**Chapter 3**

**Digital Logic Fundamentals**

*Lesson 3.1:* Logic Fundamentals

*Lesson 2.2:* Kurnugh Map

*Lesson 2.3:* Shift Registers and Counters

****

***Lesson 3.1***

***Logic Fundamentals***

**3.1.0 Objectives**

*On completion of this lesson you will know:*

* *Primary logic gates*
* *Secondary Logic gates*
* *Elementary concepts of Boolean algebra*
* *Elementary concepts of DeMorgan’s Theorem*
* *Universal logic gates*
* *Logic Simplification*

**3.1.1 Introduction**

Data and control instructions move inside a computer by means of pulses of electricity. Certain components of computers combine these pulses following a set of rules. These components are the logic elements.

Pulses of electricity are called digital signals. It has two discrete levels or values. These two discrete signal levels, HIGH and LOW, can also be represented by binary 1 or 0 respectively. A binary digit (1 or 0) is referred to as a bit. A binary signal can have only one of the two possible levels 1 and 0, and the binary number system can be used for the analysis and design of digital systems. George Boole introduced the concept of binary system in 1854 and developed Binary algebra called Boolean Algebra.

**3.1.2 Logic Gates**

**Primary Logic Gates**

A logic gate is a circuit which uses digital signals as its inputs and outputs. The common use of logic elements is to act as switches, although they have no moving parts. They open to pass on a digital pulse and close to shut it off. That is why they are known as gates. The primary gates are OR, AND and NOT.

Digital circuits which use logic gates are usually arranged so that logic 1 appears at an output only for some definite combinations of input signals - for this reason these circuits are sometimes called combinational logic circuits.

**OR Gate:** The OR gate is a digital logic gate that implements logical disjunction. It operates according to the truth table as shown in Figure 3.1.1. A HIGH (1) output results if one or all the inputs to the gate are HIGH (1). If neither input is HIGH, a LOW (0) output results. OR operation is denoted by +.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



***Figure 3.1.1: Symbol and truth table of a two inputs OR gate***

**AND Gate:** The AND gate implements logical conjunction. It operates according to the truth table as shown in Figure 3.1.2. A HIGH (1) output results if all the inputs to the gate are HIGH (1). A dot (●) is used to show the AND operation i.e. A●B.  However, this dot is sometimes omitted and the output is denoted by AB.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



***Figure 3.1.2: Symbol and truth table of a two inputs AND gate***

**NOT Gate:** The NOT gate produces an inverted version of the input at its output.  It is also known as an inverter.  If the input variable is A, the inverted output is called NOT A. This is also shown as A', or A with a bar on the top, i.e., , as shown at the output. Symbol and truth table of a NOT gate is shown in Figure 3.1.3.

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 | 0 |



***Figure 3.1.3: Symbol and truth table of a NOT gate***

**Secondary Logic Gates**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**NAND gate:** This is equivalent to an AND gate followed by a NOT gate.  The output of a NAND gate is high if any one of the inputs is low or all the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion. Symbol and truth table of NAND gate is shown in Figure 3.1.4.



***Figure 3.1.4: Symbol and truth table of a two inputs NAND gate***

**NOR gate:** This is equivalent to an OR gate followed by a NOT gate. The outputs of a NOR gate is low if any one of the inputs is high or all the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion. Symbol and truth table of a NOR gate is shown in Figure 3.1.5.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



***Figure 3.1.5: Symbol and truth table of a two inputs NOR gate***

**EXOR gate:** The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high.  An encircled plus sign () is used to show the EXOR operation. Symbol and truth table of EXOR gate is shown in Figure 3.1.6.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



***Figure 3.1.6: Symbol and truth table of a two inputs NOR gate***

**EXNOR gate:** The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion. Symbol and truth table of EXNOR gate is shown in Figure 3.1.7.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



***Figure 3.1.7: Symbol and truth table of a two inputs EXNOR gate.***

The NAND and NOR gates are called universal functions since with either one the AND, OR and NOT functions can be generated.

**3.1.3 Boolean Theorems**

Logic circuits can be simplified by Boolean algebra and simplification of logic functions and logic circuits is an important application of Boolean algebra. A simplified circuit costs less and works efficiently. Boolean theorems are used to simply logic functions. Important Boolean theorems are given in Table 3.1.1.

**Example 3.1.1:** Simplify 



**Example 3.1.2:** Simplify



***Table 3.1.1: Boolean theorems***

|  |  |  |
| --- | --- | --- |
| Theorems based on AND logic: | | **Example** |
| 1. | A ● 0 = 0 | 1 ● 0 = 0 0 ● 0 = 0 |
| 2. | A ● 1 = A | 1 ● 1 = 1 0 ● 1 = 0 |
| 3. | A ● A = A | 1 ● 1 = 1 0 ● 0 = 0 |
| 4. | A ●  = 0 | 1 ● 0 = 0 0 ● 1 = 0 |
| 5. | A ● B = B ● A (Commutative Law) |  |
| 6. | A ● B ● C= (A ● B) ● C= A ● (B ● C)  (Associative Law) |  |
| Theorems based on OR logic: | | **Example** |
| 7. | A **+** 0 = A | 1 + 0 = 1 0 + 0 = 0 |
| 8. | A **+** 1 = 1 | 1 + 1 = 1 0 + 1 = 1 |
| 9. | A **+** A = A | 1 + 1 = 1 0 + 0 = 0 |
| 10. | A **+**  = 1 | 1 + 0 = 1 0 + 1 = 1 |
| 11. | A **+** B = B **+** A (Commutative Law) |  |
| 12. | A **+** B **+** C= (A **+** B) **+** C= A **+** (B **+** C)  (Associative Law) |  |
| Distributed Law | |  |
| 13. | A ● (B **+** C) = A ● B **+** A ● C |  |
| 14. | A + (B ● C) = (A + B) ● (A + C) |  |
| Redundancy Law | |  |
| 15a. | A **+** (A ● B) = A |  |
| 15b. | A ● (A **+** B) = A |  |

**3.1.4 De Morgan’s Theorems**

French Mathematician De Morgan formulated two theorems in 1953 and they are known as De Morgan’s theorems. For two variables, De Morgan’s theorem can be written as





The proof of the De Morgan’s theorems for two variables using truth table is as follows

***Table 3.1.2: Proof of the De Morgan’s theorems for 2 varibles***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Remark |  |  | Remark |
| 0 | 0 | 1 | 1 | De Morgan’s  Theorem is verified | 1 | 1 | De Morgan’s  Theorem is verified |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

For three variables, De Morgan’s theorems can be written as





The De Morgan’s theorems are used to simplify logic functions associated with NOT gates.

**3.1.5 Universality of NAND and NOR gates**

Any logic can be realized by OR, AND and NOT gates. But it is possible to realize any logic circuit by using only NAND or NOR gates. This is known as the universality of NAND and NOR gates. Figure 3.1.8 and 3.1.9 show realization of basic logic gates by NAND gates only and NOR gates only respectively.



***Figure 3.1.8: Realization NOT, AND and OR gates by NAND gates only***



***Figure 3.1.9: Realization NOT, AND and OR gates by NOR gates only***

**Example 3.1.3:** Realize the logic circuit of Figure 3.1.10 by NAND gates only.



***Figure 3.1.10: Example 3.1.3***

***Solution:***

The basic logic gates are first replaced by their NAND equivalents as shown in Figure 3.1.11 (a). Two serial NAND operations do not perform in any logic operation. Thus these can be eliminated as shown in Figure 3.1.11 (b).



***Figure 3.1.11: Solution of Example 3.1.3***

**3.1.7 Key points**

* Data and control instructions move inside a computer by means of pulses of electricity.
* Pulses of electricity are called digital signals.
* HIGH and LOW levels of a pulse can also be represented by binary 1 or 0 respectively.
* A logic gate is a circuit which uses digital signals as its inputs and outputs.
* The OR gate is a digital logic gate that implements logical disjunction.
* The AND gate implements logical conjunction.
* The NAND and NOR gates are called universal functions since with either one the AND, OR and NOT functions can be generated.
* Logic circuits can be simplified by Boolean algebra, and simplification of logic functions and hence logic circuits, is an important application of Boolean algebra.
* De Morgan’s theorem can be written as  and .

**2.1.6 Practice Set**

**Multiple Choice Questions**

1. An OR function is denoted by \_\_\_\_\_\_\_\_\_\_\_.
   1. ×
   2. ©
   3. 
   4. +
2. The AND gate implements logical \_\_\_\_\_\_\_\_\_\_.
   1. conjunction
   2. disjunction
   3. pn junction
   4. None
3. The primary gates are \_\_\_\_\_\_\_\_\_\_
   1. OR, AND, NAND
   2. NOR, NOT, AND
   3. OR, AND, NOT
   4. NOR, NAND, EX-OR
4. The Pulses of electricity represents\_\_\_\_\_\_\_\_\_\_ signal.
   1. voltage
   2. analog
   3. current
   4. digital
5. Which one is true for De Morgan’s Theorem \_\_\_\_\_\_\_\_\_\_.
   1. 
   2. 
   3. All.
   4. None.
6.  = \_\_\_\_\_\_\_\_\_\_.
   1. .
   2. 
   3. 
   4. 

**Review Questions**

1. What do you know about Boolean algebra?
2. Draw the symbols and truth tables of a two input (a) OR gate (b) AND gate?
3. Explain the operation of AND, NOT, NOR and XOR gates
4. What is a universal gate?
5. Define K-map? Why is it necessary?
6. Write down De Morgan’s Theorem for three variables.

**Analytical Questions**

1. Prove that NAND and NOR gates are universal gates.
2. Explain different types of secondary logic gates.
3. State De Morgan’s Theorem. Verify the theorem using a truth table.
4. Write De Morgan’s Theorem for four variables.
5. Realize the following logic circuit by using NOR gates only. 
6. Simply the following expressions using Boolean algebra
   1. 
   2. 
   3. 
   4. 
   5. 
   6. 
7. Give the truth tables and draw logic circuits for the following:
   1. 
   2. 
8. Realize logic circuits for the following
   1. 
   2. 
9. Replace the basic gates by their NAND or NOR equivalents and convert them for NAND, or NOR gate only.

****

***Lesson 3.2***

***Karnaugh Map***

**3.2.0 Objectives**

*On completion of this lesson you will know:*

* *Use of Karnaugh Map*
* *Two-variable K-map*
* *Three-variable K-map*

**3.2.1 Karnaugh Map**

The Karnaugh map (K-map) provides a simple method of minimizing Boolean expressions. Boolean expressions having four and even six variables can be simplified with the K-map . 

A K-map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The K-map can also be described as a special arrangement of a truth table.

**3.2.2 Two-variable K-map**

Figure 3.2.1 shows two-variable K-map and the truth table for the general case of a two-variable problem. Around the edge of the K-map are the values of the two input variables A and B. A is along the top and B is down the left hand side. The values around the edge of the map can be thought of as coordinates. As an example, the square on the top right hand corner of the map in the map has coordinates and . This square corresponds to the row in the truth table where  and and . Note that the value in the F column represents a particular function to which the K-map corresponds.



**Figure 3.2.1: A Two-variable K-map**



**Figure 3.2.2: K-map for** 

Figure 3.2.2 shows the truth table and K-map for the function.

In this figure, the and  condition of the truth table corresponds to the  cell whereas the  and  condition of the truth table corresponds to the  cell.

**Example 3.2.1:** Simply the following using K-map.



Solution: The function is plotted in Figure 3.2.3. In the two-variable K-map the adjacent cells are grouped for simplification. A cell may be used more than once for making groups of two cells. For the left vertical group, i.e., for the 1st column,  and , thusshould not be considered. The value for this group is  only. Similarly for the lower horizontal group, i.e., 2nd row,  and , thusis not considered. The value for this group is  only. Thus for the two groups from the map 



**Figure 3.2.3:** Example 3.2.1

**3.2.3 Three-variable K-map**

Figure 3.2.4 shows three variables K-map with the value of each cell. There are eight cells in the map. In this map, groups of two or four adjacent cells are considered for simplification. For this map,

* A single cell is identified by three variables.
* A group of two adjacent cells is identified by two variables.
* A group of four adjacent cells is identified by one variable only.

**Example 3.2.2:** Simply the following using K-map.





**Figure 3.2.4:** Example 3.2.2

The K-map for this function is shown in Figure 3.2.4. For simplification the groups should be as large as possible and the number of groups should be minimized and all the 1’s should be considered. The two possible groups consisting of two 1’s are shown in figure. From the figure we can write 

**Example 3.2.3: Find the simplified expression from the following K-map.**



**Figure 3.2.5:** Example 3.2.3

From Figure 3.2.5: (a) , (b), (c)and (d)

**3.2.4 Key points**

* Karnaugh map (K-map) provides a simple method of minimizing Boolean expressions.
* A K-map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables.
* The K-map can also be described as a special arrangement of a truth table.

**2.1.6 Practice Set**

**Multiple Choice Questions**

1. The Karnaugh map (K-map) provides a simple method of \_\_\_\_\_\_\_\_\_\_.
2. minimizing Boolean expressions.
3. maximizing Boolean expressions.
4. minimizing electricity
5. None
6. Which are not the feature of three-variable K-map?
7. A single cell is identified by three variables
8. A group of two adjacent cells is identified by three variables
9. A group of two adjacent cells is identified by three variables
10. A group of four adjacent cells is identified by one variables

**Review Questions**

1. What is K-map?
2. Why is it necessary?
3. List features of three-varible K-map

**Analytical Questions**

1. Obtain the simplest expression form the following K-map



 

****

***Lesson 3.3***

***Shift Registesr and Counters***

**3.3.0 Objectives**

*On completion of this lesson you will know:*

* *Latch and clock signal*
* *Different types of Flip-flops*
* *Shift Register and its configurations*
* *Asynchronous and Synchronous counter*

**3.3.1 Latch and Clock Signal**

**Latch**

The basic memory circuit is known as a latch. A latch is used for storing bits (0 or 1). It has two states: state 1 and state 0. A latch can be realized by NAND gates or NOR gates only.

**NOR latch**: The logic diagram and the truth table for a NOR latch is shown in Figure 3.3.1. The circuit has the following input-output characteristics:

* Input : This is the resting state of the NOR latch. It has no effect on the outputs state. That is, and  will remain in the previous state without any change.
* Input and : This produces the set state of the NOR latch. In this case and .
* Input and : This produces the reset state of the NOR latch. In this case and .
* Input and : In this case. This is not allowed and the condition should not be used.



***Figure 3.3.1: Circuit and truth table of a NOR latch***

**NAND latch**: The logic diagram and the truth table for a NAND latch is shown in Figure 3.3.2. The circuit has the following input-output characteristics:

* Input and : This is the normal resting state of the NAND latch. It has no effect on the output state. and  will remain in the previous state without any change.
* Input and : This produces the set state of the NAND latch. In this case and .
* Input and : This produces the reset state of the NOR latch. In this case and .
* Input : In this case, . This is not allowed and the condition should not be used.



***Figure 3.3.2: Circuit and truth table of a NAND latch***

**Clocked Latch:** It is often required to set or reset the memory cell in synchronism with an enable pulse. Such a circuit is called a clocked latch (Figure 3.2.3) and it changes its output state only if it is enabled, that is when 



***Figure 3.3.3: Clocked latch***

**Clock Signal:**

Digital systems can operate either in asynchronous or synchronous mode. In asynchronous mode, the outputs of the logic circuits can change state at any time when one, or more, input changes, whereas the exact time at which an output can change state is determined by a signal called the clock in synchronous mode. A clock is generally a square wave as shown in Figure 3.3.4. The output can change state only when the clock makes a transition. The transitions are called edges. When a clock changes from 0 to 1, it is called positive edge transition and when a clock changes from 1 to 0, it is called negative edge transition.

There are several types of clocked flip-flops (FFs), these are positive-edge triggered FF and negative edge triggered FF as shown in Figure 3.3.4.



***Figure 3.3.4: Clock signal***

**3.2.3 Flip flops (FFs):**

The popular edge triggered FFs are listed below

* SR FF
* JK FF
* D FF and
* T FF

These can be either positive-edge triggered FF or negative edge triggered FF.

**SR FF:** The block representation and truth table for positive-edge triggered SR FF and negative edge triggered SR FF are shown in Figure 3.3.5 (a) and (b) respectively. The changes in the output states take place at the edge transition of the clock only.



***Figure 3.3.5: SR FF***

**JK FF:** The block representation and truth table for positive-edge triggered JK FF and negative edge triggered JK FF are shown in Figure 3.3.6 (a) and (b) respectively. Unlike SR FF, the outputs of JK FF toggles, i.e., goes to opposite state when both the inputs are high. It is to be noted that the changes in the output states take place at the edge transition of the clock only.



***Figure 3.2.6: JK FF***

**T and D FF:**

T and D FFs can be realized by the JK or SR FFs. The block representations and truth tables for positive-edge triggered T and D FFs are shown in Figure 3.3.7 (a) and (b) respectively.



***Figure 3.3.7: T FF***

**3.3.4 Shift register**

A flip-flop (FF) can store only 1-bit of information (1 or 0). Thus, it is also called a 1-bit register. A number of FFs is required to store binary information. A shift register is a group of FFs put together so that the binary number stored in the FFs can be shifted FF for each clock pulse. Shift registers can have both [parallel](http://en.wikipedia.org/wiki/Parallel_communication) and [serial](http://en.wikipedia.org/wiki/Serial_communication) inputs and outputs as shown in Figure 3.3.8. These are often configured as

* Serial-in, serial-out (SISO)
* Serial-in, parallel-out (SIPO)
* Parallel-in, serial-out (PISO).
* Parallel-in, parallel-out (PISO).

## Beside these, there are other configurations based on the above classification. In a bidirectional register data can be shifted serially from left to right or from right to left using a mode control input. A register is referred to as a universal register if it can be operated in all the four possible modes.

## 

Figure 3.3.8: Shift-register configurations

**3.3.5 Counter**

A Counter is used for counting pulses. A counter is constructed by connecting a number of FFs (Figure 3.2.9). The number of states that a counter can count is called its modulus. For example a decimal or decade counter has modulus 10 and it can count from 0 to 9. When the pulses to be counted are applied to a counter, it goes from one state to the next state and the counter comes back to its starting state after counting  pulses, in the case of modulo *N-*counter.

A FF has two stable states and a group of  FFs has  states. Thus it is possible to make a modulo- counter using  FFs. Basically there are two types of counters, these are:

* Asynchronous counter or ripple counter and
* Synchronous counter

In an asynchronous counter, the FFs are not clocked simultaneously whereas in the synchronous counter all the FFs are clocked simultaneously. Synchronous counters are faster than the asynchronous counters.



***Figure 3.3.9: Block representation of a 4-bit counter***

**Decade counter:** Figure 3.3.10 shows the simplified block diagram of a decade counter together with its one input and four outputs: A, B, C, D. The input pulses are applied to the input and the output indicates the count state. The count table for the counter is shown in Table 3.3.1.



***Figure 3.3.10: A simplified block diagram of a decade counter***

***Table 3.3.1: Count table for a decade counter***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of pulses | Counting states | | | |
| D | C | B | A |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

**3.3.6 Key points**

* The basic memory cell is known as a latch. A latch is used for storing bits (0 or 1).
* A latch can be realized by NAND gates or NOR gates.
* It is often required to set or reset the memory cell in synchronism with an enable pulse. Such a circuit is called a clocked latch.
* Digital systems can operate either in asynchronous or synchronous mode.
* In asynchronous mode, the outputs of the logic circuits can change state at any time when one or more input changes.
* The exact time at which an output can change state is determined by a signal called the clock in synchronous mode.
* A flip-flop (FF) can store only 1-bit of information (1 or 0). It is also called a 1-bit register.
* A number of FFs is required to store binary information.
* A shift register is a cascade of FFs, sharing the same clock.
* A counter is used for counting pulses.
* A FF has two stable states and a group of  FFs has  states.

**3.2.7 Practice Set**

**Multiple Choice Questions**

1. A latch can be realized by \_\_\_\_\_\_\_\_\_\_\_.
   1. NOR
   2. NAND
   3. both
   4. none
2. The outputs of the logic circuits can change state at any time when one or more input changes in \_\_\_\_\_\_\_\_\_\_ mode.
   1. synchronous
   2. asynchronous
   3. synchos
   4. none
3. A flip-flop (FF) can store only \_\_\_\_\_\_\_\_\_\_ of information.
   1. 1 bit
   2. 2 bits
   3. n bits
   4. 2n bits
4. A \_\_\_\_\_\_\_\_\_\_ is a cascade of FFs.
   1. fixed register
   2. variable register
   3. shift register
   4. none
5. Counters are used for counting\_\_\_\_\_\_\_\_\_\_
   1. analog signal
   2. pulses
   3. FF
   4. none

**Review Questions**

1. Mention the main functions of a shift register and a counter.
2. What is a universal counter?
3. What do you understand by modulus of a counter?
4. Distinguish between synchronous and asynchronous counters.

**Analytical Questions**

1. Explain the four possible modes of shift registers and show their operation with the help of illustrations.
2. Draw the block diagram of a decade counter and give its truth table.
3. Explain different type of latches.