ECE 2372 Modern Digital Design July 8

Reading assignment for today: Chapter 1, then thru page 43 of Ch 2, and then p 46 thru 49.

The Gray Code

The Gray code is a binary code in which only one bit position changes at a time when moving from one step in the code sequence to the next. Such a code is shown below compared to a regular binary code:

Binary code			Gray Code		
2^2	2^1	2^0			
4's	2's	1's			
A	B	C			
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

On page 31 of your textbook are shown two "encoding wheels" which could be used to encode the position of a rotating shaft. This type of encoding is used, often with optical sensors (an LED emitter and photo transistor receiver pair) to encode a rotary position. Note that with the binary code, there are cases where multiple bits change at the same boundary. However, this does not occur with the Gray Code. No matter how exactly we place optical sensors with such an encoding wheel, with a binary code we have the possibility to get "false" codes briefly during those transition edges. This does not occur

Negative numbers in computers

Sometimes we need to use negative numbers inside a computer. A question that arises is how should we represent negative numbers. There is no explicit place for a "sign" to indicate negative. We could select a special sign bit to represent a negative sign. One such approach is shown below. For example, we could make the MSbit represent the sign and arrive at the following table:

2^3	2^2	2^1	2^0	
8's	4's	2's	1's	
A	B	C	D	decimal equivalent
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
1	0	0	0	-0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Note that although this certainly works, we wind up with a 0 (zero) and a -0 (negative zero). Perhaps there is another solution which works better. It turns out there is a better solution and it is called 2's complement form. A table for 2's complement form is given below. At first it may seem confusing, but there is a lot of power available with this approach.

2^3	2^2	2^1	2^0	2's Complement Form
8's	4's	2's	1's	
A	B	C	D	decimal equivalent
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
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

Note that with the 2's complement form, we eliminated the negative 0, and that entry in the table becomes minus 8. That may not seem like enough advantage, so what else is there? For one thing, note that the MSbit does represent the sign. If the MSbit is 1, then it is a negative number, otherwise it is a positive number. That was true of the other approach as well. One major advantage of the 2's complement form is the ease of converting a number to its negative. It is as simple as inverting all the bits and adding 1. Some examples follow:

Convert the 4-bit nibble representing 3 to its negative in 2's complement form:

0011

1100 invert all the bits

+ 1 add 1

1101 compare to the table above

Now, to verify the above is reversable, take the negative 3 number and negate it

1101 negative 3 in 2's complement form

0010 invert all the bits

+ 1 add 1

0011 result is positive 3

Now lets try converting 1 to negative 1

0001 positive 1

1110 invert all the bits

+ 1 add 1

1111 result is negative 1

Now reverse the process

1111 negative 1

0000 invert all the bits

+ 1 add 1

0001 result is positive 1

One advantage of the 2's complement form is that regardless of how many digit positions are in the number, a binary number consisting of all 1's is simply negative 1. So, in 2's complement form, 1111 is negative 1. 11111111 is negative 1. 1111111111111111 is negative 1.

If we add negative 1 to negative 1, we should get negative 2. Let's try that:

1111

+1111

1 1110

Note we got a 4 bit result but with a carry. Looking at just the 4 bit result, and ignoring the carry bit we have 1110. That is negative 2 according to our table above. Let's convert to its negative to see if we get positive 2:

1110

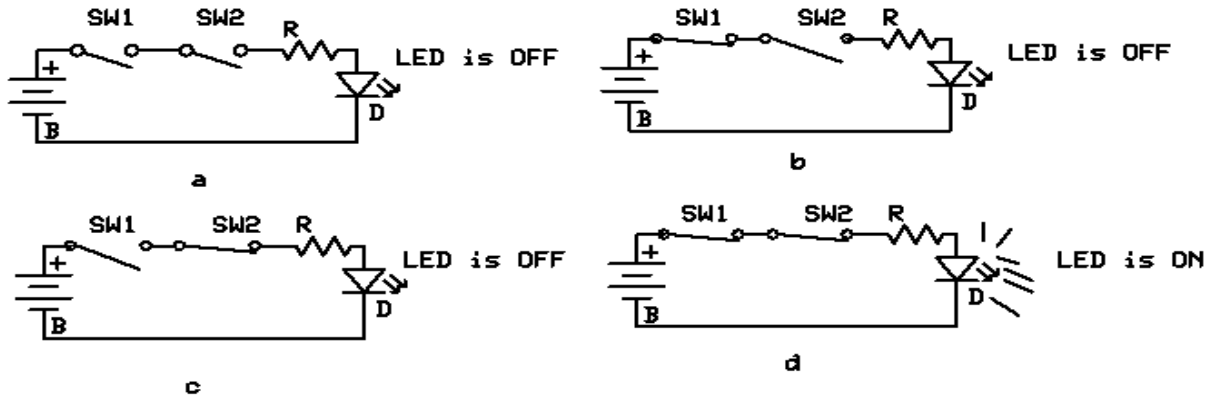
0001 invert all the bits

+ 1 add 1

0010 is positive 2

Using 2's complement makes it easy to convert an add operation to a subtract operation. We will see this later in the semester when we will design an adder and then convert it into a subtractor thru the use of 2's complement to form a negative number which is then added. Adding a negative number is the same as subtracting that number. It turns out it is easier to make an adder in digital logic circuits than a subtractor, so 2's complement form is used to simply subtract operations to an addition using a negative number.

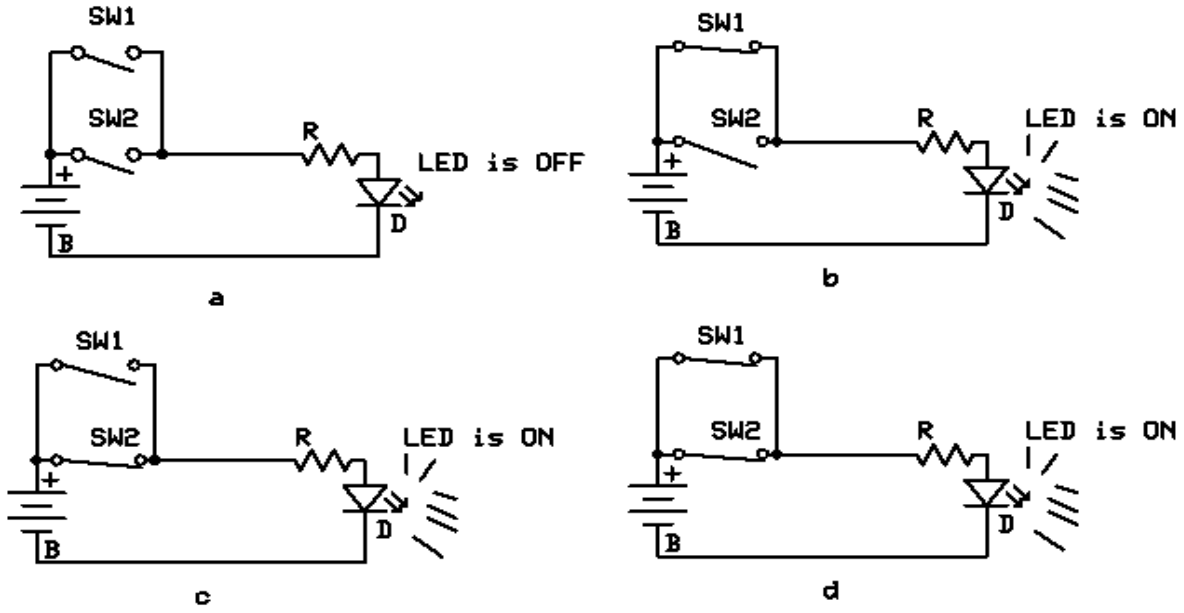
That covers the chapter 1 material. We now get into chapter 2. I think the easiest way to understand really basic logic functions of AND and OR is with a simple circuit diagram. Below is a circuit schematic demonstrating AND logic function of two switches. Both SW1 AND SW2 have to be ON for the LED to be ON.



The two switches are SW1 and SW2. The state of the LED (ON/OFF) is an indication of the combination of states of SW1 and SW2. In this diagram, we achieve the AND logic function. If we consider the ON state of the switch to be when it is in the closed position, then we arrive at the following truth table:

SW1	SW2	LED state (SW1 AND SW2)		
0	0	0	(OFF)	FALSE
0	1	0	(OFF)	FALSE
1	0	0	(OFF)	FALSE
1	1	1	(ON)	TRUE

We can in a similar way show the OR logic function. Here, if either SW1 OR SW2 is ON, the LED will be ON.



The two switches are SW1 and SW2. The state of the LED (ON/OFF) is an indication of the combination of states of SW1 and SW2. In this diagram, we achieve the OR logic function. If we consider the ON state of the switch to be when it is in the closed position, then we arrive at the following truth table:

SW1	SW2	LED state (SW1 OR SW2)		
0	0	0	(OFF)	FALSE
0	1	1	(ON)	TRUE
1	0	1	(ON)	TRUE
1	1	1	(ON)	TRUE

We have above seen two of the very basic logic functions, AND and OR. From these two basic logic functions, and the use of an inverting function, we can derive or synthesize the remaining logic functions we need.

One additional logic function with need is INVERT. It does as the name implies, simply inverts the input state to the opposite state. If input is 1, then the output of an inverter is 0.

Truth table

In	Out
0	1
1	0

We now need to discuss some notation. To write out the logic equations, the + symbol denotes the OR logic function, and the · symbol for the AND logic function.

The OR function

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 1$

The AND function:

$0 \cdot 0 = 0$

$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

When using some name such as A or SW1 to represent a logic state of some variable, if we want to indicate the opposite state, we can put a bar across the top of the symbol.

We will wrap it up here for today. I will be posting a small homework assignment on BB. Tomorrow we will begin an introduction to Boolean Algebra and the first steps in designing a logic circuit system.

Below is a table with many of the common gate types:

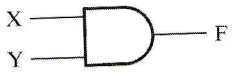
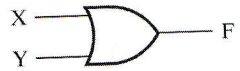
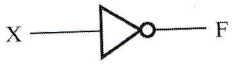
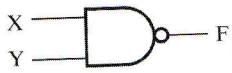
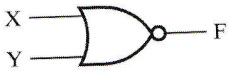
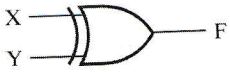
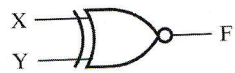
Name	Distinctive-Shape Graphics Symbol	Algebraic Equation	Truth Table
AND		$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)		$F = \overline{X}$	X | F 0 | 1 1 | 0
NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR (XNOR)		$F = XY + \overline{X}\overline{Y}$ $= X \oplus Y$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

FIGURE 2-3
Commonly Used Logic Gates