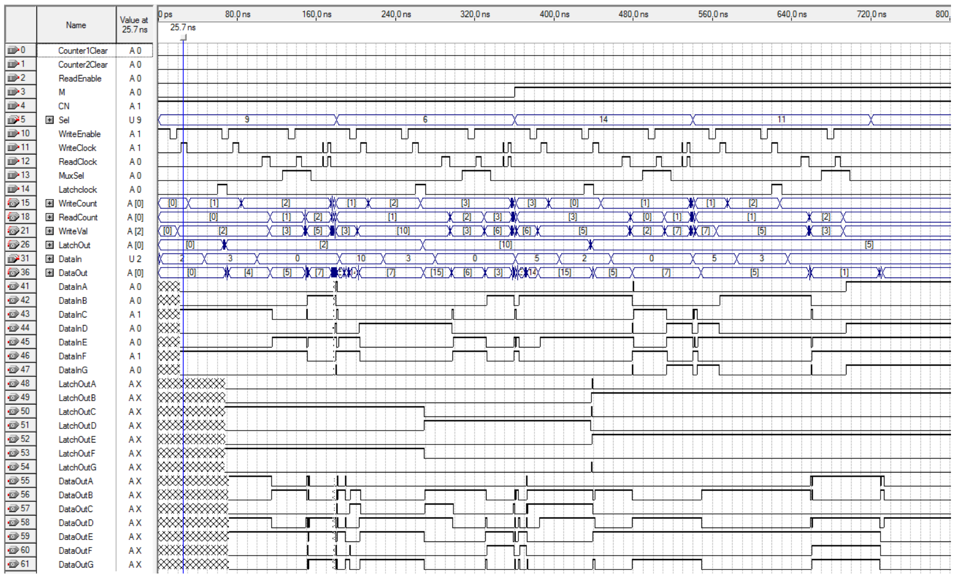
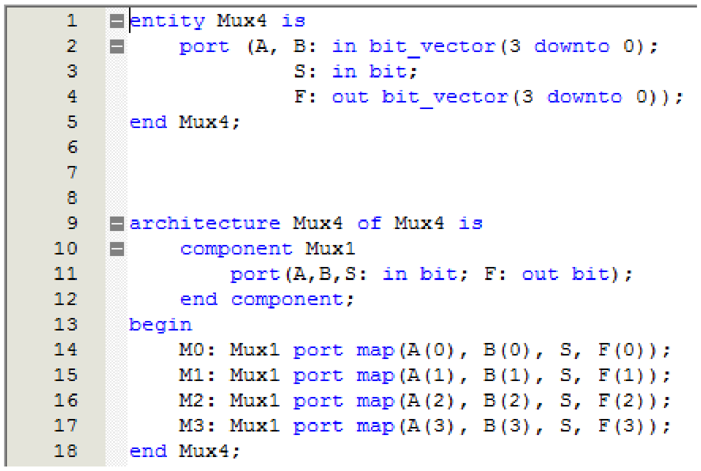


Figure 11 above shows the simulation report for all three improved VHDL displays added. All inputs and outputs are identical to those in *Figures 8 and 9*, except for display values above 9, which now display proper hex values A-F. Functionality was confirmed on the FPGA.

Figure 12: VHDL Code for 4x2:1 MUX



```
entity Mux4 is
    port (A, B: in bit_vector(3 downto 0);
              S: in bit;
              F: out bit_vector(3 downto 0));
end Mux4;



architecture Mux4 of Mux4 is
    component Mux1
        port(A,B,S: in bit; F: out bit);
    end component;
begin
    M0: Mux1 port map(A(0), B(0), S, F(0));
    M1: Mux1 port map(A(1), B(1), S, F(1));
    M2: Mux1 port map(A(2), B(2), S, F(2));
    M3: Mux1 port map(A(3), B(3), S, F(3));
end Mux4;
```

Figure 12 above shows the VHDL code for the 4x2:1 MUX. It was created by referencing the display VHDL code for structure, and the VHDL code for a single 2:1 MUX, which was imported into the code. The 1x2:1 MUX code was used to extrapolate a 4x2:1 MUX component.

Figure 13: Waveform Simulation with added VHDL 4x2:1 MUX

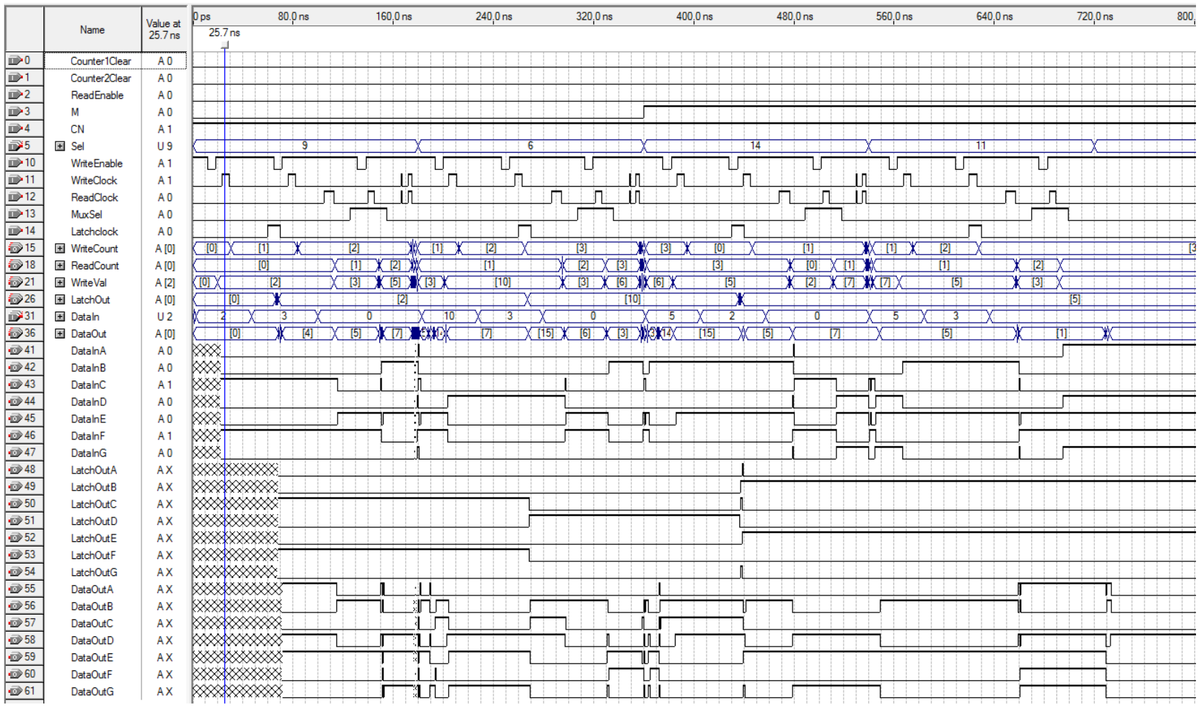
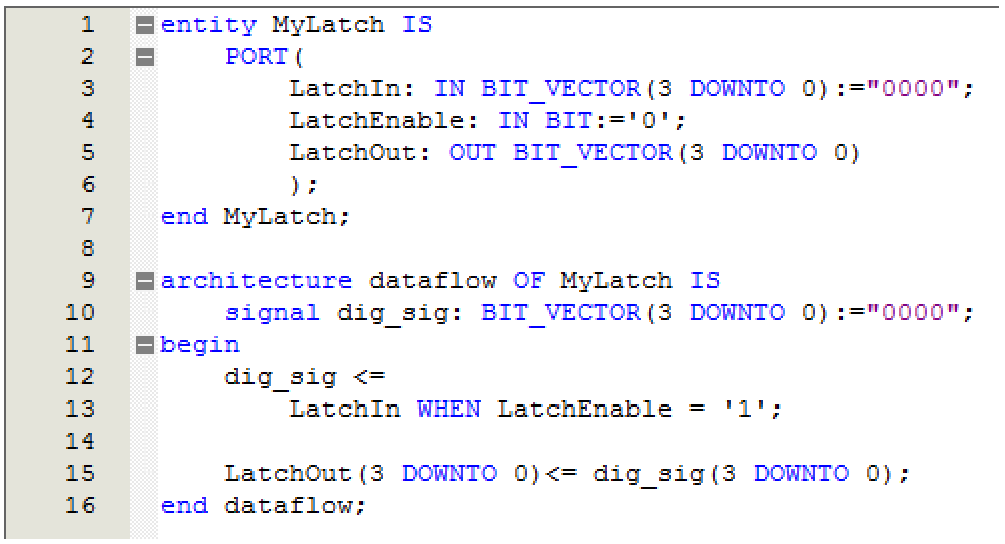


Figure 13 above shows the output waveform with the added VHDL MUX component. Again, the simulation waveform is identical to the waveform before component replacement, indicating that the component is functioning identically to the old one as intended. Functionality was verified on the FPGA.

Figure 14: VHDL Code for 4-Bit Latch



```
entity MyLatch IS
    PORT(
        LatchIn: IN BIT_VECTOR(3 DOWNTO 0):="0000";
        LatchEnable: IN BIT:='0';
        LatchOut: OUT BIT_VECTOR(3 DOWNTO 0)
        );
end MyLatch;

architecture dataflow OF MyLatch IS
    signal dig_sig: BIT_VECTOR(3 DOWNTO 0):="0000";
begin
    dig_sig <=
        LatchIn WHEN LatchEnable = '1';

    LatchOut(3 DOWNTO 0)<= dig_sig(3 DOWNTO 0);
end dataflow;
```

Figure 14 above shows the VHDL code for the replacement 4-bit latch. It operates by setting a conditional output based on whether LatchEnable is HIGH or LOW, just as the physical component acts.

Figure 15: Waveform Simulation with added Single VHDL 4-Bit Latch

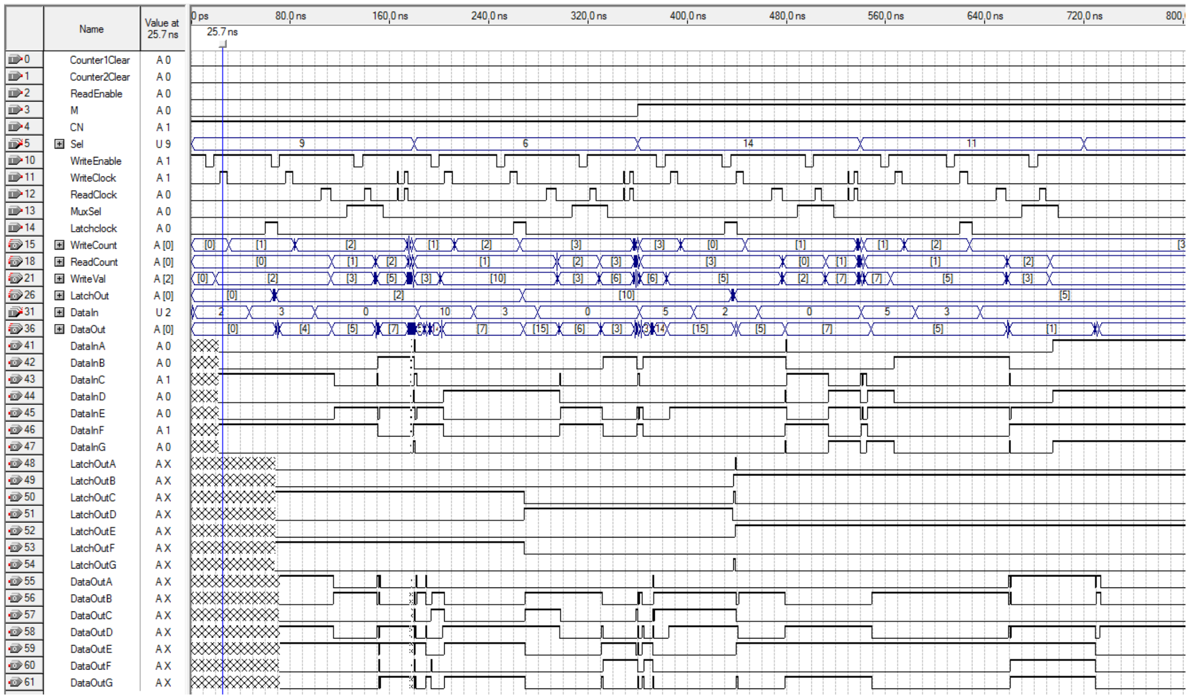
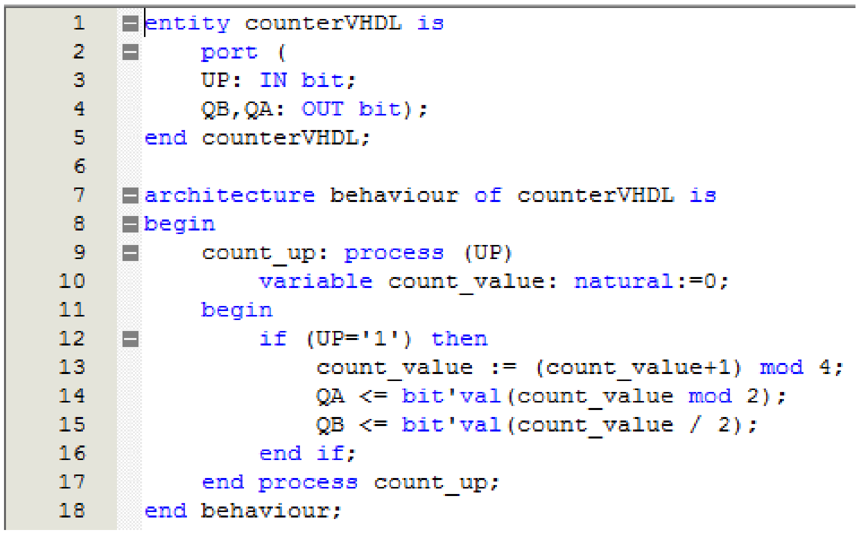


Figure 14 above shows the output waveform with the added VHDL Latch component. Again, the simulation waveform is identical to the waveform before component replacement, indicating that the component is functioning as intended. Functionality was verified on the FPGA.

Figure 16: VHDL Code for Counter



```
entity counterVHDL is
    port (
    UP: IN bit;
    QB,QA: OUT bit);
end counterVHDL;

architecture behaviour of counterVHDL is
begin
    count_up: process (UP)
        variable count_value: natural:=0;
    begin
        if (UP='1') then
            count_value := (count_value+1) mod 4;
            QA <= bit'val(count_value mod 2);
            QB <= bit'val(count_value / 2);
        end if;
    end process count_up;
end behaviour;
```

Figure 16 above shows the VHDL code for the replacement Counter. It operates by using modulus 4 to keep the count value from 0-3, and then splits up that modded binary value via a combination of modulus 2 and division by 2, a trick learned in ECE 142 Computer Organization. Doing so allows the count value to be split into two output bits, just as in the schematic.

Figure 17: Waveform Simulation with added Single VHDL Counters

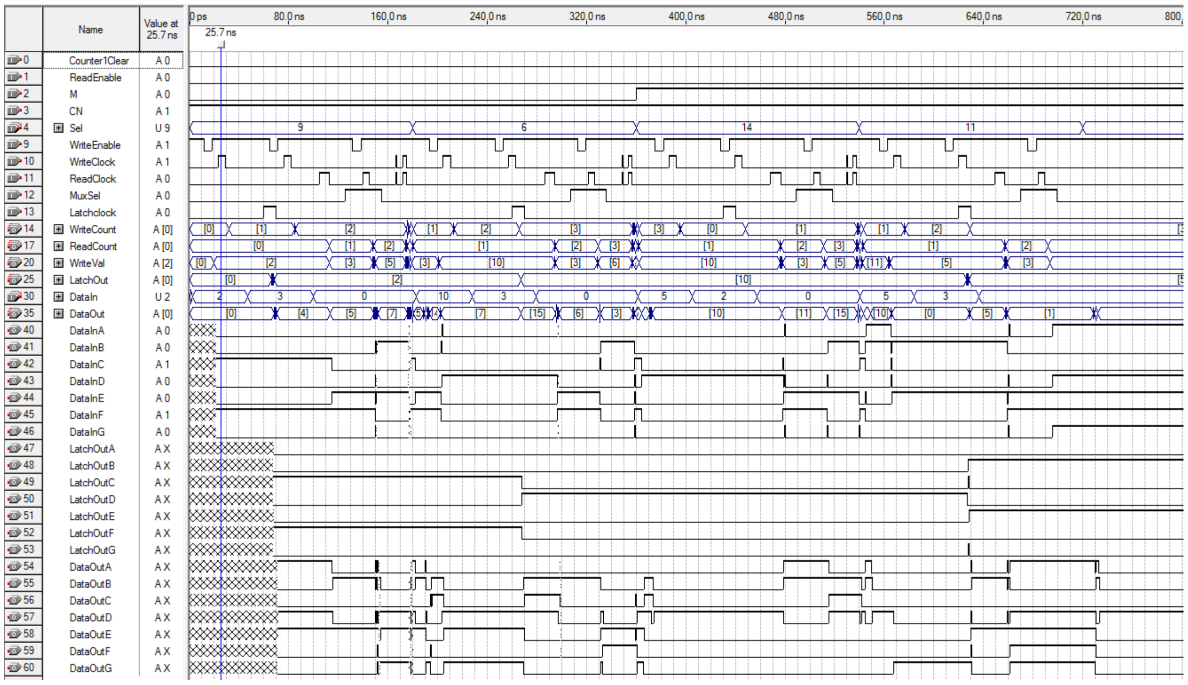
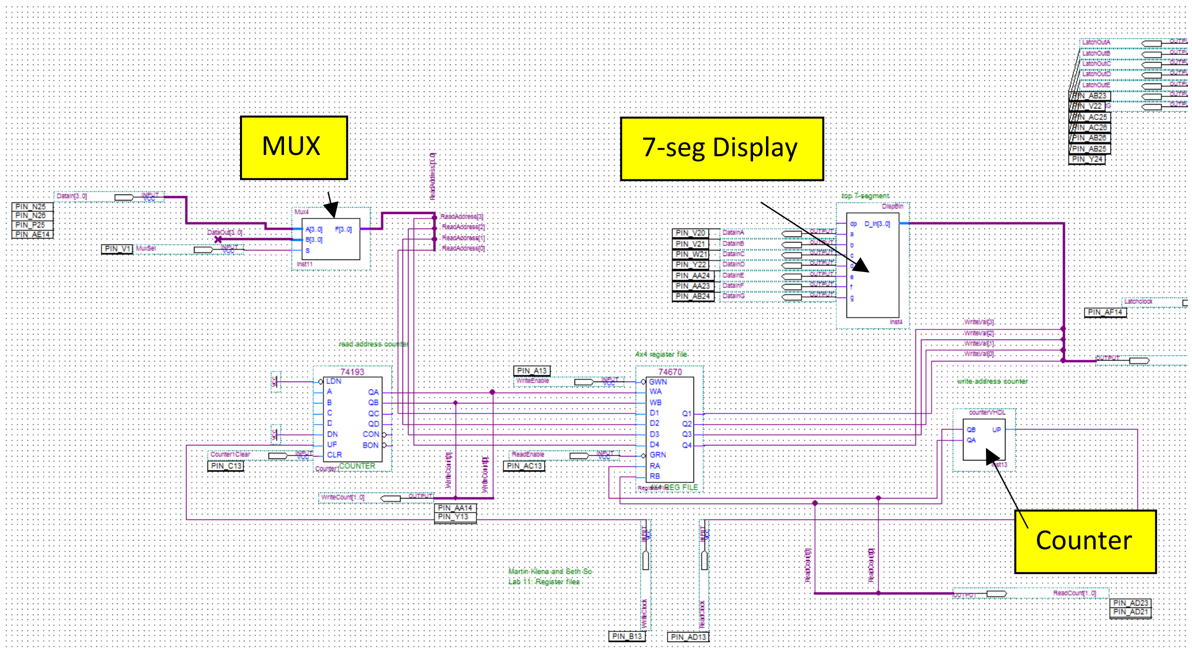
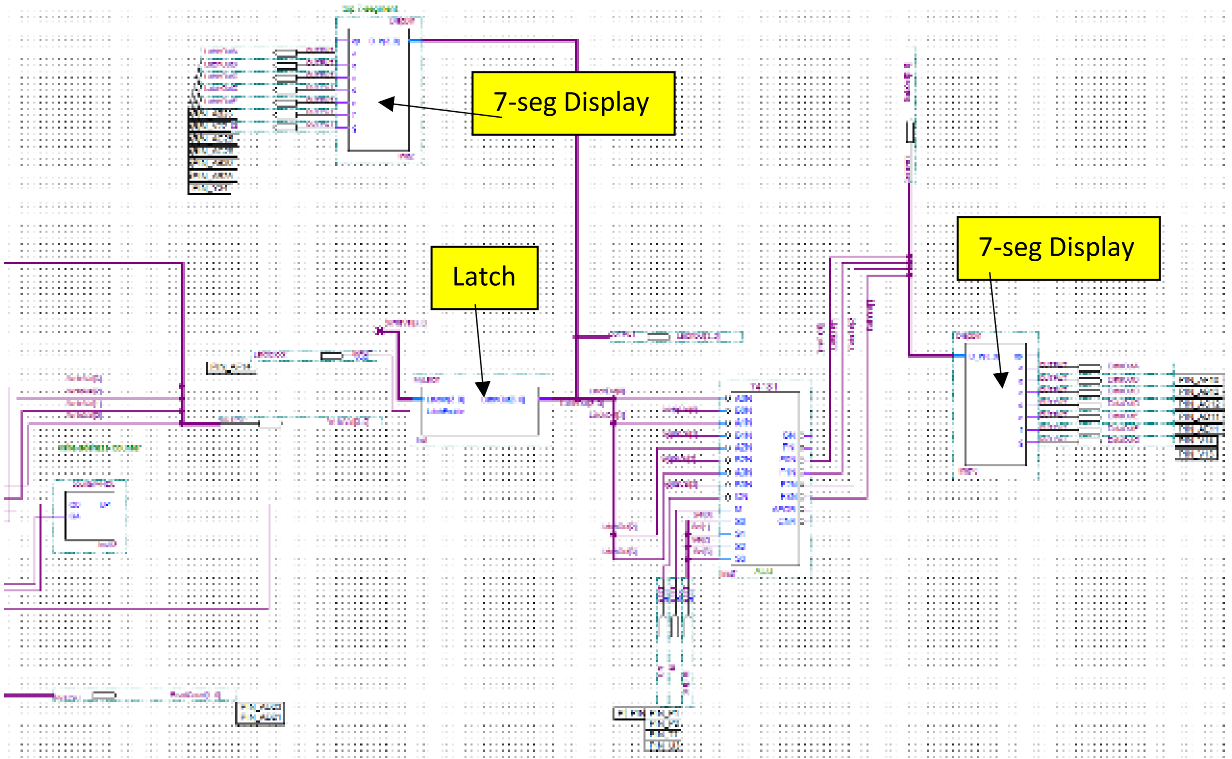


Figure 13 above shows the output waveform with the added VHDL Counter component. Again, the simulation waveform is identical to the waveform before component replacement, indicating that the component is functioning as intended. Functionality was verified on the FPGA.

Figure 18: Final Schematic with VHDL Components.



(Left Half)



(Right Half)

Figure 18 above shows the final schematic, with the different components replaced by VHDL equivalents. A successful demonstration of the board was shown to Vianney Mixtur.

Conclusion:
All results were as expected. The FPGA board was able to perform as it was intended, producing the same outputs as those generated in the Quartus II simulation and on the protoboard ciruit. The major advantage of the FPGA was that nothing had to be built, since everything was on one integrated circuit chip. It was fast, since different versions of the cicuit could rapidly be tested by downloading the code to the board. VHDL coding was also introduced.

Summary:

The goal of Laboratory 13 was to adapt the 4-bit ALU circuit from lab 12 to the Altera DE2 FPGA board. After successful upload and testing of the schematic to the FPGA, parts were sequentially replaced with VHDL equivalents to demonstrate the FPGA's ability to read VHDL coding. Rigorous testing was conducted at each stage to prevent build up of error and find problems at their root cause. In the end, the board was able to successfully perform all functions intended.

(References on next page)

References:

- 74193 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464854_2/courses/2184_UPITT_ECE_0501_SEC1010/74193.pdf

- 74247 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464855_2/courses/2184_UPITT_ECE_0501_SEC1010/74247.pdf

- 74670 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464856_2/courses/2184_UPITT_ECE_0501_SEC1010/74670.pdf

- 7400 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464846_2/courses/2184_UPITT_ECE_0501_SEC1010/7400.pdf

- 74157 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464843_2/courses/2184_UPITT_ECE_0501_SEC1010/74AC157.pdf

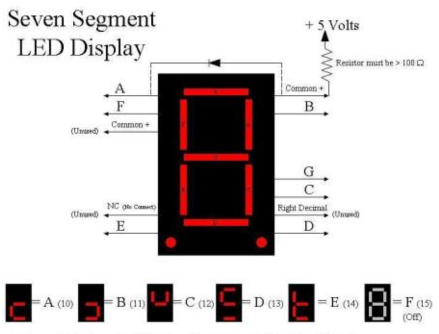
- 7474 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464850_2/courses/2184_UPITT_ECE_0501_SEC1010/7474.pdf

- 74181 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464853_2/courses/2184_UPITT_ECE_0501_SEC1010/74181.pdf

- Altera DE2 Manual: https://courseweb.pitt.edu/bbcswebdav/pid-24291885-dt-content-rid-23464866_2/courses/2184_UPITT_ECE_0501_SEC1010/DE2_UserManuall.pdf

- Intro to VHDL PPT: https://courseweb.pitt.edu/webapps/blackboard/execute/content/file?cmd=view&content_id=_24291848_1&course_id=_448088_1&framesetWrapped=true

- LED outputs to Display Conversion:



7 Segment Display symbols for Hexadecimal values A-F in old set of LEDs.