Elements of Gate Logics

0.1 Introduction

The instructions and data are in binary, i.e., $\{1, 0\}$ format. The binary Levels in logics, for example, in TTL are: 0-0.8 volts for logic 0 (false), and 2.0-5.0 volts for logic 1 (true). Other logics are ECL, DTL, RTL, etc.

A CPU identifies a binary string as Data or Instructions, with reference to start location in the memory, which is instruction, and other instructions are in consecutive locations, advanced by the size of the instructions. The data locations' addresses are specified as part of the instructions.

A minimum circuit to store a bit (0 / 1) is called memory cell. A single memory cell is a Flip-flop or Bipolar Transistor-pair, where one transistor is always in saturation and other in cutoff. The figure 1 shows a single memory cell made of two bipolar transistors and other register components.

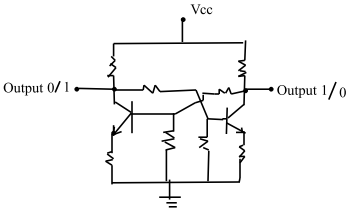

Figure 1: Single cell to store 1 or 0.

0.2 Logic Gates

The basic gates are : *AND* gate, *OR* gate and *NOT* gate. The universal gates are NAND and NOR gates. Figure 2 shows the various gates, and circuits made of gates.

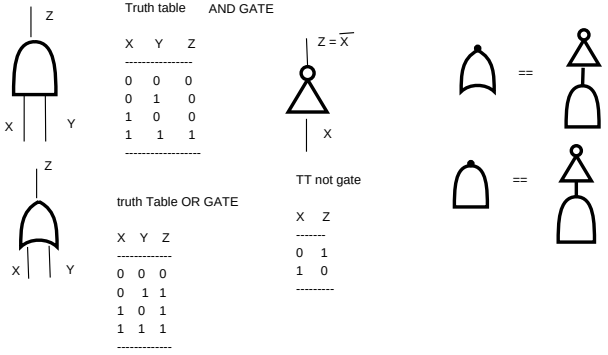

Figure 2: Basic Logic Gates.

The figure 3 show a two-input diode-transistor logic gate. If any of the inputs A or B is true (high), it will forward bias the transistor, and the output is false (low). When both the inputs are low, the transistor is reversed bias, and output is high. This gives the nor operation.

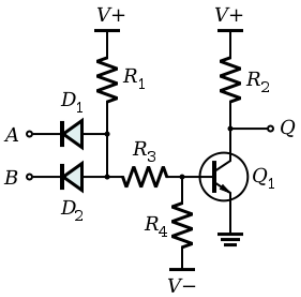

Figure 3: Two input DTL NOR Gate.

The TTL is a NAND operation is shown in figure 4. When all the inputs are held at high voltage, the base–emitter junctions of the multiple-emitter transistor are reverse-biased. A small "collector" current (approximately 10μA) is drawn by each of the inputs. An approximately constant current flows from the positive supply, through the resistor and into the

base of the multiple emitter transistor. This current passes through the base-emitter junction of the output transistor, allowing it to conduct and pulling the output voltage low (logical zero).

For an input of logical zero, the base-collector junction of the multiple-emitter transistor and the base-emitter junction of the output transistor are in series between power supply and ground. If one input voltage becomes zero, the corresponding base-emitter junction of the multiple-emitter transistor is in parallel with these two junctions. A phenomenon called current steering means that when two voltage-stable elements with different threshold voltages are connected in parallel, the current flows through the path with the smaller threshold voltage. As a result, no current flows through the base of the output transistor, causing it to stop conducting and the output voltage becomes high (logical one). During the transition the input transistor is briefly in its active region; so it draws a large current away from the base of the output transistor and thus quickly discharges its base. This is a critical advantage of TTL over DTL that speeds up the transition over a diode input structure.

However, this simple TTL circuit has disadvantages also, which can be improved by adding two more transistors and and diode, to improve its output resistance.

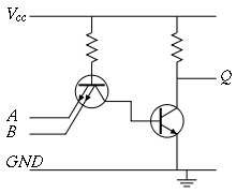

Figure 4: Two input TTL NAND Gate.

The number of connections from a gate, which can be taken is called fan-out, similarly the fan-in is defined. Hence,

Fan-In: The maximum number of gates which can be connected input to a gate.
Fan-Out: The maximum number of gates which can be connected at the output of a gate.

0.3 Simple Boolean Logic Circuits

The entire computer system is made of using these basic gates, called **Combinational Circuits**. The combinational circuits have no memory

element. Figure 5 shows a simple combinational circuit.

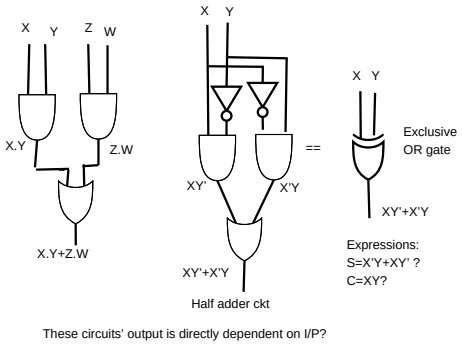

Figure 5: Adder Circuits.

In contrast to combinational circuits, the *sequential* circuit's output is a function of current input and previous input, hence the output depends on past history also.

0.4 Standard forms of expressions

We can write the boolean expressions in many ways, but some ways are more useful than others.
A **sum of products** (SOP) expression contains: Only OR (sum) operations at the "outer-most" level, each term that is summed must be a product of literals. For example, $f(x, y, z) = y' + x'yz' + xz$ is a SOP expression. The advantage is that any sum of products expression can be implemented using a two-level circuit. In the expression, The literals and their complements at the "0th" level, AND gates at the first level, and a single OR gate at the second level.

A **Minterms** is a special product of literals, in which each input variable appears exactly once. A function with n variables has 2^n minterms (since each variable may also appear complemented). A three-variable function, such as $f(x, y, z)$, has $2^3 = 8$ minterms. For example,

$$x'y'z', x'y'z, x'yz', x'yz,$$
$$xy'z', xy'z, xyz', xyz$$

is a minterm. This can be represented in short form as

$$(m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7).$$

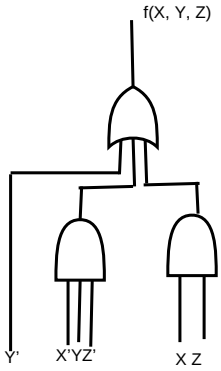

Figure 6: Sum of Product

The *minterm* m_i assumes a value 1 for unique value of variables.

A **Product of sums** expression contains only AND (product) operations at the "outermost" level, and each term must be a sum of literals. A Product of sums expressions can be implemented with two-level circuits: literals and their complements at the "0th" level, and OR gates at the first level, a single AND gate at the second level. For example, $f(x, y, z) = y'(x' + y + z')(x + z)$ is a SOP expression.

A **Maxterms** is a sum of literals , in which each input variable appears exactly once. A function with n variables has 2^n maxterms. Each maxterm is false for exactly one combination of inputs. A maxterm for a three-variable function $f(x, y, z)$ is

$$(x' + y' + z')...(x + y + z)$$

or

$$(M_0, ..., M_7).$$

Note that, *maxterm* defines the 0s in the truth table.

Product of maxterms: If you have a truth table for a function, you can write a product of maxterms expression by picking out the rows of the table where the function output is 0 (be careful if you are writing the actual literals!)

Table 1: Boolean function's table.

x	y	z	$f(x,y,z)$	$f'(x,y,z)$
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

0.4.1 Maxterms

Consider the table 1 for boolean function, whose expressions are given in the last two columns as well as the function values.

$$\begin{aligned} f &= (x'+y+z)(x'+y+z')(x'+y'+z') \\ &= M_4M_5M_7 \\ &= \Pi M(4,5,7) \\ f' &= (x+y+z)(x+y+z')(x+y'+z)(x+y'+z')(x'+y'+z) \\ &= M_0M_1M_2M_3M_6 \\ &= \Pi M(0,1,2,3,6) \end{aligned}$$

0.4.2 Conversion between standard forms

The minterms can be converted into maxterms and vice-versa using De-Morgan's laws. Considering maxterms' expression as

$$\begin{aligned} f' &= \Pi M(0,1,2,3,6) \\ &= M_0M_1M_2M_3M_6 \end{aligned}$$

$$\begin{aligned} (f')' &= (M_0M_1M_2M_3M_6)' \\ f &= (M_0' + M_1' + M_2' + M_3' + M_6') \\ &= (m_4m_5m_7) \\ &= \Sigma m(4,5,7) \end{aligned}$$

That means, given the maxterms, list all the maxterms which can be created with those many variables, eliminate the maxterms from these which are already existing, and from the remaining, complement the variables from each, and join them as minterms.

In the similar way, the minterms can be converted into maxterms.

0.5 Combinational Circuits

The circuits, like, Adders, Subtractors, Multipliers, Dividers, Multiplexers, Demultiplexers, whose output depends on current input, are called combinational circuits. The figure 7 shows some of the combinational circuits. Half adder adds two bits and produces the sum and carry. A circuits, which adds two bits and the previous addition's carry is called a full-adder. Thus, two add 4-bits you need three full-adders, and one half-adder for the two least significant bits.

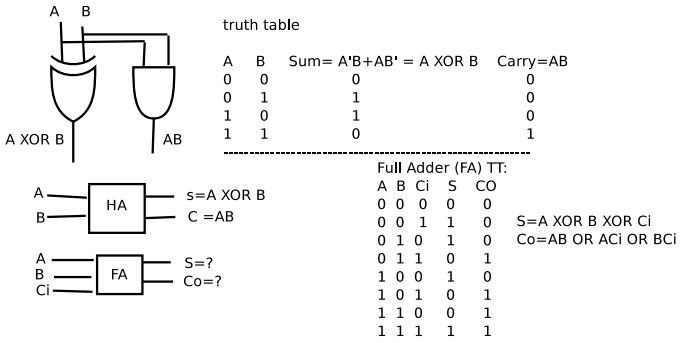

Figure 7: Combinational Circuits.

The following are the steps for determining the Max and Min terms expressions for the full adder.

In general, we say,

$$f(x_1, \ldots, x_n) = \sum_i (x_{i1}^{\cdot} \ldots x_{in}^{\cdot})$$

where, $x_{ij}^{\cdot} = x_{ij} | \overline{x_{ij}}$ is a SOP.

The product of sums can be expression by,

$$f(x_1, \ldots, x_n) = \prod_i (x_{i1}^{\cdot} + \cdots + x_{in}^{\cdot}).$$

The above methods are called, *Quinne-McClusky Method*. From the truth table shown in figure 7, we have,

$$S = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i$$

and,

$$C_o = \bar{A}BC_i + A\bar{B}C_i + AB\bar{C}_i + ABC_i$$

The general format for C_o is,

$$C_o(A, B, C_i) = (m_3, m_5, m_6, m_7)$$

and the sum S is: $(A \oplus B) \oplus C_i$, where ($\oplus$ is ex-or).

$$\begin{aligned} S &= (\bar{A}B + A\bar{B}) \oplus C_i \\ &= ABC_i + \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i \end{aligned}$$

The carry out of full adder, $C_o = AB + AC_i + BC_i$.

0.6 Karnaugh Map for minimization of Gate circuits

The karnaugh map is after *Maurice Kaurnaugh*, Bell Labs, 1950.

K-map minimization using minterms:

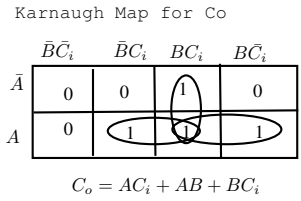

Figure 8: Minimization using Karnaugh maps.

The function $C_o(A, B, C_i)$ Can be implemented by three *AND* gates plus one *OR* gate. Alternatively by: four gates each having fan-in 2.

K-map minimization using maxterms:

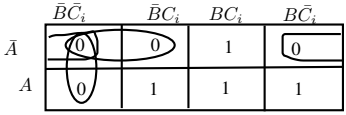

Figure 9: Minimization using Karnaugh maps.

$$
\begin{aligned}
C_o(A, B, C_i) &= (A + B)(A + C_i)(B + C_i) \\
&= (A + AC_i + AB + BC_i)(B + C_i) \\
&= \ldots \\
&= AB + BC_i + AC_i
\end{aligned}
$$