ADDERS AND SUBTRACTORS

September 18th, 2007
CSC343
Fall 2007
Prepared by: Steven Medina

PURPOSE

The purpose of this lab is to show you how to implement an adder using Quartus. As the name implies, adders are used to add two sets of values together. Adders are a very common design in digital design. For example, a CPU will use an adder to have its program counter point to its next instruction. This is done by adding a constant value of 4 to the current instructions memory address. You will be using adders both here, and in future labs.

You will be shown three different kinds of adders. They are the half-adder, the full-adder. And the ripple carry adder. The purpose is to show you not only what each is, but why they are important. You will learn why each is important as you go through this lab.

After creating our adder designs in Quartus, you will test your design on Altera's DE2 programmable board. The DE2 board will be explained later in this lab.

HALF-ADDER

A half-adder (HA) is an adder that accepts two inputs and gives two outputs. The two inputs are the two single bit binary values that will be added to each other. The two outputs represent the sum. We need two outputs (rather than one output) because the sum may have a carry bit. For example, in binary, 1+0 = 1. This situation has no carry bit in the output. In other words, the output itself is 1 bit. However, if we add 1+1, we get 10. This output is 2 bits long. This is a case where the carry-bit for the output is needed.

A half adder consists of two logic gates. These are an AND gate, and an Exclusive OR gate. A diagram of a half adder is shown below.

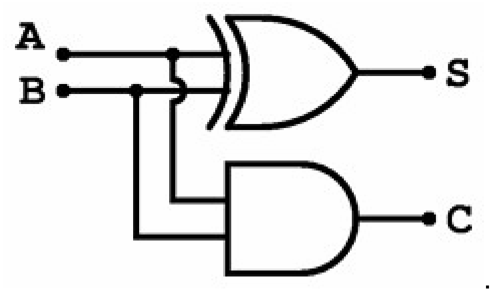


Figure 1: Half Adder using XOR and AND gate.

For the design of the half adder, do the following.

1) Create a new project in Quartus. Where it asks for the family or device you wish to target for compilation, select the Cyclone II board. Under the list of available devices, click EP2C35F672C6.

2) Design the half-adder shown in figure 1 in a block diagram file (bdf). Remember that the XOR and AND gates are available by clicking the "Symbol Tool" icon.

3) Compile your design.

4) Simulate your design using a vector waveform file. Set your inputs to have every possible combination of adding before simulating. For example, set the inputs in a way that you add all the combination 0+0, 0+1, 1+0, and 1+1. Do your outputs show that the inputs are being added together?

5) In your report, include a logic table for your half-adder.

FULL-ADDER

The main difference between a half-adder (HA) and a full-adder (FA) is that a full adder takes 3 inputs rather than 2. Why is this significant? A full adder can add the same two input bits as a full adder PLUS an extra bit for an incoming carry. This is important for cascading adders together to create N-bit adders. Before we cascade adders together, we will design a simple full-adder.

A full-adder is made up of two XOR gates and a 2-to-1 multiplexer. A diagram below shows how a full adder is connected.

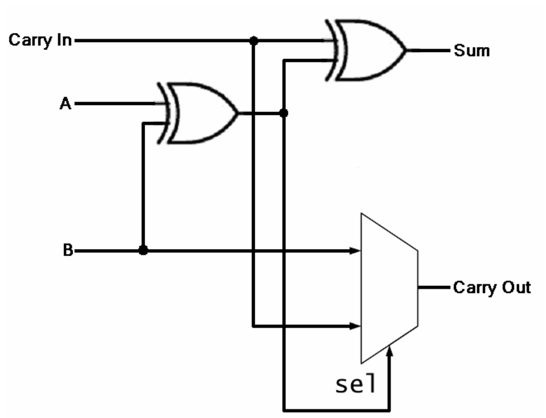


Figure 2: Full Adder using two XOR gates and a multiplexer.

For the design of the full-adder, do the following.

1) Create a new project in Quartus. Use the same board type as when creating a project for the half-adder.

2) Design the full-adder shown in figure 2. When creating the multiplexer, DO NOT USE AN LPM! Create the multiplexer by taking the 4-to-1 multiplexer code given in lab 1, and altering it to be a 2-to-1 multiplexer.

3) Open a blank VHDL file. Type in the following code and save it as dec_7seg.vhd.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

-- Hexadecimal to 7 Segment Decoder for LED Display
```

```
ENTITY dec_7seg IS
PORT(   hex_digit : IN STD_LOGIC;
      segment_a, segment_b, segment_c, segment_d, segment_e, segment_f,
      segment_g : OUT std_logic);
END dec_7seg;

ARCHITECTURE a OF dec_7seg IS
      SIGNAL segment_data : STD_LOGIC_VECTOR(6 DOWNTO 0);

      BEGIN
            PROCESS (Hex_digit)
                  -- HEX to 7 Segment Decoder for LED Display
            BEGIN -- Hex-digit is the four bit binary value to display
in hexadecimal
            CASE Hex_digit IS
                  WHEN '0' =>
                        segment_data <= "1111110";
                  WHEN '1' =>
                        segment_data <= "0110000";
                  WHEN OTHERS =>
                        segment_data <= "0111110";
            END CASE;
      END PROCESS;

            -- extract segment data bits and invert
            -- LED driver circuit is inverted

      segment_a <=      NOT segment_data(6);
      segment_b <=      NOT segment_data(5);
      segment_c <=      NOT segment_data(4);
      segment_d <=      NOT segment_data(3);
      segment_e <=      NOT segment_data(2);
      segment_f <=      NOT segment_data(1);
      segment_g <=      NOT segment_data(0);
END a;
```

4) Create a symbol block out of this VHDL file. Remember that the symbol block will be automatically named the same name as the entity of your VHDL file, which is dec_7seg. A picture of what the block should look like is shown below.

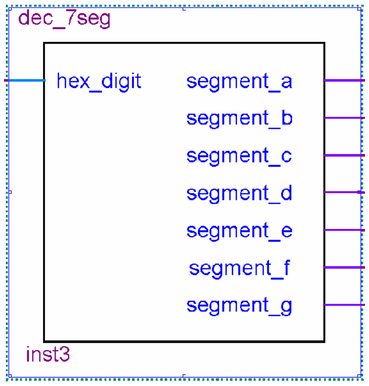


Figure 3: Image of dec_7seg block.

5) Add one of the dec_7seg blocks to both the carry output and the sum output.

6) Add a dec_7seg block to each wire coming out of the input pins. You have three input pins going into three inputs. Simply split the wire coming out of each input pin between your original design and the dec_7seg blocks.

6) Create output pins for each of the seven outputs of both dec_7seg blocks. Naming them something such as “a1, b1, c1…g1” and “a2, b2, c2…g2” from top to bottom would be a good naming convention.

7) Simulate your design using a vector waveform file. Just as in the half-adder, set your inputs to have every possible combination of adding before simulating. Of course, since this is a full-adder, you will have more possible combinations of inputs. Do your outputs show that the inputs are being added together?

8) In your simulation output, you will have outputs for each of the pins from the dec_7seg outputs. These represent a seven-segment display like the one shown below.

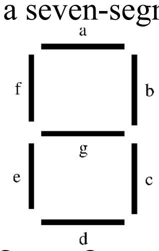


Figure 4: Seven-Segment Display

Each of the 7 lines is labeled according to the labels you would have on your dec_7seg block if you used the naming convention I suggested.

9) The logic values output from the dec_7seg are INVERTED logic values from what would really be displayed on the seven-segment display on the DE2 board. Therefore, if output “a1” in your vector waveform file gives a logic value 1, the seven segment display would actually turn “a” off from figure 3. Using the output of your vector waveform file, make sure that for any given sum of two numbers, the outputs from dec_7seg will display the correct value on a seven-segment display. Again, the logic values coming from the dec_7seg blocks are INVERTED from what will actually appear on the DE2 board.

4-BIT RIPPLE CARRY ADDER

Now that you have created a full-adder, you can begin adding numbers that are larger than 1-bit in length. In this case, you will design a circuit that will add two 4-bit numbers together. This will be done by cascading four full-adders together. For example, if we wanted to add the numbers 1011 and 1110 together, we should get the proper result of 11001 by using nothing but full adders. In other words we need to implement a design that can add numbers such as:

```
  1 0 1 1
+ 1 1 1 0
----------
1 1 0 0 1
```

in Quartus using full adders. Below is a diagram of what a 4-bit ripple carry adder looks like.

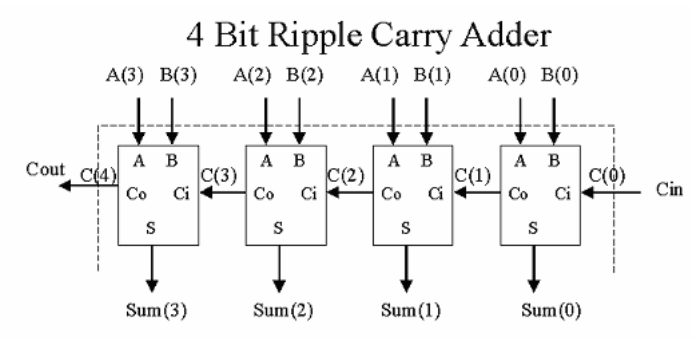


Figure 5: 4-bit ripple carry adder.

As you can see in figure 3, each block represents one full-adder. The full-adder to the far right takes a total of three inputs; two for adding and one for a carry-in. It will then have two outputs. These outputs are labeled "sum" and "carry". The "sum" output will be the value of the first bit of our result, while the carry output will be driven to the carry input in the full-adder to its left. This process repeats for all four full-adders. As you can see, since there is a final carry (labeled C(4)), the resulting sum can be up to 5-bits long.

For the design of the 4-bit ripple carry adder, do the following.

1) Create a new project in Quartus. Use the same board type as when creating a project for the half-adder and full-adder. This is the exact type of board you will test your design on.

2) Design the ripple-carry adder shown in figure 3. As a tip, you can use the "create symbol file for current file" option for block diagram files, not just VHDL files.

3) Open a new blank VHDL file, and copy and paste the VHDL code from dec_7seg into your new file. The dec_7seg code is written to display a 0 or a 1 only. Alter the code from dec_7seg to be able to output any hexadecimal value from 0 to F, rather than just a 0 or a 1 that it currently is able to output. Name the altered file "dec_7seg_hex".

4) Create a symbol file for your altered dec_7seg_hex VHDL file.

5) We want to display your inputs and outputs on the DE2 board in hexadecimal. Therefore, you will need to write a simple VHDL code that converts binary into hexadecimal. Imagine this as a black box. The black box would have a single 4-bit input and four 1-bit outputs. To help you understand what you are writing code for, there is an illustration below in figure 6.

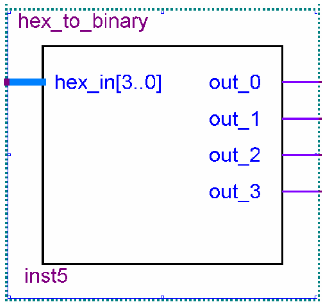


Figure 6: Black box of hexadecimal-to-binary converter.

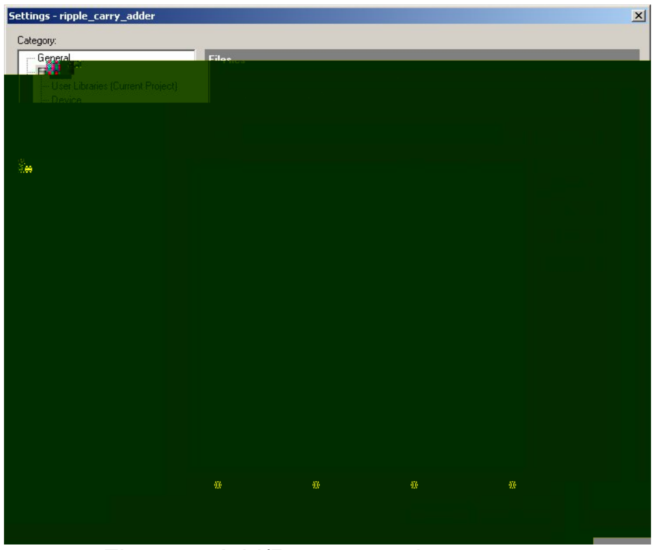
6) Create a symbol block for the VHDL code you just wrote. Connect an input pin into your hex-to-binary converter (remember, it is a 4-bit input). One of each output should connect to the first input for each full-adder in your design.

7) Add another hex-to-binary converter in your diagram. Connect an input pin into the block. Connect one of each output into the other input for each of the full-adders in your design.

8) Split both input pins that go into your hex-to-binary converter into a dec_7seg_hex block. You should have already created this block in step 4.

9) The input pin that goes into the "carry" input can remain the same, with no need for converting it into hexadecimal. However, split the wire coming out of this input pin into a regular dec_7seg block. You must import this block into your design if you have not done so already. To import a file:

   a) Click project=>Add/remove files in project. The screen that appears is shown below.


Settings - ripple_carry_adder
Category:
General
User Libraries (Current Project)
Figure 7: Add/Remove project screen.

   b) Click on the box with the three dots. A screen will appear asking which file you wish to add.

c) Add the “dec_7seg” file. If it is not in the same directory as your Ripple-Carry Adder project, you must copy and paste it there first, and redo this process.

10) When the values are output from each full-adder, they must be converted back to hexadecimal. Therefore, you must now write a small VHDL file that will create the exact *opposite* as figure 6. In other words, you want to convert binary to hexadecimal. You must then create a symbol block out of your VHDL code. A image of what a block like this would look like is shown below.

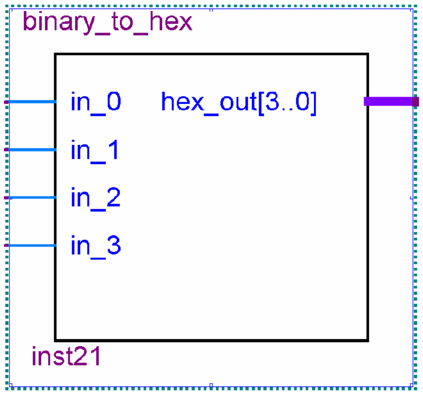


Figure 8: Binary-to-hexadecimal converter.

11) Connect one data outputs from each full-adder into one of the inputs of the binary-to-hex converter.

12) Connect a wire from the carry output of the last full adder into a regular dec_7seg block. Create output pins for all seven outputs.

13) Output the binary-to-hex converter into a dec_7seg_hex block. Create output pins for all seven outputs.

14) Compile your design.

15) Simulate your design in a vector waveform file.

16) See if you can trace a combination of inputs to see if the two dec_7seg_hex blocks have the right logic values. Remember; invert the logic values to see what the ACTUAL seven-segment decoder on the DE2 board would display.

Shown below in figure 9 is a sample of what the final design might look like.

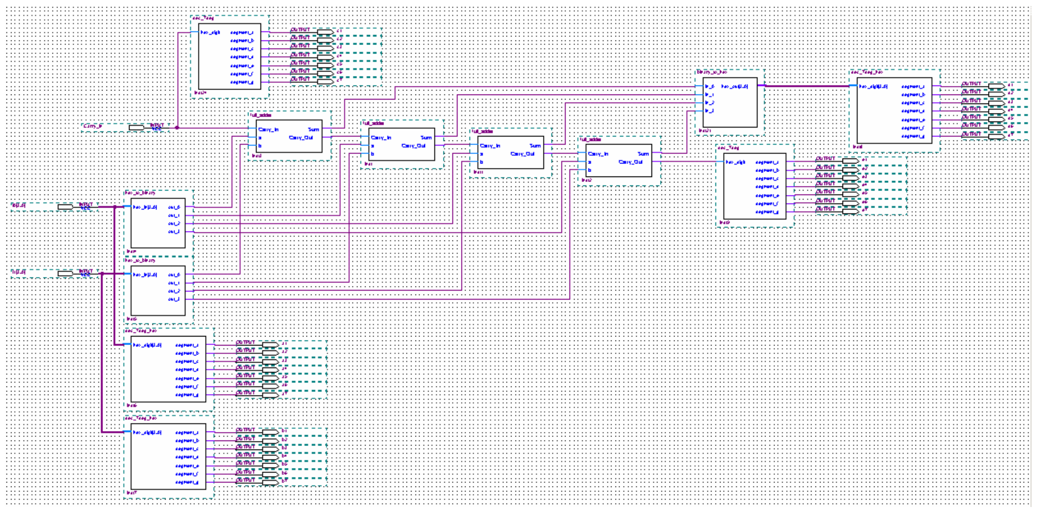


Figure 9: Complete 4-bit ripple-carry adder.

ADDER/SUBTRACTOR

You will now use your ripple-carry adder to make an adder/subtractor. The difference here is that there will be an extra input bit that controls if you are adding or subtracting. How do you subtract one binary number from the other? You will need to use you knowledge of two's compliment.

1) Create a new project called "adder_subtractor".

2) Create a symbol block out of your ripple-carry adder design. Before actually making the new symbol block, it would be a good idea to remove all the dec_7seg and dec_7seg_hex blocks and replace them with input and output pins.

3) Use this block in a new block diagram file. Your block will look similar to the image shown below.

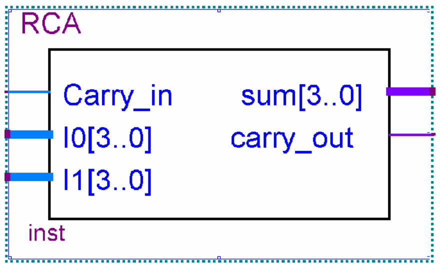


Figure 10: Symbol block of Ripple-Carry Adder

4) In order to have the block subtract, you will need to be able to take the two's compliment of the value going into EITHER I0 or I1. You do not need to do it to both. In order to do this, you will need to write VHDL code that will take the two's compliment coming from an input pin. As a hint, you can alter the dec_7seg_hex code to do this.

5) Create a symbol block out of your two's compliment VHDL code.

6) Connect all the inputs and outputs to either a dec_7seg or a dec_7seg_hex block; whichever is appropriate for that input or output.

7) Compile your design.

8) Simulate your design in a vector waveform file to see if your design adds and/or subtracts, depending on the value of the input pin going into the two's compliment block.

   A final design of your Adder/Subtractor will look something like the image below.

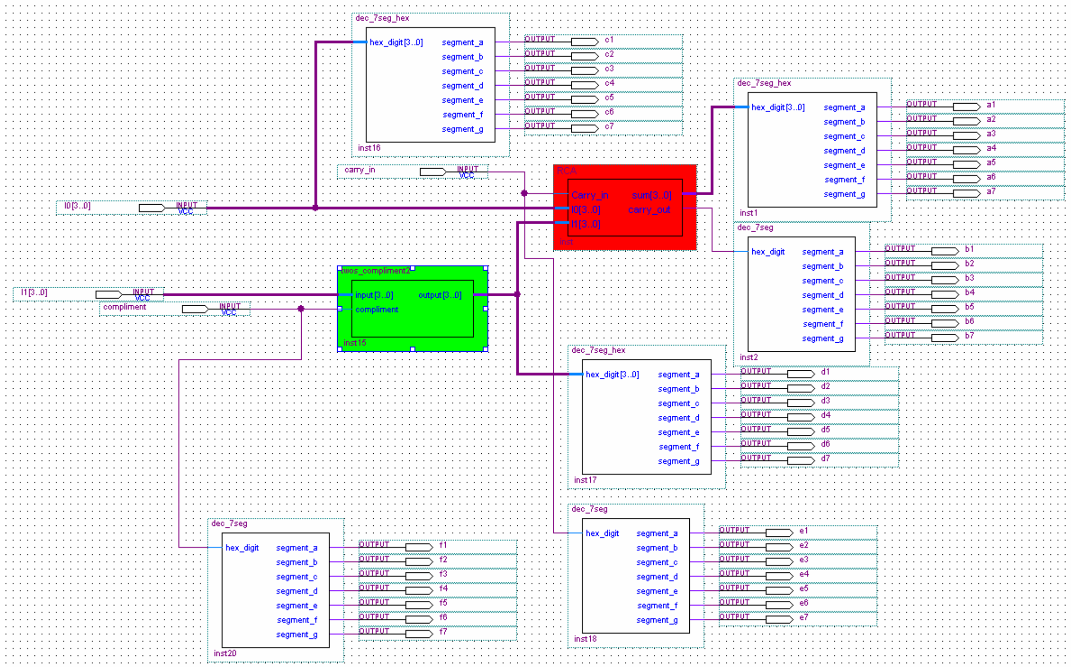


Figure 11: Image of an Adder/Subtractor using a Ripple-Carry Adder.

The block that is highlighted in red is the symbol block for the ripple-carry adder. The block highlighted in green is the symbol block for the two's complimentor. All the rest of the blocks are either dec_7seg blocks or input/out pins.