THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
Department of Electrical and Computer Engineering

CPE 324 Advanced Logic Design Laboratory

Laboratory Assignment #2

Design Capture and FPGA-based Rapid Prototyping using the DE2-115 Platform

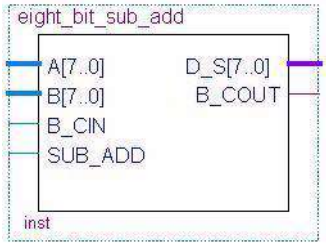
(Implementation of an 8-bit Subtractor/Adder)

(7% of Final Grade)

The purpose of this laboratory project is for you to gain experience using the Intel FPGA Quartus Prime® Computer Aided Design tool suite to implement a simple digital design. You will first gain experience creating a hierarchical design using traditional schematic capture techniques. You will then enter the same design using the Verilog Hardware Description Language using all three styles of Hardware Description Language encoding schemes discussed in this course. These styles include structural, data-flow, and behavioral. You will reconfigure the Cyclone IV E® Field Programmable Gate Array, FPGA with a schematic, structural Verilog, and Behavioral Verilog representation of the design. In each case, you will use the DE2-115's slide switches, and discrete LEDs on the DE2-115 to verify the correctness of the implementation. When you have completed this laboratory, you will be able to use the Quartus CAD tool to capture, compile, and download any type of design that you are likely to encounter in this course.

Subtractor/Adder Module

The example design that you are to enter, compile, and download in this course is an eight-bit two's complement subtractor/adder module that is shown in Figure 1. This design is to be created in three different ways.



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 1: Eight -Bit Two's Complement Subtractor/Adder Module

The design's main function is to output the valid two's complement difference or sum between two eight-bit quantities that are present on its *A* and *B* input buses. This module is designed to be cascaded with itself to create proportionately larger size subtraction/addition functional units. This is accomplished through the use of the borrow/carry input, *B_CIN*, and a borrow/carry output, *B_COUNT*. The *SUB_ADD* input is used to determine whether the other inputs will be subtracted or added to one another. Whenever the *SUB_ADD* line is set to a logic high, the *D_S* output produces a two's complement difference of *A-B* and the *B_COUNT* indicates if there is a borrow is required at the most significant bit position. Whenever the *SUB_ADD* line is set to a logic low, the *D_S* output produces the sum of *A+B* and the *B_COUNT* line indicates if a borrow or carry is generated at the most significant bit position. The module itself can be implemented in a number of ways (such as a ripple or look-ahead type logic) with each style of implementation presenting various complexity/performance trade-offs.

Prelab Assignment:

In this lab you are to enter designs into Intel FPGA Quartus Prime® using both schematic capture and Verilog HDL techniques. This is discussed in some detail in the manual "Design Entry, Simulation, and Emulation using the Quartus Prime® and ModelSim® CAD Software and the Terasic DE2-115 Rapid Prototyping Platform" that is present on the course Canvas® course delivery website. Simulation will not be used in this laboratory. Instead, you will verify designs experimentally using an actual hardware implementation made on the Teraasic DE2-115® rapid prototyping platform. This means that the designs that are created will be downloaded into the FPGA of the DE2-115 and will temporarily replace the Logic Trainer application that was utilized in the first laboratory assignment in this class.

Part I -- Hierarchical Schematic Capture Design Entry and Implementation using Quartus Prime

Background:

In this part of the laboratory you will be asked to enter a gate-level design of a two's complement ripple Subtractor/Adder module using hierarchical schematic capture design entry methods. Hierarchical design is a functional decomposition method that employes a hierarchy of levels of abstraction to represent the design. In this type of design, you should not implement the design in a flat manner where the entire design is placed on a single design sheet that utilizes the lowest-level gate/flip-flop components (i.e. primitives) but rather, you should create the design at multiple levels that are used to form a hierarchy of components. In this manner, complex components are created using a set of less complex components, which themselves may be made up of more basic components. This continues until the components that are the lowest level to be modeled are encountered. These components are called primitives and are usually simple gates, flip-flops, or latches.

Design Capture Assignment:

In this part of the assignment you are to implement a multi-level hierachical design of a ripple adder using schematic capture techniques. The top level of the design is to be made up of two four-bit adders that are cascaded together by connecting the *B_COUT* output of one four_bit module to the *B_Cin* input of the other module as shown below in Figure 2.

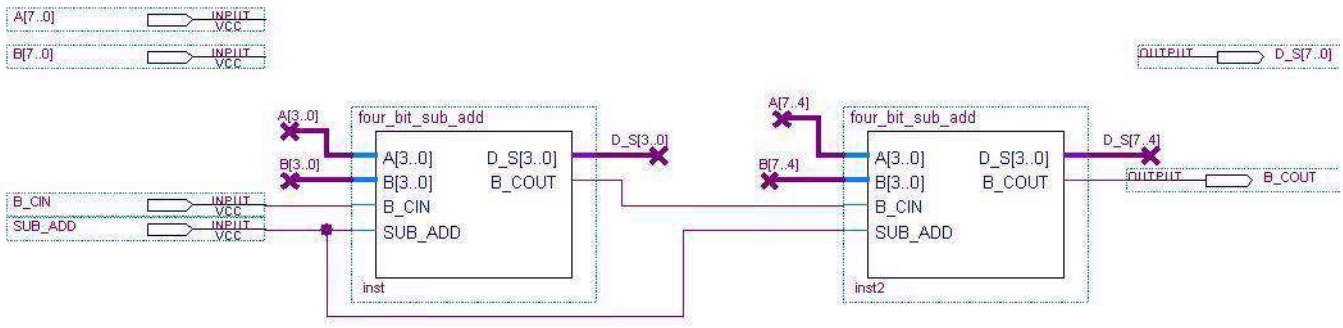


Figure 2: Top-Level Representation of Eight-bit Adder in Quartus Prime (*eight_bit_sub_add* module)

(Note: Part 1 is the most time consuming part of this assignment).

Each four-bit module should be composed of four one-bit full subtractor/adder modules as shown in Figure 3.

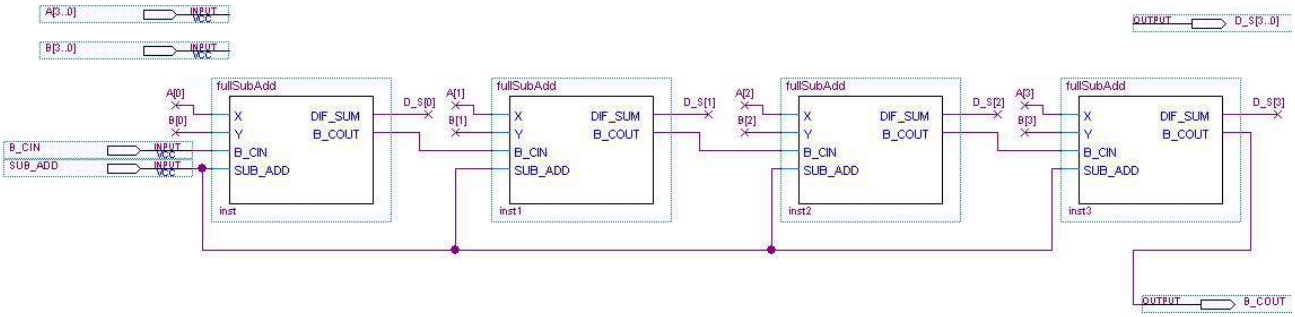


Figure 3: Representation of Four-bit Adder In Quartus Prime (*four_bit_sub_add* module)

In as similar style each full Subtractor/Adder module should then be represented as shown in Figure 4.

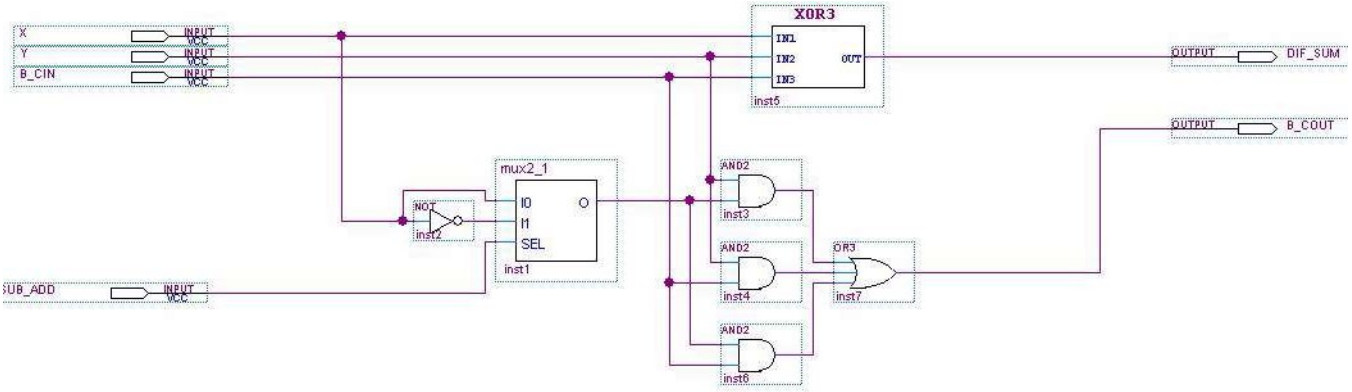


Figure 4: Representation of full Subtractor/Adder (*fullSubAdd* module)

The only component which is not a primitive in Figure 4 is the 2-to-1 multiplexer. It should be represented in the manner shown in Figure 5.

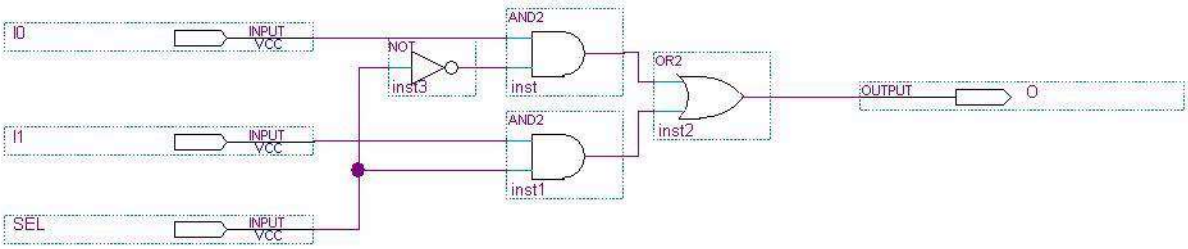


Figure 5: Representation of 2 to 1 Multiplexer (*mux2_1* module)

Components are to be created and added to the project library by progressing from bottom to top of the hierarchy so that the less complex modules can be used as building blocks to create the more complex ones. After the design has been entered, it should be successfully compiled. Note that the *XOR3* gate can also be represented in the same manner by using two *XOR2* gates if desired but there does exist an *XOR3* component in one of the Quartus Prime libraries so you can treat the *XOR3* as a primitive.

Implementation:

After successful compilation, ***record the number of Logic Elements and other FPGA resources used in this version of the design***. This should appear as part of the main compilation window. After successful compilation, the design should then be verified by skipping the simulation phase and implementing it directly on the DE2-115 rapid prototyping platform. In most cases simulation would be a precursor to this step since it provides the more detailed information but in some cases it makes sense to proceed directly to implementation. Skipping the simulation is made possible because of the rapid hardware prototyping aspects of Field Programmable Gate Arrays (much faster that wiring up TTL logic on a wireless protoboard!). Before doing this you will need to logically map the inputs and outputs of the design to the assign the correct **EP4CE115F29C7** Cyclone IV FPGA pin numbers on the Terasic DE2-115's Educational Trainer. The inputs should be connected to the slide switches and the outputs connected to the discrete green LEDs on the DE2-115. This mapping is shown in Table 1

Table 1: Cyclone IV Pin Location to be used on DE2-115 Board

Design I/O		DE2-115 Connections	
I/O Signal Name	Direction	Cyclone IV (EP4CE115F29C7) Pin Number	Element
A[7]	Input	PIN_AB26	SW[7]
A[6]	Input	PIN_AD26	SW[6]
A[5]	Input	PIN_AC26	SW[5]
A[4]	Input	PIN_AB27	SW[4]
A[3]	Input	PIN_AD27	SW[3]
A[2]	Input	PIN_AC27	SW[2]
A[1]	Input	PIN_AC28	SW[1]
A[0]	Input	PIN_AB28	SW[0]
B[7]	Input	PIN_AA22	SW[15]
B[6]	Input	PIN_AA23	SW[14]
B[5]	Input	PIN_AA24	SW[13]
B[4]	Input	PIN_AB23	SW[12]
B[3]	Input	PIN_AB24	SW[11]
B[2]	Input	PIN_AC24	SW[10]
B[1]	Input	PIN_AB25	SW[9]
B[0]	Input	PIN_AC25	SW[8]
B_CIN	Input	PIN_Y24	SW[16]
SUB_ADD	Input	PIN_Y23	SW[17]
B_COUT	Output	PIN_F17	LEDG[8]
D_S[7]	Output	PIN_G21	LEDG[7]
D_S[6]	Output	PIN_G22	LEDG[6]
D_S[5]	Output	PIN_G20	LEDG[5]
D_S[4]	Output	PIN_H21	LEDG[4]
D_S[3]	Output	PIN_E24	LEDG[3]
D_S[2]	Output	PIN_E25	LEDG[2]
D_S[1]	Output	PIN_E22	LEDG[1]
D_S[0]	Output	PIN_E21	LEDG[0]

As a minimum the correct functionality of the design should be demonstrated to the laboratory instructor which will

- 1) Add two unsigned 8-bit numbers that do not generate a carry on the B_COUT output.
- 2) Subtract two unsigned 8-bit numbers that do not generate a borrow on the B_COUT output.
- 3) Add two unsigned 8-bit numbers in which there is a pending carry on the B_CIN input.
- 4) Subtract two unsigned 8-bit numbers when there is a pending borrow on the B_CIN input.
- 5) Add two unsigned 8-bit numbers where an internal carry in propagates all the way from LSB to B_COUT output
- 6) Subtract two unsigned 8-bit numbers where a borrow in propagates all the way from LSB and generates a borrow request on the B_COUT output.
- 7) Add two numbers that causes an overflow into the carry bit.
- 8) Assuming that the numbers are represented in 2's complement and the MSB is interpreted as the sign bit, subtract a larger positive 8-bit signed number from a smaller 8-bit signed positive number thereby generating a negative result.

Part II -- Verilog HDL Entry and Implementation using Quartus Prime

In this part of the laboratory experiment, you are to implement two functionally equivalent designs of the two's complement adder/subtractor module within the Intel FPGA Quartus Prime environment using the Verilog HDL.

Structural Verilog HDL Design Implementation:

The Verilog HDL design shown in Figure 6 is a ***Structural*** (hierarchical) Verilog model of the ripple two's complement subtractor/adder design. It is an equivalent design to that which was implemented using schematic capture techniques in Part 1 of this laboratory. A soft copy of this listing can be found on Canvas. As in the first part of the laboratory the purpose of this part is not design creation but is for each student to develop proficiency with the Quartus Prime design tool and to make relevant observations that can be generally applied to designs that the student may create in the future.

```
// Structural Representation of Adder/Subtractor Module
// presented in Part I of Laboratory Experiment
// B. Earl Wells -- University of Alabama Huntsville, 2024
// This version places all component modules in a single file
// Note: it is also possible to include each of the component
// modules in separate files and set up the profile such that
// it references each of these files.

// It is good practice in Quartus to give the project the same
// name as the top-level module

// Top level module
// eight-bit subtractor/adder
module eight_bit_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
```

Figure 6: Representation of Eight-bit Adder/Subtractor in Structural Verilog HDL

```
// 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 6 (continued): Representation of Eight-bit Adder/Subtractor in Structural Verilog HDL

Implementation Assignment (structural design):

After successful compilation, you should ***record the number of Logic Elements and other FPGA resources used in this version of the design***. This information should appear as part of the main compilation window. Then you should verify the functionality of the design using the DE2-115 rapid prototyping platform. Before doing this you will need to assign the correct FPGA pin numbers that correspond to the DE2-115's connection to switches and discrete LED's to the top level Input and Output pins of the design. These pin numbers are listed in Table 1 from the first part of this laboratory.

As in the schematic design entry case of Part 1, you should demonstrate the correct functionality of the design to the laboratory instructor. The set of inputs should

- 1) Add two unsigned 8-bit numbers that do not generate a carry on the B_COUT output.
- 2) Subtract two unsigned 8-bit numbers that do not generate a borrow on the B_COUT output.
- 3) Add two unsigned 8-bit numbers in which there is a pending carry on the B_CIN input.
- 4) Subtract two unsigned 8-bit numbers when there is a pending borrow on the B_CIN input.
- 5) Add two unsigned 8-bit numbers where an internal carry in propagates all the way from LSB to B_COUT output
- 6) Subtract two unsigned 8-bit numbers where a borrow in propagates all the way from LSB and generates a borrow request on the B_COUT output.
- 7) Add two numbers that causes an overflow into the carry bit.
- 8) Assuming that the numbers are represented in 2's complement and the MSB is interpreted as the sign bit, subtract a larger positive 8-bit signed number from a smaller 8-bit signed positive number thereby generating a negative result.

This can be the same set of design stimulus as was used previously.

Behavioral Verilog HDL Design Implementation:

The Verilog HDL design described in Figure 7 below represents a ***Behavioral*** Verilog model of the two's complement adder/subtractor design that was implemented using schematic capture techniques and structural Verilog design methods. Enter this version of the subtractor/adder design and verify its functionality through simulation and implementation. A soft copy of this listing can be found on the Canvas page associated with the laboratory.

```
// Behavioral Representation of Adder/Subtractor Module
// presented in Part Prime of Laboratory 3 Experiment
// B. Earl Wells -- University of Alabama Huntsville, 2014
// Single -- Top-Level Module
module eight_bit_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 7: Representation of Eight-bit Adder/Subtractor in Verilog HDL

Implementation Assignment (behavioral design):

After successful compilation, ***record the number of Logic Elements and other FPGA resources used in this version of the design***. This should appear as part of the main compilation window. After successful compilation, you should then verify the design in the same way you did previously using the DE2-115 rapid prototyping platform. Before doing this students will need to assign the correct FPGA pin numbers that correspond to the DE2-115's connection to switches and discrete LED's input and output pins of the design. The manner in which this is to be done was shown in Table 1.

You should use the same set of stimulus input from previous designs to verify the correct functionality of the design to the laboratory instructor. This stimulus should as a minimum

- 1) Add two unsigned 8-bit numbers that do not generate a carry on the B_COUT output.
- 2) Subtract two unsigned 8-bit numbers that do not generate a borrow on the B_COUT output.
- 3) Add two unsigned 8-bit numbers in which there is a pending carry on the B_CIN input.
- 4) Subtract two unsigned 8-bit numbers when there is a pending borrow on the B_CIN input.
- 5) Add two unsigned 8-bit numbers where an internal carry in propagates all the way from LSB to B_COUT output
- 6) Subtract two unsigned 8-bit numbers where a borrow in propagates all the way from LSB and generates a borrow request on the B_COUT output.
- 7) Add two numbers that causes an overflow into the carry bit.
- 8) Assuming that the numbers are represented in 2's complement and the MSB is interpreted as the sign bit, subtract a larger positive 8-bit signed number from a smaller 8-bit signed positive number thereby generating a negative result.

Deliverables for Final Lab Report

In completing this assignment you will be expected to present to the laboratory instructor the following set of deliverables as part of their laboratory report. The requirements for laboratory reports are specified in the Lab Report Format handout which can be referenced on the Laboratory Canvas website.

1. Part I screen shots of Quartus Prime Schematics for the schematic capture design assignment.
2. Stimulus input used to verify the functionality of the design.
3. Date and time of the DE2-115 demonstrations for each of the four implementations (one schematic capture and the three Verilog implementations).

Post Laboratory Questions

Students are also expected to answer the following questions that reflect upon their laboratory experiences and the output data of their simulations.

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?
2. How does hierarchical schematic capture and hierarchical structural Verilog HDL design techniques help one manage the complexity of the design process?
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?
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?
5. Describe the design trade-offs associated with the structural and behavioral Verilog versions in terms of ease of design entry and the ability to control the low-level attributes of the design (such as what type of subtractor/adder is created).