Abem Lucas & Alan Rieger & Dylan Thornburg
ELEN 21L (59602)
5/11/23

Lab 5: 4-bit Ripple-Carry Adder

Introduction:

In this lab we were able to add together two 4-bit numbers using Hierarchical Design Methods instead of utilizing truth tables and a 2-level SoP or PoS circuit which would have been long and impractical. We started off by implementing the 1-bit Full-Adder then creating the 4-bit Ripple Carry Adder and testing our design.

Procedures:

We worked ahead of the lab on the verilog code for the four-bit adder so we didn't have to do much thinking on that. We had some verilog problems that Allen Shelton (our TA) assisted us in solving such as bad hierarchy and minor incorrect format. After fixing these conventions we had a successful upload. Once uploaded, it worked perfectly on the first try.

Figure 1: Final Schematics

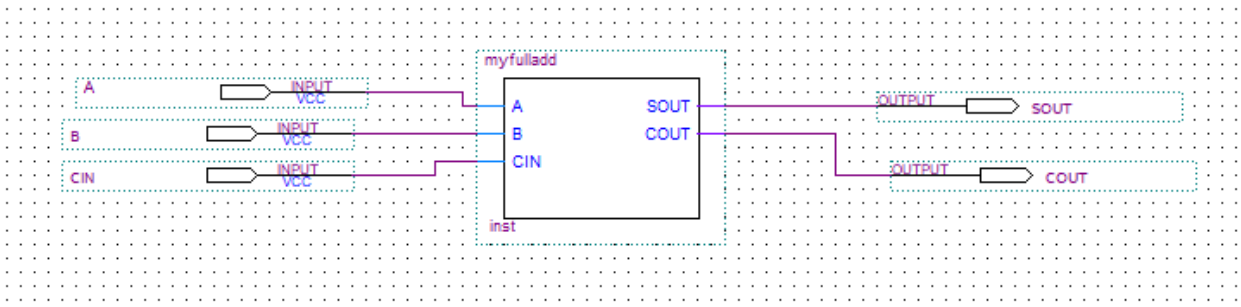


Figure 2: Verilog Code

```
module myfulladder(A,B,CIN,SOUT,COUT);
        input A,B,CIN;
        output SOUT,COUT;
        assign SOUT=A^B^CIN;
        assign COUT=(A&B)|(A&CIN)|(B&CIN);
endmodule

module fourbitadder (A,B,CIN,SOUT,COUT,V);
        input [3:0] A, B;
        input CIN;
```

```
        output [3:0] SOUT;
        output COUT, V;
        wire [3:1] C;

        myfulladder zero (A[0],B[0],CIN,SOUT[0],C[1]);
        myfulladder one (A[1],B[1],C[1],SOUT[1],C[2]);
        myfulladder two (A[2],B[2],C[2],SOUT[2],C[3]);
        myfulladder three (A[3],B[3],C[3],SOUT[3],COUT);
        assign V = C[3]^COUT;
endmodule

module bin_7seg(BI_DIGIT,SEG);
  input [3:0] BI_DIGIT;
  output [6:0] SEG;
  reg [6:0] SEG;

  // seg = {g,f,e,d,c,b,a};
  // ---a----
  // |     |
  // f     b
  // |     |
  // ---g----
  // |     |
  // e     c
  // |     |
  // ---d----

  always @(BI_DIGIT)
    case (BI_DIGIT)
      4'h0: SEG = ~7'b0111111;
      4'h1: SEG = ~7'b0000110;
      4'h2: SEG = ~7'b1011011;
      4'h3: SEG = ~7'b1001111;
      4'h4: SEG = ~7'b1100110;
      4'h5: SEG = ~7'b1101101;
      4'h6: SEG = ~7'b1111101;
      4'h7: SEG = ~7'b0000111;
      4'h8: SEG = ~7'b1111111;
      4'h9: SEG = ~7'b1100111;
      4'ha: SEG = ~7'b1110111;
      4'hb: SEG = ~7'b1111100;
      4'hc: SEG = ~7'b1011000;
```

```
        4'hd: SEG = ~7'b1011110;
        4'he: SEG = ~7'b1111001;
        4'hf: SEG = ~7'b1110001;
    endcase
endmodule
```

Figure 3: Simulation Waveforms of all the Components

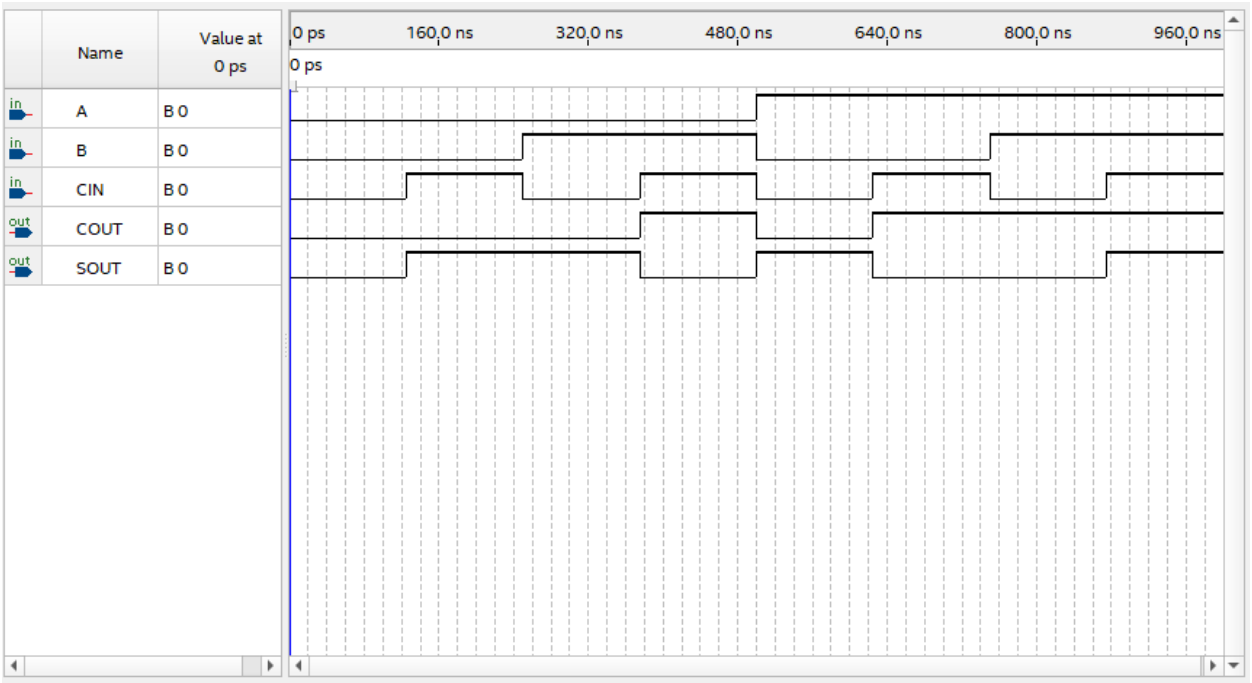


Figure 4: Circuit Diagram for 4-bit Ripple Carry Adder

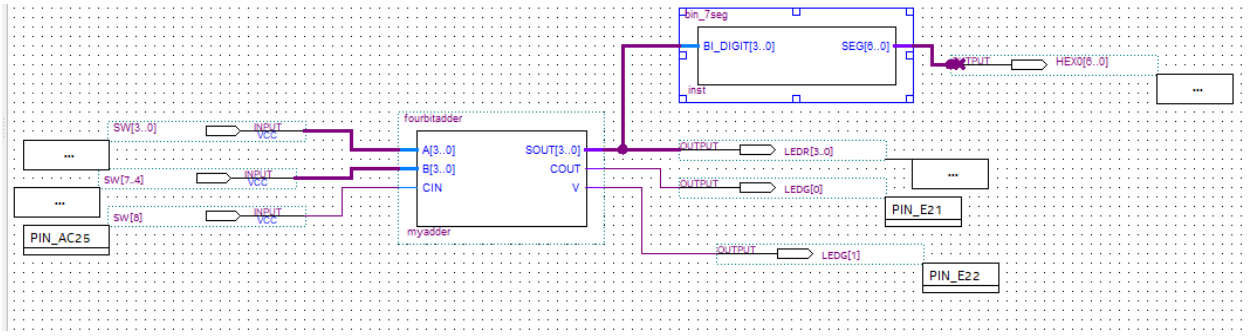
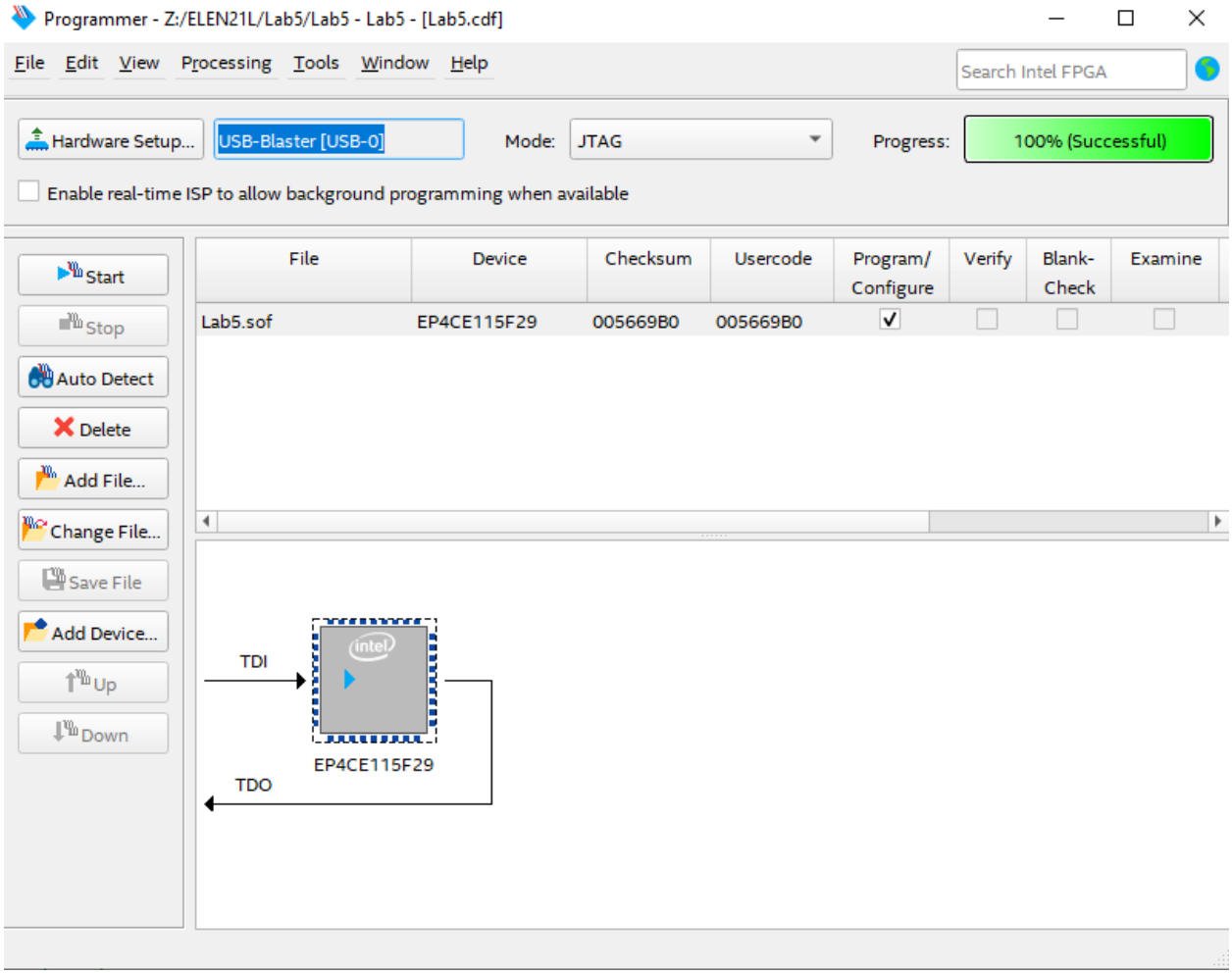
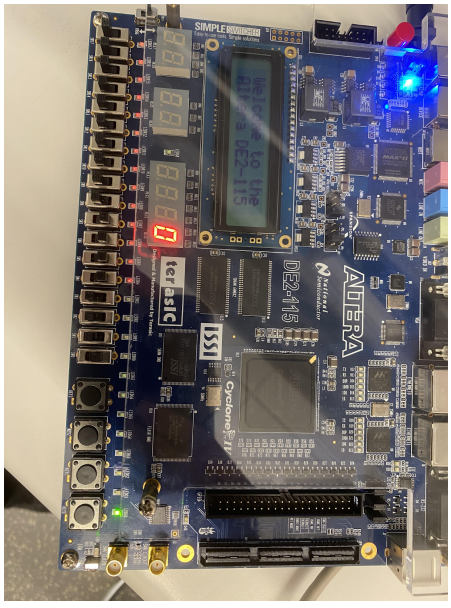
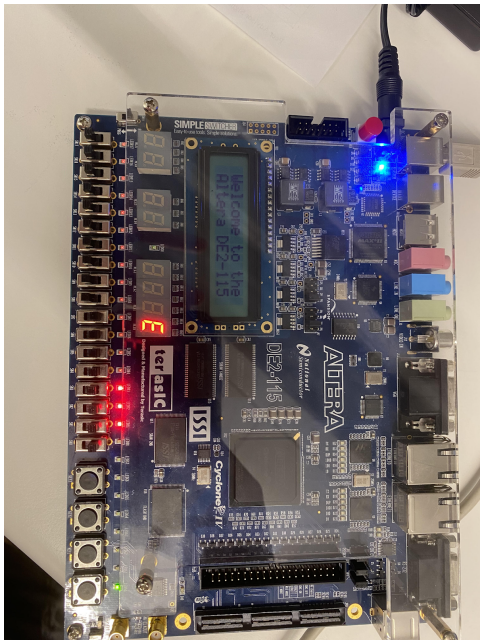


Figure 5: Successful Upload to FPGA



Figures 6 and 7: Working FPGA





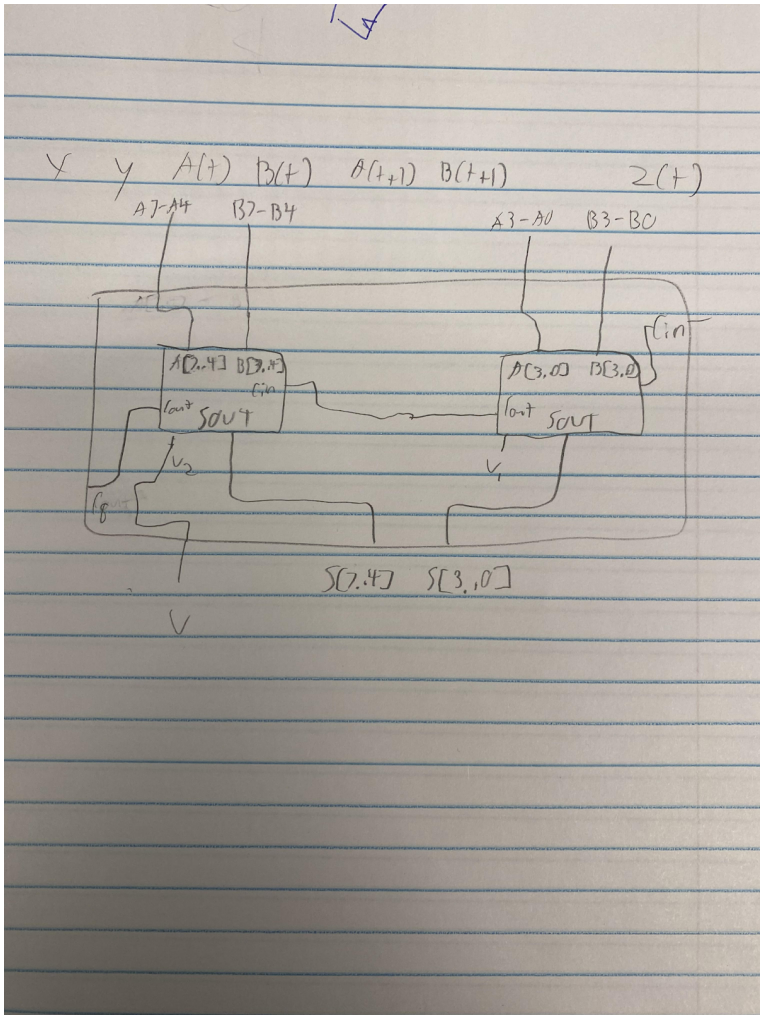
Questions:

1. **Find one pair of X and Y inputs (with C0=0) that would result in the following: o C4 = 0 and V = 0 o C4 = 0 and V = 1 o C4 = 1 and V = 0 o C4 = 1 and V = 1**

   X=0 Y=0
   X=4 Y=4
   X=9 Y=7
   X=C Y=A

2. **If you had to make an 8-bit adder, show how would you do it using only instances of the 4-bit module you have built in this lab? Specifically, show how you would create the C8 output and the V output.**

The c8 output and v would remain similar and we would structure our 8-bit adder using the same method that allowed for the full adder to construct the 2-bit adder like so:



3. **Did the initial testing of your circuit go smoothly or did you encounter incorrect results? If the latter, describe how you determined what was wrong.**

   The circuit testing went well on the FPGA, however, working on creating the circuit had some bumps as we were having trouble importing the pins and compiling overall. However, once we uploaded the circuit it worked perfectly.

4. **If the circuit were completely correct except that X[1] and Y[1] were interchanged in a full adder input, would you be able to detect this based on observing outputs during testing? Why or why not?**

   No you would not be able to detect it because addition is commutative and thus this wouldn't cause a problem. Since X[1] and Y[1], when flicked up, add 2 to the total, thus the solution will not change.

5. **Your TA will make available to you a waveform showing an adder circuit that is not behaving correctly; there is some incorrect wiring inside the adder. o Identify exactly where it's misbehaving, i.e., where it showing incorrect results. o Make a guess (a hypothesis) about what might be wrong, which would explain the incorrect behavior. o Identify another test (i.e., a set of input stimulus) that might help prove or disprove your hypothesis. § [See waveform_incorrect.png on Camino**

The first segment (fourth) the output is two while we're adding 0+0 with the carry in as 1 so the output should be one. The second segment, we're adding 1+0 with carry in as 0 and we're getting two when the output should be one. The third segment, we're adding 0+1 + carry in of 0 and it is still giving us two when it should be one. Lastly the fourth segment is adding 1+1 carry in of 0 and we get 1 when it should be 2. The sum could have the 2^0 bit and 2^1 bit switched around. We could test larger numbers that don't involve the first two bits, and if the math is correct, we would know the problem is with the first two bits and that we were right in our hypothesis.

Testing Design Table:

Inputs			Outputs Observed		
4-bit inputs (in hex)			4-bit sum	Extra Outputs	
X	Y	C0	S	V	C4
0	0	0	0	0	0
0	0	1	1	0	0
1	0	0	1	0	0
0	1	0	1	0	0
2	0	0	2	0	0
4	3	0	7	0	0
4	4	0	8	1	0
9	6	0	F	0	0
9	7	0	0	0	1
C	A	0	6	1	1
E	A	0	8	0	1

E	**A**	**1**	**9**	**0**	**1**

Conclusion:

Overall, we were able to implement the 4-bit Ripple Carry Adder and observe the resulting inputs in 4-bit hex and both 4-bit sum and extra outputs. Through the experiment and results we were able to get, we know how useful using hierarchical design methods is compared to using truth tables and SoP, or PoS equations when dealing with 4-bit numbers.