

EER-018 Laboratory #4

Design and Simulation of an Adder/Subtractor Circuit

Objective

In this lab you will use MSI devices in the MAX+plus II library as well as primitive gates to implement an adder/subtractor circuit.

Equipment

- Laboratory PC
- MAX+plus II
- 3.5 inch floppy disk, DOS format (bring to lab)

Prelab

Recall that in 2's complement representation, all negative numbers start with a 1, and positive numbers start with a 0.

1. Fill in the missing entries in the table below with the signed decimal equivalent of each 4-bit 2's complement number.

2's complement	decimal	2's complement	decimal
0000	0	1000	-8
0001		1001	
0010		1010	
0011		1011	
0100		1100	
0101	5	1101	-3
0110		1110	
0111		1111	

Table 1. 4-bit 2's complement numbers

An adder/subtractor circuit will add or subtract 2's complement binary integers and produce a correct 2's complement result. The value of a single control signal, we will call add_sub, determines whether addition or subtraction is performed. Some examples for 4-bit numbers are shown below.

Addition

3 0011	-7	1001	-7	1001		7	0111
+2 0010	+6	0110	+8	1000		+-6	1010
- - - - -	- - - - -	- - - - -	- - - - -	- - - - -		- - - - -	- - - - -
5 0101	-1	1111	-15	1 0001 (result overflows)		1 1	0001

Subtraction

3 0011	7	0111	-7	1001		7	0111
-6 0110	-6	0110	--8	1000		--6	1010
- - - - -	- - - - -	- - - - -	- - - - -	- - - - -		- - - - -	- - - - -
-3 1101	1	0001	1 1	0001 (result overflows)		13 1	0001 (result overflows)

Notice that whenever the result of an operation produces a number larger than 7 or smaller than -8, there is an overflow because these numbers cannot be represented in 4-bit 2's complement.

To design the adder/subtractor circuit we will use the basic idea that

$$\mathbf{A - B = A + (-B)}$$

Therefore, we will accomplish subtraction by simply adding the negative of the second number. The negative of B will be implemented using the 2's complement.

The 7483A Adder is a 4-bit adder that we will use to design the adder/subtractor circuit. Figure 1 is a top level block diagram of the adder/subtractor circuit and the function table that defines the add_sub control signal. When add_sub=1, the plus/minus block

generates the 1's complement of B and the carry in (CI) is one. This converts B to -B. When add_sub=0, the plus/minus circuit does not change B, and CI=0.

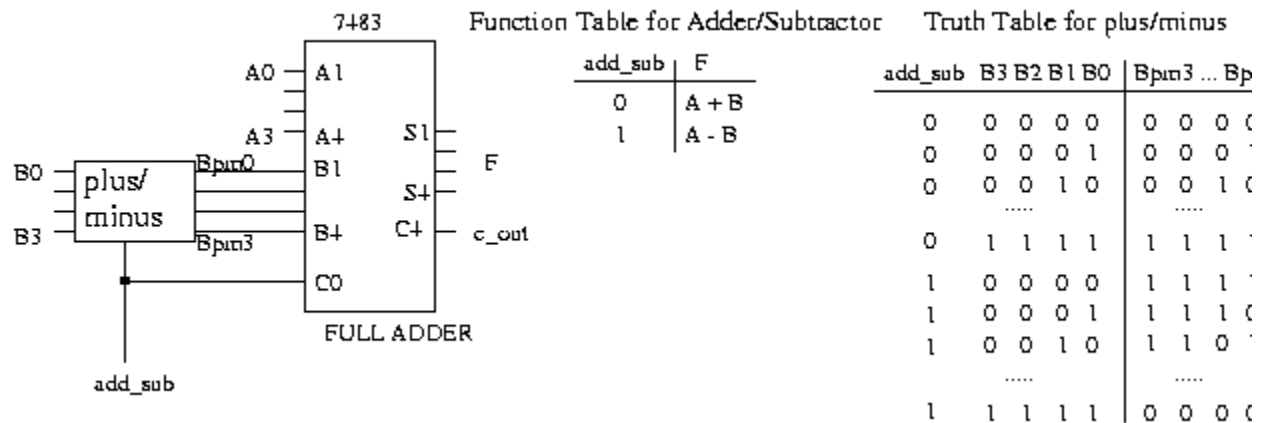


Figure 1. Adder/Subtractor Block Diagram

- Using the truth table for plus/minus, find the expression for Bpm(i) as a function of B(i) and the add_sub input.
- Using this equation, draw the logic for the "plus/minus" block. Make use of XOR gates to reduce the number of gates required.

Procedure

Enter Circuit

- Using the block diagram that you produced in for the prelab as a guide, enter the adder/subtractor circuit. Use the following hints:
 - Find the 7483 component in the ../mf library.
 - Use "View - zoom out" to get more space to move components apart if more room is required for wiring.
 - Sometimes it is convenient to rotate components. This can be done by selecting them and then using the "Edit - Rotate.." commands.
 - Save your file often, just in case. Name it add_sub.gdf.
 - Print a copy of the circuit.

Test the Circuit

Create waveform file:

- Using the procedure from Lab #2, create a waveform file to test your circuit. Group the signals so that you have A[3..0], B[3..0], add_sub, F[3..0] and Cout signals.
- Since this circuit has many more inputs than the circuit in Lab #2, we will have to run the simulation much longer to test it. Choose "File - End time" and set the end of simulation to 3.2us.
- First we will generate inputs using a simple method of assigning count sequences.
 - To define the A[3..0] and B[3..0] inputs, first select the A[3..0] signal and then click on the count sequence button on the left side of the window. (third button from bottom). Assign a binary count sequence, starting at 0, and incrementing every 100ns. Repeat this for the B[3..0] signal, but set the initial count to F, and the count increment to -1.
 - Zoom out in the waveform window and you will see that the test pattern starts to repeat at 1.6us. We will set the add_sub input to 1 to subtract for the second set of repeated inputs. Drag the cursor on the add_sub signal between 1.6us and 3.2us. Once this region is selected, click on the "overwrite with logic 1" button - the 8th button from the bottom on the left hand side of the window.
 - Save the waveform file.
 - Before you simulate, fill in the "expected" values for F in the table below for the selected values.

add_sub	A input	B input	F expected (hex)	F actual (hex)	F (decimal)
0	0	F			
0	1	E			
0	2	D			
0	3	C			

1	0	F			
1	1	E			
1	2	D			
1	3	C			

- o Now simulate and fill in the actual values for F along with the decimal version. If they are not correct, you will have to go back and revise your design.
4. What is the total number of tests that would be applied if all combinations of the 9 inputs were applied?_____ Give the formula for the number of test combinations (T) in terms of the number of inputs (I).
 5. Notice that although it was easy to create the test vectors in the previous test, not all cases (negative + negative, positive + positive, positive - negative, etc) were tested.
 6. To enter different simulation values, follow this procedure:
 - o Zoom in so that approximately 1.6us of time is visible in the waveform window.
 - o Drag the mouse across the section of the B[3..0] waveform where the value is 0. This section should darken.
 - o Click on the "overwrite group waveform" button (the second from the bottom on the left side of screen).
 - o Type a "3" for the value. The value will change on the waveform window. Continue this process and enter the values for A and B shown below. (notice these values are given in decimal in the table below and must be converted to hex 2-s complement)
 - o A similar procedure can be used to set the add_sub value as shown in the table below.
 7. Bring up the simulator and test the circuit for the following input combinations and record the results. Indicate whether there was an overflow or not.
 8. Print the waveform.

add_sub	A input	B input	F (hex)	F (decimal)	Overflow?
0	0	3			
0	5	5			
0	-6	3			
0	-4	-3			
0	-2	-7			
1	3	4			
1	7	5			
1	-3	7			
1	7	-3			
1	-1	-7			

Table 2. Simulation Results

Simulation of 8-bit Adder-subtractor

In this section we will examine an 8-bit version of the adder-subtractor circuit. This circuit has already been constructed for you.

Examine the 8-bit circuit

1. Choose "File - Open" and navigate to the c:\EE18\Lab3 folder.
2. Open the add_sub8.gdf file. Notice that it is composed of two 4-bit add_sub components.
3. Double-click on one of the add_sub components to examine the internal structure of the component.
4. How does this 4-bit circuit differ from the circuit that you designed?
5. Why is this modification necessary to create the 8-bit version?

Simulate the 8-bit circuit

1. To prepare for simulation, choose "File - Project - Set Project to Current File".
2. Now compile the circuit.
3. Follow the procedures in this lab for simulating the circuit. Here are some hints:
 - o If the add_sub waveform window is open, close this before creating one for the add_sub8 circuit.

- o When entering nodes from snf, choose only the bus signals (A[7..0], etc) and add_sub to be added.
4. Fill in the values that you get for the following input cases.
 5. Print the waveform results.
 6. If you save the waveform, be sure to save it on your floppy disk, and not on the hard drive. If you accidentally save it (the add_sub8.snf file) to the hard drive, be sure to remove it so that the next lab section can have the joy of creating it.

add_sub	A input (decimal)	A input (hex)	B input (decimal)	B input (hex)	F (hex)	F (decimal)	Overflow?
0	0	0	17	11			
0	95	5F	59	3B			
0	-68	BC	32	20			
0	-44	D4	-13	F3			
0	-99	9D	-79	B1			
1	33	21	44	2C			
1	79	4F	56	38			
1	-39	D9	79	4F			
1	27	1B	-39	D9			
1	-100	9C	-7	F9			

Report

- Include a few paragraphs that give an overview of how you entered and simulated the 4-bit adder/subtractor circuit.
- Include an explanation of how you derived the "plus/minus" logic.
- Derive an expression and a circuit that can detect the overflow condition. Hint: Notice that overflow can occur only for the following conditions:

```

positive + positive : overflow if result is negative
negative + negative : overflow if result is positive
positive - negative : overflow if result is negative
negative - positive : overflow if result is positive

```

The expression will be a function of A3, B3, F3 and add_sub.

- How would the expression change for the 8-bit version?