Cover Page
CPE 324-02: Advanced Logic Design Laboratory

Lab 2

8 Bit Adder Subtractor

Submitted by: Nolan Anderson

Date of Experiment: 02/07/21

Report Deadline: 02/07/21

1. Introduction

1.1 - What is to be studied, what is the purpose, and how is this purpose accomplished?

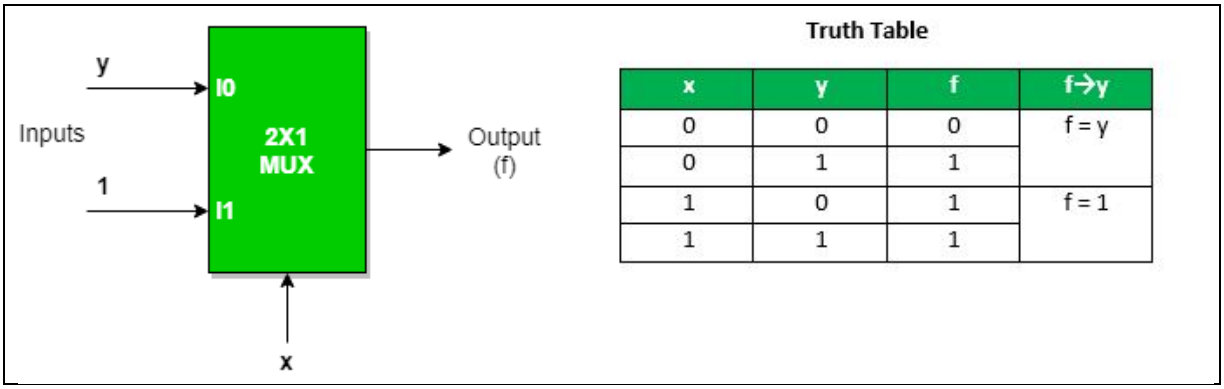
Lab 2 covers a variety of topics but most notable is understanding the eight-bit sub adder by using LED's and switches on the board. Continually, this lab helps us understand how an eight-bit sub adder is implemented all the way down to the full subtractor and adder. Lastly, this lab continues to develop our skills and knowledge of the Quartus software, design schematics, and Verilog code. To attain this knowledge, we need to read the lab 2 assignment and create the designs shown in the screenshot. Continually, there is a lab 2 instructions file that states the process in more detail.

2. Experiment Description

2.1 Theory, analysis, and purpose

2.1.1 - 2 to 1 Multiplexer

A 2 to 1 multiplexer implemented in Quartus is displayed in figure 7 of this report. Figure 1 describes the inputs and outputs. Multiplexers are designed to send one or more analogue or digital signals through an output on a transmission line. Their purpose is to switch several input lines to a common output of a control signal. They can be either analogue or digital.



Truth Table

x	y	f	f→y
0	0	0	f = y
0	1	1	
1	0	1	f = 1
1	1	1	

Figure 1: 2 to 1 multiplexer truth table

2.1.2 Full Subtractor / Adder

Full subtractor / adders are another form of a combinatorial and its quartus implementation can be seen in Figure 6. Figure 2 displays its truth table. Adder subtractors use 2 to 1 multiplexer and an assortment of combinatorial gates to produce an output.

INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

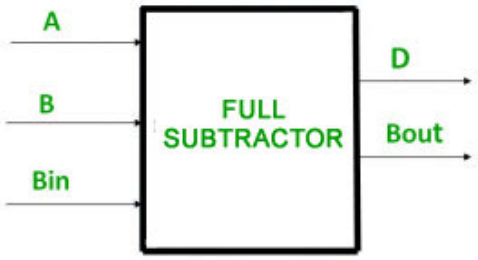


Figure 2: Full adder / subtractor

2.1.3 Four-bit Adder / Subtractor

Four-bit adder subtractors are another important piece when working up to an eight-bit adder subtractor. These designs use four full subtractors and adders to produce an ouput. Figure 5 displays the design in figure 8, and figure 3 displays its truth table.

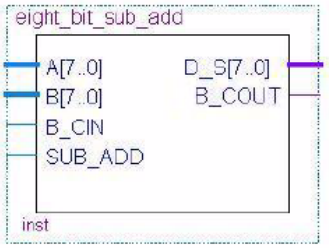
	A				B				Sum				Carry
Cin	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3: Four-bit adder subtractor truth table

2.1.4 Eight-bit Adder Subtractor

Lastly, the eight bit adder subtractor allows us to perform operations on 2 eight bit numbers and consists of 2 four-bit adder subtractors. This design is what we will be using for the rest of

the lab report. Figure 5 displays the implementation in Quartus and figure 4 displays the overall functionality.



Function:

```
if (SUB_ADD) = '1' then
  D_S <= A - B - B_Cin --subtract B and B_Cin from A
  if (B>A) B_Cout <= '1' {borrow}
  else B_Cout <= '0' {no borrow}
else { -- if SUB_ADD='0'
  D_S <= (A + B )[7.0]+B_Cin; {lower 8 bits of A + B + B_Cin}
  B_Cout <= (A+B+B_Cin)[8]; {most significant bit of A+B+B_Cin}
}
```

Figure 4: Overal functionality of an eight-bit adder subtractor.

2.1.5 Analysis and Purpose

Implementing an eight-bit adder subtractor hierarchically is imperative in grasping the fundamentals of digital hardware design and will prove to be useful knowledge in the coming labs. Analyzing an eight-bit adder subtractor, we see that it calls down all the way to a 2 to 1 multiplexer. Looking further to the structural model, this is a direct replacement for the schematic design entries as previously described. It is a simple, yet very effective approach to an eight-bit adder subtractor using Verilog code. The structural model introduces Verilog code as a direct replacement of the designs we just created which increases our understanding of both Verilog and Schematic design. Lastly, we can see the much more concise and complex behavioral model. This is also a replacement for the previous two designs but would be much harder to service and understand. Its concise nature is impressive, however, and highlights the power of Verilog.

2.2 Design and implementation procedure

2.2a Using Design Schematics

The first approach to the adder subtractor design was taking advantage of Quartus' powerful schematic design tool. The eight-bit adder subtractor, provided in the Lab 2 assignment PDF, in this design takes advantage of hierarchical methods to perform its operations. Figure 5 displays the highest level of the design, which calls the four-bit adder subtractor and so on. The following figures (6-8) display the remaining pieces of the puzzle.

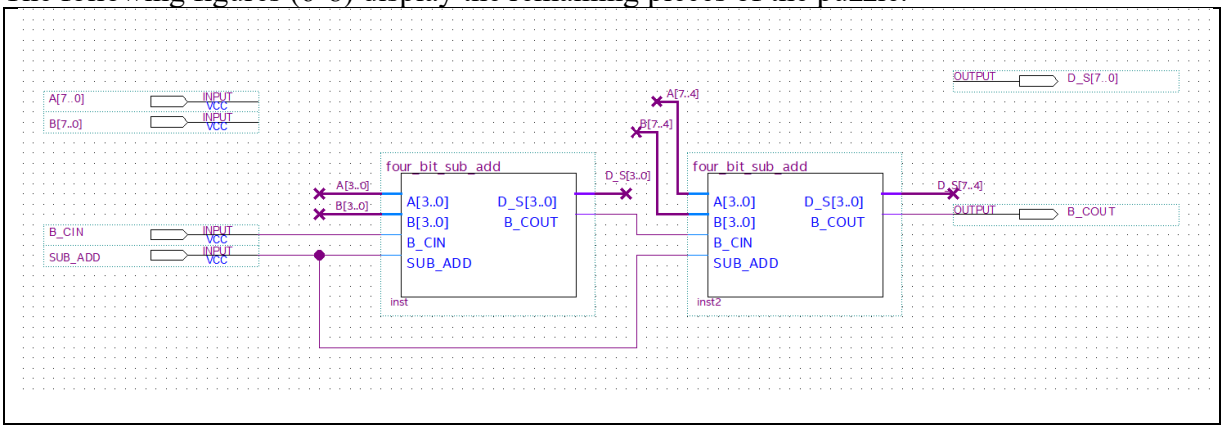


Figure 5: Eight-bit adder subtractor

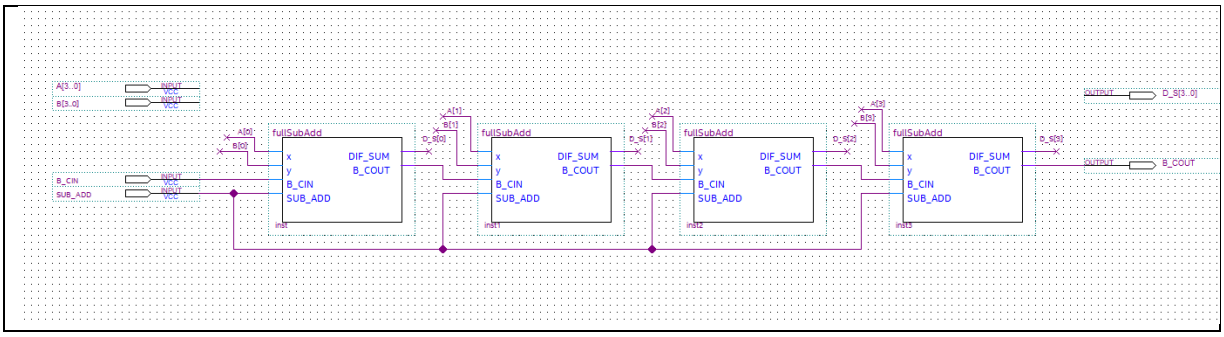


Figure 6: Four-bit adder subtractor

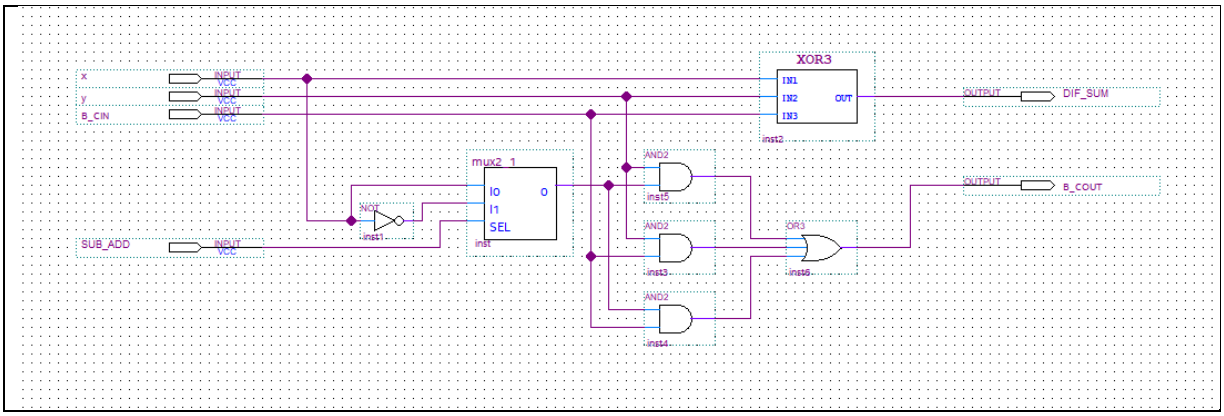


Figure 7: Full adder subtractor

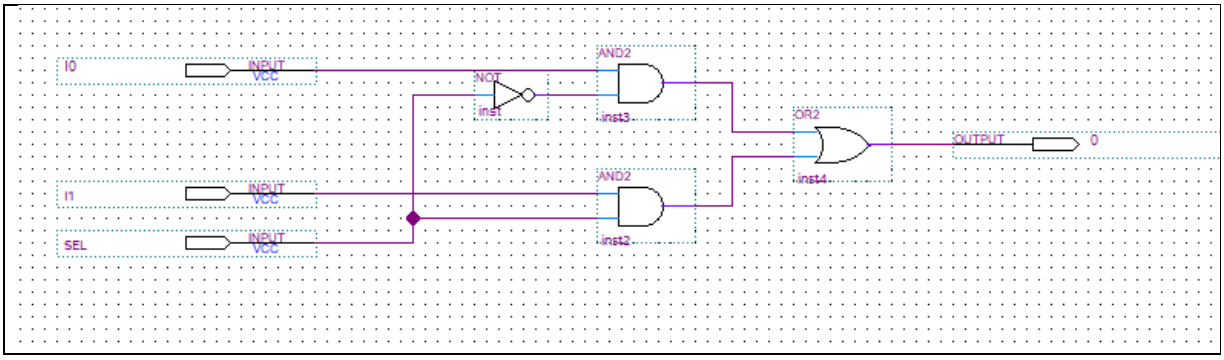


Figure 8: 2 to 1 Multiplexer

Figures 5-8 accurately show the complete eight-bit adder subtractor and highlights the importance of separating the design and using Quartus' symbol feature to limit complexity and simplify the overall design.

2.2b Using Structural Verilog

The structural model, see figure 9, is a direct replacement for the hierarchical schematic design as described in section 2.2a. The only difference is the design is implemented in Verilog. It has the same function calls, and is just as easy to read and understand – if you are well versed in Verilog.

```
// Top level module
// eight-bit subtractor/adder
module eight_bit_sub_add(D_S,B_COUT,A,B,B_CIN,SUB_ADD);
    output [7:0] D_S;
    output B_COUT;
    input [7:0] A,B;
    input B_CIN,SUB_ADD;
    wire n0;
    four_bit_sub_add C0(D_S[3:0],n0,A[3:0],B[3:0],B_CIN,SUB_ADD);
    four_bit_sub_add C1(D_S[7:4],B_COUT,A[7:4],B[7:4],n0,SUB_ADD);
endmodule

// four-bit subtractor/adder
module four_bit_sub_add(d_s,b_cout,a,b,b_cin,sub_add);
    output [3:0] d_s;
    output b_cout;
    input [3:0] a,b;
    input b_cin,sub_add;
    wire n0,n1,n2;
    fullsubadd C0(d_s[0],n0,a[0],b[0],b_cin,sub_add);
    fullsubadd C1(d_s[1],n1,a[1],b[1],n0,sub_add);
    fullsubadd C2(d_s[2],n2,a[2],b[2],n1,sub_add);
    fullsubadd C3(d_s[3],b_cout,a[3],b[3],n2,sub_add);
endmodule
```

```
// full subtractor/adder
module fullsubadd(dif_sum,b_cout,x,y,b_cin,sub_add);
    output dif_sum,b_cout;
    input x,y,b_cin,sub_add;
    wire n0,n1,n2,n3,n4;
    not C0(n0,x);
    mux2_1 C1(n1,x,n0,sub_add);
    and C2(n2,y,n1);
    and C3(n3,y,b_cin);
    and C4(n4,n1,b_cin);
    or C5(b_cout,n2,n3,n4);
    xor C6(dif_sum,x,y,b_cin);
endmodule

// 2 to 1 multiplexer
module mux2_1(o,i0,i1,sel);
    output o;
    input i0,i1,sel;
    wire n0,n1,n2;
    not G0(n0,sel);
    and G1(n1,i0,n0);
    and G2(n2,i1,sel);
    or  G3(o,n1,n2);
endmodule
```

Figure 9: Eight-bit adder subtractor structural implementation

2.2c Using Behavioral Verilog

The last, and most complex, implementation is the behavioral Verilog model. The behavioral model favors a concise nature in place of simplicity. Continually, it is much harder to understand the implementation and has weird tendencies when borrowing and carrying. The code is displayed in figure 10.

```
module eight_bit_sub_add(D_S,B_COUT,A,B,B_CIN,SUB_ADD);
    output reg [7:0] D_S;
    output reg B_COUT;
    input [7:0] A,B;
    input B_CIN,SUB_ADD;

    reg [8:0] Cbuf;

    always @(A or B or B_CIN or SUB_ADD) begin
        if (SUB_ADD) begin
            Cbuf = A - B - B_CIN;
        end
        else begin
            Cbuf = A + B + B_CIN;
        end
        D_S = Cbuf[7:0];
        B_COUT = Cbuf[8];
    end
endmodule
```

Figure 10: Eight-bit adder subtractor Behavioral implementation

3. Demonstration

3.1 Video Link:

Folder:
https://drive.google.com/drive/folders/16QoZFWimvjmE4P-8_F79kciyIY5xSQq?usp=sharing
Video:
https://drive.google.com/file/d/1lsHX1GjoZ2fhpAKrEA70GJ8x_067d3vq/view?usp=sharing

4. Experimental Results

4.1 Observations:

The results of the experiment match the expected output of an eight-bit adder subtractor. The outputs are clearly defined in section 4.2, using the stimulus questions as input.

4.2 Questions

4.2a Schematic Design Entry

Total logic elements: 13 / 49,760
Displayed below are the following inputs and outputs I used for the different questions. The questions can be found below in section 6.1

Question	Cin / borrow	A (right)	B (left)	Result	Overflow
1	0	0000 0000	0000 1111	0000 1111	0
2	0	0011 0000	0001 0000	0010 0000	0
3	1	0001 0000	0000 1111	0010 0000	0
4	1	0011 0000	0001 0000	0001 1111	0
5	1	1111 1111	0000 0000	1111 1111	1
6	1	1111 0000	1111 1111	1111 1111	1
7	0	1111 1111	0000 1111	0000 1110	1
8	0	0001 0000	0110 0000	1011 0000	1

4.2b Structural Verilog Entry

Total logic elements: 14 / 49,760
Displayed below are the following inputs and outputs I used for the different questions. The questions can be found below in section 6.1

Question	Cin / borrow	A (right)	B (left)	Result	Overflow
1	0	0000 0000	0000 1111	0000 1111	0
2	0	0011 0000	0001 0000	0010 0000	0
3	1	0001 0000	0000 1111	0010 0000	0
4	1	0011 0000	0001 0000	0001 1111	0
5	1	1111 1111	0000 0000	1111 1111	1
6	1	1111 0000	1111 1111	1111 1111	1
7	0	1111 1111	0000 1111	0000 1110	1
8	0	0001 0000	0110 0000	1011 0000	1

4.2c Behavioral Verilog Entry

Total logic elements: 16 / 49,760
Displayed below are the following inputs and outputs I used for the different questions. The questions can be found below in section 6.1

Question	Cin / borrow	A (right)	B (left)	Result	Overflow
1	0	0000 0000	0000 1111	0000 1111	0
2	0	0011 0000	0001 0000	0010 0000	0
3	1	0001 0000	0000 1111	0010 0000	0
4	1	0011 0000	0001 0000	0001 1111	0
5	1	1111 1111	0000 0000	1111 1111	1
6	1	1111 0000	1111 1111	1111 1111	1
7	0	1111 1111	0000 1111	0000 1110	1
8	0	0001 0000	0110 0000	1011 0000	1

4.3 Pre-Laboratory results:

See section 2.2a-c, all models are implemented and discussed there.

4.4 Post lab questions:

4.4.1 For your three versions of this design, what kind of compilation errors or warnings did you encounter? How did you correct the errors and how did you determine which warnings were important and which ones could be ignored?

I received several errors while working on this project, but the main ones came from me not understanding Verilog properly. One error in particular, "Top-level design entity is undefined" made this lab take significantly longer than expected, especially for the second portion. This was usually fixed by either changing module names, setting the correct top-level entity, and having erroneous file names in the .qsf file.

For warnings, I did not pay attention to them as much. One notable warning that does not apply here is "No clocks defined in design." Clocks are not important for this lab and therefore the error is not important either. There were several errors designating incomplete I/O assignments. This is most likely not that important as the .qsf file was given to use and the .v header module takes care of the assignments. All the other assignments for the DE10 Lite board are present in the .qsf file but are not important for these implementations.

4.4.2 How does hierarchical schematic capture and hierarchical structural Verilog HDL design techniques help one manage the complexity of the design process?

Using a hierarchical approach throughout the design process helps not only simplifies the design process, it also helps to develop good habits. Instead of limiting the design to one module, developing submodules allows the user to break down the eight-bit subtractor / adder into smaller chunks. This helps increase understanding and limit complexity as you are prioritizing the micro to understand the macro.

4.4.3 Do you think the stimulus you used is good enough to verify the full functionality of your design? Why or why not? Would this stimulus be good enough to be used to automatically test the design for manufacturing defects? Why or why not?

The stimulus questions provided broadly verifies the three design entries, but not all functionality. To cover every possibility would require further research and testing, but for most cases, the stimulus should prove functionality if we work on assumptions (if carry out works, it should work for all tests of carry out). If we do not assume anything, the stimulus would be far too precise and simply a waste of time. Overall, the stimulus proves the design well enough for most cases, including manufacturing defects.

4.4.4 Were there any differences between the amount of FPGA resources that were used for each design. If so how did they differ? If not do you believe that the designs are implemented in the same manner by the Quartus Prime synthesis tool?

All 3 designs differed in their resource usage, with the behavioral model being the highest at 16 where as the structural model used 14 and the schematic design at 13. The total number of logic elements on the DE10 Lite board is 49,760, so their impact comparatively is minimal. This shows that Quartus' synthesis tool compilation output is different based on the manner of

the input. Concise and complex programs allocate more resources while simple and longer programs have less impact.

4.4.5 Describe the design trade-offs associated with the structural and behavioral Verilog versions.

While the behavioral implementation flexes its concise nature, it cannot match the simplicity of its structural counterpart. Further, the behavioral model is buggy – especially with modifying the carry in value. While understandable, most would prefer the structural model over the behavioral as the behavioral would be harder to modify and debug. Overall, the behavioral is concise, buggy – but functional in most situations whereas the structural model is simple and robust.

5. Conclusions

5.1 - Results and lessons learned:

The most notable knowledge gained from this lab was a better understanding of the eight-bit adder subtractor and the features of the Quartus application. Continually, it was a much-needed review and practice of binary operations. This lab also introduced the process, and power, of hierarchical schematic and code design. Hierarchical design is essential in understanding digital hardware and design. A structural and behavioral replacement for schematic design also helps to increase understanding of hardware design. Using Verilog code, we can represent schematic design without needing complex software. Part 2 of the laboratory assignment provides us with the Verilog code and running it in Quartus renders the same results as the schematic designs. However, the behavioral model was rather buggy, but it is concise and impressive to analyze. Overall, this lab shows its importance in its difficulty and broad range of topics it asks us to understand.

6. Appendix

6.1 Implementation questions

- 1) Add two numbers that do not generate a carry.
- 2) Subtract two numbers that do not generate a borrow.
- 3) Add two numbers in which there is a pending carry.
- 4) Subtract two numbers when there is a pending borrow.
- 5) Add two numbers where a carry in propagates all the way from LSB to carry out
- 6) Subtract two numbers where a borrow in propagates all the way from LSB and generates a borrow out request.
- 7) Add two numbers that causes an overflow into the carry bit.
- 8) Subtract a larger positive number from a smaller one generating a negative result if the MSB is interpreted as the sign bit.