**Chip Design for High School**

Name: Dhuvarak Sharma

Class: 10th Grade

School: Sri Kumaran Public School

**Week 4:**

1. **Section 1: Logic Realization Using Switches:**
   1. Exclusive OR (Ex – OR) Operations:
      1. There are 2 inputs X and Y and an output Z:

|  |  |  |
| --- | --- | --- |
| Inputs | | Output |
| X | Y | Z |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* + 1. This operation is denoted as:  
       Z = X + Y
    2. When the inputs are same, the output is 0, but when the inputs are different, the output is 1. This is called an Ex-OR Operation
    3. In other words, Ex-OR can also be represented as:  
       Z = (X bar & Y) + (X & Y Bar)
    4. Proof:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Inputs | | Outputs | | | |
| X | Y | Z (Output of Ex-OR Operation | X Bar & Y | X & Y Bar | (X bar & Y) + (X & Y Bar) |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

* + - 1. From the above table, it is evident that the output of an Ex-OR Operation is equal to the values of (X bar & Y) + (X & Y Bar).
      2. Hence, proved.
  1. Exclusive NOR (Ex-NOR) Operations:
     1. There are 2 inputs X and Y and an output Z:

|  |  |  |
| --- | --- | --- |
| Inputs |  | Output |
| X | Y | Z |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* + 1. This operation is denoted as:  
       Z = X & Y
    2. This is also represented as:  
       Z = (X Bar & Y Bar) + (X & Y)
    3. Proof:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Inputs | | Outputs | | | |
| X | Y | Z (Output of Ex-NOR Operation | X Bar & Y bar | X & Y | (X bar & Y bar) + (X & Y) |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |

* + - 1. From the above table, it is evident that the output of an Ex-NOR Operation is equal to the values of (X bar & Y bar) + (X & Y).
      2. Hence, proved.
  1. Ex-OR and Ex-NOR can also be realized using NOR and NAND operations.
  2. Ex-OR is represented symbolically as:

Ex-OR

X

Z

Y

* 1. Ex-NOR is represented symbolically as:

Ex-NOR

X

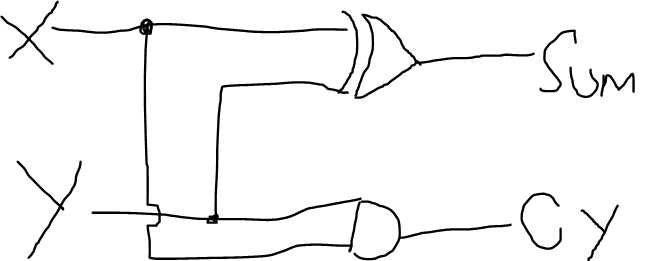
Z

Y

* + 1. Reason:  
       (X + Y) Bar = (X Bar & Y) Bar + (X & Y Bar) Bar = [(X Bar & Y) Bar] + [(X & Y Bar) Bar] = (X Bar bar + Y bar) + (X Bar + Y Bar bar) = (X + Y bar) + (X Bar + Y) = 0 + (X & Y) + (X Bar & Y Bar) + 0 = X&Y + (X Bar & Y Bar) = Ex-NOR
    2. Therefore, inverting Ex-OR gives Ex-NOR

1. **Section 2: Switches and Logic Relationship:**
   1. The output of all the Boolean operations switches from 0 and 1 (on or off state) for the given input combination.
   2. All Boolean functions are switching functions.
   3. So, using switches we can realize the Boolean function in the physical world.
   4. For example:
      1. If there is a circuit given with a battery, a switch S and a bulb, it is observed that when the switch is on, the bulb is on, and when the switch is off, the bulb is also off.
      2. Now, let another switch be connected in parallel to the first switch. Let the 2 switches be X and Y.
         1. When both the switches are off, the bulb doesn’t glow because there is no path for current to flow.
         2. If X is on but Y is off, the bulb glows because current flows through the switch X.
         3. If Y is on but X is off, the bulb still glows because current now flows through the switch Y.
         4. When both switches are on, the bulb glows because current flows through both the switches.
         5. Hence, when 2 switches are connected in parallel, an OR operation is formed.
      3. Say, instead of in parallel, the 2nd switch is connected in series.
         1. When both the switches are off, the bulb is off.
         2. When X is off but Y is on, the bulb doesn’t glow.
         3. When X is on but Y is off, the bulb still doesn’t glow.
         4. When both the switches are on, the bulb is on.
         5. Hence, when 2 switches are connected in series, an AND operation is formed.
   5. In a slide switch, let any position of the switch be considered its default state. In this state, the contact between the sliding switch and the terminal it is touching is called normally closed (or on) contact, and the lack of contact between the sliding switch and the other terminal is called as normally open (or off) contact.
      1. In a circuit, if there is a slide switch, and a bulb is connected to the normally closed part of the switch with nothing connected to the normally open part of the switch, the output connected to the normally closed contact is an inverted version of the input.
         1. Without doing anything, the bulb is on.
         2. When the switch is operated, the bulb turns off.
         3. Hence, this switch forms a NOT operation.
2. **Section 3: Combinational Logic Design – Part 1:**
   1. Example:
      1. Design a combinational logic to add (regular addition; not OR) 2 1-bit numbers.

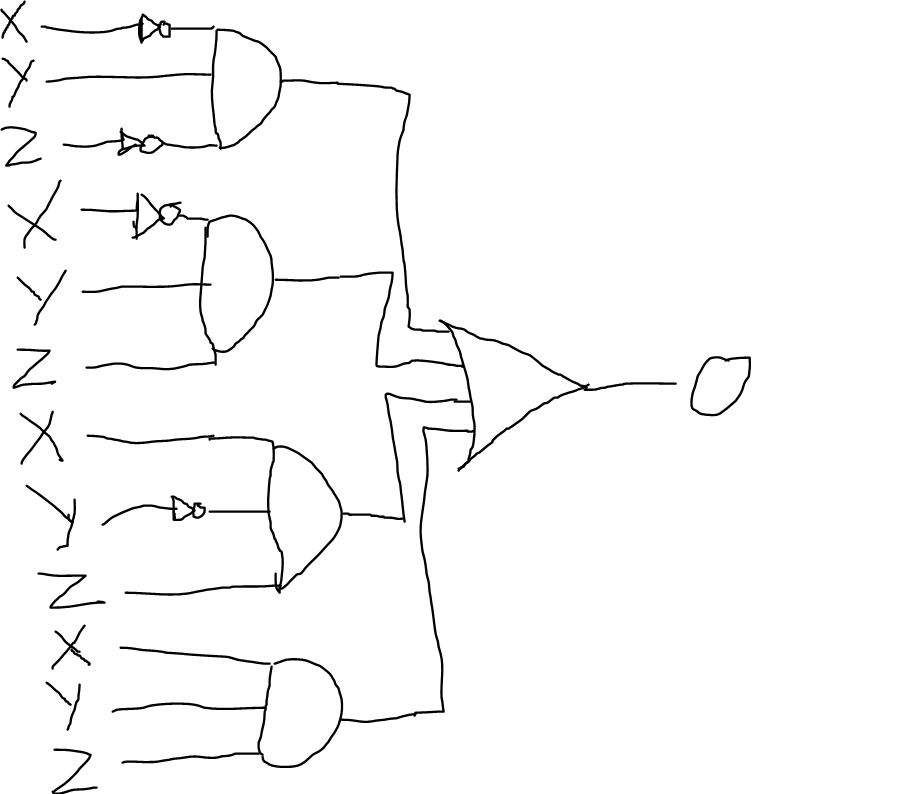
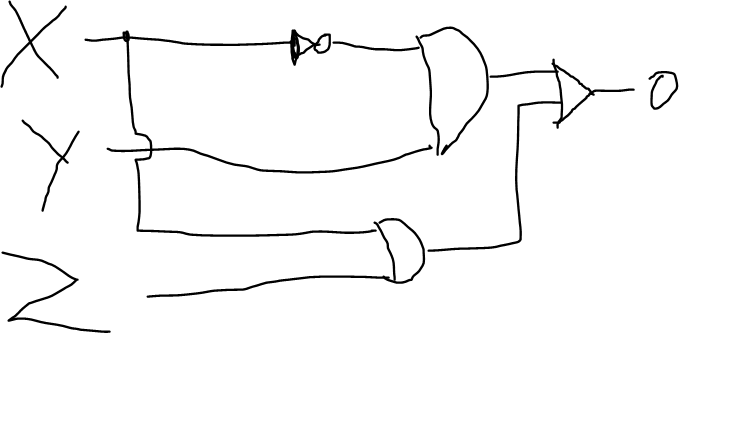
|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| X | Y | Carry (CY) | Sum |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

* + - 1. If we are to design a logic which takes in 2 inputs X and Y, and gives 2 outputs CY and Sum, Sum is created out of X, Y.
         1. This implies Sum is a function of X, Y, that is, value taken by sum depends on X, Y.
         2. Sum = f(X,Y)
         3. From the table, it is clear that Sum = X + Y
      2. Similarly, CY = f(X,Y)
         1. From the table, it is clear that CY = X & Y  
            
      3. This logic module performs 1-bit addition.
    1. We have a 3-bit input (X, Y, Z).
       1. (X, Y, Z) is not equal to (Z, Y, X) or (Y, Z, X) or any other combination as they are different numbers completely.
       2. In (X, Y, Z), where Z is the rightmost bit called LSB (Least Significant Bit) and X is the leftmost bit called MSB (Most Significant Bit).
       3. For the 3-bit number XYZ, we need to generate an output bit O as 1 whenever XYZ is a prime number.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number in Decimal | Input | | | Output |
|  | X | Y | Z | O |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |

* + - 1. As each input can take 2 values, and there are 3 inputs, therefore we can have 23 input combinations = 8 combinations
      2. To find the logic for O:
         1. O is 1 when [(X=0)&(Y=1)&(Z=0)] or [(X=0)&(Y=1)&(Z=1)] or [(X=1)&(Y=0)&(Z=1)] or [(X=1)&(Y=1)&(Z=1)]
         2. (X=0)&(Y=1)&(Z=0) is the same as X Bar & Y & Z Bar

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input |  |  | Output |  |  |
| X | Y | Z | X Bar | Z Bar | X Bar & Y & Z Bar |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

* + - * 1. Hence, (X=0)&(Y=1)&(Z=0) is the same as X Bar & Y & Z Bar
        2. Similarly, (X=0)&(Y=1)&(Z=1) will be X Bar & Y & Z  
           (X=1)&(Y=0)&(Z=1) will be X & Y Bar & Z  
           (X=1)&(Y=1)&(Z=1) will be X & Y & Z
        3. Therefore, O = (X Bar & Y & Z Bar) + (X Bar & Y & Z) + (X & Y Bar & Z) + (X & Y & Z) = f(O)  
             
           
        4. Therefore, if XYZ is a prime number, then O = 1
      1. Simplified Form of O:
         1. O = (X Bar & Y & Z Bar) + (X Bar & Y & Z) + (X & Y Bar & Z) + (X & Y & Z)  
              
            O = X Bar & Y(Z Bar + Z) + X & Z (Y Bar + Y)  
            O = (X Bar & Y) + (X & Z)
         2. Therefore, simplified version of the circuit will be:  
              
            

1. **Section 4: Combinational Logic Design – Part 2:**
   1. Let there be any random Boolean function:

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | | Output |
| X | Y | Z | O |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

* 1. In order to get O = 1, the input combinations should be (X Bar Y Bar Z Bar) + (X Bar Y Z) + (X Y Z Bar)
  2. To know that we got a simplified version of the logic, we use an algorithmic approach called K-Map.
  3. K-Map:
     1. Full Form: Karnaugh Map
     2. If only 1 bit is changing between 2 expressions, then we can group it and simplify it.
     3. If we can arrange input combinations in such a way that there is only 1-bit change in the adjacent neighbour, then we can group and optimize the inputs, if the adjacent neighbour is also present (if both are 1 as output).
     4. Hence, in the above table we can’t group 2 and 3 as, even though both have 1 bit change, 2 is not present (output is 0). Hence they can’t be grouped.
     5. In order to achieve grouping, the numbers can be re-arranged as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| 000 0 | 001 1 | 011  3 | 010  2 |
| 100  4 | 101  5 | 111  7 | 110  6 |

* + 1. By arranging the numbers (input combinations) this way, we are seeing only 1-bit variation across any 2 adjacent cells.
    2. This arrangement is known as K-Map.
    3. For Prime Numbers:

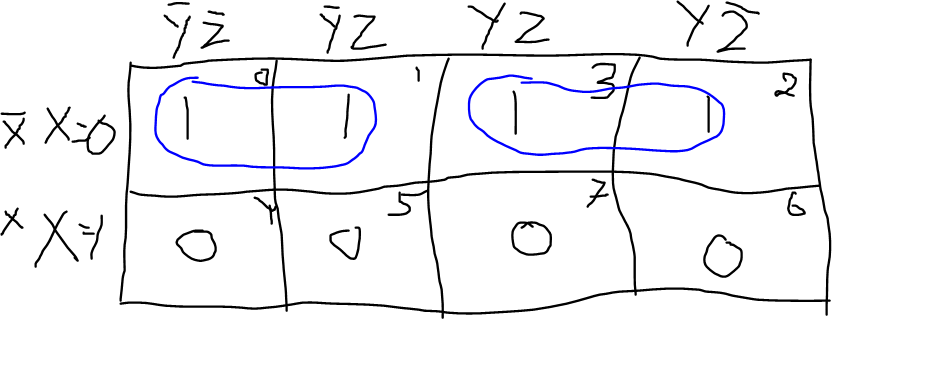
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Y Bar Z Bar | Y Bar Z | Y Z | Y Z Bar |
| X = 0 | 000 0 | 001 1 | 011 1 3 | 010 1 2 |
| X = 1 | 100  4 | 101 1 5 | 111 1 7 | 110  6 |

* + - 1. If XYZ = 2, O = 1, that is X Bar Y Z Bar results in O = 1
      2. XYZ = 3, O = 1, so X Bar Y Z results in O = 1
      3. Hence (X Bar Y Z Bar) and (X Bar Y Z) can be grouped, and whatever goes away can be removed, that is, as Z is changing, it can go away, and the resulting answer is X Bar Y
      4. In the same way, (X Y Bar Z) and (X Y Z) can be grouped, and as Y is changing, it can be removed, giving X Z as the answer.
    1. For current example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Y Bar Z Bar | Y Bar Z | Y Z | Y Z Bar |
| X = 0 | 000 1 0 | 001 1 | 011 1 3 | 010  2 |
| X = 1 | 100  4 | 101  5 | 111  7 | 110 1 6 |

* + - 1. Here, no adjacent cell grouping is possible.
      2. Therefore, whatever expression we already have is the simplified expression, that is (X Bar Y Bar Z Bar) + (X Bar Y Z) + (X Y Z Bar) is already simplified.
    1. Another Example:

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | | Output |
| X | Y | Z | O |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

* + - 1. This table in the form of a K-Map along with grouping is:  
           
         
      2. In 0 and 1, Z is changing, so the resulting expression will be (X Bar Y Bar)
      3. In 3 and 2, Z is again changing, so the resulting expression will be (X Bar Y).
      4. So the logical expression will be (X Bar Y Bar) + (X Bar Y) = X Bar (Y Bar + Y) = X Bar
      5. Hence, the logical expression of the output is X Bar.