Benny Chen
1/25/2022
Lab 2

Theory

In this lab we delve further into both how logic gates work and how converting bits from one format to another works. Converting between number systems is extremely important, especially in this modern age of computing, as it is essential in communications between devices or circuits and is especially true for a human interface. Each bit, for a certain number there is always a weight associated with it which is the base of the number to the power of what value place it is. For example, most bases would have 1 as their first value as base^0 is 1. Taking binary as an example, the base is 2 so each value from that would be an exponent of that, so the first value would be 2^0 or 1, then 2^1, which is 2, 2^2 or 4 and so on. Each value can hold from a certain amount of bits also based on the base or radix of the number. For binary each bit can have 0 or 1, making it base 2 while in ternary its 0, 1, 2, making it base 3. For this lab, we converted 2 number systems, a binary number representation, DCBA, to a ternary number representation of XYZ, with each ternary bit stored as a 4-bit binary number. In this lab we only needed to convert only one bit of ternary through 11 iterations, 0-11. We first write out the table for all DCBA terms and its XYZ counterparts. We then isolate the Z bit and find the binary representation of its iterations to 11, with each binary bit corresponding as Z0, Z1, Z2, Z3. We then create equations from each Z bit in terms of D, C, B, A with AND and OR gates which are then used to make a circuit.

The radix of a number system is the "base" of the number or how many values can fit in a bit of the system. The radix economy is the number of values needed for that number times the radix or base of the number. For example, the radix economy for 1000 with a radix of 10 would be 40 or 1111 in ternary would be 12. The radix economy is used to measure the amount of storage or memory needed to efficiently store that number. By this logic, the lower the radix economy is, the more efficient that number system would be and the opposite would be true for a higher one. Any bit of information added in a number system would contribute to the increase of memory and storage used, from an increase in the number of bits a number can store to the number of values that are stored. The most economical and efficient base would be *e* or 2.718.

Deliverables

Table:

	D	**C**	**B**	**A**	**->**	**Z3**	**Z2**	**Z1**	**Z0**	
0	0	0	0	0	->	0	0	0	0	0
1	0	0	0	1	->	0	0	0	1	1
2	0	0	1	0	->	0	0	1	0	2
3	0	0	1	1	->	0	0	0	0	0
4	0	1	0	0	->	0	0	0	1	1
5	0	1	0	1	->	0	0	1	0	2
6	0	1	1	0	->	0	0	0	0	0
7	0	1	1	1	->	0	0	0	1	1
8	1	0	0	0	->	0	0	1	0	2
9	1	0	0	1	->	0	0	0	0	0
10	1	0	1	0	->	0	0	0	1	1

Equations:
Z3 Z2 Z1 Z0 0001:
DCBA 0001, 0100, 0111, 1010

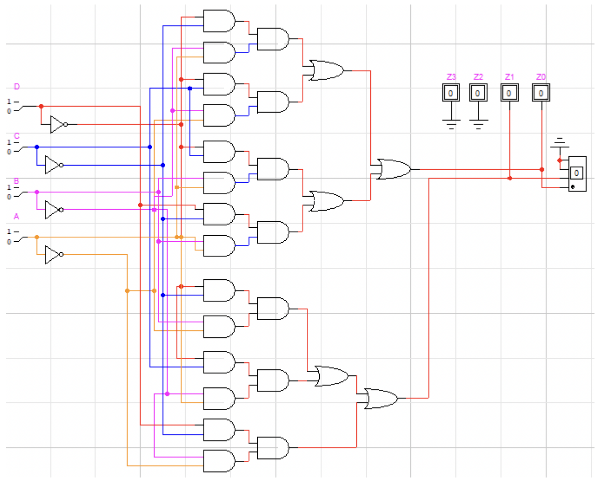
Z3 Z2 Z1 Z0 0010:
DCBA 0010, 0101, 1000

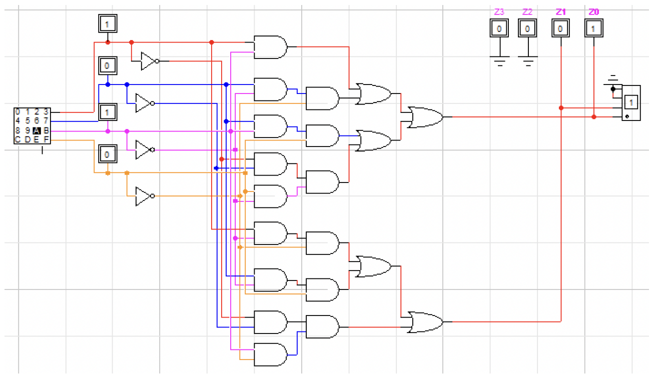
Logic Equation:
z0: D’C’B’A + D’CB’A’ + D’CBA + DC’BA’
z1: D’C’BA’ + D’CB’A + DC’B’A’
z2: 0
z3: 0

Simplified Logic Equation:

z0: DB + CB'A' + CBA + D'C'B'A
z1: DB'A' + CB'A + D'C'BA'
z2: 0
z3: 0

I simplified my equation by creating a karnaugh map of the values needed for each bit to be its corresponding ternary Z bit value. For the don’t cares, It would be the binary values of 11-15 as we dont use them ever. By using those don’t care values, we can simplify the equation a little.





Discussion

In this lab we learned more about how logic gates worked and about how bits of one numbering scheme are converted to another. I also become more familiar and comfortable with the workings of logicworks. With the conversion of a number scheme, I also learned the various methods of which it can be converted from the simple adding or dividing method. Along with this I learned about what a radix economy is and how it's important in the effective storage of these numbering systems. For the circuit itself, throughout the lab I tried to think of ways to improve the organization and efficiency of it. We were limited to only using 2 input gates, which became very cluttered but if we were to use 4 input gates, It would help in organizationing each input case. I could also simplify it using XOR gates however we cannot use them.

Questions

Question 1:

Convert the decimal number 97 (Base 10) to binary, octal, and hexadecimal.

Method 1: Division method

Method 2: Addition method

Binary: 1100001

1/2=0.5 | 0.5 * 2 = 1	**3/2=1.5 0.5 * 2 = 1**	**6/2=3**	**12/2=6**	**24/2=12**	**48/2=24**	**97/2=48.5 0.5 * 2 = 1**
64	**32**	**16**	**8**	**4**	**2**	**1**
1	**1**	**0**	**0**	**0**	**0**	**1**

Octal: 141

1/8=0.125 | 0.125 * 8 = 1	**12/8=1.5 | .5 * 8 = 4**	**97/8=12.125 | .125 * 8 = 1**
64	**8**	**1**
1	**4**	**1**

Hexadecimal: 61

6/16 = .375 | .375 * 16 = 6	**97/16 = 6.0625 | 0.0625 = 1**
16	**1**
6	**1**

Question 2:

Convert the signed binary number 101001112 to 2's complement and decimal.

Binary: 10100111

Flip/XOR all bits then add 0000 0001

2's Complement: 01011001

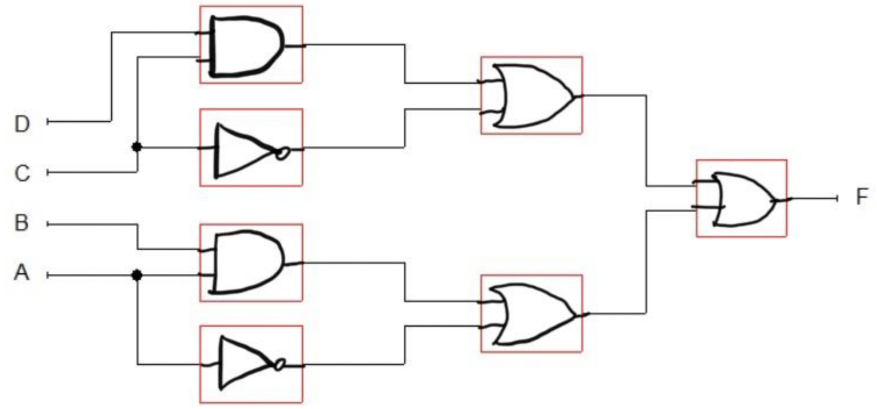
Method 1: Doubling method

Method 2: Addition method

Decimal:
128 + 32 + 4 + 2 + 1 = 167

0*2+1=1	**1*2+0=2**	**2*2+1=5**	**5*2+0= 10**	**10*2+0= 20**	**20*2+1= 41**	**41*2+1= 83**	**83*2+1= 167**
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	**0**	**1**	**0**	**0**	**1**	**1**	**1**

Question 3:



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