

Implementing Combinational Circuits

Due: By 6:00pm on Wednesday April 9.

While Multiplexers are primarily thought of as “data selectors” because they select one of several inputs to be logically connected to the output, they can also be used to implement Boolean functions. Similarly, while n -bit Decoders are primarily thought of as n -bit binary to 1 of 2^n code converters or as Demultiplexers, they can also be used to implement Boolean functions of n variables.

1 Multiplexers Used in Boolean Functions

Consider the following truth table that describes a function of 4 Boolean variables.

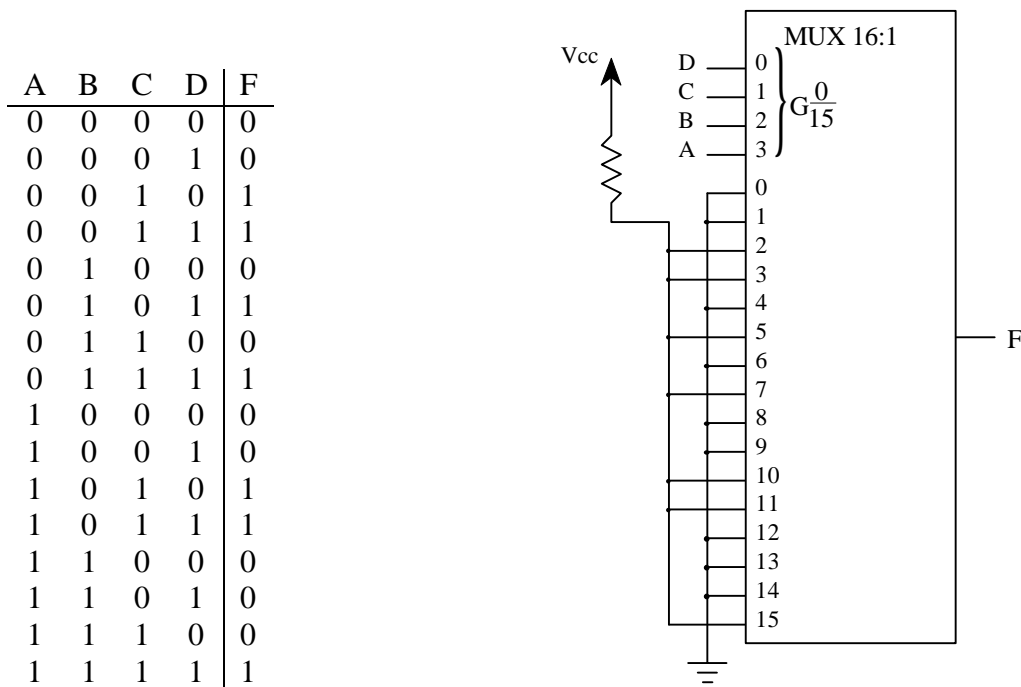


Figure 1: A 16 to 1 Multiplexer Implementation

A 16 to 1 Multiplexer with A , B , C , and D applied to its S_3 , S_2 , S_1 , and S_0 inputs respectively would select one of its 16 inputs for each of the 16 possible combinations of A , B , C , and D . We can implement the function described by the truth table by connecting a voltage source for logic level 1 or ground for a logic level 0 to each of the Multiplexer inputs corresponding to the required value of the function associated with the combination of A , B , C , and D that selected the input. Therefore, the inputs to the Multiplexer will be the same as the F entries in the truth table provided A , B , C , and D are connected to the Multiplexer select inputs in the right order.

Consider what happens when, instead of using a 16 to 1 Multiplexer, we use an 8 to 1 Mux. If we choose to connect A , B , and C to the inputs of the Multiplexer, then for each combination of A , B and C ,

although only one Mux input is selected, we need to realize two potentially different values of F . If they are different, they will be functions of D (either D or \overline{D}). To see this, we partition the truth table into sections where A , B and C have the same value. This is shown in Figure 2.

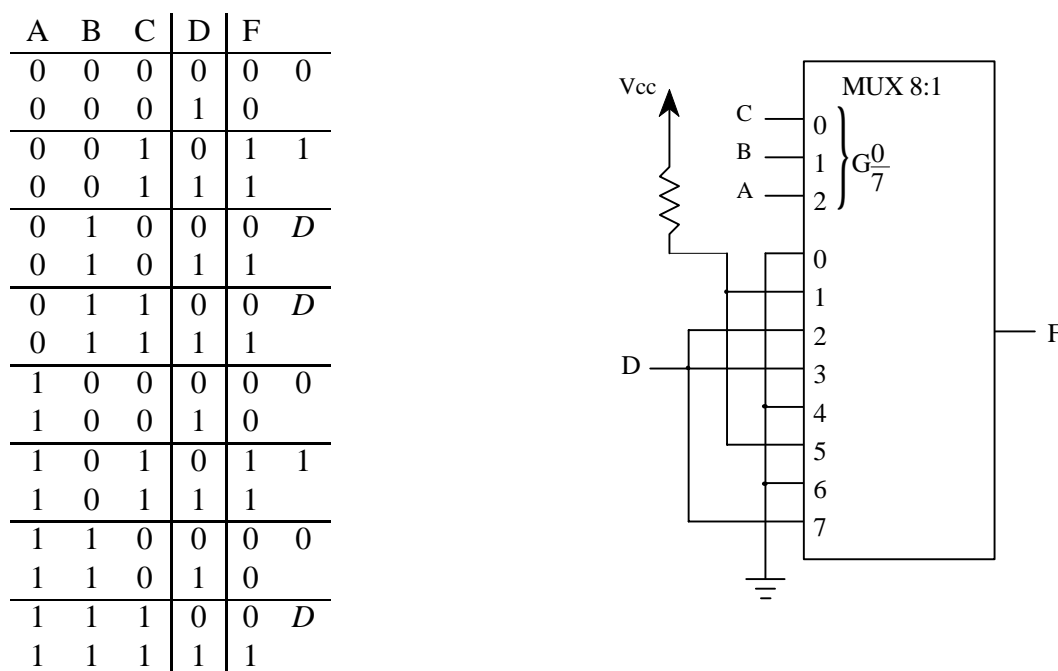


Figure 2: An 8 to 1 Multiplexer Implementation

This idea can be extended to a 4 to 1 Mux implementation. In this case we partition the table into sections for which A and B have the same value. This is shown in Figure 3. Now Multiplexer inputs will be either 1, 0 or some function of C and D . K-maps may be used to simplify the functions of C and D for each of the table partitions, but in the case of two variables, the functions are usually obvious.

The final possibility, shown in Figure 4, is a 2 to 1 Mux implementation. In this case only A is used as a select input and the table is partitioned into two parts based on A . Inputs to the Mux will be functions of B , C and D . Here it is more likely that you will need to use K-maps to simplify the functions.

It should be apparent that there is no explicit importance to the order of A , B , C and D in the truth table. If we rearranged them, we would have chosen different variables as Mux select inputs, and we might end up using less logic in the rest of the circuit. Consider rearranging the table so that B is the Mux select input. Using this approach, the realization shown in Figure 5 uses less hardware.

2 Multiplexers as Data Selectors

In larger designs you may want to perform the same operations on signals coming from a number of different sources, or share a resource, such as a Seven-Segment display or an ALU (Arithmetic Logical Unit). If this is the case Multiplexers and Decoders can be used to route data. When used in this fashion

A	B	C	D	F	
0	0	0	0	0	
0	0	0	1	0	<i>C</i>
0	0	1	0	1	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	<i>D</i>
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	0	<i>C</i>
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	0	<i>CD</i>
1	1	1	0	0	
1	1	1	1	1	

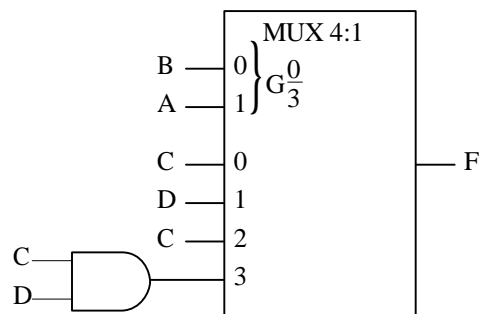


Figure 3: A 4 to 1 Multiplexer Implementation

the inputs to a Multiplexer are split between control signals and data signals (also known as “control path” and “data path”). In the design process the two are designed independently of each other with separate truth tables and K-maps. That is when designing the control path you are not concerned with the values on the data path. Similarly when designing the data path you assume the correct values are on the control path without being concerned with how they are generated.

Several Multiplexers can be combined to create Multiplexer configurations of different path widths and numbers of inputs. In Figure 6(a) two 74157s (a quadruple 2-to-1 Multiplexer with common select and active-low enable) are used to create an octal (8-bit) 2-to-1 Multiplexer by controlling the select line *S* on both modules with the same select signal. Figure 6(b) shows a quad 4-to-1 Multiplexer realized with two 74157 modules. Select signal *S1* enables one of the two modules and disables the output of the other. Signal *S0* is used as the select signal for both modules.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$$\overline{B}C + BD$$

$$\overline{B}C + CD$$

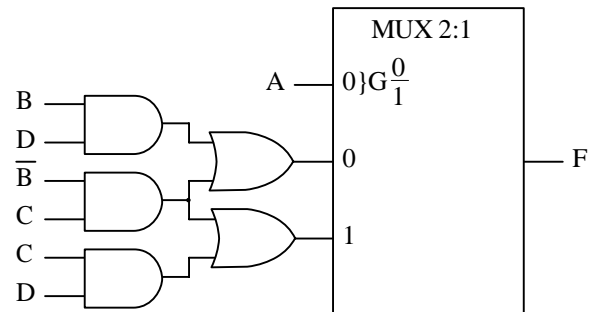


Figure 4: A 2 to 1 Multiplexer Implementation

B	C	D	A	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

$$D(\overline{A} + C)$$

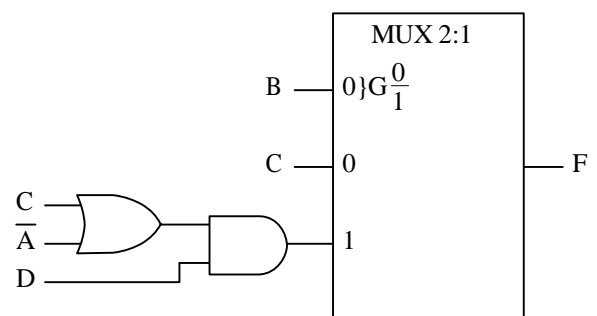
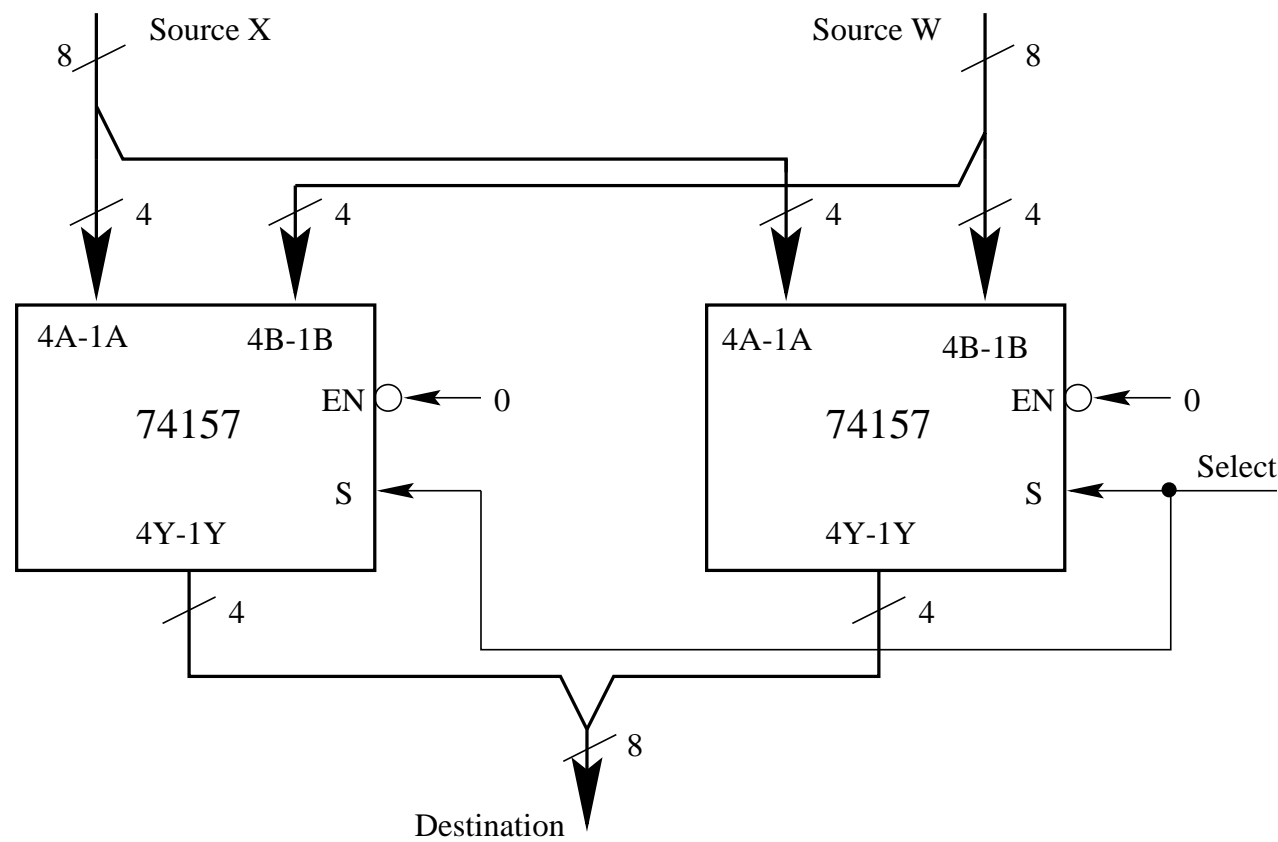
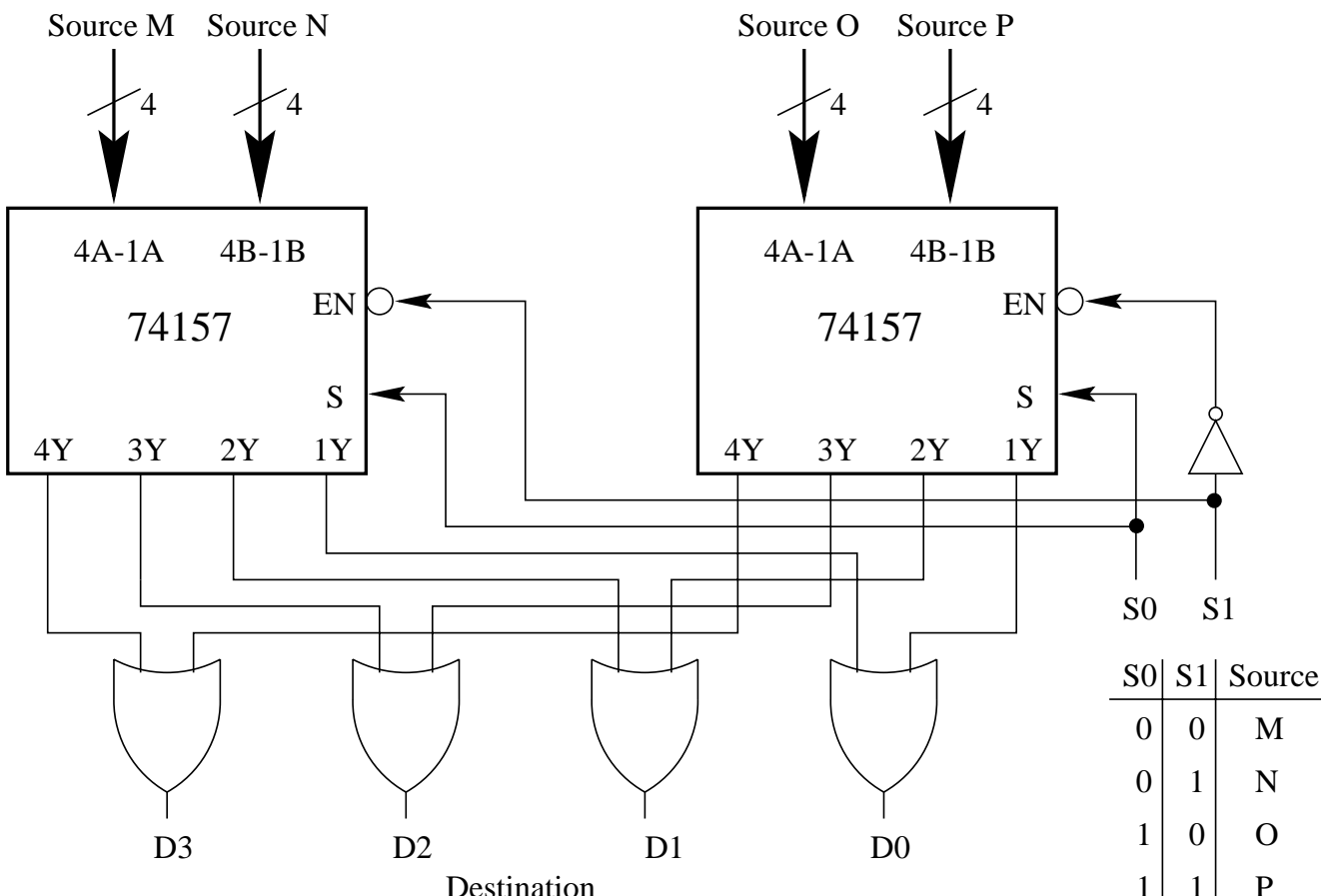


Figure 5: Another 2 to 1 Multiplexer Implementation

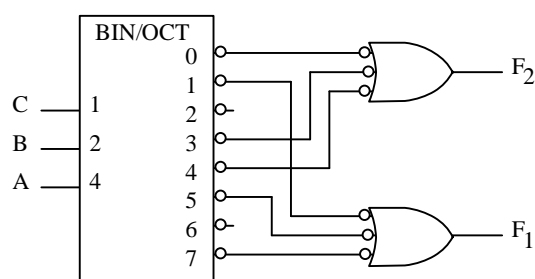


(a)



3 Decoders Used in Boolean Functions

For an n -bit to 2^n Decoder, the only output asserted is the one corresponding to the decimal value of the binary number presented to the n inputs. Another way to state this is to say that each Decoder output represents one of the 2^n minterms of n variables. Thus, a Decoder can be used to implement a function of n variables simply by connecting the outputs of the Decoder that correspond to the minterms of a function to a multi-input OR gate. Consider the Decoder circuit shown in Figure 7 which implements two separate functions of three variables. Notice that we have used a 74138 3-to-8 Decoder. As in most Decoders, the outputs are asserted low. This allows us to use NAND gates to implement the minterm sums.



$$F_1(ABC) = \sum_m(1, 5, 7), \quad F_2(ABC) = \sum_m(0, 3, 4)$$

Figure 7: A Decoder Circuit

We can see that, unlike Multiplexers, Decoders can be used to implement more than one Boolean function at a time. On the other hand, a Multiplexer implementation may not require any external circuitry, while a Decoder circuit always needs external OR gates.

4 Seven-Segment Displays

A Seven-Segment display consists of seven LEDs arranged in a figure 8 as shown in Figure 8. Lighting combinations of the LEDs allows one to display various symbols. To display the symbol 0, we light segments A, B, C, D, E and F. Lighting segments B and C produces the symbol 1. Because it is a common anode display, to select a Seven-Segment display a high voltage is applied to the corresponding anode (A1-A4 on the Digilab Board). Individual LED segments are lit by taking the inputs (the cathodes CA-CG, DP) associated with the segment low.

Important Note - The seven segment LED's are active low, meaning you will have to drive the segment low on the CA - CG, DP pins low in order to light up the LED segments. Example - if CA = 0, then segment A will light up.

Also, in order to select the particular 7 segment group, the common Anodes A1 - A4 must be driven by Vcc, meaning you will have to connect the A1 - A4 pins to the FPGA and pull them up internally through an IO Buffer in your schematic entry.

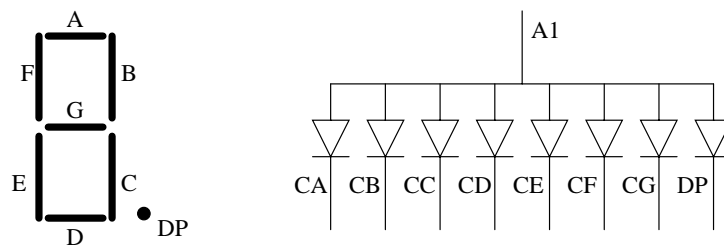


Figure 8: Seven Segment Display

5 Subtractor

In Lab 3 you learned how to create a Full Adder and then construct a four-bit Adder from 4 Full Adders. The same Adder can be used for two's complement addition and subtraction. Recall that subtracting a positive number in two's complement is the same as adding a negative number. So, to create a Subtractor one of the inputs of an Adder can be two's complemented as shown in Figure 9. Two's complementing

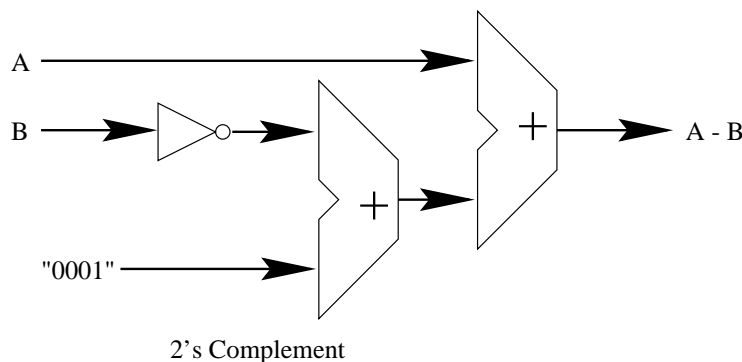


Figure 9: A Subtractor Circuit Using Two's Complement

a number consists of inverting the bits and adding one. Implementing this method in hardware requires implementing a second Adder, an expensive task in terms of size and speed. A better way is to take the one's complement (inverting the bits) of the subtrahend and take advantage of the carry-in bit of a Ripple-Carry Full Adder to add one as shown in Figure 10.

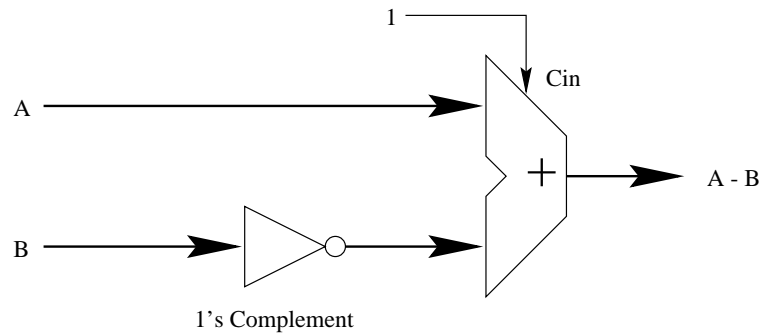


Figure 10: A Subtractor Circuit Using One's Complement

Pin #	FPGA Pin #	Function	Pin #	FPGA Pin #	Function
1		GND	21	87	BTN4
2		VU	22	86	BTN3
3		VCC	23	85	BTN2
4	122	LDG	24	84	BTN1
5	121	NC	25	83	DP
6	120	LD8	26	80	SW8
7	118	NC	27	79	CG
8	117	LD7	28	78	SW7
9	115	NC	29	77	CF
10	114	LD6	30	75	SW6
11	113	NC	31	74	CE
12	112	LD5	32	67	SW5
13	103	A4	33	66	CD
14	102	LD4	34	65	SW4
15	100	A3	35	63	CC
16	99	LD3	36	62	SW3
17	96	A2	37	60	CB
18	95	LD2	38	59	SW2
19	94	A1	39	58	CA
20	93	LD1	40	57	SW1

6 Preliminary Questions

1. Design a 4 bit Adder/Subtractor using one 4 bit Adder with carry in, carry out and overflow; Multiplexers; and any circuit needed to complement one of the inputs. You may implement either the two's complement or one's complement design presented in Section 5. Your adder/subtractor should be able to take two 2's complement numbers, positive or negative, and add or subtract them according to a control signal. Your output should consist of the result in two's complement form and an overflow signal. Some test cases are shown in Figure 11. You do not need to build a circuit to perform the 2's complement conversion, negative numbers will be input in 2's complement form. Draw the complete circuit on paper and bring the final design with you to the lab.

	A					B				C		R				O
3	0	0	1	1	1	0	0	0	1	0	4	0	1	0	0	0
3	0	0	1	1	1	0	0	0	1	1	2	0	0	1	0	0
2	0	0	1	0	3	0	0	1	1	0	5	0	1	0	1	0
2	0	0	1	0	3	0	0	1	1	1	-1	1	1	1	1	0
4	0	1	0	0	6	0	1	1	0	0	10	X	X	X	X	1
4	0	1	0	0	6	0	1	1	0	1	-2	1	1	1	0	0
5	0	1	0	1	-2	1	1	1	0	0	3	0	0	1	1	0
5	0	1	0	1	-2	1	1	1	0	1	7	0	1	1	1	0
-2	1	1	1	0	4	0	1	0	0	0	2	0	0	1	0	0
-2	1	1	1	0	4	0	1	0	0	1	-6	1	0	1	0	0
-3	1	1	0	1	7	0	1	1	1	1	-10	X	X	X	X	1
-3	1	1	0	1	7	0	1	1	1	0	4	0	1	0	0	0

Figure 11: Subtractor Test Data

2. Design a two's complement binary to 7-Segment converter shown in Figure 12 using a 4 to 16 Decoder and OR gates. Remember that 7-Segment displays are active low. Use an eighth output signal for the minus sign. Create a truth table and draw the complete circuit on paper and bring the final design with you to the lab.

A	B	C	D	X(7-Seg)
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

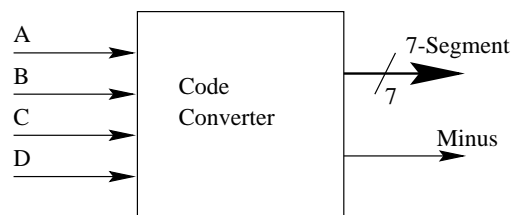


Figure 12: 2's Complement to Seven-Segment Converter

7 The Lab

1. Implement and simulate the Adder/Subtractor circuit you designed in question 1. You do not need to create the adder; use the "ADD4" part supplied in the Xilinx ISE Tools. Obtain a printout of the timing simulation waveform and compare it with the table in preliminary question 1 and hand it in with the lab report.
2. Implement and simulate the code converter you designed in question 2. You can use the D4_16E in the Xilinx ISE Tools. Obtain a printout of the timing simulation waveform and compare it with your truth table and hand it in with the lab report.
3. Combine the two circuits as shown in Figure 13, and show the Addition/Subtraction on the 7 segment display. For simplicity use the LED1 as the minus sign and use LED2 as the overflow indicator. Implement and pass off the entire design.

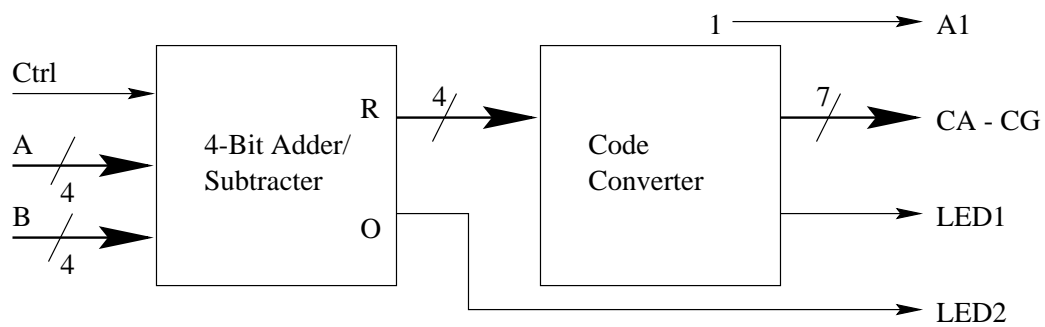


Figure 13: Adder/Subtractor and 7-Segment Decoder Block Diagram