Module 3 – Digital Gates and Combinational Logic

INTRODUCTION:

The principles behind digital electronics were developed hundreds of years before technology allowed us to build the first logical gate. A system of controlling processes and making decisions based on simple "yes" or "no" answers was developed by the Irish mathematician, George Boole in the 1850’s. He formalized the way we express logical expressions and documented the AND, OR, NOT functions that are the foundation of all modern logical design today.

CONCEPT 3-1: Basic Logic Gates

All logical operations can be narrowed down to three basic gates:

- **AND** - A logical "1" is output only if all inputs are a logical "1".
- **OR** - A logical "1" is output if any of the inputs are a logical "1".
- **NOT** – A single input, single output device that changes a logical "1" to a logical "0" or a logical "0" to a logical "1".

CONCEPT 3-1A: AND Gates

An AND gate has two or more input lines for parallel binary bits, and one output bit. The rule is that if all inputs are a high, or a logical "1", then the output is a high, or a logical "1". The output is a logical "0" for any other binary combination of ones and zeros on the input lines.

The Boolean Algebra expression for an AND gate is: \( A \times B = Q \). If the letters \( A \) and \( B \) that designate the two inputs of the gate in the figure are horizontally adjacent, or side by side with nothing in between, it indicates an AND operation. If the above gate had three inputs designated as \( A, B, \) and \( C \) with the output designated as \( Q \); the Boolean Algebra expression would be:
A B C = Q.

**CONCEPT 3-1B: OR Gates**

An OR gate has two or more input lines for parallel binary bits, and one output bit. The rule is that if any of the inputs are a high, or a logical "1", then the output is a high, or a logical "1". The only time that the output is a low, or a logical "0", is when all inputs are low, or at logical "0".

The Boolean Algebra expression for an OR gate is: \( A + B = Q \). If the letters \( A \) and \( B \) that designate the two inputs of the gate in the figure are horizontally adjacent, or side by side with a "+" sign in between, it indicates an OR operation. If the above gate had three inputs designated as \( A, B, \) and \( C, \) with the output designated as \( Q; \) the Boolean Algebra expression would be:

\[ A + B + C = Q. \]

**CONCEPT 3.1C: INVERT OR NOT GATE**

An INVERTER is the simplest of all gates. It outputs the opposite binary level that it inputs. A logical "1" becomes a logical "0", and a logical "0" becomes a logical "1". It does nothing else to the data.
Another way to indicate that a function or a variable is inverted is to draw a line over the top. This is done in the mathematical expression. In a digital schematic diagram that shows the interconnections of an electronic circuit, a logical inversion is shown with a small circle. A line over the top of $A$ is $\text{NOT } A$ and means that if $A$ is a logical 1, $\text{NOT } A$ is a logical 0. A device in a schematic diagram that has a small circle takes the logical level of the signal going to the circle and inverts it to the opposite logical level. Notice in the figure above. The small circle on the point at the output of the INVERTER changes $A$ to $\text{NOT } A$.

**CONCEPT 3-2: Advanced Gates**

By using combinations of AND, OR and NOT gates, three new types of gates are produced, Exclusive OR (XOR), Negative AND (NAND), and a Negative OR (NOR). The new gates have the following rules:

- **NAND** – A logical "0" is output anytime all inputs are a logical "1".
- **NOR** – A logical "0" is output if any of the inputs are a logical "1".
- **XOR** – A logical "1" is output anytime the inputs have an odd number of logical "1"s.

**CONCEPT 3-2A: NAND Gates**

NAND gates are AND gates with the output inverted, or OR gates with all inputs inverted. If all of the inputs are at a logical "1" state, the output is zero. The output will be at a logical "1" state for all other input combinations.

A NAND gate is produced by inverting the output of an AND gate, or by inverting all of the inputs to an OR gate. The rule simply becomes the inverse of the AND gate, or if all inputs to the NAND gate are high, or at a logical "1", then the output is low, or at a logical "0". The output is high, or at logical "1" for all other input combinations.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>NAND</th>
<th>TRUTH TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$Y$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

LOGIC SYMBOLS
The Boolean Algebra expression for the NAND gate has two forms. One form is for the AND gate with the output inverted:

\[ Y = \overline{A \cdot B} \]

The other form is for the OR gate with the inputs inverted:

\[ Y = A + \overline{B} \]

This gives rise to two Boolean Algebra expressions that mean exactly the same thing, and two schematic symbols that express the same identical gate.

\[ Y = \overline{A \cdot B} = A + \overline{B} \]

**CONCEPT 3-2B: NOR Gates**

NOR gates are OR gates with the output inverted, or AND gates with all inputs inverted. If any of the inputs are at a logical "1" state, the output is zero. The output will be at a logical "1" state only when all inputs are at a logical "0" state.

A NOR gate is produced by inverting the output of an OR gate, or by inverting all of the inputs to an AND gate. The rule simply becomes the inverse of the OR gate, or if any of the inputs to the NOR gate are high, or at a logical "1", then the output is low, or at a logical "0". The output is high or at logical "1" only when all inputs are zero.

![NOR Gate Truth Table](image)

The Boolean Algebra expression for the NOR gate has two forms. One form is for the OR gate with the output inverted:

\[ Y = A + \overline{B} \]

**Definition 3.1: Gate Equivalence**

When different configurations of logic gates produce the same outcome, they are said to be equivalent.
The other form is for the AND gate with the inputs inverted:

\[ Y = \overline{A} \overline{B} \]

This gives rise to two Boolean Algebra expressions that mean exactly the same thing, and two schematic symbols that express the same identical gate.

\[ Y = A + B = \overline{A} \overline{B} \]

**CONCEPT 3-2C: XOR GATES**

The last gate considered here is the EXCLUSIVE OR, or XOR gate. The XOR gate has the rule that if there is an odd number of logical "1" or high states, then the output is at a high, or a logical "1".

An XOR gate is produced by building a circuit that has AND gates, an OR gate, and NOT or INVERT gates.

The Boolean Algebra expression for the XOR gate comes from the truth table in the above figure. The output \( Q \) will be high for two of the input combinations:

\[ Q = \overline{A} B + A \overline{B} \]

The above Boolean Algebra expression will yield the correct result in a larger expression, but the common shorthand way of writing the XOR expression is like an OR expression with a circle around the + sign. A double parenthesis can also be used when typing the expression:

\[ Q = A (\oplus) B \]

To describe the circuit, the inputs of a two-input OR gate are two AND gates. One of the AND gates has the input coming from signal A inverted, and the
signal coming from B is input to the same AND gate without being inverted. The other AND gate has the B signal input inverted, and the A signal input not inverted. The following figure illustrates how this works.

**Definition 3.2: Multiplexer**
A multiplexer is digitally controlled switch that switches data from multiple sources to a single output.

**Definition 3.3: Boolean Algebra**
Boolean Algebra uses two state, or binary logic, to control outcomes based on all possible combinations of numerous input, or driving factors. Boolean Algebra was formalized by the Irish Mathematician, George Boole in the mid 1800’s.

**Definition 3.4: Truth Table**
A truth table is a tabulation of all possible outcomes for combinations of binary, or two state, logic. Truth Tables are used in digital logic designs to describe which combination input states result in a positive output, and which combinations result in no action.
**Definition 3.5: Combinational Logic**

This is a logic system made up of basic logic gates that will interpret all possible combinations of inputs, and will output a signal (logical "1" or "0"), when a desired combination is present.

**CONCEPT 3.3: Truth Tables Translate into Combinational Logic Circuits**

A Truth Table is organized into rows and columns with all possible combinations of the inputs to the left, and resulting outputs to the right. For each combination of inputs that result in a desired output, a combination of AND, OR and INVERT gates can be assembled that will output a logical "1" or "0" when the input combinations are present.

Consider the following example. A farmer has a pump “P” that pumps water onto his crop. The pump will only come on if the crop is dry “D”, and there is water in the canal “C”. The pump also has an over ride switch that the farmer “F” can press and turn the pump on any time.

The design starts by generating a truth table that will control the pump. You may assume that the sensors for wet or dry crop are there, and there is a sensor that tells if there is water in the canal. You may also assume that a logical “1” signal to a control circuit will turn on the pump.

1. Generate the truth table with all possible combinations on the inputs F, D, and C. Notice the inputs can be taken in any order.

<table>
<thead>
<tr>
<th>F</th>
<th>D</th>
<th>C</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

   The logic that controls the pump reduces down to an OR gate, an AND gate, and an INVERTER.
The Boolean Algebra expression for the pump control logic is:

\[ P = F + \overline{D} \overline{C} \]

**CONCEPT 3.4: Families and Types of Logic**

Families refer to the gate design and intended application. Different families of logic are faster or slower, can drive larger or smaller electrical loads, and require more or less of a driving source. Type refers to how the logic is packaged, and what configurations of gates are in the package. There can be just a few gates in a package, or there can be millions of transistors and complex logic matrices in a single IC chip, such as the circuits in programmable logic devices.

The table below outlines some of the more common families and types of logic gates, gives their common part reference numbers, and lists speed, input loads, output drive capability, and package considerations. Family characteristics in the table are compared to the original Transistor-Transistor Logic (TTL) logic specifications of

1 TTL load = 1.4 mA of current, called “Fan In”, and an output drive capability of 10 TTL loads called, “Fan Out”.

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>DESCRIPTION</th>
<th>PART #</th>
<th>DELAY</th>
<th>FAN IN</th>
<th>FAN OUT</th>
<th>APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTL</td>
<td>Early Logic</td>
<td>74</td>
<td>10 ns</td>
<td>1.4 mA</td>
<td>14 mA</td>
<td>General</td>
</tr>
<tr>
<td>S</td>
<td>Schottky</td>
<td>74S</td>
<td>2 ns</td>
<td>5 mA</td>
<td>20 mA</td>
<td>High Speed</td>
</tr>
<tr>
<td>LS</td>
<td>Low Pwr Schottky</td>
<td>74LS</td>
<td>8 ns</td>
<td>1.4 mA</td>
<td>14 mA</td>
<td>Med. Speed</td>
</tr>
<tr>
<td>AS</td>
<td>Adv. Schottky</td>
<td>74AS</td>
<td>1.5 ns</td>
<td>3 mA</td>
<td>20 mA</td>
<td>High Speed</td>
</tr>
<tr>
<td>F</td>
<td>Fast</td>
<td>74F</td>
<td>4 ns</td>
<td>5 mA</td>
<td>20 mA</td>
<td>High Speed</td>
</tr>
<tr>
<td>HC</td>
<td>High Speed CMOS</td>
<td>74HC</td>
<td>5 ns</td>
<td>1 uA</td>
<td>2 mA</td>
<td>Low Power</td>
</tr>
<tr>
<td>HCT</td>
<td>TTL Compat. HC</td>
<td>74HCT</td>
<td>5 ns</td>
<td>0.1 mA</td>
<td>14 mA</td>
<td>CMOS to TTL</td>
</tr>
<tr>
<td>AC</td>
<td>Advanced CMOS</td>
<td>74AC</td>
<td>4 ns</td>
<td>1 uA</td>
<td>2 mA</td>
<td>Low Power</td>
</tr>
<tr>
<td>ACT</td>
<td>TTL Compat. AC</td>
<td>74ACT</td>
<td>4 ns</td>
<td>0.1 mA</td>
<td>20 mA</td>
<td>CMOS to TTL</td>
</tr>
<tr>
<td>LVC</td>
<td>Low Voltage CMOS</td>
<td>74LVC</td>
<td>8 ns</td>
<td>1 uA</td>
<td>2 mA</td>
<td>3.3 V Power</td>
</tr>
<tr>
<td>BICMOS</td>
<td>TTL &amp; CMOS</td>
<td>74BCT</td>
<td>3 ns</td>
<td>1 uA</td>
<td>20 mA</td>
<td>Fast&amp;Lo Pwr</td>
</tr>
</tbody>
</table>

The table above provides a comparison of the characteristics of different logic families and types, including speed, fan in, fan out, and application areas.
PINOUTS OF SEVERAL COMMON GATES:

CONCEPT 3.5: Programmable Logic

Special devices now exist that have arrays of AND, OR and INVERT gates with their inputs and outputs connected together in a grid pattern, much like the wires in a window screen. The intersections of each horizontal and vertical line in the grid can be electrically connected or disconnected by a computer controlled programmer. The ability to connect any gate inputs and outputs to those of any other gate give the designer the ability to program any combinational logic circuit imaginable involving the gates in the programmable logic array. The capability of programmable logic is so vast that entire computers are now being built using one of these devices.