Santa Clara University

Lab #5: 4-bit Ripple-Carry Adder

ELEN 21L 51306 Tuesday 2:15-5pm
May 9, 2017

Group 7
Matt Kordonsky
Yutong Li

Lab #5: 4-bit Ripple-Carry Adder

I. Objectives:

- Learn to do hierarchical design and create symbols for circuits to be used as building blocks.
- Learn how to use busses for multi-bit inputs such as numbers and ASCII characters.
- Design a 4-bit ripple carry adder using half adders and full adders as building blocks.
- Use 7-segment displays to show inputs and outputs of the design, and add a Verilog module to generate the 7-segment display.

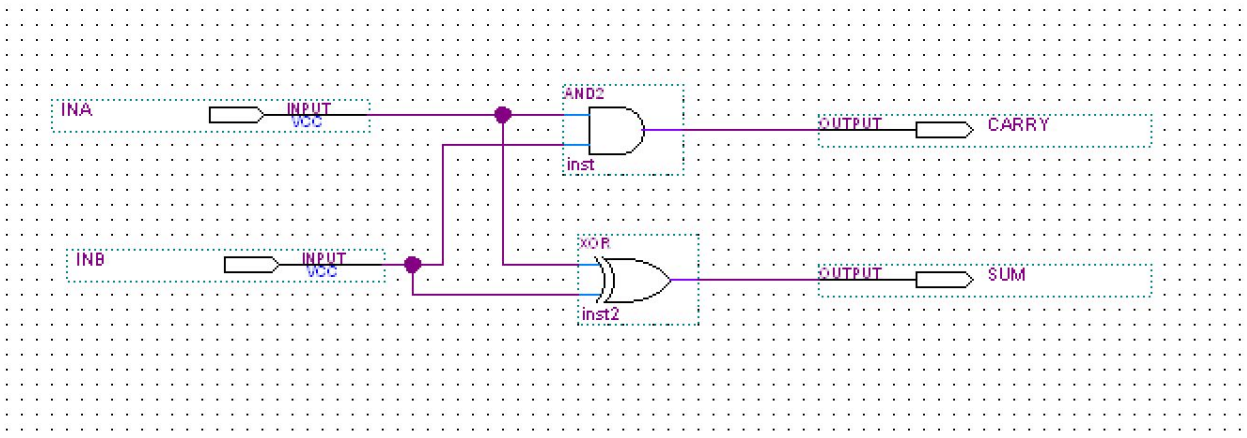
II. Introduction:

From this lab, we should learn how to do hierarchical design and how to use busses for multi-bit inputs. We also should learn design a 4-bit carry adder using half adders and full adders as building blocks. Then we should use 7-segment displays to show inputs and outputs of the design.

III. Procedure:

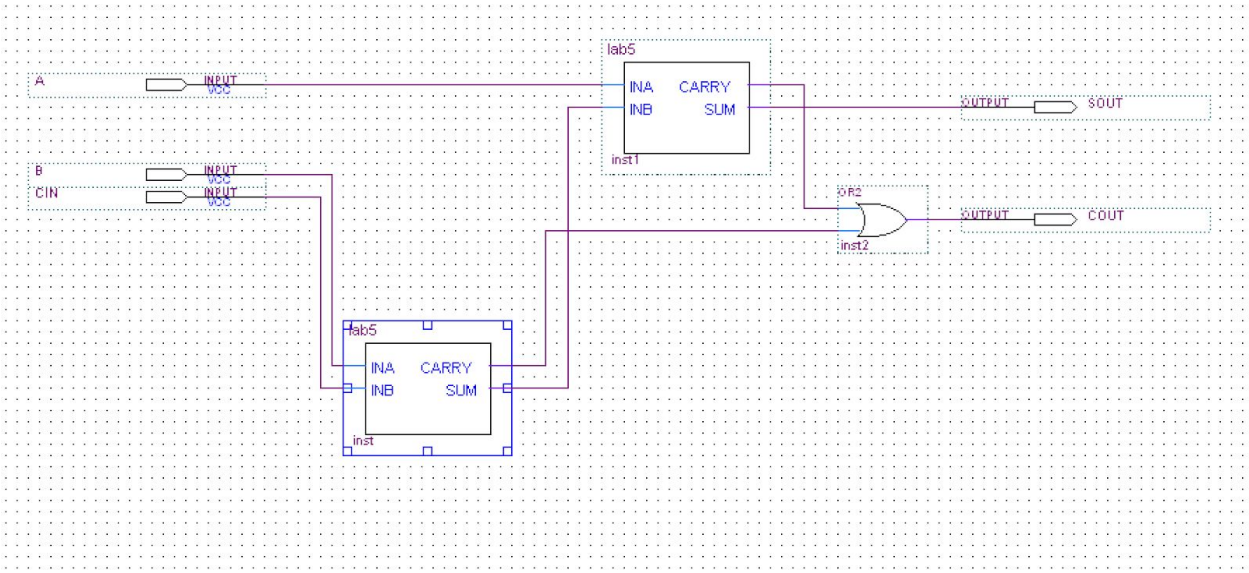
When we first started the lab, we made sure to test our half adder (from the pre-lab) to make sure it works (As seen in figure 1). This consisted of two inputs connected to an AND gate which resulted in the output CARRY and we also put them through an XOR gate which was connected to the output SUM. Once we knew it was working we set it as a usable gate input.

Figure 1: Half Adder



Next we tested our full adder (figure 2). This consisted of three inputs and two outputs. The 3 inputs were connected to 2 half adders and an or gate and outputs resulted in a sum and a carry.

Figure 2: Full Adder



Next we began to make a 4-bit ripple adder (figure 3). To do this, we had 3 inputs: a carry and 2 bus inputs with 4 values total. To make the ripple adder, we used 4 full adders and connected a single branch of the bus inputs to each full adder we also connected the carry in to the first full adder and connected the carry out of each full adder to the next full adder's carry in. The output of the circuit resulted in a bus sum which was the output of each full adder and a single carry out which was the carry output of the last full adder. Once we finished, we tested our ripple adder by making a waveform (figure 4). Once we proved that it worked we made the ripple adder a gate input (figure 5).

Figure 3: 4-bit Ripple Adder

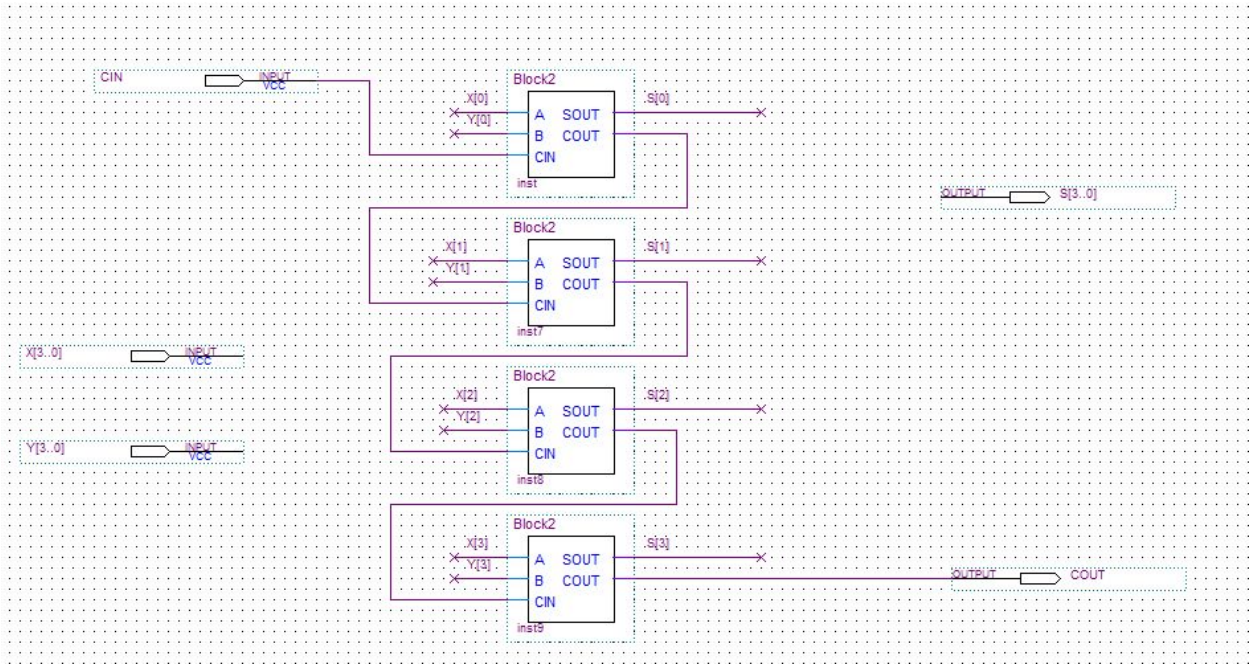


Figure 4: Ripple Adder Waveform

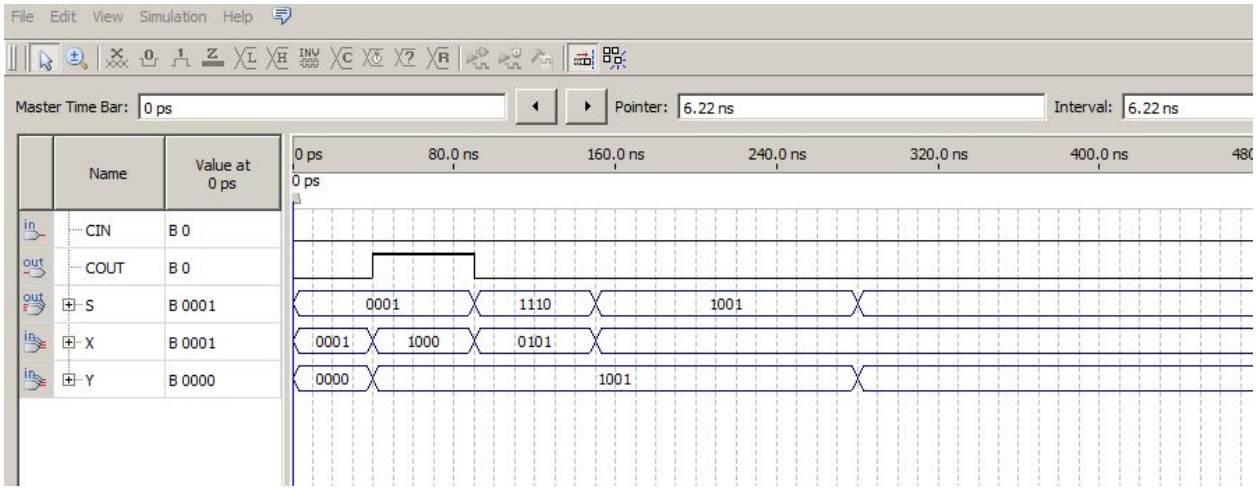
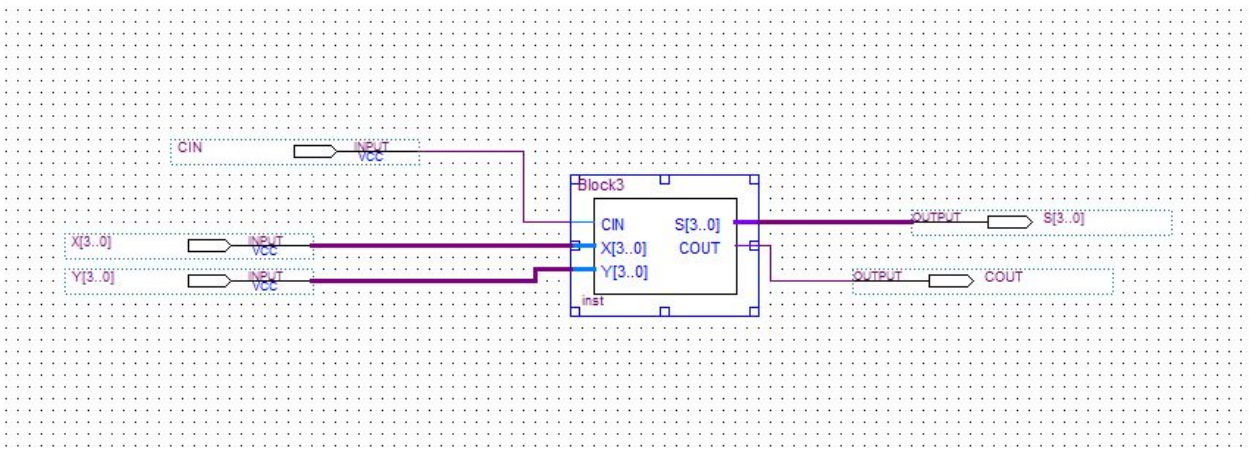


Figure 5: Ripple Adder



Next we made sure to implement our ripple adder to a 7-segment display. We did this by taking the code seen in figure 6 and altering it so that the hexadecimal numbers appeared with the given inputs. After the code was working we implemented the necessary items to our ripple carrier so that we code view it on a 7-segment display as seen in figure 7.

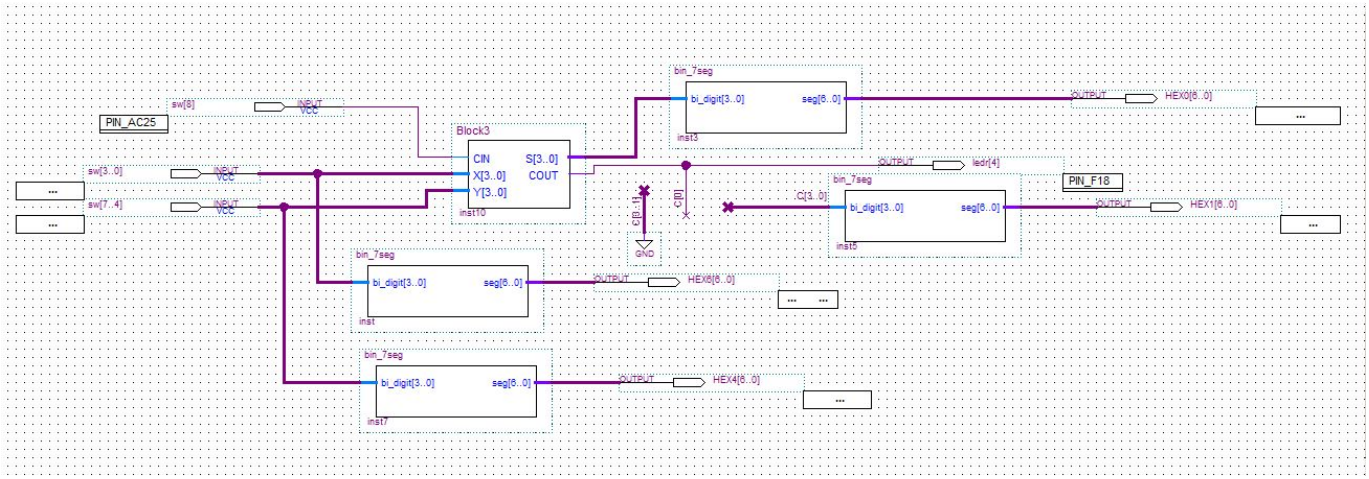
Figure 6: Code for 7-segment display

```
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};


always @ (bi_digit)
case (bi_digit)
        4'h0: seg = ~7'b0111111;
        4'h1: seg = ~7'b0000110        ;          // ---a----
        4'h2: seg = ~7'b1011011        ;          // |      |
        4'h3: seg = ~7'b1001111        ;          // f      b
        4'h4: seg = ~7'b1100110        ;          // |      |
        4'h5: seg = ~7'b1101101        ;          // ---g----
        4'h6: seg = ~7'b1111101        ;          // |      |
        4'h7: seg = ~7'b0000111        ;          // e      c
        4'h8: seg = ~7'b1111111        ;          // |      |
        4'h9: seg = ~7'b1101111        ;          // ---d----
        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 7: Ripple Adder with necessary items for 7-Segment Display



IV. Mistakes that we make:

1. I made a mistake in the half adder, connecting the same input into a gate, which causes the full adder and ripple carry adder to work incorrectly. After realizing the mistake, we redo the half adder, and adjust the full adder, and use the new full adder in out ripple carry adder.
2. When building the ripple carry adder, we don't need any wire to connect the inputs X[0], X[1], X[2], and X[3] to X[3..0], we just need a wire to show this is the input and specify its property.
3. When building the 7-segment display, we are confused by matching the number of the ripple carry adder and COUT, because COUT is only 1 element. So we use bus wire to connect GND and set its property as C[3..1], and set the COUT using node wire and set its property as C[0]. At last, we set the property of the input of the ripple carry adder as C[3..0].
4. When writing the code for the 7-segment display, we didn't get the right display for 4 and c, so we rewrite the code.

V. Final question:

1. When the sum of two numbers is greater than 15, for example, we are adding 9 and 7, the result will be 16 in decimal, but it will be 10 in hexa, because 10 will indicates the number after f. For 16 to 31, the bit of the Sum will start from 0 to f, and the bit of Cout will be 1, and the Cout LED will be on. Whenever the Cout LED is on, we can tell that the addition is overflowing.
2. In the 4-bit ripple carry adder, there is 1 full adder, and in each full adder, there are two half adder, which contains 2 gates. Therefore, we will need 20 logic gates in total.
3. We think that we should use 8 full adder to create a 8-bit ripple carry adder instead of combining 2 4-bit ripple carry adders.
4. If we are building an 8-bit ripple carry adder, we will need 40 logic gates in total.

VI. Conclusion

In this lab, we use the full adder that we created in prelab using the half adder to create the ripple carry adder. The ripple carry adder functions in the way that the output, separated into a 4-bit output called Sum and a 1-bit output called Carry, will be the sum of 2 4-bit inputs. We learn how to use input in group so we don't need so many wires and connections if we use individual inputs and outputs. We also learn how to give the input and output property so we don't need to really assign an input or output to it. When simulating the circuit, we learn to group the inputs and outputs so we don't need to assign 1 or 0 to single inputs or outputs, instead, we only need to assign a 4-bit binary number for example 0010, or 1101 to a selected interval. Besides, when building the 7-segment display circuit, we know that we need to match the number of inputs and outputs or the circuit will not work. Also, in the implementing part to the TA, we learn that if we add 9 and 7 together, we will get 10 as the result, which indicates this is an overflow, which means 10 will be 16 in decimal, also if we add b and 5 together, we will also get 10 as result, which is also an overflow.

VII. Reference lists:

- Dr. Sally Wood, Dr. Samiha Mourad, Dr. Radhika Grover. *Laboratory #5: 4-bit Ripple-Carry Adder* . Spring 2017. Print
- Bin_7seg_temp.txt. Spring 2017. Print
- Dr. Sally Wood, Dr. Shoba Krishnan. *7-Segment Display Tutorial.* Spring 2017. Print

- Altera Corporation - University Program. *Quartus II Introduction Using Schematic Designs.* Spring 2017. Print
- Altera Corporation - University Program. *Quartus II Introduction to Simulation of Verilog Designs.* Spring 2017. Print
- Terasic Technologies Inc. *DE2-115 User Manual.* Spring 2017. Print