



**Birzeit University**

**Faculty of Engineering and Technology**

**Electrical and Computer Engineering**

**Department**

**Manual for**

**Digital Electronics and Computer**

**Organization Lab**

**ENCS 2110**

# Table of Experiments

LABORATORY REGULATIONS AND SAFETY RULES .....	III
Experiment No. 1 - Combinational Logic Circuits.....	1
Experiment No. 2 - Comparators, Adders, and Subtractors.....	21
Experiment No. 3 - Encoders, Decoders, Multiplexers, and Demultiplexers .....	44
Experiment No. 4 - Digital Circuits Implementation using Breadboard.....	63
Experiment No. 5 - Sequential Logic Circuits.....	73
Experiment No. 6 - Sequential Logic Circuits using Breadboard and IC's .....	94
Experiment No. 7 - Constructing Memory Circuits Using Flip–Flops .....	101
Experiment No. 8 - Introduction to QUARTUSII Software .....	111
Experiment No. 9 - A Simple Security System Using FPGA.....	137
Experiment No. 10 - Simple Computer Simulation.....	147
Experiment No. 11 - Arithmetic Elements .....	159

## **LABORATORY REGULATIONS AND SAFETY RULES**

The following Regulations and Safety Rules must be observed in the laboratory:

- 1) It is the duty of all concerned who use any electrical laboratory to take all reasonable steps to safeguard the HEALTH and SAFETY of themselves and all other users and visitors.
- 2) Be sure that all equipment is properly working before using them for laboratory exercises. Any defective equipment must be reported immediately to the Lab. Instructors or Lab. Technical Staff.
- 3) Students are allowed to use only the equipment provided in the experiment manual or equipment used for senior project laboratory.
- 4) Power supply terminals connected to any circuit are only energized with the presence of the Instructor or Lab. Staff.
- 5) Students should keep a safe distance from the circuit breakers, electric circuits or any moving parts during the experiment.
- 6) Avoid any part of your body to be connected to the energized circuit and ground.
- 7) Switch off the equipment and disconnect the power supplies from the circuit before leaving the laboratory.
- 8) Observe cleanliness and proper laboratory housekeeping of the equipment and other related accessories.
- 9) Double check your circuit connections before switching “ON” the power supply.
- 10) Make sure that the last connection to be made in your circuit is the power supply and first thing to be disconnected is also the power supply.
- 11) Equipment should not be removed, transferred to any location without permission from the laboratory staff.
- 12) Software installation in any computer laboratory is not allowed without the permission from the Laboratory Staff.
- 13) Computer games are strictly prohibited in the computer laboratory.
- 14) Students are not allowed to use any equipment without proper orientation and actual hands-on equipment operation.
- 15) Smoking and drinking in the laboratory are not permitted.



**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 1 - Combinational Logic Circuits**

## **1.1 Objectives**

- To become familiar with AND, OR, NOT, NAND, NOR, and OR operations and implementation.
- To construct NOT, AND, OR, and XOR gates using NAND gates.
- To become familiar with the concept of the Truth table.
- To implement the different Boolean functions using NAND gate only.
- To learn techniques for solving logic design problems.
- To become familiar with minimization techniques and using Karnaugh maps.
- To construct an AOI gate with basic gates.

## **1.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab.
- KL-33002 Combinational Logic Circuit Experiment Model (1)

## 1.3 Introduction

Logic gates are digital circuits capable of performing a particular logic function by operating on several binary inputs. Logic gates can be broadly classified as Basic, Universal, and other logic gates.

### 1.3.1 Basic logic gates:

Basic Logic Gates are the fundamental logic gates using which universal logic gates and other logic gates are constructed. These gates are associative and commutative in nature. AND, OR, and NOT are the famous examples of basic logic gates.

#### 1.3.1.1 AND GATE:

The AND gate performs a logical multiplication commonly known as AND function. The output is high when both inputs are high. The output is low level when any one of the inputs is low. Figure 1.1 shows all the information for the AND gate.

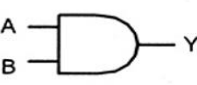
Logic function	Logic symbol	Truth table	Boolean expression															
2-input AND gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	$Y = A \bullet B$
A	B	Y																
0	0	0																
0	1	0																
1	0	0																
1	1	1																

Figure 1.1: The AND gate function, symbol, truth table, and Boolean expression.

#### 1.3.1.2 OR GATE:

An OR gate is a logic gate that performs logical OR operation. The output is high when any one of the inputs is high. The output is low level when both the inputs are low. Figure 1.2 shows all the information for the OR gate.

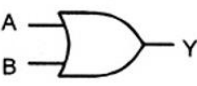
Logic function	Logic symbol	Truth table	Boolean expression															
2-input OR gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$Y = A + B$
A	B	Y																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

Figure 1.2: The function, symbol, truth table, and Boolean expression for OR gate.

### 1.3.1.3 NOT GATE:

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high. Figure 1.3 shows all the information for the NOT gate.

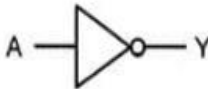
Logic function	Logic symbol	Truth table	Boolean expression						
Inverter (NOT gate)		<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	$Y = \bar{A}$
A	Y								
0	1								
1	0								

Figure 1.3: The function, symbol, truth table, and Boolean expression for NOT gate.

## 1.3.2 Universal logic gates

Universal logic gates are those capable of implementing any Boolean function without requiring any other type of gate. We called these gates Universal gates because Universal gates are not associative in nature, but they are commutative in nature. There are two universal logic gates NAND gate and NOR gates.

### 1.3.2.1 NAND GATE:

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the inputs is low. The output is low level when both inputs are high. Figure 1.4 shows all the information for the NAND gate.

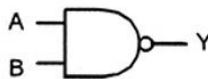
Logic function	Logic symbol	Truth table	Boolean expression															
2-input NAND gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	$Y = \overline{A \cdot B}$
A	B	Y																
0	0	1																
0	1	1																
1	0	1																
1	1	0																

Figure 1.4: The function, symbol, truth table, and Boolean expression for NAND gate.

### 1.3.2.2 NOR GATE:

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high. Figure 1.5 shows all the information for the NOR gate.

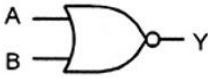
Logic function	Logic symbol	Truth table	Boolean expression															
2-input NOR gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	$Y = \overline{A + B}$
A	B	Y																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

Figure 1.5: The function, symbol, truth table, and Boolean expression for NOR gate.

## 1.3.3 Derived Logic Gates

There are two remaining gates of the primary electronics logic gates: XOR, which stands for Exclusive OR, and XNOR, which stands for Exclusive NOR.

### 1.3.3.1 EX-OR GATE:

The output is high when any one of the inputs is high. The output is low when both the inputs are low, and both the inputs are high. Figure 1.6 shows all the information for the EX-OR gate.


Logic function	Logic symbol	Truth table	Boolean expression															
2-input EX-OR gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</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></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$Y = A \oplus B$
A	B	Y																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Figure 1.6: The function, symbol, truth table, and Boolean expression for X-OR gate.

### 1.3.3.2 EX-NOR GATE:

The X-NOR gate is a contraction of X-OR and NOT. The output is high when the number of ones is even. The output is low when the number of one is odd. Figure 1.7 shows all the information for the X-NOR gate.


Logic function	Logic symbol	Truth table	Boolean expression															
2-input EX-NOR gate		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1	$Y = \overline{A \oplus B}$
A	B	Y																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Figure 1.7: The function, symbol, truth table, and Boolean expression for the X-NOR gate.



## 1.4 Procedure

In section 1.4 we will learn how to build different gates using NAND and NOR gates.

### 1.4.1 The characteristics of NOR gate.

- a) Set module KL-33002 and locate (block a) as shown in Figure 1.8. Connect the +5V of module KL-33002 to the +5V output of the fixed power supply KL-31001. And do the same for ground (GND).

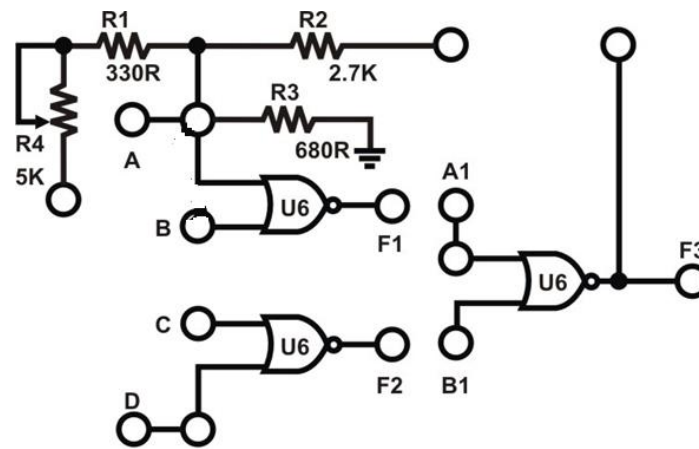


Figure 1.8: KL-33002 NOR Gate Block.

- b) Then select the first gate OR gate in Figure 1.8. Connect inputs A and B with Data switch TTL level in power supply SW0 and SW1. connect output F1 with Logic Indicator (LED) L0 in the power supply. write the result in Table 1.1.

Table 1.1: Data for part 1.4.1 (b).

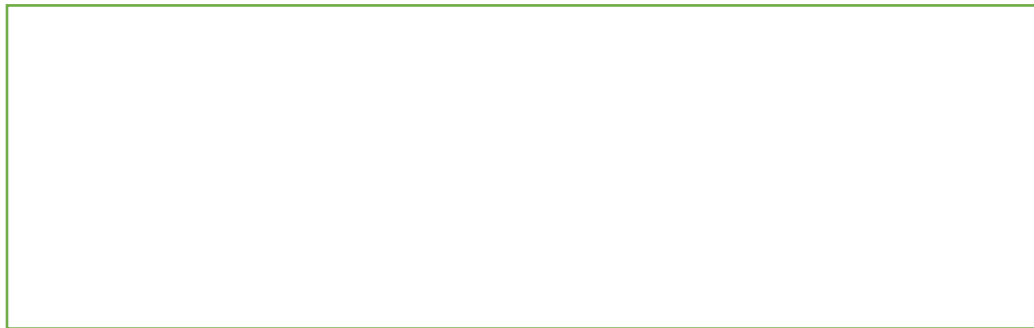
SW0(A)	SW1(B)	L0 (F1)
0	0	
0	1	
1	0	
1	1	

- c) Use the same connection in part b. Set SW0 to “0” and change SW1 to “0”, see the result of the output F1 then change SW1 to “1”, and see the output to make it clear go to Table 1.2 and write the result in the table.

**Table 1.2: Data for part 1.4.1 (c).**

SW0(A)	SW1(B)	L0 (F1)
0	0	
0	1	

- Does the circuit above act as a NOT gate? -----
- Find another way to build a NOT gate using the NOR gate and draw the circuit in the box below. (Hint: return to Figure 1.5 and see the truth table of NOR gate look when the inputs are the same the output will be the opposite of inputs).



- d) Use two OR gates to construct a buffer as shown in Figure 1.9. Insert connection clips between A and B, connect F1 with A1 then connect A1 with B1. Connect input A to SW0 and output F3 to L1. write the result in Table 1.3.

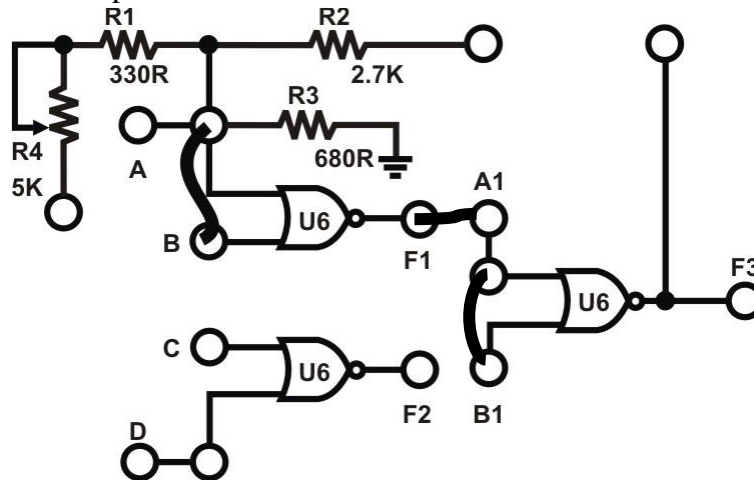
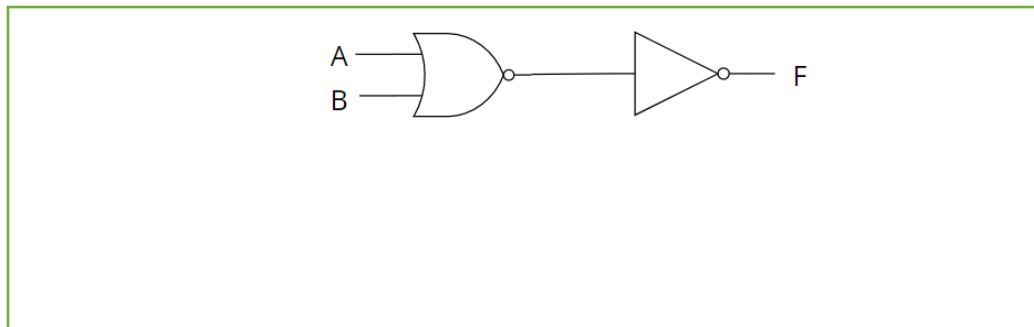


Figure 1.9: build buffer using NOR gate

**Table 1.3: Data for part 1.4.1 (d).**

SW0(A)	L1 (F3)
0	
1	

- Does the circuit above act as a buffer gate? -----
- Build OR gate using NOR gate only gate and draw the circuit in the box below.  
(Hint: we can build OR gate using NOR and NOT gates. we learn how to build a NOT gate using NOR gate).



- e) Use two OR gates to construct an OR gate. Insert connection clips between F1-A1 and A1-B1. Connect inputs A to SW0, B to SW1; and output F3 to L1. Follow the input sequences shown below and record the output states in Table 1.4.



Figure 1.10: build OR using NOR gate.

**Table 1.4: Data for part 1.4.1 (e).**

SW0(A)	SW1(B)	L1 (F3)
0	0	
0	1	
1	0	
1	1	

- f) Know we need to build AND gate using NOR gate only. to learn how to construct it, let's solve the following equation. We know that AND gate equation is  $F=A.B$  if we take the inverse for this equation and then use De morgan law the equation becomes

$$F' = \dots\dots\dots$$

If we take the inverse for the above equation, then the equation becomes:

$$F'' = \dots\dots\dots$$

- g) From the above equation ( $F''$ ) we can see that we can build AND gate using one NOR gate and two NOT gates. Insert connection clips according to Figure 1.11 below. Connect A to SW0; D to SW1; F1 to A1; F2 to B1; F3 to L1. Follow the input sequences given below, and record the output states in Table 1.5.

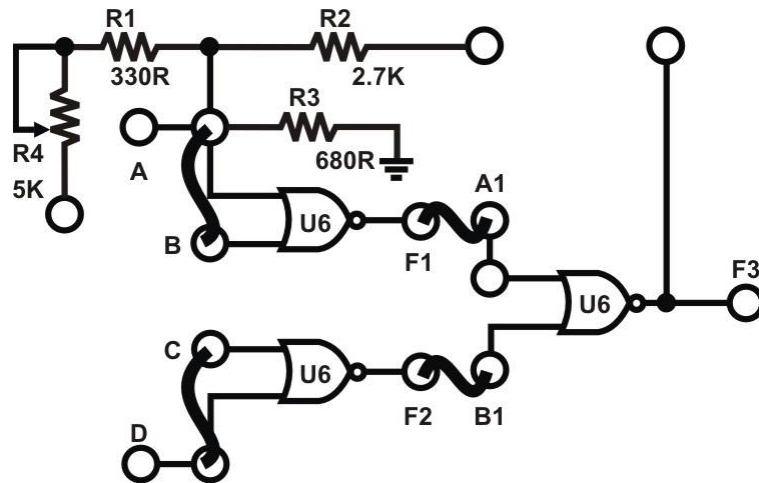


Figure 1.11: build AND gate using NOR gate using kits.

Table 1.5: Data for part 1.4.1 (g).

SW0(A)	SW1(D)	L1 (F3)
0	0	
0	1	
1	0	
1	1	

### 1.4.2 The characteristics of NAND gate.

- a) Set module KL-33002 and locate (block b) gate as shown in Figure 1.12. Connect the +5V of module KL-33002 to the +5V output of the fixed power supply KL-31001 and do the same for the ground (GND).

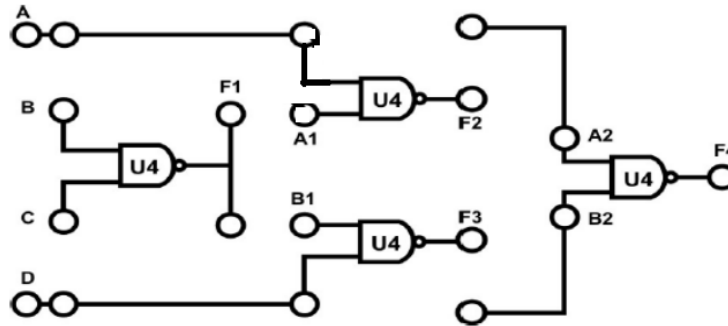


Figure 1.12: KL-33002 NAND Gate Block.

- b) Then select the first gate in Figure 1.12. Connect A and A1 in block a with Data switch SW0 and SW1 TTL level in the power supply sequentially and connect output F2 with Logic Indicator L0 in the power supply. write the result in Table 1.6.

Table 1.6: Data for part 1.4.2 (b).

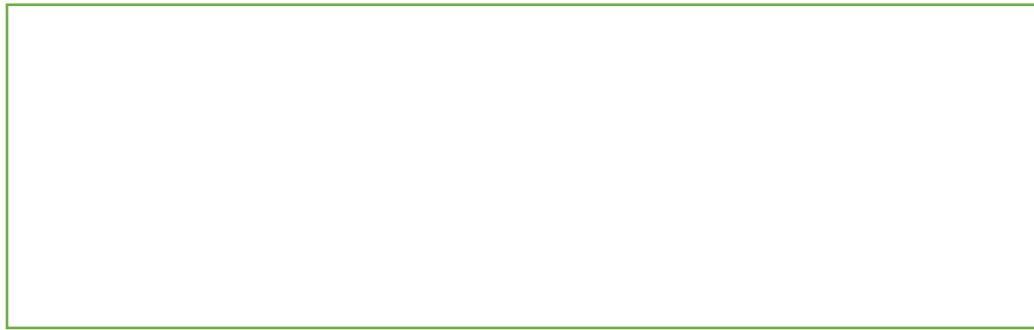
SW0(A)	SW1(A1)	L0 (F2)
0	0	
0	1	
1	0	
1	1	

- c) Use the same connection in part b. Set SW0 to “1” and change SW1 to “0”, see the result of the output F2 then change SW1 to “1”, and see the output to make it clear go to Table 1.7 and write the result in the table.

**Table 1.7: Data for part 1.4.2 (c).**

SW0(A)	SW1(A1)	L0 (F2)
1	0	
1	1	

- Does the circuit above act as a NOT gate? -----
- Find another way to build a NOT gate using a NAND gate and draw the circuit in the box below. (Hint: return to Figure 1.4 and see the truth table of the NAND gate look when the inputs are the same the output will be the opposite of the inputs).



- d) Use two of NAND gates to construct a buffer as shown in Figure 1.13. Insert connection clips between A and A1, connect F2 with A2 then connect A2 with B2. Connect input A to SW0 and output F4 to L1. write the result in Table 1.8.

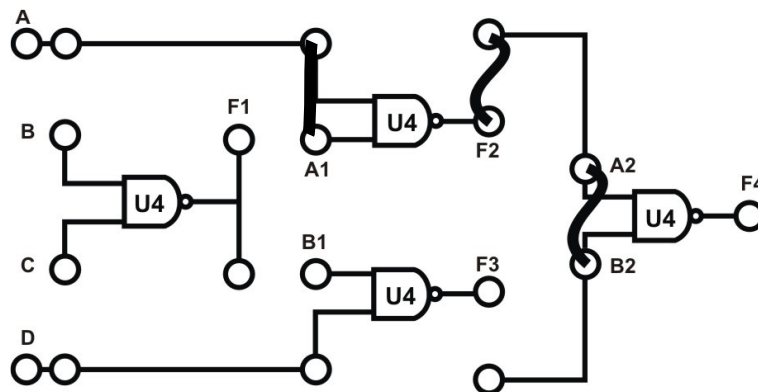
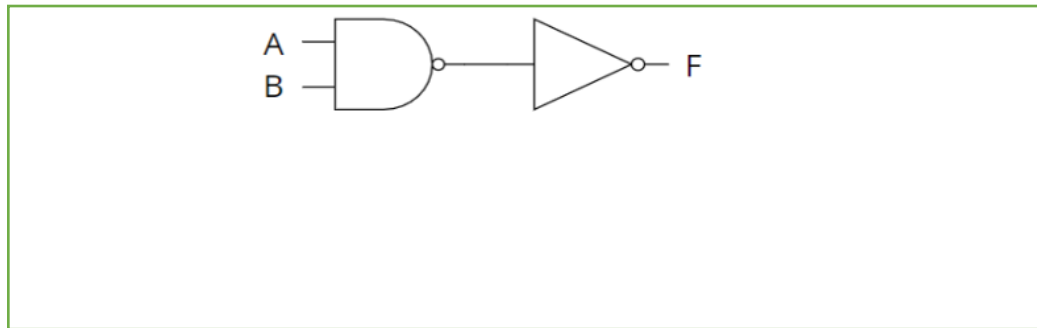


Figure 1.13: build buffer using NAND gate

**Table 1.8: Data for part 1.4.2 (d).**

SW0(A)	L1 (F)
0	
1	

- Does the circuit above act as a buffer gate? -----
- Build AND gate using NAND gate only gate and draw the circuit in the box below. (Hint: we can build AND gate using NAND and NOT gates. we learn how to build NOT gate using NAND gate).



- e) Use two NAND gates to construct an AND gate. Insert connection clips between F2-A2 and A2-B2. Connect inputs A to SW0, A1 to SW1; and output F4 to L1. Follow the input sequences shown below and record the output states in Table 1.9.

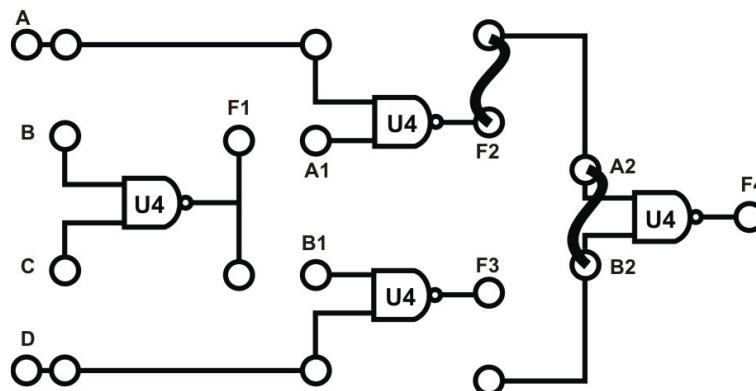


Figure 1.14: build AND using NAND gate.

**Table 1.9: Data for part 1.4.2 (e).**

SW0(A)	SW1(A1)	L1 (F4)
0	0	
0	1	
1	0	
1	1	

- f) Know we need to build OR gate using NAND gate only to learn how to construct it, let's solve the following equation. We know that OR gate equation is  $F=A+B$  if we take the inverse for this equation and then use De Morgan law the equation becomes

$F' = \dots\dots\dots$

If we take the inverse for the above equation, then the equation becomes:

$F'' = \dots\dots\dots$

- g) From the above equation ( $F''$ ) we can see that we can build OR gate using one NAND and two NOT gates. Insert connection clips according to Figure 1.15 below. Connect A to SW0; D to SW1; connect A with A1; and D with B1 and F2 to A2; F3 to B2 then connect the output F4 with L1. Follow the input sequences given below, and record the output states in Table 1.10.

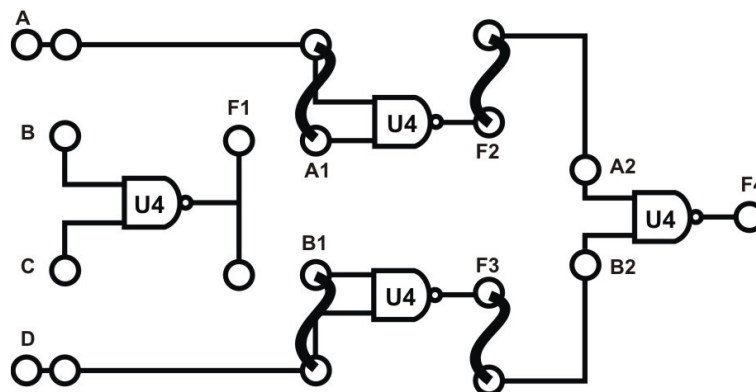


Figure 1.15: build OR gate using NAND gate using kits.



**Table 1.10: Data for part 1.4.2 (g).**

SW0(A)	SW1(D)	L1 (F4)
0	0	
0	1	
1	0	
1	1	

### 1.4.3 XOR Gate Circuit.

there are two ways to build an XOR gate. first using basic gates as shown in Figure 1.16(a) the second way is by using NAND gates only as shown in Figure 1.16(b).

Note: Remember XOR and the equation is  $F = AB' + A'B$

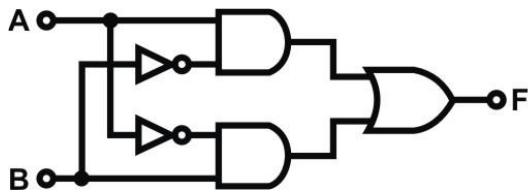


Figure 1.16 (a): Constructed with basic gates.

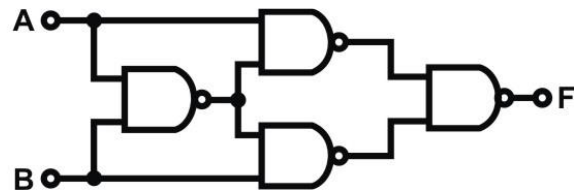


Figure 1.16 (b): Constructed with NAND gates.

### 1.4.3.1 Constructing XOR gate with NAND gate (Module KL-33002 block NAND gates)

Insert connection clips according to Figure 1.17. a) to construct the circuit of Figure 1.17. Connect inputs A to SW1, D to SW2; outputs F1 to L1, F2 to L2; F3 to L3 and F4 to L4. record the output in Table 1.11.

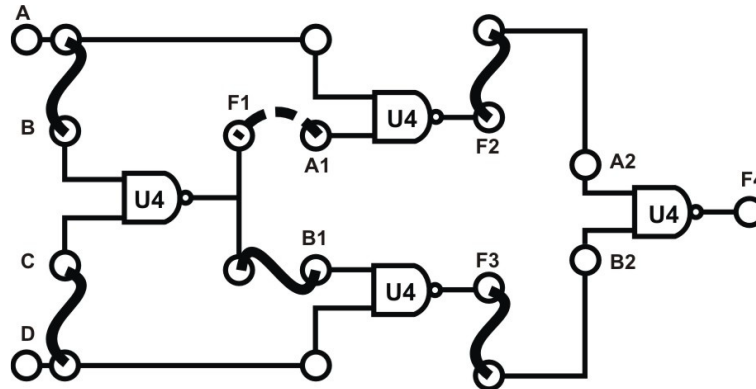


Figure 1.17: Constructed with NAND gates.

Table 1.11: Data for part 1.4.3.1.

SW1(A)	SW2(D)	L1(F1)	L2(F2)	L3(F3)	L4 (F4)
0	0				
0	1				
1	0				
1	1				

- Now determine the Boolean expression for F1, F2, F3 and F4.

F1=.....

F2=.....

F3=.....

F4=.....

- Since  $F = A'B + AB'$ , when  $B=0$ , then  $F=.....$  and the circuit act as ..... When  $B=1$ ,  $F=.....$ , the circuit act as an ..... In other words, the input state of an XOR gate determines whether it will act as a .....or .....

### 1.4.3.2 Constructing XOR gate with Basic Gate (Module KL-33002 block c)

Insert connection clips according to Figure 1.18(a). a) to construct the circuit of Figure 1.18(b). Connect inputs A, B to SW1, SW2; outputs F1, F2, F3 to L1, L2, L3 as shown in Figure 1.18 (a). write the result in Table 1.12.

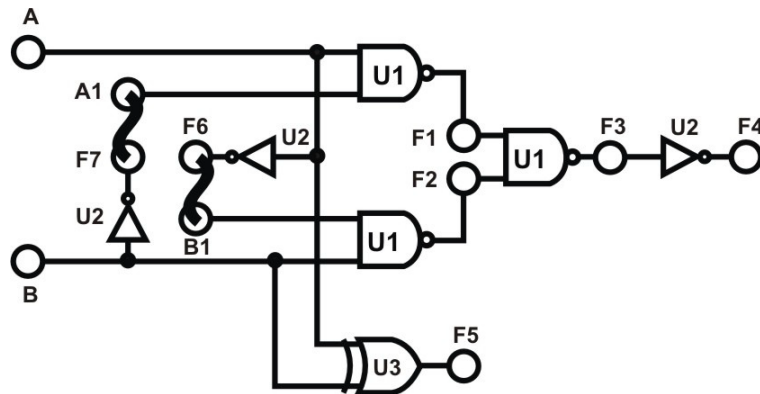


Figure 1.18(a): Constructed with basic gates using kits.

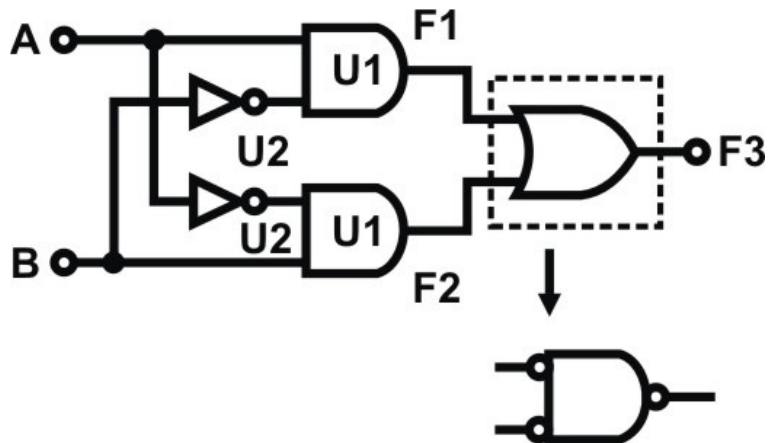


Figure 1.18 (b): equivalent circuit.

Table 1.12: Data for part 1.4.3.2.

SW1(A)	SW2(B)	L1(F1)	L2(F2)	L3(F3)
0	0			
0	1			
1	0			
1	1			

#### 1.4.4 AOI Gate Circuits.

AND-OR-INVERTER (AOI) gates consist of two AND gates, one OR gate, and one INVERTER (NOT) gate. The symbol of an AOI gate is shown in Figure 1.19. The Boolean expression for the output F is  $F = (AB + CD)'$  ..... (1)

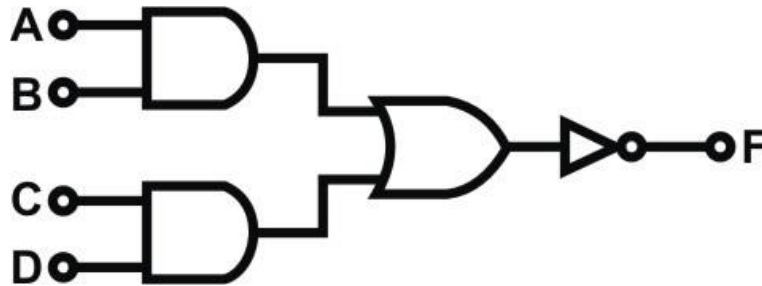


Figure 1.19: AOI gate.

By De Morgan's theorem, Eq. (1) can be converted to:

$$F = (A' + B')(C' + D') \text{ ..... (2)}$$

Eq. (1) is also referred to as the "Sum of Products".

Eq. (2) is also referred to as the "Product of Sums".

The A-Q-I gate is a "Sum of Products" logic combination

- a) Use gates on block c of module KL-33002, shown in Figure 1.20 (a), to construct the A-O-I gate of Figure 1.20 (b). Figure 1.20 (c) is the equivalent A-O-I circuit.

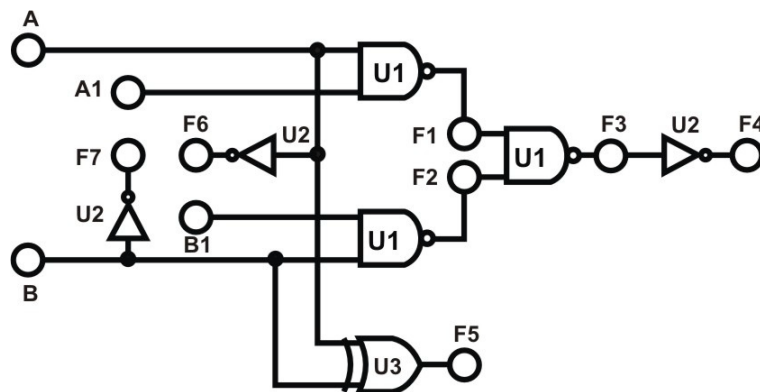


Figure 1.20(a): AOI circuit Wiring diagram

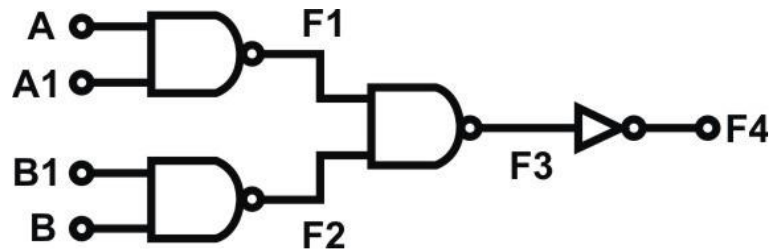


Figure 1.20(b): AOI circuit Actual circuit

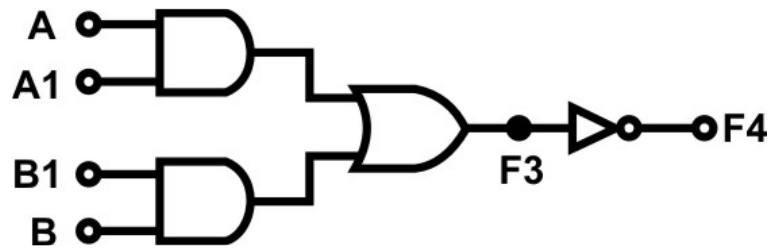


Figure 1.20(c): AOI circuit Equivalent circuit

- b) Connect inputs A, A1, B, B1 to Date Switches SW0, SW1, SW2 and SW3 respectively. Connect outputs F3, and F4 to Logic Indicators L1 and L2 respectively.
- c) Set  $B \times B1$  to "0", follow the input sequences for A, A1 in Table 1.13, and record the outputs

Table 1.13: Data for part 1.4.4 (C).

SW0(A)	SW1(A1)	L1(F3)	L2(F4)
0	0		
0	1		
1	0		
1	1		

- Does F3 act as an AND gate between A and A1?
  - When  $B \times B1$  is "0", does F3 act as an AND gate between A and A1? ( $F3 = A \times A1$ ).
- d) When  $A1 \times A$  is "0", follow the input sequences for B, B1 in Table 1.14 and record the Outputs.

**Table 1.14: Data for part 1.4.4 (d).**

<b>SW2(B)</b>	<b>SW3(B1)</b>	<b>L1(F3)</b>	<b>L2(F4)</b>
<b>0</b>	<b>0</b>		
<b>0</b>	<b>1</b>		
<b>1</b>	<b>0</b>		
<b>1</b>	<b>1</b>		

- Does F3 act as an AND gate between B and B1?
- When  $A \times A1$  is “0”, does F3 act as an AND gate between B and B1? Does F3 equal to  $A \times A1 + B \times B1$ ?

- **Selected Tasks by the instructor.**

## 1.5 Post Lab

- Draw the logic diagram showing the implementation of the following Boolean equation using “NAND” gates
  - a)  $F = AB(CA)$ .
  - b)  $F = (D.A) + (C.B)$
- Draw the logic diagram of the following Boolean equations using NOR gates.
  - a)  $F = (A+B)(CD+A)$
  - b)  $F = (ABC+D)C$
- Implement the OR operation using AND, NOT gate. Draw the logic diagram and write the Boolean equation.
- Implement the AND gate using OR, NOT gate. Draw the logic diagram and write the Boolean equation.
- Prove that the equality operation  $F_1 = AB + A'B'$  is the inverse of exclusive OR operation  $F_2 = AB' + A'B$  (use Demerger's theorem).
- Show how is it possible to reduce Boolean expressions using the Karnaugh map:
  - a)  $F_1 = A'B'C + ABC' + A'BC' + AB'C$
  - b)  $F_2 = A'D + A'C + BD + AB'D'$



## **Faculty of Engineering and Technology**

### **Department of Electrical and Computer Engineering**

#### **ENCS 2110**

### **Digital Electronics and Computer Organization Lab**

#### **Experiment No. 2 - Comparators, Adders, and Subtractors**

#### **2.1 Objectives**

- To understand the construction and operating principle of digital comparators
- To construct comparators with basic gates and ICs
- To implement half- and full-adders using basic logic gates and ICs
- To implement a 4-bit adder unit(s)/ICs to add 4-bit numbers
- To understand the theory of complements
- To construct half- and full-subtractor circuits

#### **2.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab
- KL-33002 Combinational Logic Circuit Experiment Model (1)
- KL-33003 Combinational Logic Circuit Experiment Model (2)
- KL-33004 Combinational Logic Circuit Experiment Model (3)



## 2.3 Pre-Lab

- 1) Prepare all sections and work out all the required designs.
- 2) Build half adder using basic gates.
- 3) Build the above circuit using universal gates.
- 4) Build a full adder using basic gates.
- 5) Build a full adder using a half adder and another gate.
- 6) Build a 4-bit adder using a full adder.
- 7) Build a 4-bit subtractor using basic gates.

## 2.4 Theory

### 2.4.1 Half- and Full-Adder Circuits:

#### 2.4.1.1 Half Adder:

The half adder accepts two binary digits on its inputs and produces two binary digits outputs, a sum bit, and a carry bit. The half-adder is an example of a simple, functional digital circuit built from two logic gates. The half-adder adds to one-bit binary numbers (AB). The output is the sum of the two bits (S) and the carry (C) as shown in Figure 2.1.

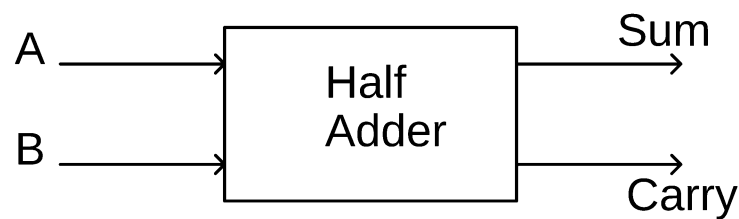


Figure 2.1: Half-Adder Functional Diagram.

Note how the same two inputs are directed to two different gates. The inputs to the XOR gate are also the inputs to the AND gate. The input "wires" to the XOR gate are tied to the input wires of the AND gate; thus, when voltage is applied to the A input of the XOR gate, the A input to the AND gate receives the same voltage. As shown in Figure 2.2.

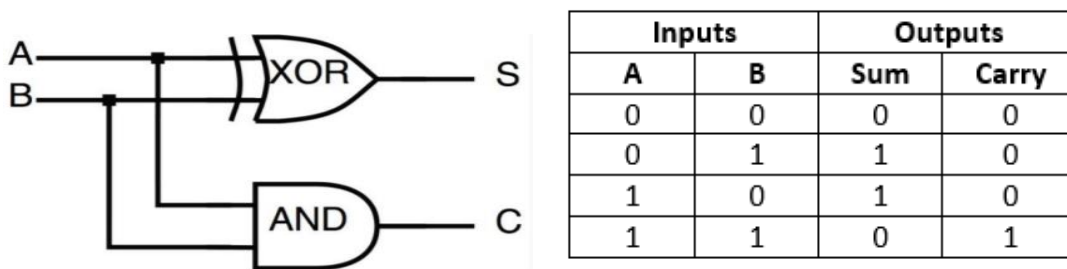


Figure 2.2: Half-Adder circuit and truth table.

### 2.4.1.2 Full Adder:

The full adder accepts two input bits and an input carry and generates a sum output and an output carry. The full-adder circuit adds three one-bit binary numbers ( $C_{in}$ ,  $A$ ,  $B$ ) and outputs two one-bit binary numbers, a sum ( $S$ ) and a carry ( $C_{out}$ ) as shown in Figure 2.3. The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.

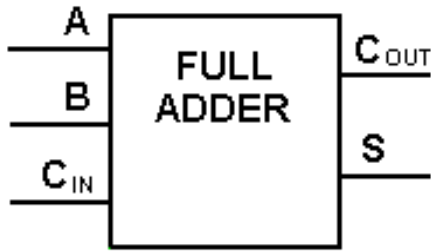


Figure 2.3: Full-Adder Functional Diagram.

In Figure 2.4 (a), if you look closely, you'll see the full adder is simply two half adders joined by an OR. We can implement a full adder circuit with the help of half-adder circuits. The first half adder will be used to add  $A$  and  $B$  to produce a partial Sum. The second half adder logic can be used to add  $C_{in}$  to the Sum produced by the first half adder to get the final  $S$  output. If any of the half-adder logic produces a carry, there will be an output carry. Thus,  $C_{out}$  will be an OR function of the half-adder Carry outputs. We can see the truth table of the full adder in Figure 2.4 (b).

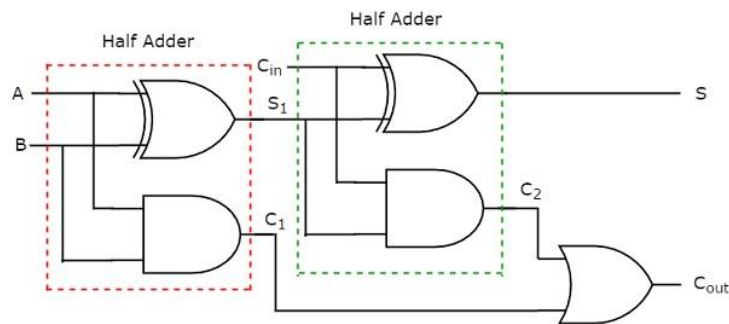


Figure 2.4 (b): Full-Adder Circuit.

Input			Output	
A	B	C <sub>in</sub>	Sum	C <sub>out</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2.4 (b): Full-Adder Truth table.

To perform additions of numbers greater than 2 bits in length, the connection shown in Figure 2.5, or "Parallel Input" should be used to generate sums simultaneously. However, the sum of the next adder will be stable only after the previous adder's carry has stabilized. For example, in Figure 2.5, the sum of FA2 will not be stable unless the carry of FA1 is stable.

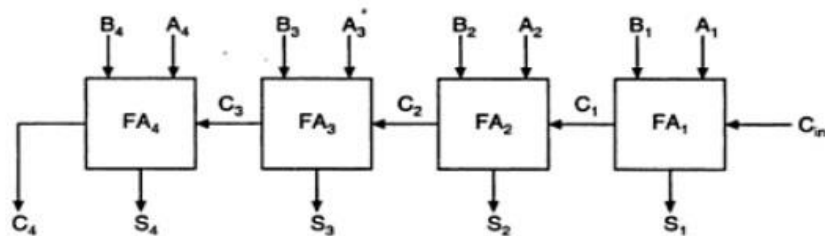


Figure 2.5: 4-bit adder.

When FA1 adds A1 and B1, a sum S1 and a carry C1 is generated. C1 will be added to A2 and B2 by FA2, generating another sum S2 and another carry C2. In the case of Figure 2.5, the sum of the four adders does not stabilize at the same time, delaying the adding process. This delay can be eliminated by using the "Look-Ahead" adder.

**Look-ahead adders** is it is the faster circuit in performing binary addition by using the concepts of Carry Generate and Carry Propagate. A CLA is termed as the successor of a ripple carry adder. A CLA circuit minimizes the propagation delay time through the implementation of complex circuitry.

The operation of the carry lookahead is based on two scenarios:

- Calculate every digit position to know whether that position is propagating a carry bit that comes from its right position.

- Then combine the calculated values to produce the output for every set of digits where the group generates a propagation bit that comes from the right position.

Carry lookahead adders operate by generating two bits called Carry Propagate and Carry Generate which are represented by Cp and Cg. The Cp bit gets propagated to the next stage and the Cg bit is used for generating the output carry bit and this is independent of the input carry bit. The below picture shows the 4-bit carry lookahead adder architecture.

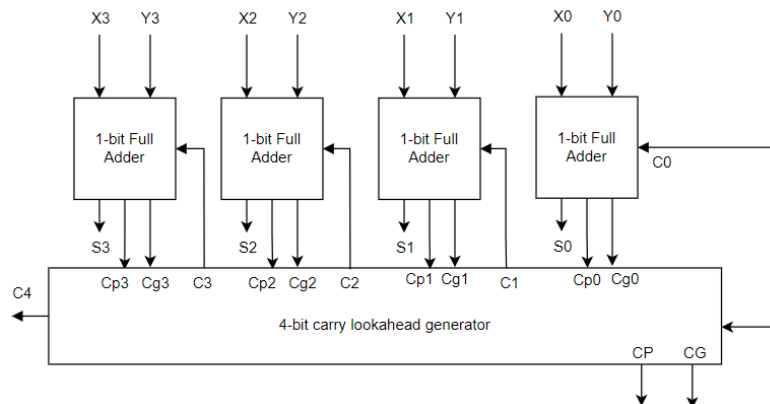


Figure 2.6: 4-bit Carry Lookahead Adder Architecture.

The total number of gate levels in the circuit for carry propagation can be known from the full adder circuit. From input  $C_{in}$  to output  $C_{out}$ , two gates are required which are AND and OR gates. As we are considering a 4-bit circuit, the total number of gate levels will be 8. In the same way, for an  $n$ -bit parallel adder circuit, there is a  $2n$  number of gate levels.

For the construction of the carry-lookahead adder, we need two Boolean expressions which are for carry-lookahead adder formula for carry propagate Cp and carry generate Cg.

$$C_{pi} = X_i \oplus Y_i$$

$$C_{gi} = X_i \cdot Y_i$$

With the above expressions, the sum and carry at the output can be given as:

$$Sum_i = C_{pi} \oplus C_i$$

$$C_{i+1} = C_{gi} + (C_{pi} \cdot C_i)$$

With the above fundamental equations, the boolean expression for carry output at every stage can be known. So

$$C1 = Cg0 + (Cp0 \cdot C0)$$

$$C2 = Cg1 + (Cp1 \cdot C1) = Cg1 + (Cp1 \cdot [Cg0 + (Cp0 \cdot C0)])$$

$$Cg1 + Cp1 \cdot Cg0 + Cp1 \cdot Cp0 \cdot C0$$

$$C3 = Cg2 + (Cp2 \cdot C2)$$

$$Cg2 + (Cp2 \cdot [Cg1 + Cp1 \cdot Cg0 + Cp1 \cdot Cp0 \cdot C0])$$

$$C4 = Cg3 + (Cp3 \cdot C3)$$

$$Cg3 + (Cp3 \cdot Cg2 + (Cp2 \cdot [Cg1 + Cp1 \cdot Cg0 + Cp1 \cdot Cp0 \cdot C0]))$$

**Binary adders can be converted into BCD adders.** Since BCD has 4 bits with the largest number being 9, and the largest 4-bit binary number is equivalent to 15, there is a difference of 6 between the binary and the BCD adder: Under the following conditions 6 must be added when binary adders are used to add BCD codes:

1. When there is any carry.
2. When the sum is larger than 9.

If the order of priority is S<sub>8</sub>, S<sub>4</sub>, S<sub>2</sub>, S<sub>1</sub> and the sum is larger than 9 then S<sub>8</sub> x S<sub>4</sub> + S<sub>8</sub> x S<sub>2</sub>. If any carry is involved, assuming the carry is CY, under this term, 6 must be added: CY + S<sub>8</sub> X S<sub>4</sub> + S<sub>8</sub> X S<sub>2</sub>.

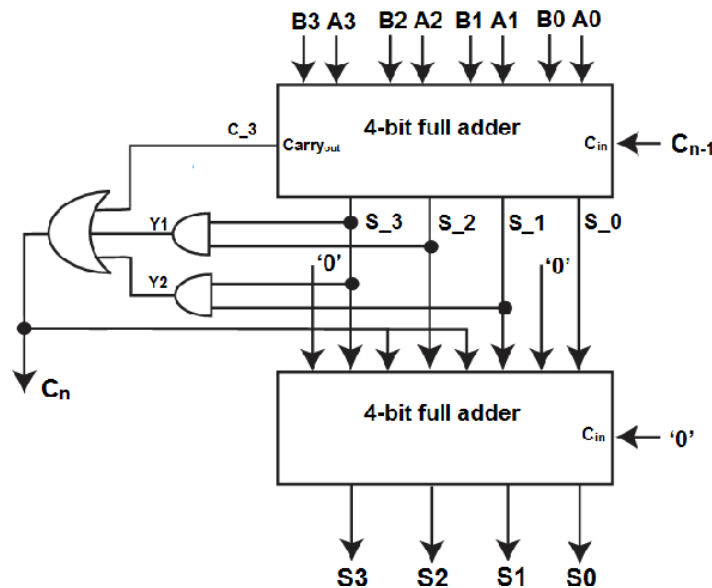


Figure 2.7: BCD adder.

## 2.4.2 Half- and Full-Subtractor Circuits:

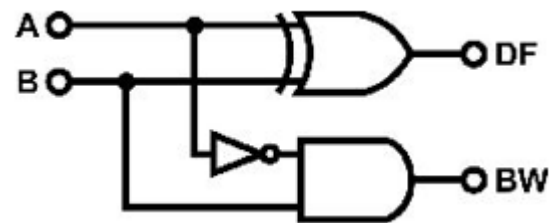
### 2.4.2.1 Half Subtractor:

Binary subtraction is usually performed by using 2's complement. Two steps are required to obtain 2's complement. First, the subtrahend is inverted to 1's complement, i.e., a "1" to a "0" and a "0" to a "1". Secondly, a "1" is added to the least significant bit of the subtrahend in 1's complement.

A half-subtractor performs the task if subtraction 1-bit at a time regardless of whether the minuend is greater or less than the subtrahend. "Borrow" from previous subtraction is not taken into consideration

Minuend	Subtrahend	Difference	Borrow
A	B	DF	BW
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(a) Truth table.



(b): Half-Subtractor circuit.

Figure 2.8: Half-Subtractor.

### 2.4.2.2 Full Subtractor:

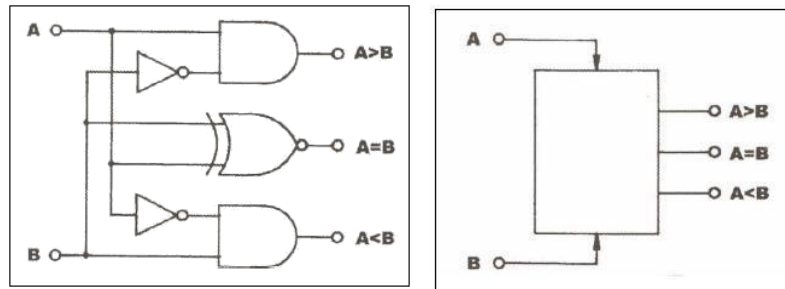
full subtractor is a combinational circuit that performs subtraction of two bits, one is minuend and the other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit has three inputs and two outputs. The three inputs A, B, and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively. Although subtraction is usually achieved by adding the complement of subtrahend to the minuend, it is of academic interest to work out the Truth Table and logical realization of a full subtractor; x is the minuend; y is the subtrahend; z is the input borrow; D is the difference; and B denotes the output borrow. The corresponding maps for logic functions for outputs of the full subtractor namely difference and borrow.





### 2.4.3 Comparator Circuit:

At least two numbers are required to perform any comparison. The simplest form of the comparator has two inputs. If the two inputs are called A and B, then there are three possible outputs:  $A > B$ ,  $A = B$ , and  $A < B$ . Figure 2.11 shows the schematic and symbol diagrams of a simple 1-bit comparator circuit.

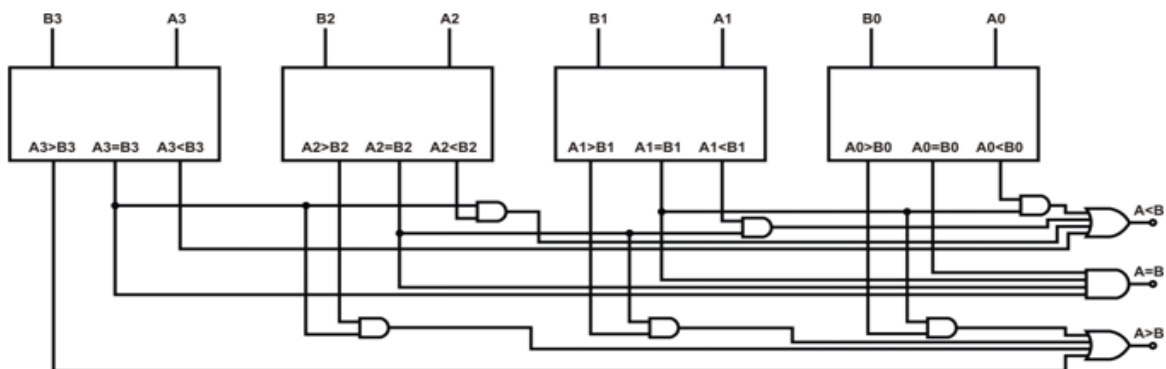


(a) Logic diagram.

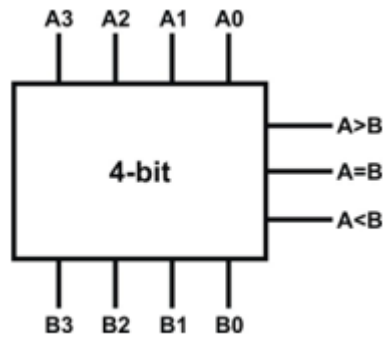
(b): Circuit symbol.

Figure 2.11: Comparator circuit.

In actual applications, 4-bit comparators are used most often. In a 4-bit comparator, each bit represents  $2^0$ ,  $2^1$ ,  $2^2$ , and  $2^3$ . Comparison will start from the most significant bit ( $2^3$ ), if input A is greater than input B at the  $2^3$  bits, the “ $A > B$ ” output will be in the high state. If A and B are equal at the  $2^3$  bits, the comparison will be carried out at the next highest bit ( $2^2$ ). If there is still no result at this bit, the process is repeated at the next bit. At the lowest bit ( $2^0$ ), if the inputs are still equal then the “ $A = B$ ” output will be in the high state. Figure 2.12 shows the schematic and symbol of a 4-bit comparator.



(a): A 4-bit comparator constructed with four 1-bit comparators.



(b): Symbol of a 4-bit comparator

Figure 2.12: Expansion of 1-bit comparators to construct 4-bit comparator.

## 2.5 Procedure

### 2.5.1 Comparator Circuits

#### 2.5.1.1 Constructing Comparator with Basic Logic Gates

- a) Set module KL-33002 block c. Insert connection clips according to Figure 2.13 (a) (connect A1 with F7 and F6 with B1). NAND and XOR gates will be used to construct the 1-bit comparator shown in Figure 2.13 (b).

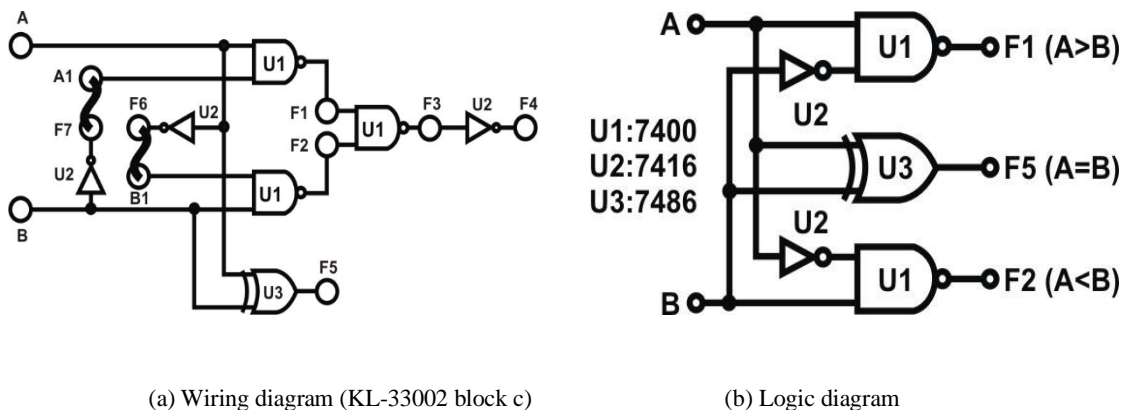


Figure 2.13: 1-bit comparator.

- b) The inputs are triggered by high-state voltage. Connect inputs A and B to Data Switches SW1 and SW2. The outputs are triggered by low-state voltage. Connect outputs F1, F2, F5 to Logic Indicators L1, L2, and L3 respectively.
- c) Follow the input sequences and the result in Table 2.1.

Table 2.1: Data for part 2.5.1.1 (c).

INPUTS			OUTPUTS		
SW2(B)	SW1(A)		F1	F2	F5
0	0	A=B			
0	1	A>B			
1	0	A<B			
1	1	A=B			

#### 2.5.1.2 Constructing Comparator with TTL IC

- a) Block d of module KL-33002 will be used in this section. U6 is a 74LS85 4-bit Comparator IC shown in Figure 2.14.

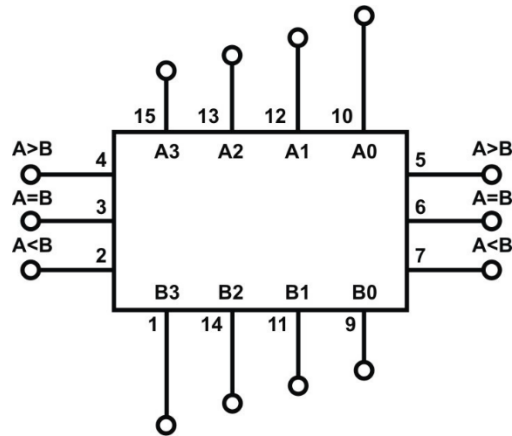


Figure 2.14: 4-bit Comparator IC (KL-33002 block Comparator 2).

- b) Connect input on the left side of the block A<B to SW1 and set it to '0', A=B to SW2 and set it to '1', A>B to SW3 and set it to '0'. The inputs A0~A3 connected to DIP Switch 1.0~1.3 and B0~B3 connected to DIP Switch 2.0~2.3 also connect output on the right side of the block A<B, A=B, and A>B to L1, L2, L3. follow cascading input sequences in Table 2.2 and record the outputs.

Table 2.2: Data for part 2.5.1.2 (c).

				INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	X3	X2	X1	X0	A>B	A=B	A<B
0	0	0	0	0	0	0	0			
0	0	0	0	1	1	1	1			
0	0	0	1	0	0	1	1			
0	0	0	1	0	1	1	0			
0	0	0	1	1	0	0	0			
0	0	1	1	0	1	1	0			
0	1	0	0	1	0	0	0			
0	1	0	0	1	1	1	1			
1	0	0	0	0	1	1	1			
1	0	0	1	1	0	0	1			
1	0	1	0	1	0	1	0			

Compare the results with the function table of **74LS85** (page 2 of the datasheet)<sup>1</sup> as shown in Table 2.3.

<sup>1</sup> <https://www.futurlec.com/74LS/74LS85.shtml>

Table 2.3: function table of 74LS85.

Comparing Inputs				Cascading Inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	A = B	A > B	A < B	A = B
A3 > B3	X	X	X	X	X	X	H	L	L
A3 < B3	X	X	X	X	X	X	L	H	L
A3 = B3	A2 > B2	X	X	X	X	X	H	L	L
A3 = B3	A2 < B2	X	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	X	X	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	H	L	L	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	L	H	H	L

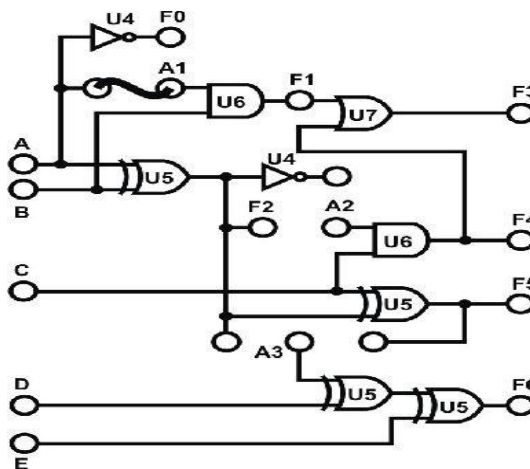
H = HIGH Level, L = LOW Level, X = Don't Care

**Design a three-bit comparator (using the basic comparator) and hand it out to your TA. (Pre-Lab).**

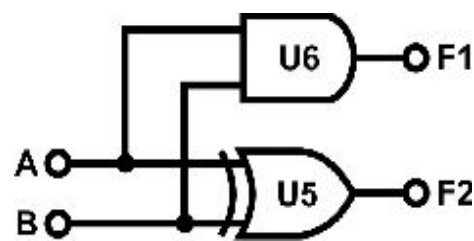
## 2.5.2 Half- and Full-Adder Circuits

**Hand out the design, Boolean function, and truth table of half- and full-adder to your TA. (Pre-Lab).**

- a) Set module KL-33004 and locate block a. Insert connection clips according to Figure 2.15 (a), using XOR and AND gate to assemble the half-adder circuit of Figure 2.15 (b). Connect the +5V of module KL-33004 to the +5V output of the fixed power supply.



(a) Wiring diagram (KL-33004 Half-Adder block)



(b) Half-Adder circuit

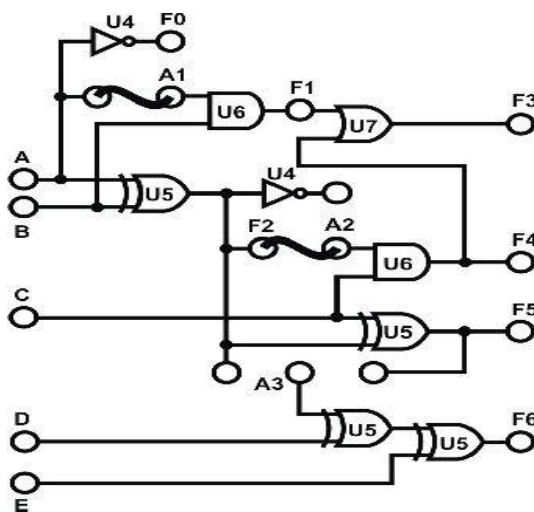
Figure 2.15: Half -Adder.

- b) Connect inputs A, B to Data Switches SW0, SW1 and connect outputs F1, F2 to logic indicators L1 and L2. Follow the input sequences for A and B in Table 2.4 and record the output states.

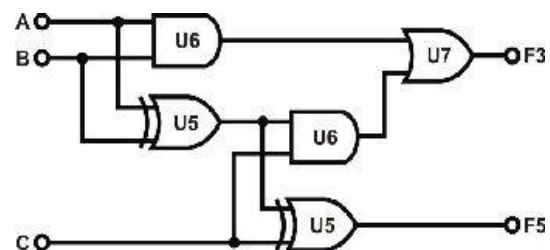
**Table 2.4: Data for part 2.5.2 (b).**

INPUTS		OUTPUTS	
SW1 (B)	SW0 (A)	F1 (CARRY)	F2 (SUM)
0	0		
0	1		
1	0		
1	1		

- c) Reassemble the circuit according to Figure 2.16 (a) to construct the full-adder circuit shown in Figure 2.16 (b).
- d) Connect A, B, C to SW1, SW2 and SW3. A and B are augends while C is the previous carry. Connect F3 to L1, F5 to L2. Follow the input sequences in Table 2.5 and record the output states.



(a) Wiring diagram (KL-33004 Full-Adder block) .



(b) Full-Adder circuit.

Figure 2.16 Full adder

Table 2.5: Data for part 2.5.2 (d)

OUTPUTS			OUTPUTS	
SW3 (C)	SW2 (B)	SW1 (A)	F3 (CARRY)	F5 (SUM)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

### 2.5.3 Half- and Full Subtractor Circuits

**Hand out, Design, Boolean function, and truth table of half- and full-subtractor to your TA. (Pre Lab).**

- Set module KL-33004 and locate block a. Insert connection clips according to Figure 2.17.
- Connect inputs A~C to Data Switches SW0~SW2; Outputs F2 to Logic Indicator L1; F1 to L2; F3 to L3; and F5 to L4. When C=0 the circuit is a half-subtractor. F1 is the borrow output; F2 is the difference. F5=F2; F4=0; F3=F1. When C=1 the circuit is a full-subtractor. F3 is the borrow output and F5 is the difference output.

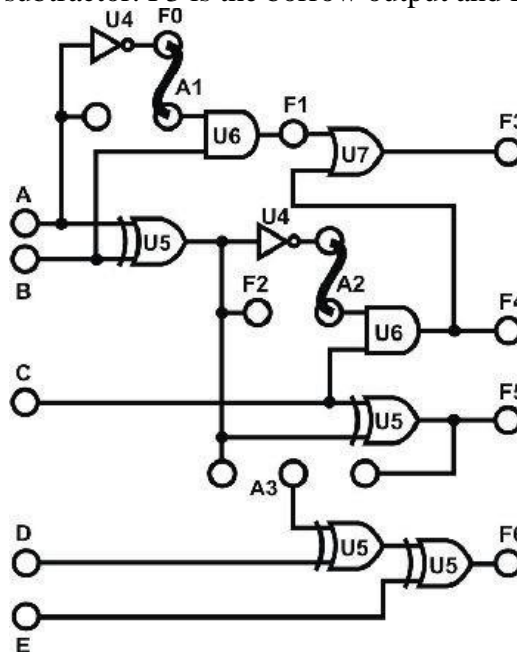


Figure 2.17: Wiring diagram (Half-subtractor).

- c) Follow the input sequences in Table 2.6 and record output states.

**Table 2.6: Data for part 2.5.3 (c)**

	INPUTS			OUTPUTS			
	C	A	B	F1	F2	F3	F5
Half-subtractor Half-adder	0	0	1				
	0	0	0				
	0	1	1				
	0	1	0				
Full-subtractor Full-adder	1	0	0				
	1	0	1				
	1	1	0				
	1	1	1				

## 2.5.4 Constructing 4-Bit Full-Adder with IC

- a) U5 on block b of module KL-33004 is used as a 4-bit adder. Connect input Y5 to SW0, so the XOR gates, which are connected to Y0~Y3, will act as buffers.

Connect input X0~X3 (addends), and Y0~Y3 (augends) to DIP switches DIP2.0~2.3 and DIP1.0~1.3 respectively as shown in Figure 2.21. Connect F1, F8,F9,F10 and F11 to L1~L5. Follow input sequences in Table 2.7 and set SW0 to “0”; record F1 and  $\Sigma$  in binary numbers.

$X = X_3X_2X_1X_0$

$Y = Y_3Y_2Y_1Y_0$

$\Sigma = \Sigma_3\Sigma_2\Sigma_1\Sigma_0$

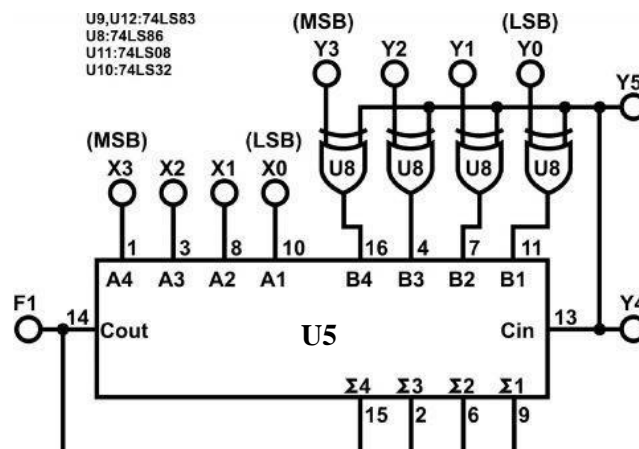


Figure 2.18: Wiring Diagram (KL-33004 4-bit block b).



Table 2.7: Data for part 2.5.4 (a)

INPUTS								OUTPUTS				
Y3	Y2	Y1	Y0	X3	X2	X1	X0	$\Sigma 4$	$\Sigma 3$	$\Sigma 2$	$\Sigma 1$	F1(CARRY)
0	0	0	0	0	0	0	0					
0	0	0	0	0	0	0	1					
0	0	0	0	0	1	1	0					
0	0	0	0	1	0	0	1					
0	0	0	0	1	1	1	1					
0	0	0	1	0	0	1	1					
0	0	0	1	0	1	1	0					
0	0	0	1	1	0	0	0					
0	0	1	1	0	1	1	0					
0	1	0	0	1	0	0	0					
0	1	0	0	1	1	1	1					
1	0	0	0	0	1	1	1					
1	0	0	1	1	0	0	1					
1	0	1	0	1	0	1	1					

## 2.5.5 Constructing 4-Bit Full-Subtractor with IC

- a) Use Module KL-33004 block b (Figure 2.22). Connect inputs X3~X0 (minuend) to DIP Switch 1.3~1.0; Y3~Y0 (subtrahend) to DIP 2.3~2.0; Y5 to SW0. Connect outputs F1 to L4; F11~F8 to L3~L0. To execute the subtract operation, set SW0 to “1” (or Cin of U5=1). Follow the input sequences below and record the output states in Table 2.8.

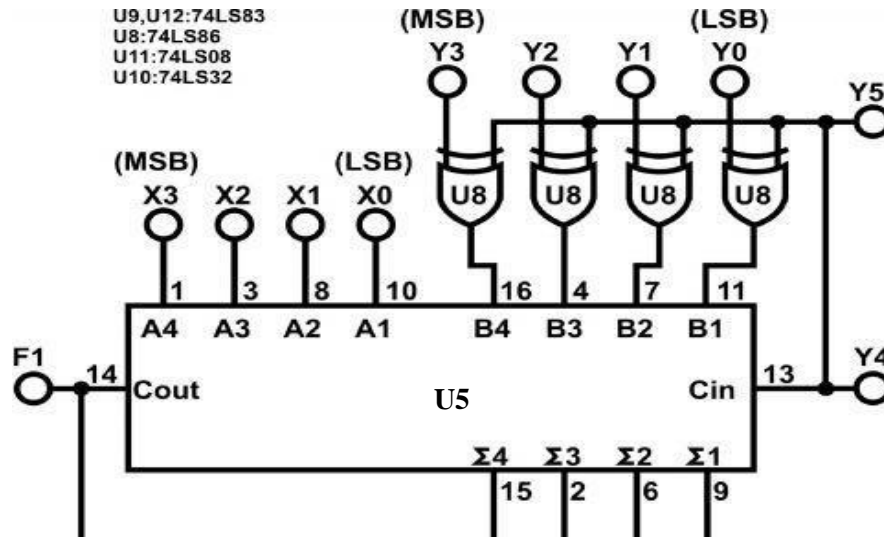


Figure 2.19: Wiring Diagram for full subtractor (KL-33004 4-bit block b)

Table 2.8: Data for part 2.5.5 (a)

INPUTS								OUTPUTS				
X3	X2	X1	X0	Y3	Y2	Y1	Y0	F11	F10	F9	F8	F1(Carry)
0	1	0	0	0	1	0	0					
0	1	0	0	0	0	1	1					
1	0	0	0	0	0	1	1					
1	0	0	0	0	0	0	1					
1	0	0	1	1	0	0	0					
1	0	0	1	0	1	1	1					
1	0	1	0	0	1	1	0					
1	0	1	0	0	1	0	1					
1	0	1	1	1	0	1	0					
1	1	1	1	1	0	1	0					

## 2.5.6 Constructing BCD Adder

- a) The circuit shown in Figure 2.20 will act as a BCD adder.

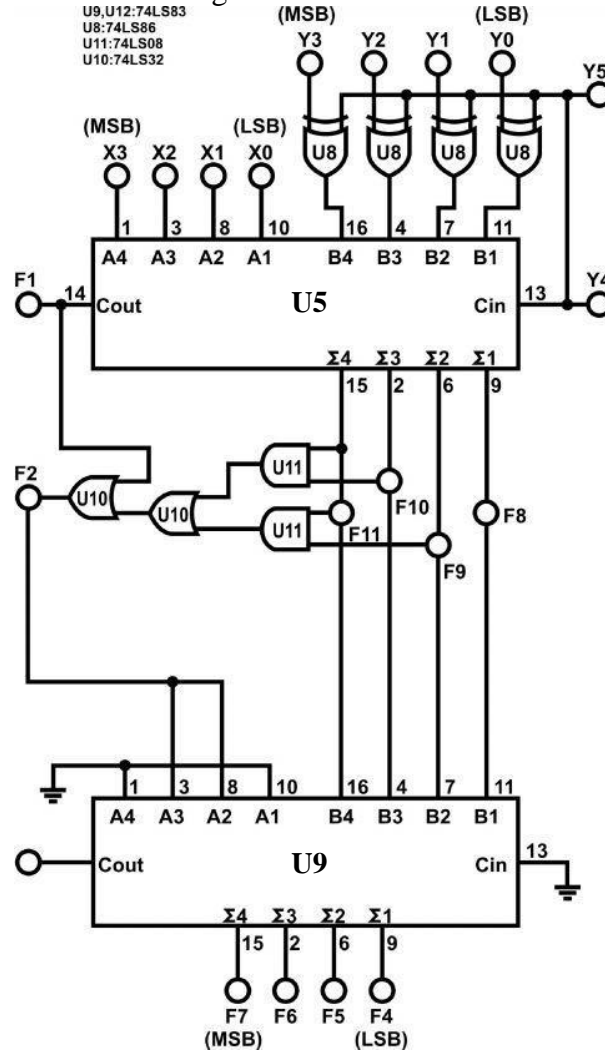


Figure 2.20: Wiring Diagram (KL-33004 BCD Adder).

- b) Connect inputs X0~X3 to DIP 1.0~1.3; Y0~Y3 to DIP 2.0~2.3; Y5 to "0".  
 U5 and U9 are 74LS83 look-ahead 4-bit BCD adders, connect outputs F8~F11 to the inputs of the 7-Segment display SEG-1. Connect F1, F2 to Logic Indicators L4 and L5. Connect outputs F4~F7 of U9 to another 7-Segment display SEG-3 and F3 to L10.
- c) F8~F11 are the sum of X0~X3 added to Y0~Y3 while F1 is the carry. Follow the input sequences for X0~X3 and Y0~Y3 in Table 2.9 and record the output states.

Table 2.9: Data for part 2.5.6 (c)

INPUTS								OUTPUTS (U5)					LAST (U9)					
X3	X2	X1	X0	Y3	Y2	Y1	Y0	F11	F10	F9	F8	F1(Carry)	F2(larger than 9)	F7	F6	F5	F4	F3(Carry)
0	0	0	0	0	0	0	0											
0	0	0	1	0	0	1	1											
0	0	1	1	0	1	0	0											
0	0	1	0	0	0	1	0											
0	0	1	0	1	0	0	0											
0	0	1	1	0	1	1	0											
0	1	0	0	0	0	1	0											
0	1	0	0	0	1	0	1											
0	1	0	0	0	1	1	0											
0	1	0	1	0	1	1	0											
0	1	1	0	0	1	1	1											
0	1	1	1	1	0	0	0											
0	1	1	1	1	0	0	1											
1	0	0	0	1	0	0	1											
1	0	0	1	1	0	0	1											
1	0	1	0	1	0	1	0											
1	0	1	0	1	0	1	1											
1	0	1	0	1	1	0	0											
1	0	1	1	1	1	1	0											
1	1	1	1	1	1	1	1											

### 2.5.7 High-Speed Adder Carry Generator Circuit

- a) 74182 on block a KL-33003 is used to construct a carry generator circuit shown in Figure 2.21.

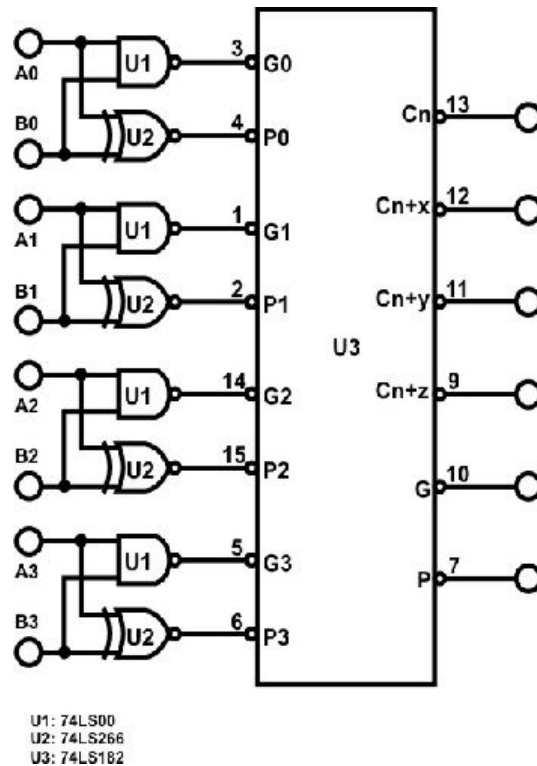


Figure 2.21: Carry generator circuit.

- b) Connect inputs A0~A3 (addends) to DIP Switches 1.0~1.3; B0~B3 (augends) to DIP2.0~2.3, connect Cn to SW0, and set SW0 to “0”. Follow the input sequences in Table 2.10 and record output states.

Table 2.10: Data for part 2.5.7 (b)

INPUTS								OUTPUTS				
B3	B2	B1	B0	A3	A2	A1	A0	Cn+x	Cn+y	Cn+z	$\overline{G}$	$\overline{P}$
0	0	0	1	0	0	0	1					
0	0	1	0	0	0	1	0					
0	0	0	0	0	0	1	0					
0	0	1	1	0	0	1	1					
1	0	1	0	1	0	0	0					
1	1	1	0	1	1	1	1					
1	1	1	1	1	1	0	1					
0	1	1	1	0	1	1	0					
1	0	0	1	0	1	0	1					

- Compare the results with the truth table of 74LS182 (datasheet).
- **Selected Tasks by the instructor.**

## 2.6 Post Lab

1. Design an 8-bit BCD adder.
2. Design an 8-bit comparator using 2 of the 4-bit comparator.
3. A 4-input, 3-output circuit that compares 2-bit unsigned numbers and outputs a '1' on one of three output lines according to whether the first number is greater than, equal to, or less than the other number. You can only use two 4×1 multiplexers.



**Faculty of Engineering and Technology**

**Department of Electrical and Computer Engineering**

**ENCS 2110**

**Digital Electronics and Computer Organization Lab**

**Experiment No. 3 - Encoders, Decoders, Multiplexers, and Demultiplexers**

### **3.1 Objectives**

- To understand the operating principles of Encoders/Decoders
- To understand the operating principles of Multiplexers/Demultiplexers
- To construct Encoders and Decoders using basic gates and ICs
- To construct Multiplexers and Demultiplexers using basic gates and ICs

### **3.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab
- KL-33004 Combinational Logic Circuit Experiment Model (3)
- KL-33005 Combinational Logic Circuit Experiment Model (4)
- KL-33006 Combinational Logic Circuit Experiment Model (5)

### 3.3 Pre-Lab

- 1) Design a circuit which uses an SN74151 to implement a sum-of-products expression, as follows:

- a) Convert the following expression into summation form (i.e.,  $F(A, B, C) = \sum (...)$ ):

$$Y = f(A, B, C) = \underline{A}\underline{B} + \underline{B}\underline{C}$$

- b) Sketch on Figure 3.1 the input connections necessary to implement the function in part (a). Observe that the inputs are connected to 0 or 1 depending on the value of the function for that min term.

**Important:** Please note that this way we can implement a 3-input function (A, B, C) using an 8-to-1 MUX. Later we will see how to implement a 4-input function (A, B, C, D) using an 8-to-1 MUX. For that we will need to inspect the additional input (say D) with the corresponding function value. The possible inputs to the MUX are 0, 1, D, D'.

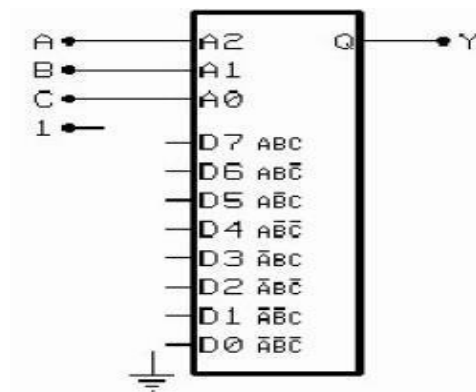


Figure 3.1: Half-Adder Functional Diagram.

- 2) Design a circuit which uses an SN74138 Demultiplexer to implement a sum- of-products expression, as follows:

- a) Convert the following expression into summation (Sum of Products –SOP-) form (i.e.  $F(A,B,C)=\sum(...)$ ):

$$Y = f(A, B, C) = \bar{A}BC + B\bar{C}$$



## 3.4 Theory

### 3.4.1 Decoder

The combinational circuit that changes the binary information into  $2^N$  output lines is known as Decoders. The binary information is passed in the form of  $N$  input lines. The output lines define the  $2^N$ -bit code for the binary information. In simple words, the Decoder performs the reverse operation of the Encoder. At a time, only one input line is activated for simplicity. The produced  $2^N$ -bit output code is equivalent to the binary information. The outputs of the decoder are nothing, but the min terms of ' $N$ ' input variables lines as shown in Figure 3.2.

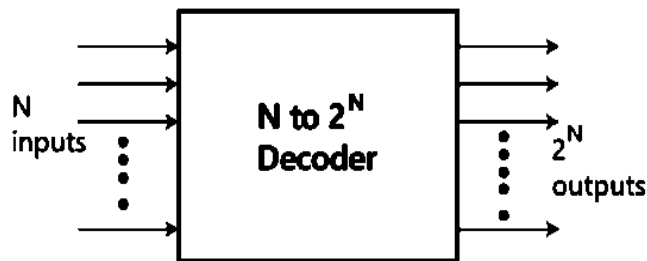


Figure 3.2: General Decoder block diagram.

Example show 2 to 4 Decoder. Let 2 to 4 Decoder has two inputs  $A_1$  &  $A_0$  and four outputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$ . The block diagram of 2 to 4 decoder is shown in the following figure.

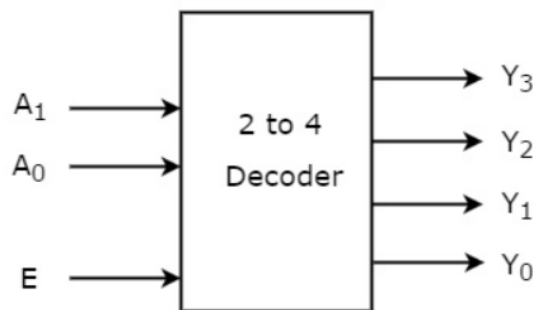


Figure 3.3: 2 to 4 Decoder with enable.

One of these four outputs will be '1' for each combination of inputs when enable,  $E$  is '1'. The Truth table of 2 to 4 decoder is shown below.

**Table 3.1: Truth table of 2 to 4 decoder with enable**

Enable	Inputs		Outputs			
E	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The circuit diagram of 2 to 4 decoder is shown in the following figure.

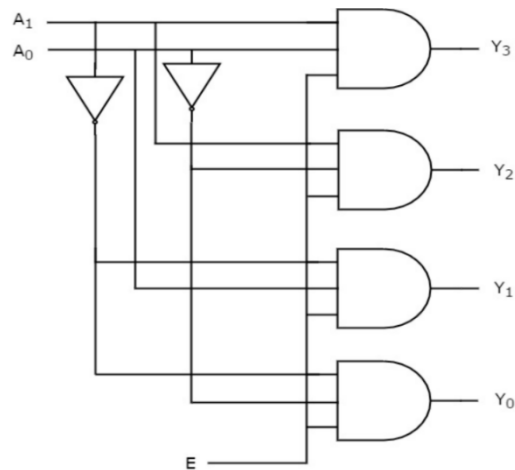


Figure 3.4: 2 to 4 Decoder circuit.

### 3.4.2 Encoder

An Encoder is a combinational circuit that performs the reverse operation of a Decoder. It has a maximum of  $2^N$  input lines and 'N' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes  $2^N$  input lines with 'n' bits. It is optional to represent the enable signal in encoders.

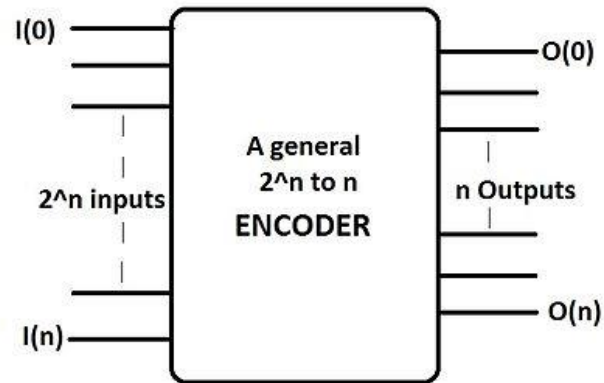


Figure 3.5: General encoder block diagram.

Example show 4 to 2 Encoder. Let 4 to 2 Encoder has four inputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$  and two outputs  $A_1$  &  $A_0$ . The block diagram of 4 to 2 Encoder is shown in the following figure.

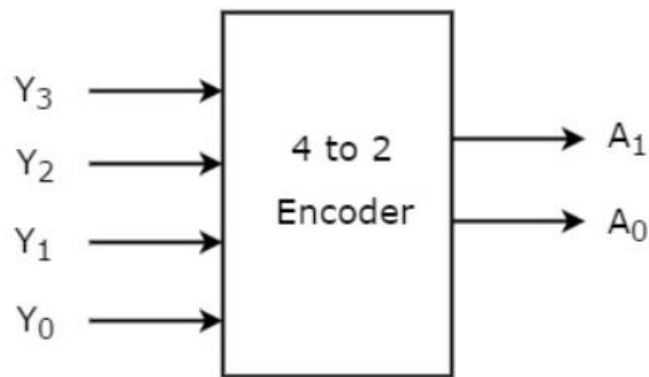


Figure 3.6: 4 to 2 Encoder.

At any time, only one of these 4 inputs can be '1' to get the respective binary code at the output. The Truth table of 4 to 2 encoder is shown below.

**Table 3.2: Truth table of 4 to 2 encoder**

Outputs				Inputs	
$Y_3$	$Y_2$	$Y_1$	$Y_0$	$A_1$	$A_0$
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

We can implement the above two Boolean functions by using two input OR gates. The circuit diagram of 4 to 2 encoder is shown in the following figure.

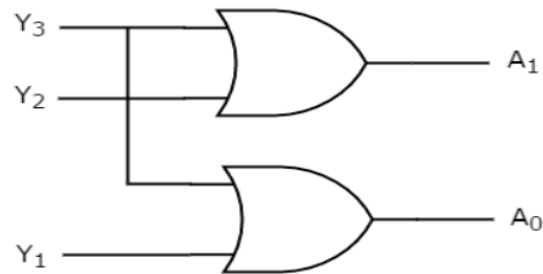
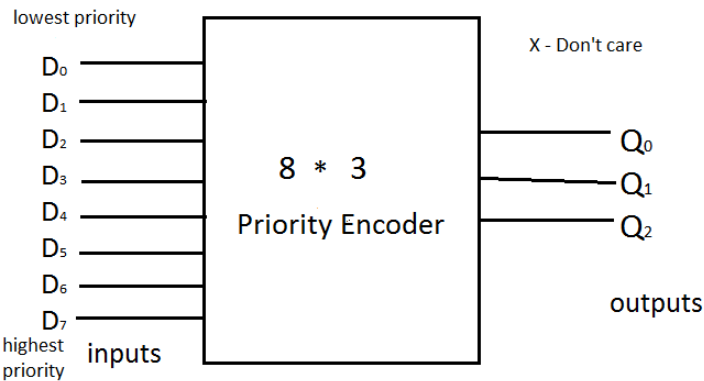


Figure 3.7: 2 to 4 Decoder circuit.

**The Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The priority encoders output corresponds to the currently active input which has the highest priority. So, when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.



(a) 8 to 3 priority Encoder block diagram.

Digital Inputs								Binary Outputs		
$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$Q_2$	$Q_1$	$Q_0$
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

(b): 8 to 3 priority Encoder truth table.

Figure 3.8: 8 to 3 priority Encoder.

### 3.4.3 Multiplexer

A multiplexer is a combinational circuit that has maximum of  $2^n$  data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of the selection lines. Since there are 'n' selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as Mux.

Example show 4 to 1 Mux. 4x1 Multiplexer has four data inputs  $I_3$ ,  $I_2$ ,  $I_1$  &  $I_0$ , two selection lines  $s_1$  &  $s_0$  and one output  $Y$ . The block diagram of 4x1 Multiplexer is shown in the following figure.

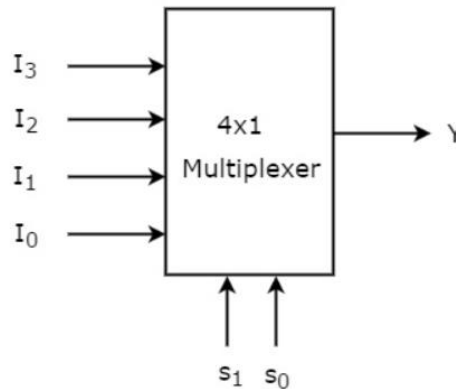


Figure 3.9: 4 to 1 Mux block diagram.

One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. Truth table of 4x1 Multiplexer is shown below.

**Table 3.3: Truth table of 4 to 1 Multiplexer**

Selections		Outputs
S1	S2	Y
0	0	$I_0$
0	1	$I_1$
0	0	$I_2$
1	1	$I_3$

We can implement this Boolean function using Inverters, AND gates & OR gate. The circuit diagram of 4x1 multiplexer is shown in the following figure.

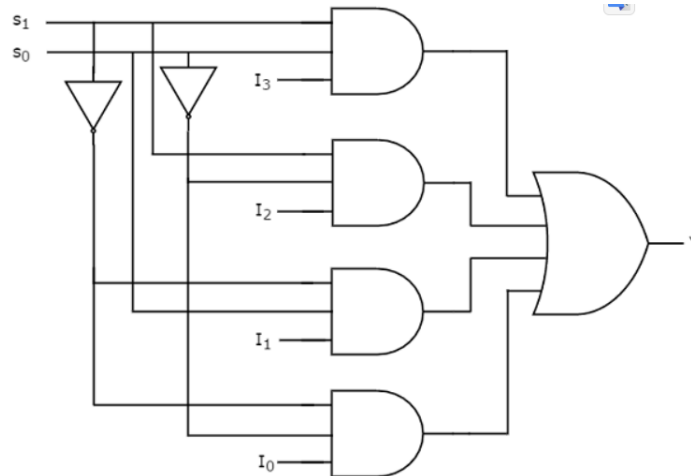


Figure 3.10: 4 to 1 multiplexer circuit.

### 3.4.4 De-Multiplexer

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of  $2^n$  outputs. The input will be connected to one of these outputs based on the values of selection lines. Since there are 'n' selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as De-Mux.

Example show 1x4 De-Multiplexer. 1x4 De-Multiplexer has one input I, two selection lines, s1 & s0 and four outputs Y3, Y2, Y1 & Y0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.

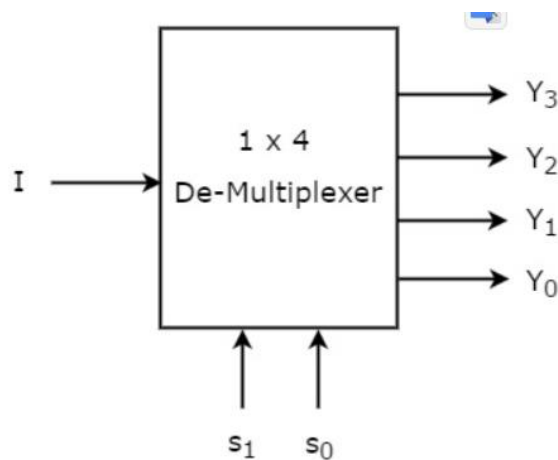


Figure 3.9: 1 to 4 De-Mux block diagram.

The single input 'I' will be connected to one of the four outputs, Y3 to Y0 based on the values of selection lines s1 & s0. The Truth table of 1x4 De-Multiplexer is shown below.

**Table 3.4: Truth table of 1 to 4 De-Multiplexer**

Selections		Outputs			
S1	S2	Y3	Y2	Y1	Y0
0	0	0	0	0	I
0	1	0	0	I	0
0	0	0	I	0	0
1	1	I	0	0	0

We can implement these Boolean functions using Inverters & 3-input AND gates. The circuit diagram of 1x4 De-Multiplexer is shown in the following figure.

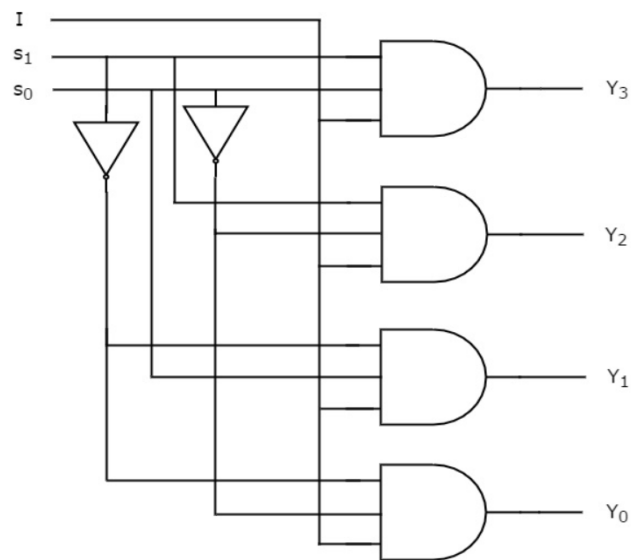


Figure 3.10:1 to 4 De-multiplexer circuit.

## 3.5 Procedure

### 3.5.1 Constructing a 4-to-2 Encoder with Basic Gates

Constructing a 4-to-2 Encoder with Basic Gates (Module KL-33005 block a).

- a) Insert connection clips according to Figure 3.11.

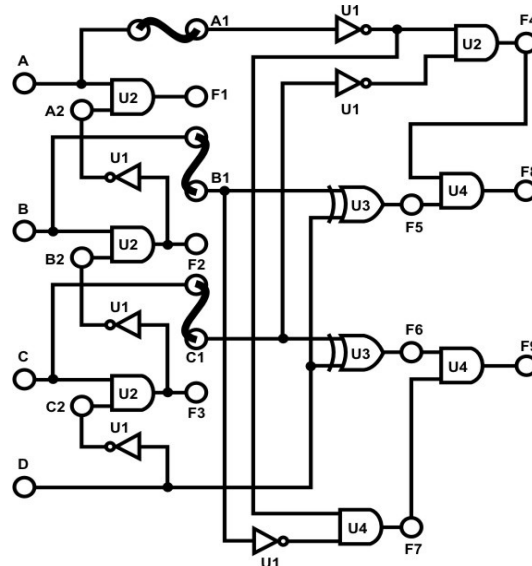


Figure 3.11: wiring diagram of 4-to-2-line Encoder.

- b) Connect the +5V of module KL-33005 to the +5V output of the fixed power supply section of KL-31001.
- c) Connect inputs A~D to Data Switches SW0~SW3 respectively; outputs F8 and F9 to Logic Indicator L0 and L1.
- d) Follow the input sequences for D, C, B, and A; in Table 5, record the output states.

Table 3.5: Data for part 3.5.1 (d)

Inputs				Outputs	
D	C	B	A	F9	F8
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		



1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

### 3.5.2 Constructing 9-to-4-Line Encoder with TTL IC

- a) The 74147 (U1) on block a of module KL-33006 is used in this experiment section. Connect the +5V of module KL-33006 to the +5V output of the fixed power supply. Note that the IC uses only 9 inputs.

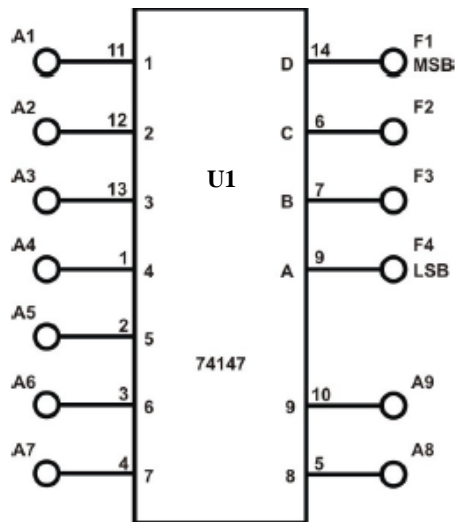


Figure 3.12: 74147 BCD Priority Encoder.

- b) As you know, the main kit has 2 sets of DIP switches, with 8 switches each: 1.1-1.8 and 2.1-2.8. Connect inputs A1~A8 to DIP Switches 1.1~1.8 and A9 to 2.1 (or one of the unused main switches S1-S4). Connect outputs F1~F4 to Logic indicators L1~L4. Follow the input sequences given in Table 6 and record output states. Be aware of the active LOW and active HIGH polarity for the inputs/outputs when interpreting the results.

Table 3.6: Data for part 3.5.2 (b)

Inputs									Outputs			
A9	A8	A7	A6	A5	A4	A3	A2	A1	F1(MSB)	F2	F3	F4(LSB)
0	1	1	1	1	1	1	1	1				
0	0	1	1	1	1	1	1	1				
1	1	1	1	1	1	1	1	0				
1	1	1	1	1	1	1	0	0				
1	1	1	1	1	1	0	1	1				
1	1	1	1	1	0	0	0	0				
1	1	1	1	0	1	1	1	1				
1	1	1	1	0	0	0	1	1				
1	1	1	0	1	1	1	0	0				
1	1	0	1	1	0	1	1	0				
1	1	0	0	0	1	1	1	1				
1	0	0	0	0	1	1	1	1				

### 3.5.3 Constructing 2-to-4 Line Decoder with Basic Gates

- a) Block c of module KL-33005 will be used in this section of the experiment. Connect +5V of module KL-33005 to the +5V output of fixed power supply.

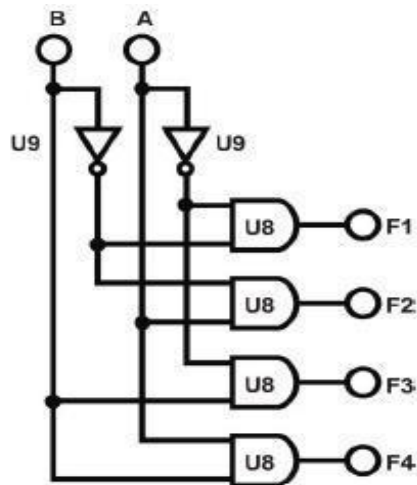


Figure 3.13: 2-to-4 Decoder.

- b) Connect inputs A, and B to Data Switches SW0 and SW1. Connect outputs F1~F4 to Logic Indicators L0~L3 respectively.
- c) Follow the input sequences for A and B in Table 3.7 and record output states.

**Table 3.7: Data for part 3.5.3 (c)**

Inputs		Outputs			
B	A	F1	F2	F3	F4
0	0				
0	1				
1	0				
1	1				

### 3.5.4 Constructing 4-to-10 Line Decoder with TTL IC

- a) U10 (7442) on block c of module KL-33004 will be used in this section of the experiment. 7442 is a BCD-to-Decimal decoder IC. BCD: Binary Coded Decimal.

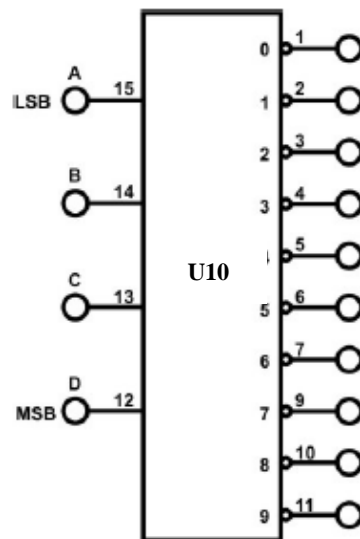


Figure 3.14: 4-to-10-line Decoder.

- b) Connect inputs A, B, C, and D to the Data Switches SW0, SW1, SW2 and SW3, respectively. Connect 10 outputs to corresponding Indicators L0~L9.
- c) Adjust the switches according to Table 3.8. Observe the output states at L0~L9. Record input and output logic states in Table 8. (Note input is the binary number for the number in the first column and the outputs active low).

**Table 3.8: Data for part 3.5.4 (c)**

Input					Output									
	D	C	B	A	0	1	2	3	4	5	6	7	8	9
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										

### 3.5.5 Constructing 2-to-1-Line Multiplexer with basic Gates

- a) Block e of module KL-33006 will be used as a 2-to-1 MUX. Connect +5V of module KL-33006 to the +5V output of fixed power supply.

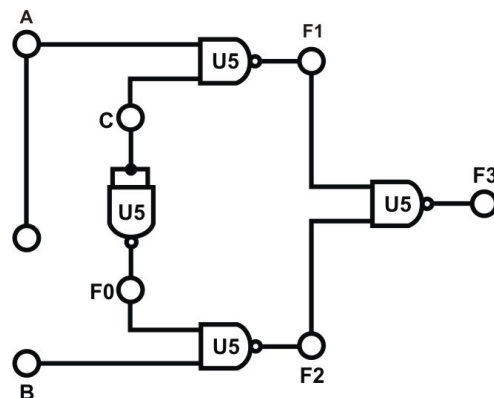


Figure 3.15: 2-to-1 Multiplexer.

- b) Connect inputs A, B to Data Switches SW0, SW1; Selector C to SW2. Connect output F3 to Logic Indicator L0.
- c) Follow the input sequences in Table 3.9 and record states of F3. Which input (A or B) determines the output for each value of C? Does this correspond to a 2-to-1 Multiplexer as you know it?

**Table 3.9: Data for part 3.5.5 (c)**

Inputs			Output
C	B	A	F3
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

### 3.5.6 Constructing 8-to-1 Line Multiplexer with IC

- a) Chip (74LS151) on block f of module KL-33006 will be used in this section of the experiment.

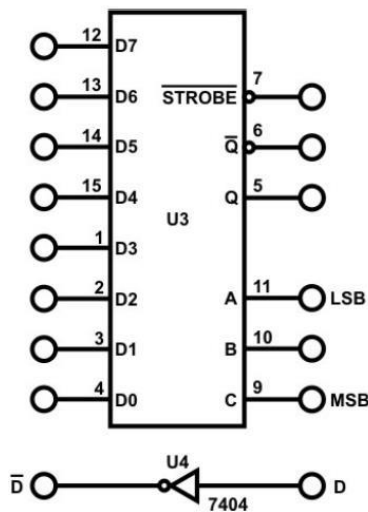


Figure (16): 8-to-1 MUX.

- b) Refer to the datasheet for specifications of the 74LS151. A, B, C are the control (selection) inputs: CBA is the value of C then B then A. So, CBA=011 means that C=0, B=1, A=1. Q is the output of the MUX.
- When CBA = “000”, data at D0 is sent to output Q.
- When CBA = “010”, data at D2 is sent to output Q.
- ⋮
- When CBA = “111”, data at D7 is sent to output Q.
- The IC will function properly only when STROBE = “0”.

When STROBE = “1” IC do not change output according to inputs, Q will remain “1”.

- c) Connect inputs D0~D7 to DIP Switch 1.0~1.7; inputs C, B, A to Data Switches SW2, SW1, SW0. Follow the input sequences in Table 6, adjust D0~D7 and CBA, and record output states. Determine which input among D0~D7 does Q depends on.

**Table 3.10: Data for part 3.5.6 (c)**

Inputs			Output
C	A	B	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

### 3.5.7 Using Multiplexer to implement a Logic Function

Given the following function:

$$F(A, B, C, D) = \sum (0, 2, 4, 5, 7, 8, 10, 11, 15)$$

- Implement the function using the 8-to-1 multiplexer using Block f of module KL-33006. Note that we have 4 inputs for this function. Three of them will be represented by the controls C B A.
- Connect inputs D, C, B, A to Data Switches SW3, SW2, SW1 and SW0 respectively. Connect output Y (Q) to Logic Indicator L0.
- Connect the Inputs D0~D7 to the proper input from: {0,1,D,D’} based on comparing the value of Y and input D for a fixed C B A value: 0 if Y=0 for D=0 and D=1, 1 if Y=1 for D=0 and Y=1 for D=1, D if Y=0 for Y=0 for D=0 and Y=1 for D=1 and D’ if Y=1 for D=0 and Y=0 for D=1.
- Record output states in Table 3.11.

Table 3.11: Data for part 3.5.7 (d)

Inputs				Output
A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

### 3.5.8 Constructing 1-to-2 Line Demultiplexer with Basic Logic Gates

- a) Block e of module KL-33006 will be used in this section. Make the connections according to Figure 3.17. Connect A to Data Switch SW0; C to SW3; F1 and F2 to Logic Indicators L1 and L2 respectively.

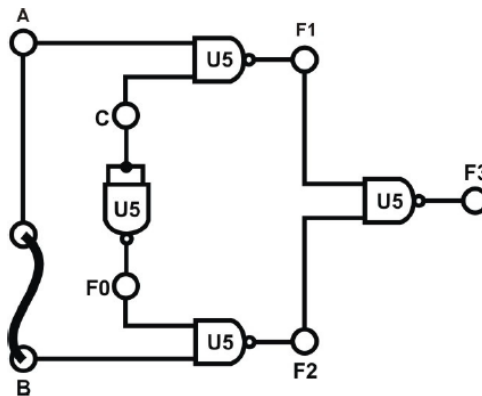


Figure 3.17: 1-to-2 Demultiplexer.

- b) Set C to “0” and change data at input A. Observe how F1 and F2 changes. Set C to “1”, change A and observe how F1 and F2 react to changes of A.

Table 3.12: Data for part 3.5.8 (b)

Inputs		Outputs	
C	A	F1	F2
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	0

### 3.5.9 Constructing 1-to-8-Line Demultiplexer with CMOS IC

- a) U2 (4051) on block b of module KL-33006 is used in this section of the experiment. Connect +5V, -5V of module KL-33006 to the +5V and -5V output of fixed power supply respectively.

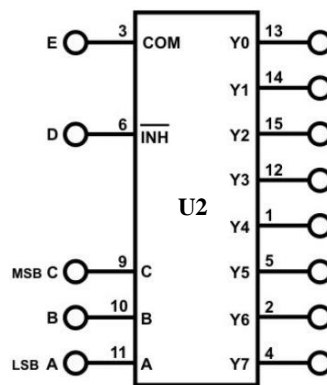


Figure 3.18: 1-to-8 Demultiplexer

- b) Connect E to DIP1.0; D to DIP1.1; A to SW0; B to SW1; C to SW2; outputs Y0~Y7 to Logic Indicators L0~L7 respectively.
- c) At D=0, apply the input sequence  $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$  to the common input E and observe outputs Y0~Y7. Did the outputs change as the input sequence is applied? Why?
- d) At D=1, apply the input sequence  $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$  to the common input E and observe outputs Y0~Y7. Did the outputs change as the input sequence is applied?
- e) Using the same sequence for E ( $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ ) with D=0, follow the sequence for A, B, and C given in Table 3.13. Record output states.



Table 3.13: Data for part 3.5.9 (e)

Inputs			Outputs							
C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

- Selected Tasks by the instructor.

### 3.6 Post Lab

1. Implement 8x1 Multiplexer using lower order Multiplexers Show how to solve it.
2. Design a Majority Circuit; a circuit that takes 4 inputs A, B, C, D and 1 output Y. Its output equals 1 when 3 or 4 of the inputs are 1. You can only use two 4x1 multiplexers.



**Faculty of Engineering and Technology**

**Department of Electrical and Computer Engineering**

**ENCS 2110**

**Digital Electronics and Computer Organization Lab**

**Experiment No. 4 - Digital Circuits Implementation using Breadboard**

#### **4.1 Objectives**

- Understanding the NAND and NOR gate characteristics and how to implement circuit
- To continue the previous experiment with simple digital devices and their operations using a breadboard.

#### **4.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab
- Breadboard
- Integrated circuits (Chips):
  - IC 7404 (inverter)
  - IC 7408 (2-input AND)
  - IC 7432(2-input OR)
  - IC 7400(2- input NAND)
  - IC 7486 (2-input XOR)

## 4.3 Pre Lab

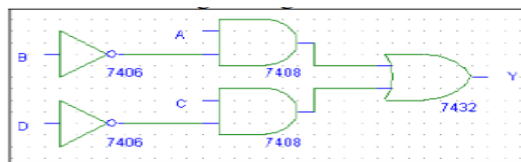
1) Watch the video in the link below to understand what the breadboard is and how it works and the way components, including chips are connected to the breadboard as was done in experiment 1.  
<https://www.youtube.com/watch?v=gwcVr5VfXwA>

2) Recall how to identify the pins of a chip. You may find the following presentation useful:  
<https://www.youtube.com/watch?v=Y9vsZTpnDDI>

3) Design and implement the following circuit using the gates on the chips as shown in Figure 4.1(.

**Your final circuit must include the ICs, their pin numbers, and the connections between the pins.).**

- a) Build Full Adder using basic gates.
- b) Build the bellow circuit using universal gates.



- c) Build a 3x8 Decoder (active low) using basic gates.
- d) Build an 8x1 Multiplexer using basic gates.
- e) Use the just constructed **4x1 multiplexer** to design a three-input network that gives 1 if the majority of its inputs are 1 and outputs a zero otherwise.
- f) Use the 2x4 decoder to implement a 2-input function that acts like an equivalence gate (XNOR): gives 1 on the output if both inputs are equal.

## 4.4 Theory

### 4.4.1 74xx ICs Family

The first family of logic chips to really catch on was Texas Instruments 74 series TTL. These contain bipolar transistors, run on 5V, are fast and use rather a lot of power. However, the 74xx numbering scheme has persisted. For example, a 7400 (seven-four-zero-zero) is a quad NAND gate. Today you can buy 74C00, 74HC00, 74HTC00, 74LS00, and 74S00. Those are 5V families. Then there is low (3.3V or lower) voltage families like 74LVC, 74AUP and so on. Some of these use bipolar transistors, others (these days most) use CMOS transistor technology.

**Note:** There are several logic ICs numbered from 74xx onwards with letters (xx) in the middle of the number to indicate the type of gate as shown in Figure 4.1. Figure 4.1 shows some digital gates with identification numbers and pin assignments.

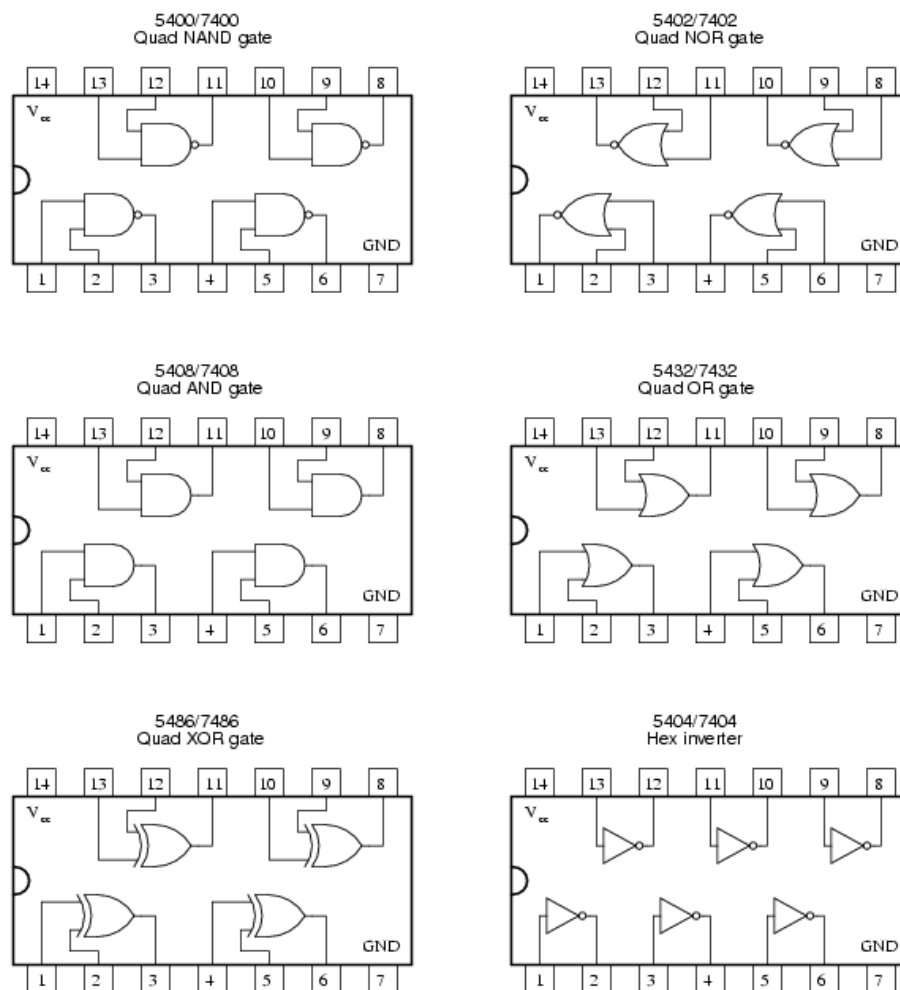


Figure 4.1: DIGITAL GATES IN IC PACKAGES.

### 4.4.2 Breadboard

The breadboard consists of two terminal strips and two bus strips (often broken in the center). Each bus strip has two rows of contacts. Each of the two rows of contacts are a node. That is, each contact along a row on a bus strip is connected (inside the breadboard).

Bus strips are used primarily for power supply connections but are also used for any node requiring a large number of connections. Each terminal strip has 60 rows and 5 columns of contacts on each side of the center gap. Each row of 5 contacts is a node as shown in Figure 4.2.

You will build your circuits on the terminal strips by inserting the leads of circuit components into the contact receptacles and making connections with 22–26-gauge wire. There are wire cutter/strippers and a spool of wire in the lab. It is a good practice to wire +5V and 0V power supply connections to separate bus strips.

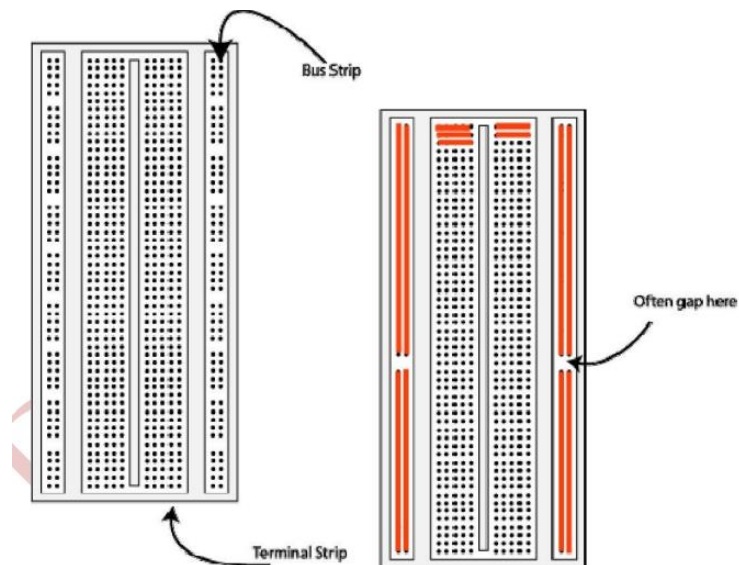


Figure 4.2: The breadboard. Lines indicate connected holes.

The 5V supply **MUST NOT BE EXCEEDED** since this will damage the ICs (Integrated circuits) used during the experiments. Incorrect connection of power to the ICs could result in them exploding or becoming very hot - with the possible serious injury occurring to the people working on the experiment! Ensure that the power supply polarity and all components and connections are correct before switching on power. You can learn more about the breadboard by click in this link <https://youtu.be/gwcVr5VfXwA>

## 4.5 Procedure

In this experiment, we will use a breadboard to implement different circuits using basic gates and NAND and NOR gates to build other gates. Before that, there were some instructions to build a circuit and the common problems.

### **Building the Circuit:**

Throughout these experiments, we will use TTL chips to build circuits. The steps for wiring a circuit should be completed in the order described below:

1. Turn the power (Trainer Kit) off before you build anything!
2. Make sure the power is off before you build anything!
3. Connect the +5V and ground (GND) leads of the power supply to the power and ground bus strips on your breadboard.
4. Plug the chips you will be using into the breadboard. Point all the chips in the same direction, with pin 1 at the upper-left corner. (Pin 1 is often identified by a dot or a notch next to it on the chip package)
5. Connect +5V and GND pins of each chip to the power and ground bus strips on the breadboard.
6. Select a connection on your schematic and place a piece of hook-up wire between the corresponding pins of the chips on your breadboard. It is better to make the short connections before the longer ones. Mark each connection on your schematic as you go, so as not to try to make the same connection again at a later stage.
7. Get one of your group members to check the connections before you turn the power on.
8. If an error is made and is not spotted before you turn the power on, Turn the power off immediately before you begin to rewire the circuit.
9. At the end of the laboratory session, collect your hook-up wires, chips and all equipment and return them to the demonstrator.
10. Tidy the area that you were working in and leave it in the same condition as it was before you started.

### **Common Causes of Problems:**

1. Not connecting the ground and/or power pins for all chips.
2. Not turning on the power supply before checking the operation of the circuit.
3. Leaving out wires.
4. wires into the wrong holes.
5. Driving a single gate input with the outputs of two or more gates  
Modifying the circuit with the power on.

#### **4.5.1 Verification of basic logic gates**

In this task you are to verify the operations of some of the ICs.

1. Test each chip using the IC tester and make sure that it is functioning properly.
2. Place each chip shown in Figure 4.1 on the breadboard in such a way that its pins are not short-circuited. Make sure power is off while you place IC's and connect wires.
3. Connect GND and +5V for each chip you want to check. Connect the gate inputs to the dip switches and the gate output to any LED. Determine the output for each possible input combination and compare your results with the expected Truth Tables.
4. Verify the function of the 7400 (2-input NAND) chips by observing how the output of the Gates changes in response to input changes.
  - a) how does the gate act if one of its two input is held at “1”?
  - b) how does the gate act if its two inputs are connected together?

#### **4.5.2 Build circuits using Gates**

- a) Implement the following circuit in Figure 4.3 using basic gates and fill the truth table (Table 4.1).

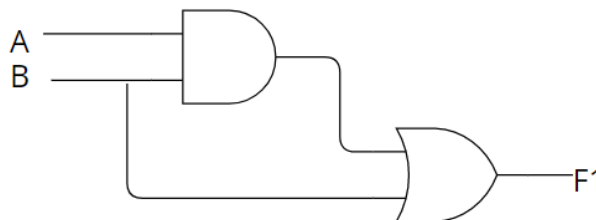


Figure 4.3: Circuit 1 using basic gates.

**Table 4.1: Data for part 4.5.2 (a)**

Inputs		Outputs
A	B	F1
0	0	
0	1	
1	0	
1	1	

What is the Boolean function for above circuit?.....

- b) Implement the following circuit in Figure 4.4 using basic gates and fill the truth table (Table 4.2).

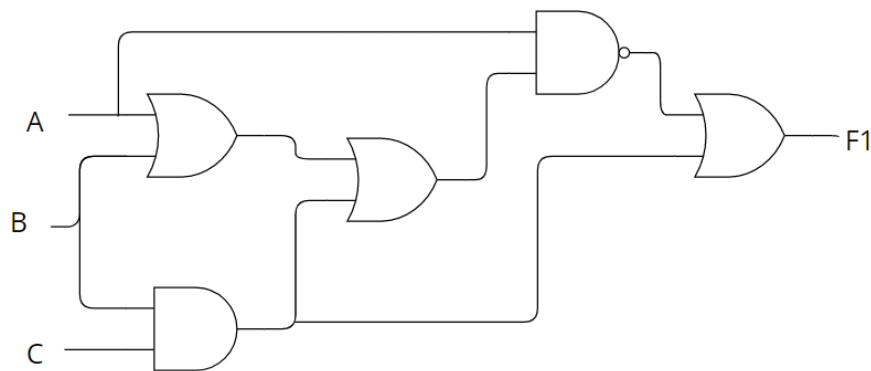


Figure 4.4: Circuit 2 using basic gates.

**Table 4.2: Data for part 4.5.2 (b)**

Inputs			Output
A	B	C	F1
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

What is the Boolean function for above circuit?.....



- c) Implement the following circuit in Figure 4.5 using basic gates and fill the truth table (Table 4.3).

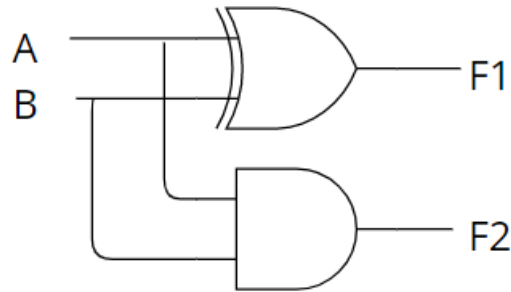


Figure 4.5: build circuit 3 using basic gates.

Table 4.3: Data for part 4.5.2 (C)

Inputs		Outputs	
A	B	F1	F2
0	0		
0	1		
1	0		
1	1		

What is the Boolean function for above circuit?.....

What this circuit do?.....

- d) (**Adder**) Use any needed gates shown in Figure 1 and the designs you prepared as a pre-lab to implement the full adder, test your design by verifying the truth table of the Full Adder.

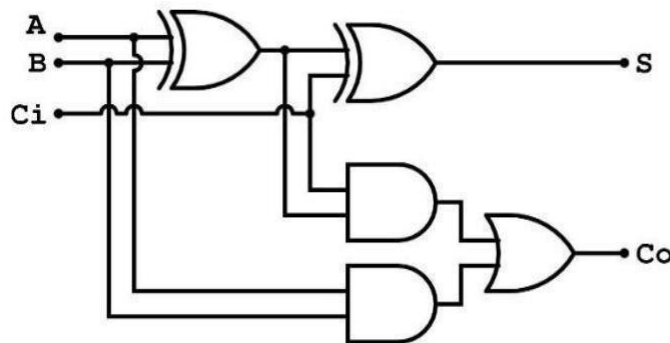


Figure 4.6: Full Adder.

e) **(Decoder)** Use any needed gates shown in Figure 1 and the designs you prepared as a pre-lab to implement the 2x4 decoder. Test your design by verifying the truth table of the DECODER.

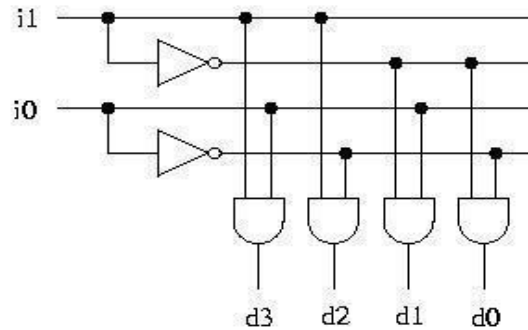


Figure 4.7: 2x4 decoder.

f) **(Multiplexer)** Use any needed gates shown in Figure 1 and the design you prepared as a pre-lab to implement the 4x1 multiplexer. Test your design by verifying the truth table of the MUX.

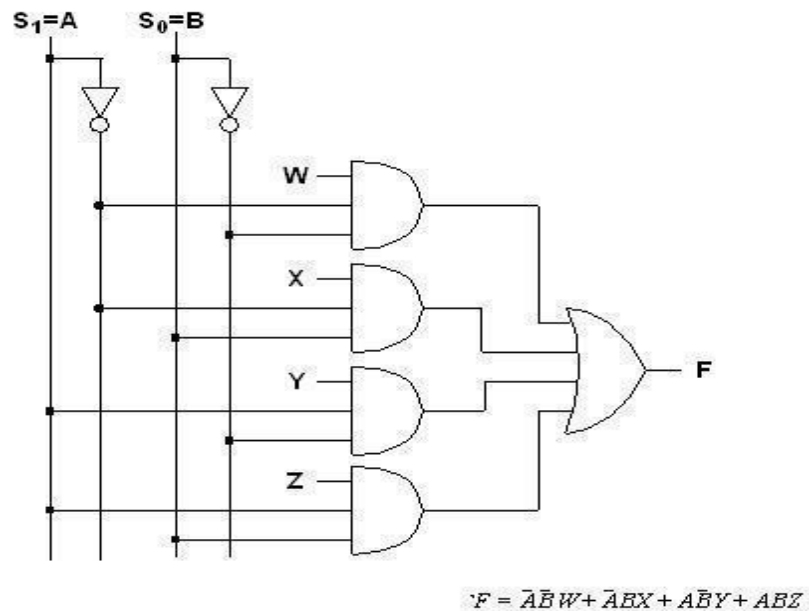


Figure 4.8: 4X1 Multiplexer.

- **Selected Tasks by the instructor.**

## 4.6 Post Lab

1. How do you go about adding an Enable (E) signal to the decoder in Figure 4.7? Modify the implementation to show that. (Design Only using chips in Figure 1).
2. How to use that to implement a 3x8 decoder using chips in Figure 1.
3. Use the just constructed 4x1 multiplexer to design a three-input network that gives 1 if the majority of its inputs are 1 and outputs a zero otherwise (Design Only using chips in Figure 1).
4. Implement  $f(x, y, z) = m(0, 1, 4, 6, 7)$ , using 4x1 MUX using chips in Figure 1.



**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 5 - Sequential Logic Circuits**

### **5.1 Objectives**

- To understand the differences between combinational and sequential Logic circuits; and the applications of various memory units.
- To study the operating principles and applications of various flip-flops.
- To understand the operating principles of counters and how to construct counters with JK flip-flops.
- To study the synchronous and asynchronous counters.

### **5.2 Equipment required**

- KL-31001 Basic Electricity Circuit Lab.
- KL-33008 Sequence Logic Circuit Experiment Model (1)
- KL-33009 Sequence Logic Circuit Experiment Model (2)
- KL-33010 Memory Circuit Experiment Model (1)

## 5.3 Laboratory Regulations and Safety Rules

The following Regulations and Safety Rules must be observed in the laboratory:

1. It is the duty of all concerned who use any electrical laboratory to take all reasonable steps to safeguard the HEALTH and SAFETY of themselves and all other users and visitors.
2. Be sure that all equipment is properly working before using them for laboratory exercises. Any defective equipment must be reported immediately to the Lab. Instructors or Lab. Technical Staff.
3. Students can use only the equipment provided in the experiment manual or used for the senior project laboratory.
4. Power supply terminals connected to any circuit are only energized in the presence of the Instructor or Lab. Staff.
5. Students should keep a safe distance from the circuit breakers, electric circuits, or any moving parts during the experiment.
6. Avoid any part of your body being connected to the energized circuit and ground.
7. Switch off the equipment and disconnect the power supplies from the circuit before leaving the laboratory.
8. Observe cleanliness and proper laboratory housekeeping of the equipment and other related accessories.
9. Double-check your circuit connections before switching “ON” the power supply.
10. Make sure that the last connection to be made in your circuit is the power supply and first thing to be disconnected is also the power supply.
11. Equipment should not be removed, or transferred to any location without permission from the laboratory staff.
12. Software installation in any computer laboratory is not allowed without permission from the Laboratory Staff.
13. Computer games are prohibited in the computer laboratory.
14. Students are not allowed to use any equipment without proper orientation and actual hands-on equipment operation.
15. Smoking and drinking in the laboratory are not permitted.

## 5.4 Pre Lab

1. Read all experiments to find the **prelab questions**
2. For each used IC (Integrated Circuits), search for its datasheet that explains exactly what a component does and how to use it.

## 5.5 Introduction

### 5.5.1 Sequential Circuits

Any digital circuit could be classified as either a combinational or a sequential circuit. Combinational logic circuits implement Boolean functions. Boolean functions are mappings of inputs to outputs. These circuits are functions of input only.

Sequential circuits are two-valued networks in which the outputs at any instant are dependent not only upon the inputs present at that instant but also upon the history (sequence) of inputs. The block diagram of a sequential circuit is shown in Figure 5.1. The basic logic element that provides memory in many sequential circuits is the flip-flop.

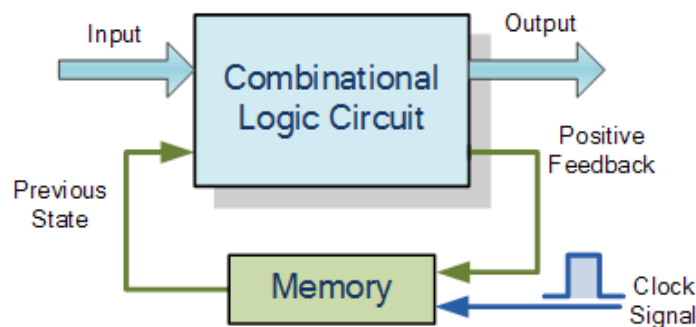


Figure 5.1: Sequential Circuit Block Diagram

### 5.5.2 Latches

Latches form one class of flip-flops. This class is characterized by the fact that the output changes' timing is not controlled. Although latches are useful for storing binary information and for the design of asynchronous sequential circuits, they are not practical for use in synchronous sequential circuits.

### 5.5.2.1 The SR (Set-Reset) Latch

It is a circuit with two cross-coupled NOR or NAND gates.

a. SR latch with NAND gates:

The one with NAND gates is shown in Figure 5.2. Note that this circuit is **active low** set/reset latch; that means the output Q goes to 1 when S (set) input is 0 and goes to 0 when R (Reset) input is 0 as shown in Table 5.1. The undefined condition is when both inputs are equal to 0 at the same time.

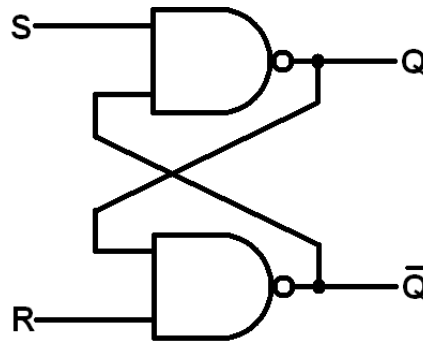


Figure 5.2: SR latch with NAND gate

Table 5.1: SR latch with NAND gate Truth table.

INPUT		OUTPUT		State
S	R	Q	$\bar{Q}$	
1	0	0	1	SET
1	1	0	1	No Change/ Memory
0	1	1	0	RESET
1	1	1	0	No Change/ Memory
0	0	1	1	Invalid

b. SR latch with NOR gates:

**Design the Logic Diagram, and function table of the SR latch using NOR gates, and explain how it works. (Pre-lab)**

c. RS latch with control input:

The RS latch with control input C is shown in Figure 5.3. If C=0 the output Q does not change regardless less the R and S values. If C= 1the circuit will work normally as shown in Table 5.2.

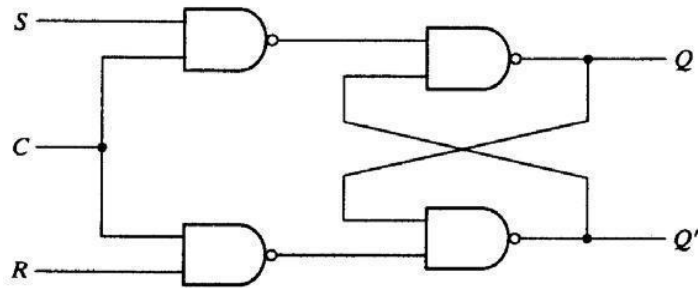


Figure 5.3: RS latch with control input.

Table 5.2: RS latch with control truth table.

INPUT			OUTPUT		State
C	S	R	$Q_{n+1}$	$Q'$	
0	X	X	$Q_n$	$Q'_n$	No Change/ Memory
1	0	0	$Q_n$	$Q'_n$	No Change/ Memory
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	0	0	Indeterminate

### 5.5.2.2 The D Latch

The D latch was developed to eliminate the undefined condition of the indeterminate state in the RS latch. The D latch and its state table is shown in Figure 5.4 and Table 5.3.

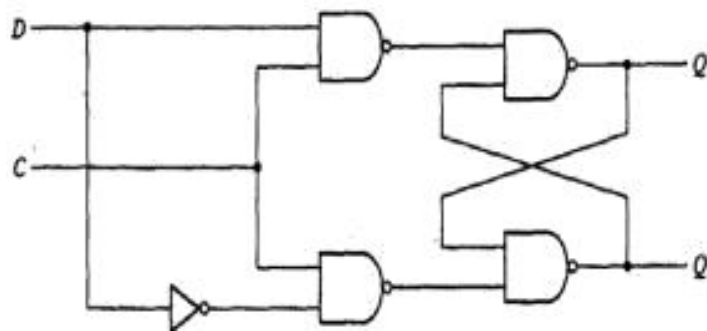


Figure 5.4: D-Latch.

Table 5.3: D latch truth table.

INPUT		OUTPUT		State
C	D	$Q_{n+1}$	$Q'$	
0	X	$Q_n$	$Q'_n$	No Change/ Memory
1	0	0	1	RESET
1	1	1	0	SET



### 5.5.3 Flip-Flops

Like latches, flip-flops are also used for storing binary information, but the difference is: The output change in the flip-flop occurs only at the clock edge while in the latch it occurs at the clock level.

A flip-flop can be implemented using two separate latches. *Figure 5.5* shows the D flip-flop implemented with two D latches.

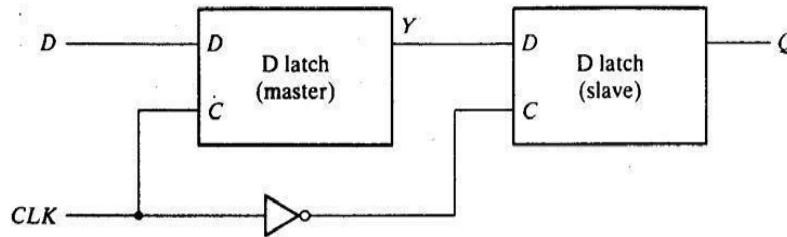


Figure 5.5: D flip flop implemented with two D latches.

There are several types of flip-flops, the common ones are D, T, and JK flip flops. Figure 5.6 shows these flip flops and their function tables.

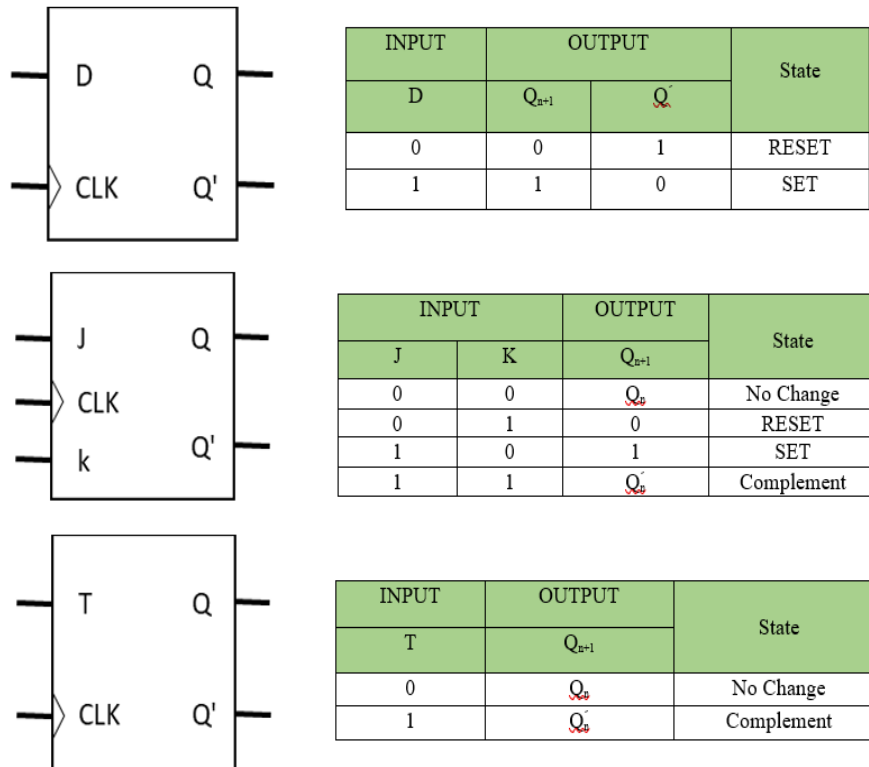


Figure 5.6: D, JK, and T flip flops.

### 5.5.4 Registers

Digital systems use registers to hold binary entities. The register is a collection of flip flops; N-bit register consists of N flip-flops. Figure 5.7 shows simple 4-bit register implemented with D- flip flops. All the flip-flops are driven by a common clock, and all are reset simultaneously.

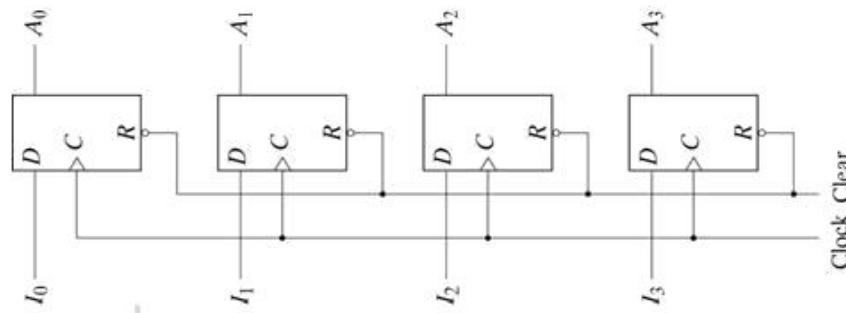


Figure 5.7: 4-bit Register.

Shift register is a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Figure 5.8 shows 4-bit shift- right register.

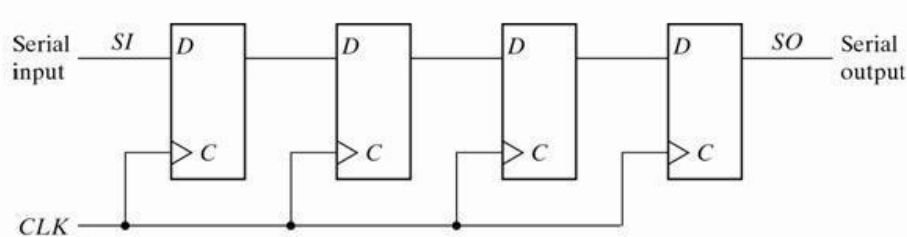


Figure 5.8: 4-bit shift- right register.

### 5.5.5 Counters

The counter is a special-purpose register; it is a register that goes through a prescribed sequence of states.

The counters are classified into two categories: Ripple and Synchronous counters. In ripple counters, there is no common clock; the flip-flop output transition serves as a source for triggering other flip-flops. In synchronous counters, all flip flops receive a common clock. Figure 5.9 shows 3-bit ripple and synchronous counters.

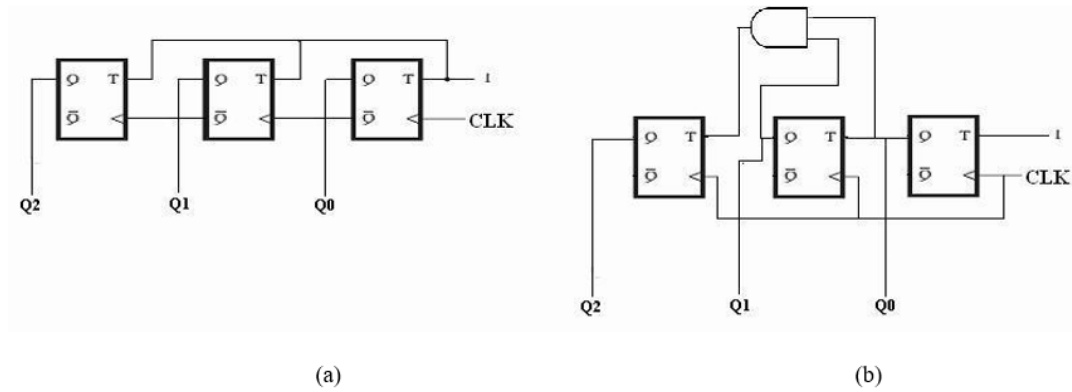


Figure 5.9: (a) 3-bit ripple counters, (b) 3-bit synchronous counter.

## 5.6 PROCEDURE

### 5.6.1 Latches and Flip Flops:

#### a) Constructing RS latch with Basic Logic Gates

1. Use the KL-33008 module to construct the circuit shown in Figure 5.10.
2. Connect +5V of module KL-33008 to the +5V output of the fixed power supply and GND of module KL-33008 to the GND output of the fixed power supply
3. Connect inputs A3, A4 to Switches SW1, SW2.
4. Connect outputs F6 and F7 to Logic Indicators L1, L2. Follow the sequences in Table 5.4.

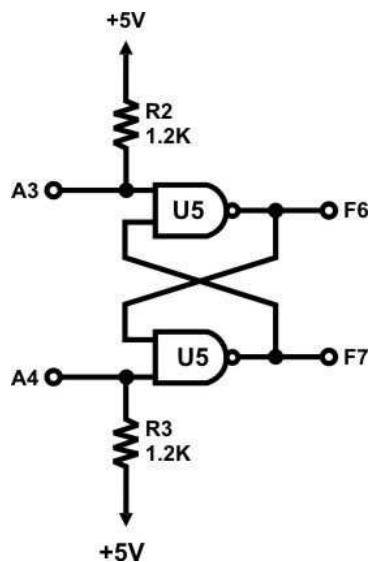


Figure 5.10: RS latch.

Table 5.4: Data for part 5.6.1 (A).

INPUT		OUTPUT	
A3	A4	F6	F7
0	0		
0	1		
1	0		
1	1		

**b) Constructing RS latch with control input**

1. Use the KL-33008 module to connect the circuit shown in Figure 5.11.
2. Connect inputs A1, A5 to Switches SW1, SW2 and follow the input sequence in *Table 5.5*.

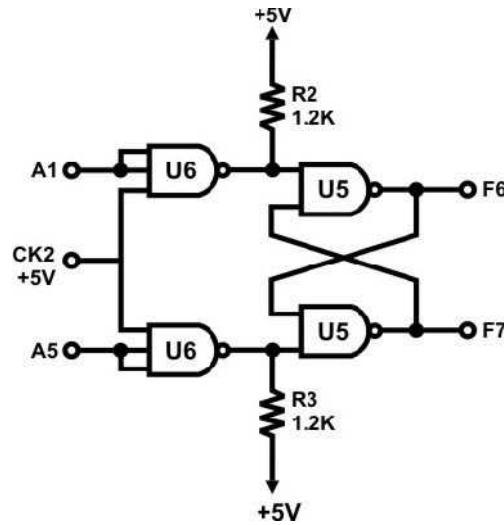


Figure 5.11: RS Latch with control input.

Table 5.5: Data for part 5.6.1 (B).

INPUT		OUTPUT	
A1	A5	F6	F7
0	0		
0	1		
1	0		
1	1		

**c) Constructing D latch with RS latch**

1. Use the KL-33008 module to construct the circuit shown in Figure 5.12.
2. Connect A1 to SW1; CK2 to SWA A and F6 to L1. Follow the sequences in Table 5.6.

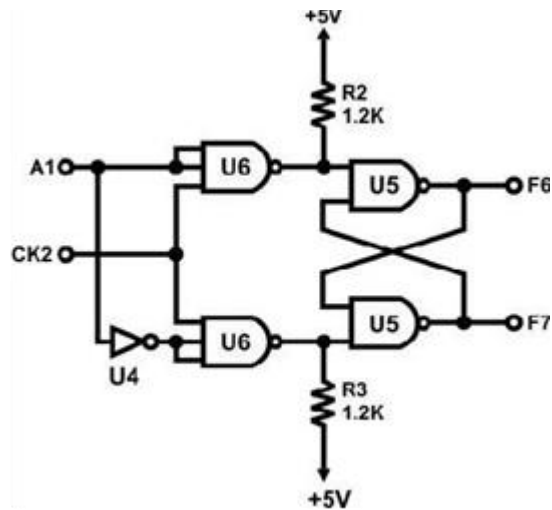


Figure 5.12: D Latch.

Table 5.6: Data for part 5.6.1 (C).

INPUT		OUTPUT
CK2	A1	F6
0	0	
0	1	
$\square$ (PULSE)	0	
$\square$ (PULSE)	1	

#### d) Constructing JK latch with RS latch

1. Use the KL-33008 module to construct the circuit shown in Figure 5.13.
2. Connect CK2 to SWB B output; A1 to SW0; A5 to SW1; F6 to L1.
3. Connect the feedback as shown in Figure 5.13.
4. Apply the following sequences and write the status of the led at each sequence value in Table 5.7.

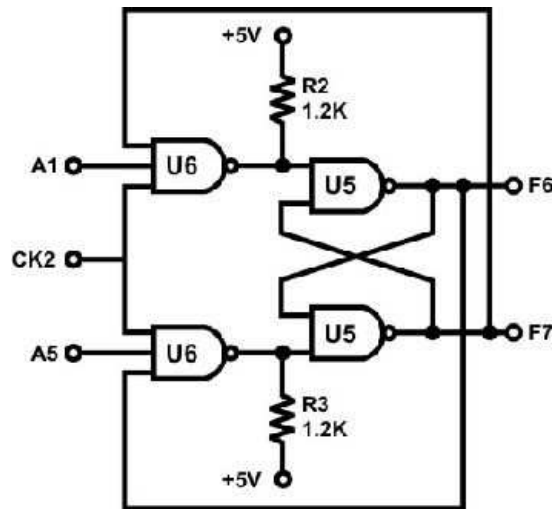


Figure 5.13: JK Latch.

Table 5.7: Data for part 5.6.1 (D).

INPUT			OUTPUT	Status
CK2	A1	A5	F6	
	1	0		
	0	0		
	1	1		
	1	0		
	0	0		
	0	1		
	1	1		

- Does the led sequence follow the expected result according to the FF state table, explain?

#### e) Constructing JK Flip-flop with master-slave RS latches

The master-slave flip-flop cancels all timing problems by using two SR flip-flops connected. The first flip-flop acts as the “Master” circuit, which triggers on the leading edge of the clock pulse while the other acts as the “Slave” circuit, which triggers on the falling edge of the clock pulse. This results in the two sections; the master section and the slave section

being enabled during opposite half-cycles of the clock signal. Let us do this part by following these steps:

1. Use the KL-33008 module to construct the circuit shown in Figure 5.14.
2. Connect CK1 to SWA A output; J to SW1; K to SW0; F1, F2, F6, F7 to L3, L2, L1 and L0, respectively.
3. Follow the sequences in Table 5.8.

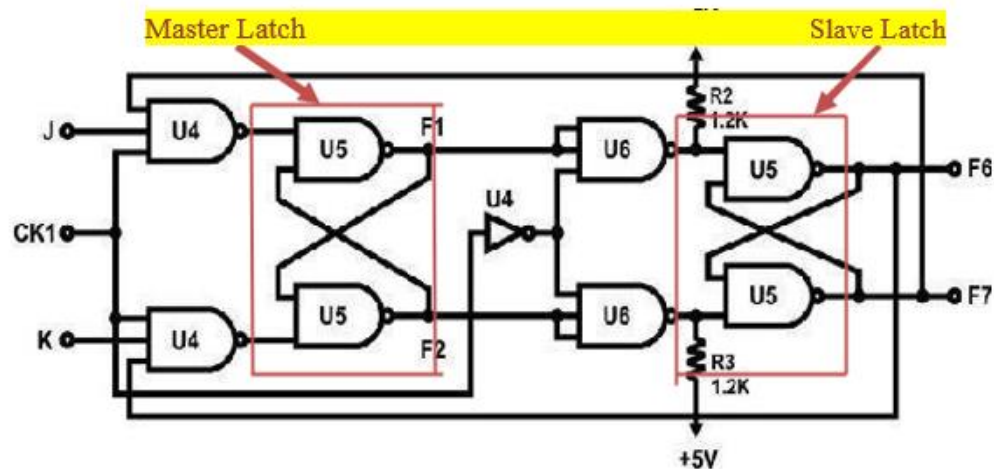


Figure 5.14: JK Flip-Flop.

Table 5.8: Data for part 5.6.1 (E).

INPUT			OUTPUT			
CK2	K	J	F1	F2	F6	F7
	0	0				
	0	1				
	1	0				
	1	1				
	1	1				



## 5.6.2 Registers

### a) Constructing Shift Register with D Flip-Flops

- Block c of module KL-33008 will be used to construct the circuit shown in Figure 5.15.

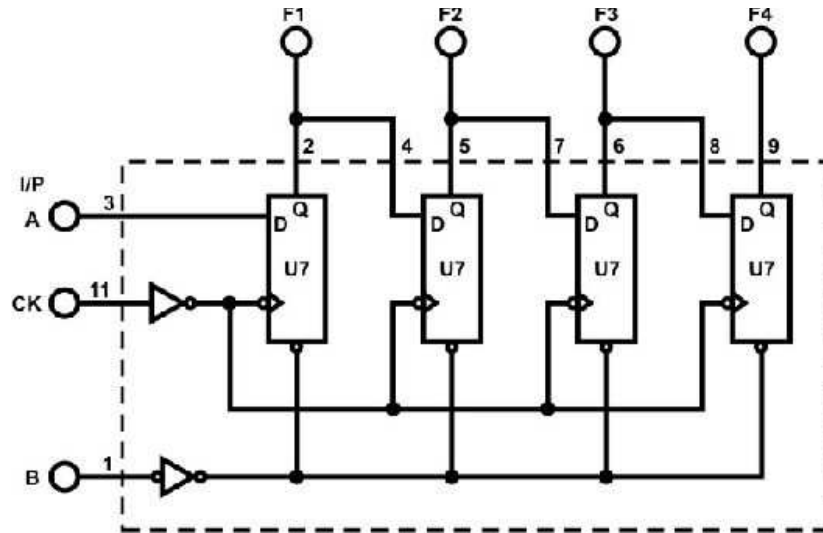






Figure 5.15: Shift Right Register.

- Connect B (clear) to SW0; A (I/P) to SW1; CK to SWA A output; F1, F2, F3, F4 to L1, L2, L3, L4, respectively.
- Set SW0 to “0” to clear B and then set SW0 to “1”.
- Follow the input sequence for A(I/P) below, observe output display at F1, F2, F3 and F4, and fill Table 5.9:

  - at A= “1”, send in a CK signal from SWA
  - at A= “0”, send in a CK signal from SWA
  - at A= “0”, send in a CK signal from SWA
  - at A= “1”, send in a CK signal from SWA

Table 5.9: Data for part 5.6.2 (A).

INPUT		OUTPUT			
A	CK	F1	F2	F3	F4
1					

0					
0					
1					

#### b) 4-Bit Shift Register with serial and parallel load

Use block b module in KL-33008 which is 4-Bit Shift Register with serial and parallel synchronous operating modes, it has serial input (B1) and four parallel (A-D) Data inputs, and four Parallel Data outputs (QA–QD) as shown in Figure 5.16.

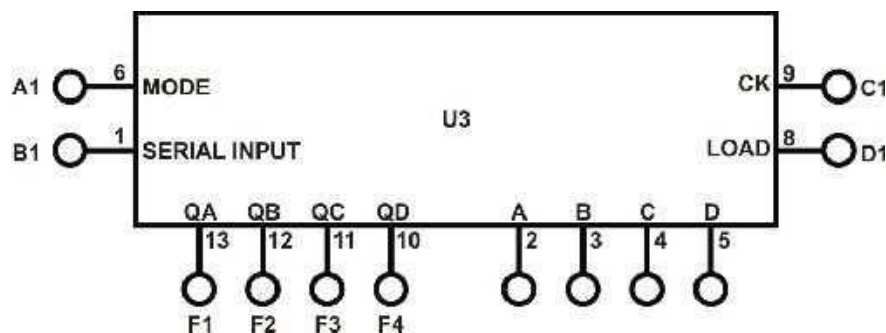




Figure 5.16: shift register with serial and parallel load.

1. Complete the following connections:





- Connect Inputs A, B, C, D to SW0, SW1, SW2, SW3 Outputs F1, F2, F3, F4 to L0, L1, L2, L3, respectively.
- B1 (I/P) to DIP2.0
- A1 (MODE) to DIP2.1 as in Table 5.10.

Table 5.10: A1 modes.

Input		
MODE Control (A1)	CK	
	C1	D1
L		X
H	X	






2. Connect CK (C1) to the clock generator TTL level output at 1Hz and change data at B1 with DIP2.0. Follow the input sequences for A1 in Table 5.11. Observe and record the outputs.

Table 5.11: Data for part 5.6.2 (B-2).

INPUT		OUTPUT			
A1	C1	L3	L2	L1	L0
1					
0					
0					
1					

3. Connect LOAD (D1) to the clock generator TTL level output at 1Hz. Set A1 to “1” and follow the input sequences for A, B, C and D in Table 5.12. Observe and record the outputs.

Table 5.12: Data for part 5.6.2 (B-3).

Input					Output			
D1	D	C	B	A	L3	L2	L1	L0
	0	0	1	0				
	1	0	1	0				
	1	1	1	0				
	0	1	1	1				
	0	1	1	0				

### 5.6.3 Counters

#### a) 2-bit Synchronous Counter

1. Use the KL-33009 module to implement the 2-bit synchronous counter shown in Figure 5.17.
2. Connect +5V of module KL-33009 to the +5V output of the fixed power supply and GND of module KL-33009 to the GND output of the fixed power supply.
3. Connect CLK input to pulser switch SWA.
4. Connect counter outputs Q1 and Q0 to indication lamps L1, L2, respectively.
5. Apply clock pulses to CLK input. Observe and record the outputs (in binary) in Table 5.13(a).
6. Apply counter outputs Q1 and Q0 to **seven segment** displays. Observe and record the outputs (in decimal) in Table 11.13(b).

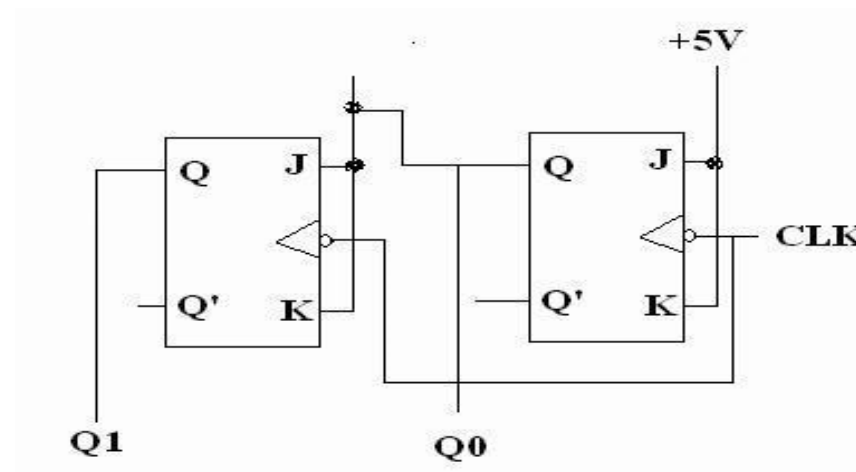


















Figure 5.17: 2-bit Synchronous Counter.

Table 5.13: Data for part 5.6.3 (A).

Input	OUTPUT	
CLK	Q1	Q0
		
		
		
		
		
		
		
		

(a)

Input	OUTPUT
CLK	D
	
	
	
	
	
	
	
	

(b)

### b) 3-bit (divide-by-eight) Ripple Counter

Divide-by-8 counter is a 3-bit counter that counts from 0-to-7:

1. Use the KL-33009 module to implement the 3-bit (divide by eight) Ripple counter shown in Figure 5.18.
2. Connect CLK input to the pulser switch.
3. Connect counter outputs Q2, Q1 and Q0 to indication lamps.
4. Apply clock pulses to CLK input. Observe and record the outputs in Table 5.14(a).
5. Apply counter outputs Q2, Q1 and Q0 to seven segment displays. Observe and record the outputs in Table 5.14(b).

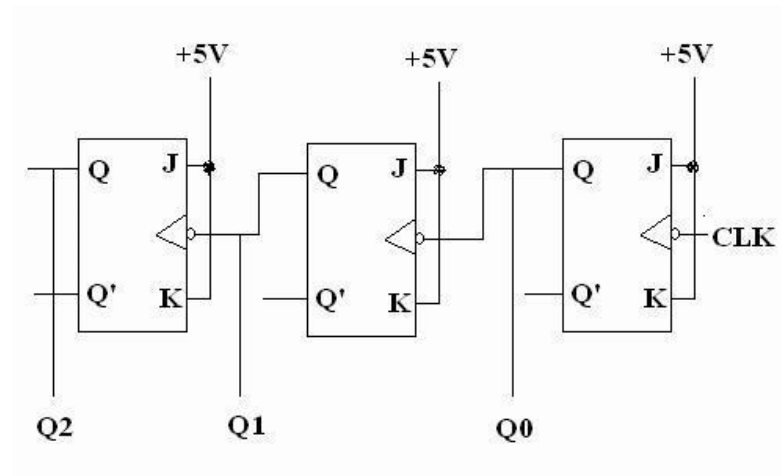


Figure 5.18: 3-bit Ripple Counter.

Table 5.14: Data for part 5.6.3 (B).

Input	OUTPUT		
CLK	Q2	Q1	Q0

(a)

Input	OUTPUT
CLK	D

(b)

**Task 2:** Modify the circuit in *Figure 5.17* to be a 3-bit **Synchronous Counter**. Attach the design with this experiment report.

### c) BCD Counter

Locate the BCD counter (IC 7490) on the KL-33010 module, which is shown in Figure 5.19. The functional block diagram of the BCD counter chip is shown in Figure 5.20.

1. Connect +5V of module KL-33010 to the +5V output of the fixed power supply and GND of module KL-33010 to the GND output of the fixed power supply.
2. Connect C3, C4 to SW0 and SW1; D1, D2 to SW2 and SW3; F1~F4 to L1~L4, A2 to SWA **A** output.
3. Connect F1 to B2, set C3, C4, D1 and D2 to ground and A2 to SWA **A** pulse. Measure and record the outputs F1, F2, F3, F4.
4. Set SW2 and SW3 to 0.

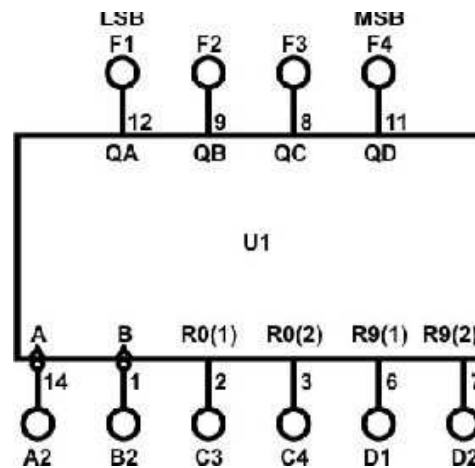


Figure 5.19: IC 7490 BCD Counter.

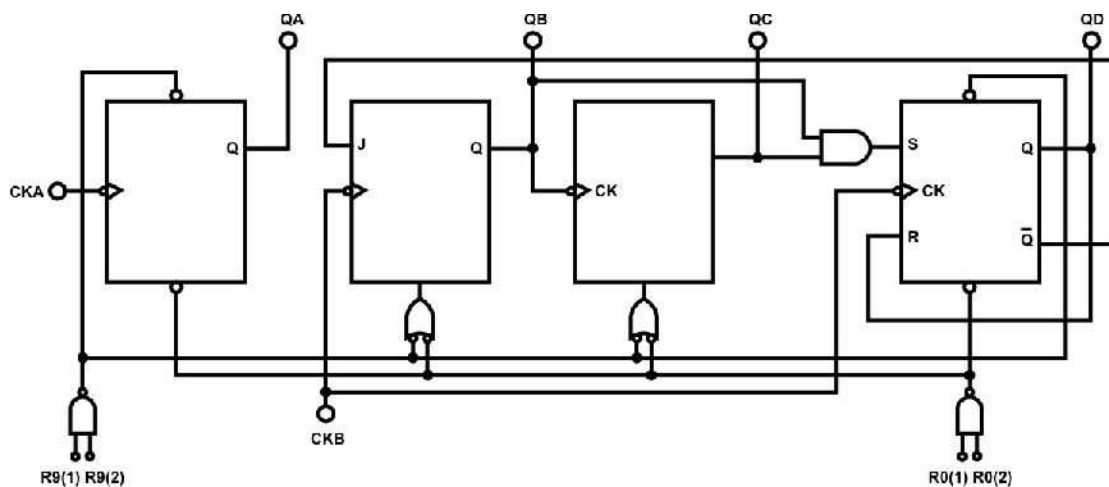


Figure 5.20: IC 7490 BCD Counter.

**d) Divide-by-8 counter using BCD chip counter:**

On the same circuit, do these modifications/changes then observe the result:

1. Change R0(2) (pin3) to +5V, and connect R0(1) (pin2) to QD (pin11) output. This will make a counter reset after 111 (or 7). **WHY?**
2. Connect clock A2 (pin14) to pulser switch.
3. Connect the outputs A, B, C, and D to indication lamps.
4. Apply clock pulses to A2 and observe the count sequence (0000-0111).

**Task3: change the connection of the counter in Figure 5.19 to count from:**

1. **0 – to – 5**
2. **0 – to – 4**

## **5.7 DISCUSSION**

Answer the following questions:

1. Although latches are useful for storing binary information, they are rarely used in sequential circuit design, why?
2. What is the disadvantage of the RS flip flop?
3. What is the difference between “synchronous” and “ripple” counters?





## **Faculty of Engineering and Technology**

### **Department of Electrical and Computer Engineering**

#### **ENCS 2110**

#### **Digital Electronics and Computer Organization Lab**

#### **Experiment No. 6 - Sequential Logic Circuits using Breadboard and IC's**

##### **6.1 Objectives**

- To learn how to use some Integrated Circuits (ICs) such as seven-segment display driver/decoder (IC7447) and counters (IC7490).
- To understand the function of the seven-segment display and how to find its pin assignment.
- To build one or more-decade counters with seven-segment displays.

##### **6.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab.
- 7447 BCD to Seven-Segment Decoders/Drivers.
- 7490 Decade and Binary Counters.

##### **6.3 Pre Lab**

- 1) What is the appropriate display type (common anode/common cathode) that must be used with 7447 display decoders? Explain your answer.
- 2) Assuming that the turn-on voltage for the LEDs is 1.7v, what is the proper value of the resistors to be connected between the 7447 decoder and the seven-segment display, to limit the current in the LED segments to 10mA?

- 3) Assume that the resistors provided in the lab are  $220\Omega$ . What would the current flowing into the LEDs be?
- 4) Design a decade counter circuit using the 7490 counters, the 7447 decoder and a seven-segment display. Show the pin numbers on the ICs in your design.

## 6.4 Theory

### 6.4.1 Seven-Segment Display

The seven-segment LED display is a common device in consumer electronics, from calculators to clocks to microwave ovens. In this lab, you will learn the basic principles of operation of the seven-segment display as well as the process of converting BCD values to the proper signals to derive this display.

The display has seven separate bar-shaped LEDs, arranged as shown below. In addition, many seven-segment displays have one (or two) circular LEDs used as a decimal point.

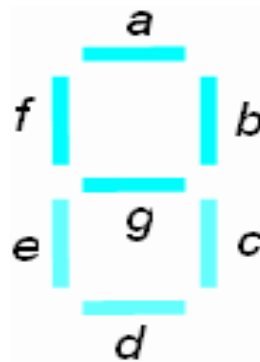


Figure 6.1: Seven-segment Display.

Inside the seven-segment display, one end of each LED is connected to a common point, which is tied either to the ground or to the positive supply, depending on the device. If the seven-segment display is designed to have the common connection tied to the positive supply (+5V), as shown in Figure 6.2 (left-hand side), it is called a common anode configuration. To turn on these LED segments, the inputs logic must be set to low.

If the seven-segment display is designed to have a common connection tied to the ground (0V), as shown in Figure 6.2 (right-hand side), it is called a common cathode configuration. To turn on these LED segments, the inputs logic must be set high.

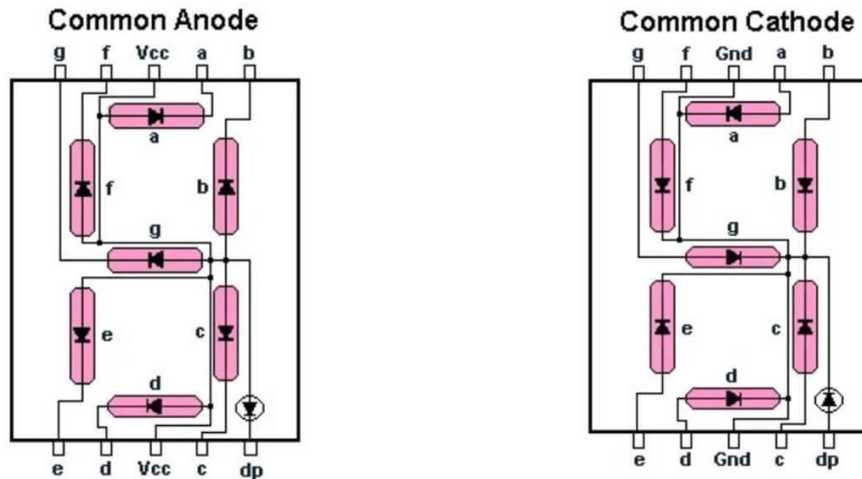


Figure 6.2: common anode/cathode displays.

In both configurations, current-limiting resistors are used to lower the amount of current that the driver sends into the LEDs. This achieves two goals:

- 1- Control the brightness of the LEDs.
- 2- Prevent over-current (that may burn the LEDs).

### 6.4.2 BCD-to-seven-segment Decoder

A BCD-to-seven-segment decoder is a logic circuit used to convert the input BCD into a form suitable for the seven-segment display.

In this lab the IC type 7447 decoder will be used. The 7447-pin assignment is shown in Figure 3. Its pin description is shown in Table 6.1.

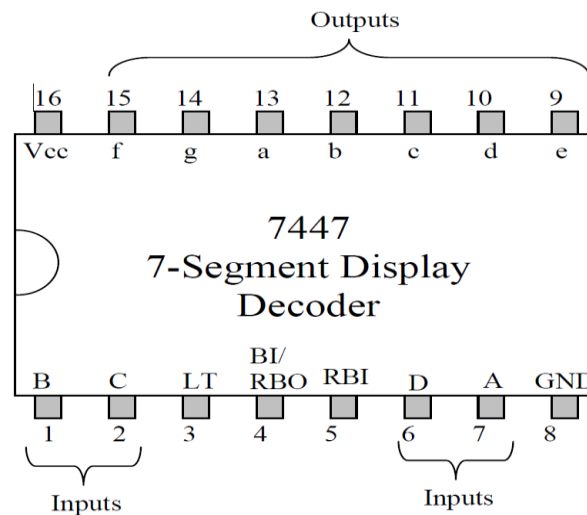


Figure 6.3: 7447 pin assignments.

Table 6.1: 7447 pin descriptions.

Pin name	Description
<b>A, B, C, D</b>	BCD inputs: D is the most significant input (DCBA)
<b>a, b, c, d, e, f, g</b>	Decoder output (Active Low)
<b>RBI</b>	Ripple Blanking Input (Active Low)
<b>BI/RBO</b>	Blanking Input (Active Low) Ripple Blanking Output (Active Low)
<b>LT</b>	Lamp Test input (Active Low)

- LT should be high for normal operation and when pulled low, all seven segments will be turned on.
- RBI must be high if blanking of a decimal zero is not desired.
- BI/RBO can be used as input or output. If BI is high, and LT is low, all 7 segments are on. This function can be used to see if all the LED segments are working. BI is used to turn off all these segments when pulled low. If A, B, C, D, and RBI are all low, and LT is high, then all 7 segments are off. In this situation, -the RBO goes low (response condition).
- For normal operation without blanking, the three inputs: LT, RBI, and BI/RBO should be connected to +5V (given that they are active low).

### 6.4.3 Counter

In this lab the IC type 7490 counter will be used. The 7490-pin assignment is shown in Figure 6.4 and reset/count function table is shown in Table 6.2.

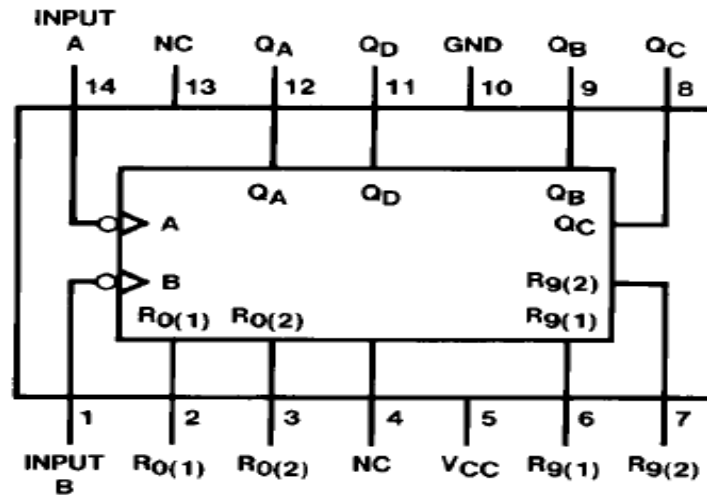


Figure 6.4: 7490 counter pin assignments.

Table 2: Reset/count function table.

Reset Inputs				Outputs			
R0(1)	R0(2)	R9(1)	R9(2)	Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

## 6.5 Procedure

### 6.5.1 BCD counter

#### A) Testing lamps in the display

1. Place the display and the 7447 chips on the breadboard.
2. Implement the circuit shown in Figure 6. 5.
3. Connect pin 4 and pin 5 of the 7447 decoders to the +5V.
4. Connect pin 3 (LT) of the 7447 decoders to the ground. All 7 segments must be turned on. (This is to verify that all segments in the display are working properly).

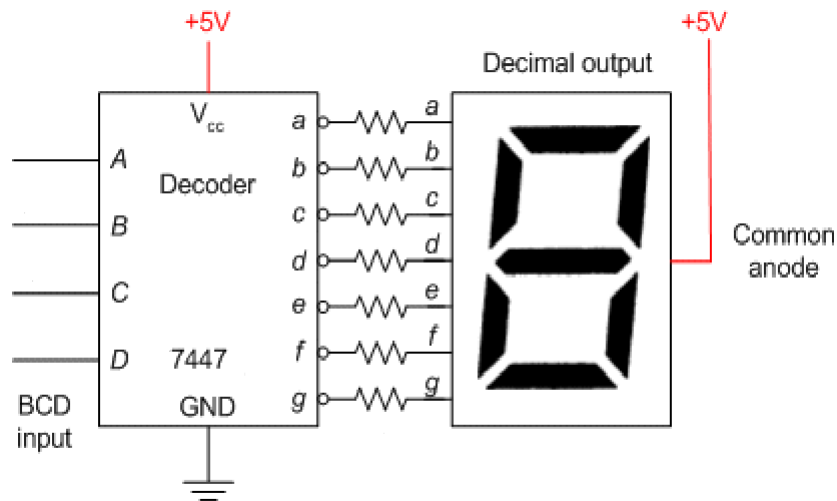


Figure 6.5: display-decoder connection.

#### B) Blanking all segments

Connect pin 4 (BI) of the decoder to the ground. All 7 segments must be turned off. You may leave the circuit connected (to be used next).

#### C) Implementing a decade counter

In this part, you will build a counter that counts from 0 to 9.

1. Connect the circuit that you designed in part 4 of the pre-lab. Show the design and the connected circuit to your instructor/assistant before turning on the power.
2. Apply clock pulses to pin 14 of the 7490 counter using Pulser Switch

(SWA).

3. Observe the counting sequence on the display D1 and complete Table 6.3
4. Apply clock pulses to pin 14 using the “pulse generator” of the KL-31001 Basic Electricity Circuit Lab. Observe the count sequence.

Table 6.3: Data for part 6.5.1 (C-4).

D1

- **Selected Tasks by the instructor.**

## 6.6 Post Lab

1. Design a two-decade counter that counts from 00 to 99.
2. Add additional input to your design that can be used to reset the counter.
3. Modify the counter to count to 59 (without Reset).



## **Faculty of Engineering and Technology**

### **Department of Electrical and Computer Engineering**

#### **ENCS 2110**

#### **Digital Electronics and Computer Organization Lab**

#### **Experiment No. 7 - Constructing Memory Circuits Using Flip–Flops**

### **7.1 Objectives**

- Understand the basic structure of Random-Access Memory (RAM).
- Understand and test the circuit of 64-bit Random Access Memory (RAM).

### **7.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab
- KL-33010 Memory Circuit Experiment Model (1)
- KL-33011 Memory Circuit Experiment Model (2)

### **7.3 Theory**

#### **7.3.1 CONSTRUCTING RANDOM ACCESS MEMORY (RAM) WITH D FLIP-FLOP**

Two different types of basic structures for RAM are given below:

- 1) In the RAM circuit of Figure 7.1 (a), the input and output are not separated. There are two control terminals: one is the R/W terminal (R for READ or OUTPUT, W for WRITE or INPUT) and the other one is the ENABLE terminal.





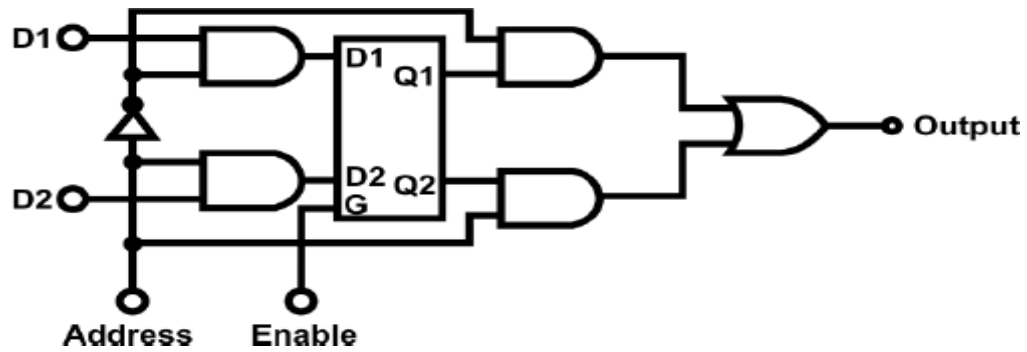


Figure 7.2: Implementation of the two-bit memory presented in Figure 1 (a).

Commercial random-access memories may have a capacity of thousands of words and each word may range from. The logical construction of a large capacity memory would be a direct extension of the configuration as shown in Figure 7.3. The two address inputs go through a 2x4 decoder to select one of the four words.

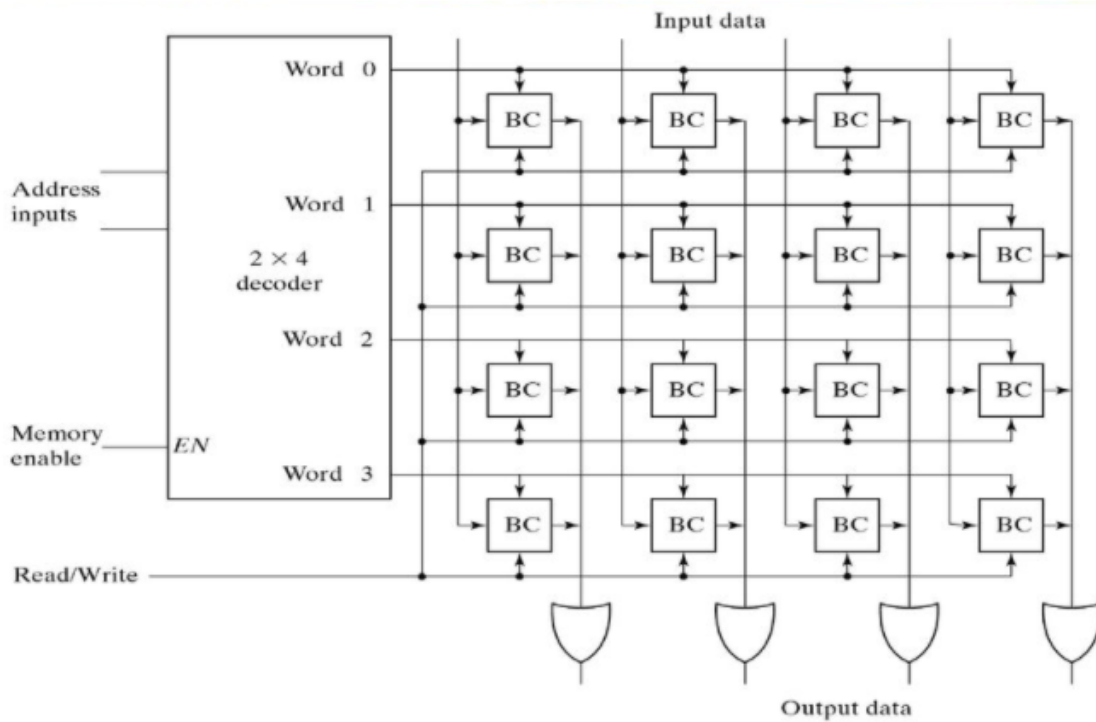


Figure 7.3: Logical construction of a 4x4 RAMS.

### 7.3.2 64-BIT RANDOM ACCESS MEMORY (RAM) CIRCUIT

Like ROM, RAM is also a memory element. The data selection process is controlled by the address selectors. The length of data is related to the number of data variations. For example, if there are 4 data then  $2^4$  or 16 data variations exist.

The number of address lines determines the number of locations. If there are 4 address lines, then  $2^4$  or 16 locations exist. A 4-bit data can be stored in each location, since the total capacity is  $16 \times 4$ , where the 4 is the number of data while 16 is the number of address lines.

Figure 7.4 shows the 7489 IC, which is a  $16 \times 4$  memory with 64 memory capacities. Also, its function table.

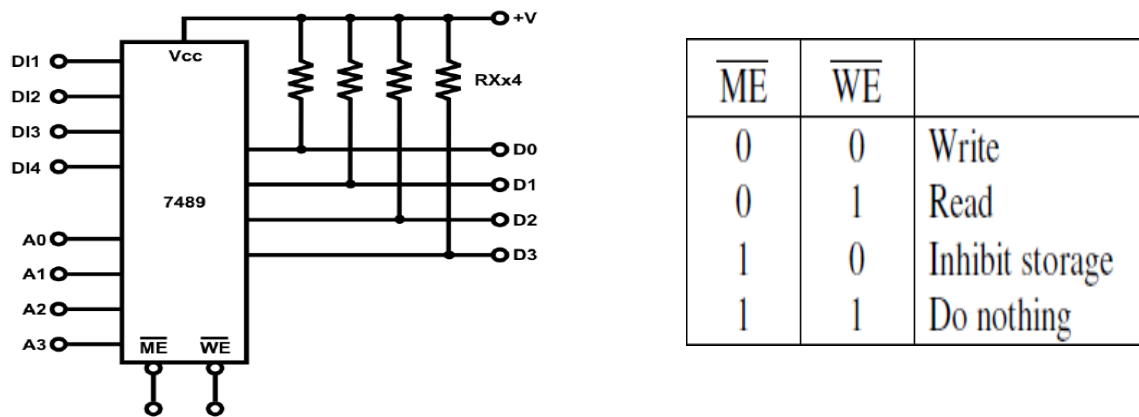


Figure 7.4: Block diagram of  $16 \times 4$  RAM chip.

- When  $\overline{ME}' = 0$  and  $\overline{WE}' = 0$ , the memory is enabled, and the input process starts. The input and output terminals are separated. The output terminals are open-collector type so resistors " $R_{X \times 4}$ " must be added to the supply voltage. Since the output terminal of 7489 is open-collector type, the outputs can be connected in parallel, as shown in Figure 7.4. The operating sequence will be controlled by  $\overline{ME}'$  and  $\overline{WE}'$ .
- When  $A_4A_5=00$ , A is selected,  $\overline{ME}'$  and  $\overline{WE}'$  of B, C and D all equal to "1". Similarly, when  $A_4A_5=01$ , B is selected,  $\overline{ME}'$  and  $\overline{WE}'$  of C and D all equal to "1". E is 2-4 decoders with "0" as its output. The unselected outputs are in high or "1" state.
- Since the outputs will have high impedance when  $\overline{ME}'$  and  $\overline{WE}'$  are both "1", each  $R/W'$  control of 7489 are connected to an OR gate to ensure that when  $\overline{ME}' = "1"$ ,  $\overline{WE}'$  will be equal to "1" too.

- 

Uploaded By: Ahmad K Hamdan

## 7.4 Procedure

### 7.4.1 Latches and Flip Flops

A) Block g of module KL-33010 which is shown in Figure 7.6 will be used in this part.

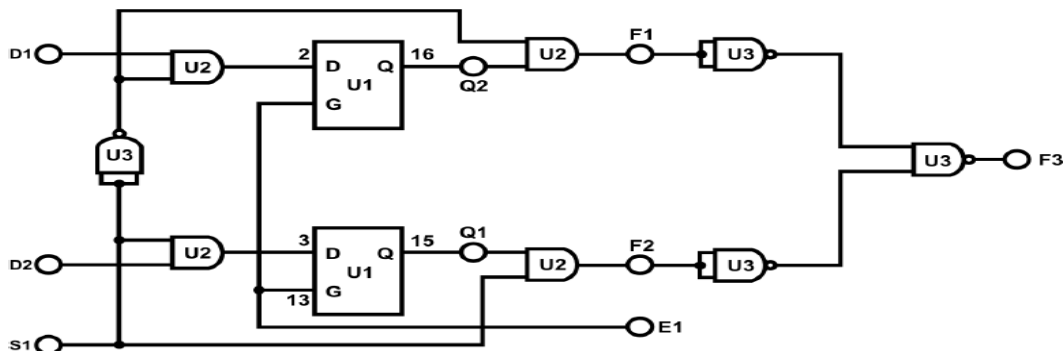


Figure 7.6: RAM block with D Flip-Flop of module KL-33010.

B) Connect E1, S1, D2, D1 to Data Switch SW0~SW3 respectively. Connect outputs F1, F2, F3 to Logic Indicators L1~L3. Refer to the input sequence in Table 7.1 and record all outputs. Discuss and explain the results to your TA.

Table 7.1. Extracted data observations and storing process of a 1x2 RAM

Input				Output		
E1	S1	D2	D1	F3	F2	F1
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

C) Then, set module KL-33011 and locate the block b. Insert connection clip according to

Figure 7.7, connect +5V, +15V of module KL-33011 to the +5V, 15V output of fixed power supply respectively.

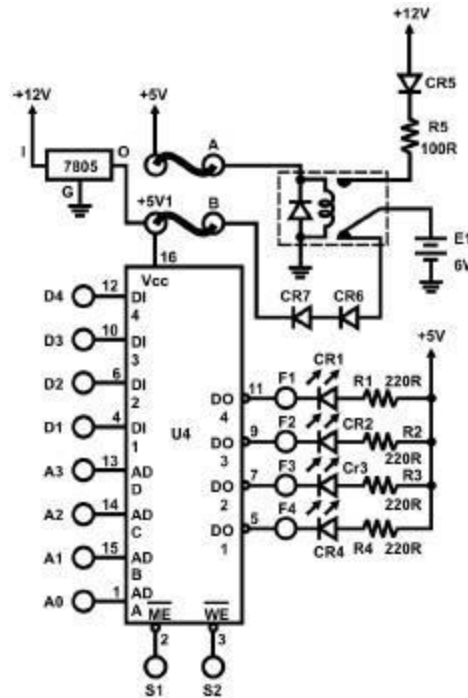


















Figure 7.7: Open Collector 4x4 RAM.

















- D) Connect inputs D4~D1 to DIP Switch 1.0~1.3; A3~A0 to DIP 2.0~2.3; S1 (ME) to Pulser Switch SWAA; S2 to Data Switch SW0. Outputs are indicated by CR1~CR4.
- E) Set SW0 (WE') to "0" for the "WRITE" task. Start from address 0000, and input data to A0~A3 by setting the DIP switch. Activate SWA once to write the data into its assigned address. Repeat this process for all the addresses, ending with 1111. Record what was written into each address in Table 7.2 under the "WRITE" column.

Table 7.2: Data for part 7.4.1 (E).

Address				Write					
A3	A2	A1	A0	$\overline{\text{ME}}$	$\overline{\text{WE}}$	D4	D3	D2	D1
0	0	0	0		0				
0	0	0	1		0				
0	0	1	0		0				
0	0	1	1		0				
0	1	0	0		0				
0	1	0	1		0				
0	1	1	0		0				
0	1	1	1		0				
1	0	0	0		0				
1	0	0	1		0				
1	0	1	0		0				
1	0	1	1		0				
1	1	0	0		0				
1	1	0	1		0				
1	1	1	0		0				
1	1	1	1		0				

F) Now set SW0 (WE') to "1" for the "READ" task and connect S1 (ME') to Pulser Switch SWA. Observe the states of CR1~CR4 and record them under the "READ" column in Table 7.3.

Table 7.3: Data for part 7.4.1 (F).

Address				Read					
A3	A2	A1	A0	$\overline{ME}$	$\overline{WE}$	F4	F3	F2	F1
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		1				
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				

- G) Disconnect SWA and "A" clip, then turn off the main power switch of IT- 3000 for about 10 seconds and turn it on again. Change the address and press SWA then attempt to read the data. Are they still stored in the RAM?
- H) Disconnect the "B" clip, and VCC disappears. Repeat Step 5 to see if the data are still stored in the RAM.

- **Selected Tasks by the instructor.**



## 7.5 Post Lab

1. Design a 4x16 RAM using four 4x4 RAMS.
2. Although D latches are useful for storing binary information, they are not used in RAM circuit design, why?



**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 8 - Introduction to QUARTUSII Software**

### **8.1 Objectives**

- To learn how to use QUARTUS II and write code using Verilog HDL language.
- To learn how to test code and make symbols from code.
- To learn about FPGA and how to download code from QUARTUS II to FPGA

### **8.2 Equipment Required**

- QUARTUS II program.
- FPGA Board

### **8.3 Pre Lab**

- 1) Here is a link to an HDL tutorial:  
<https://www.youtube.com/playlist?list=PLnyw1IVZpaTukmt80aNs7gT74U3vboDYr>
- 2) Quartus II Introduction:  
<https://www.youtube.com/watch?v=uG1GTRelG3I>  
<https://www.youtube.com/watch?v=TdLqbgrVREQ>
- 3) Download Quartus from this link below:  
[http://www.mediafire.com/file/eqd7xidoan3exqv/90\\_quartus\\_free.exe](http://www.mediafire.com/file/eqd7xidoan3exqv/90_quartus_free.exe)
- 4) Using Quartus to build the following circuit:

- a) Build half adder on data flow.
- b) Build the Full adder using half adder structural.
- c) Build a 2-bit counter on behavioral.
- d) Build an 8x1 Multiplexer on behavioral.
- e) Build a 2x4 decoder using basic gates (structural).
- f) Show the waveform for the above parts.

## 8.4 Theory

### 8.4.1 QUARTUS II Program

Quartus enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.



Figure 8.1: QUARTUS II logo.

### 8.4.2 FPGA Board

FPGA stands for field-programmable gate array. That's quite a mouthful, so let's start with a basic definition. Essentially, an FPGA is a hardware circuit that a user can program to carry out one or more logical operations. Taken a step further, FPGAs are integrated circuits, or ICs, which are sets of circuits on a chip—that's the "array" part. Those circuits, or arrays, are groups of programmable logic gates, memory, or other elements.

With a standard chip, such as the Intel Curie module in an Arduino board or a CPU in your laptop, the chip is fully baked. It can't be programmed; you get what you get. With these chips, a user can write software that loads onto a chip and executes functions. That software can later be replaced or deleted, but the hardware chip remains unchanged.

For FPGA, the programming can be a single, simple logic gate (an AND or OR function), or it can involve one or more complex functions, including functions that, together, act as a comprehensive multi-core processor.

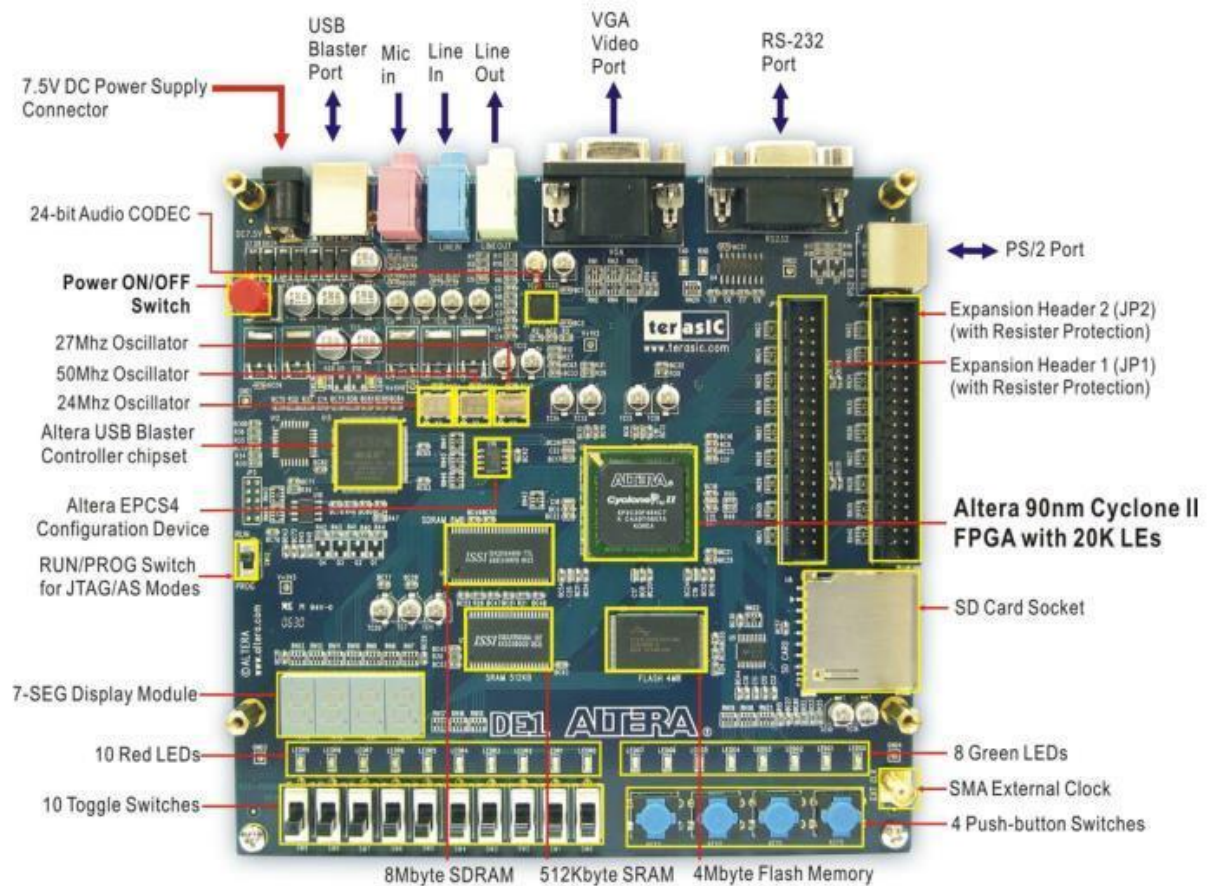


Figure 8.2: FPGA Board.

## 8.5 Procedure

### 8.5.1 How to create project in QUARTUS II

- **Step1:** Run the QUARTUSII software: double click on the QUARTUS II item in the desktop.
- **Step2:** When you open QUARTUS II the windows in Figure 8.3 will appear, then select option Create New Project Wizard or you can start new project by select option File then select New Project Wizard as shown in Figure 8.4.



Figure 8.3: How to create new project (first way).

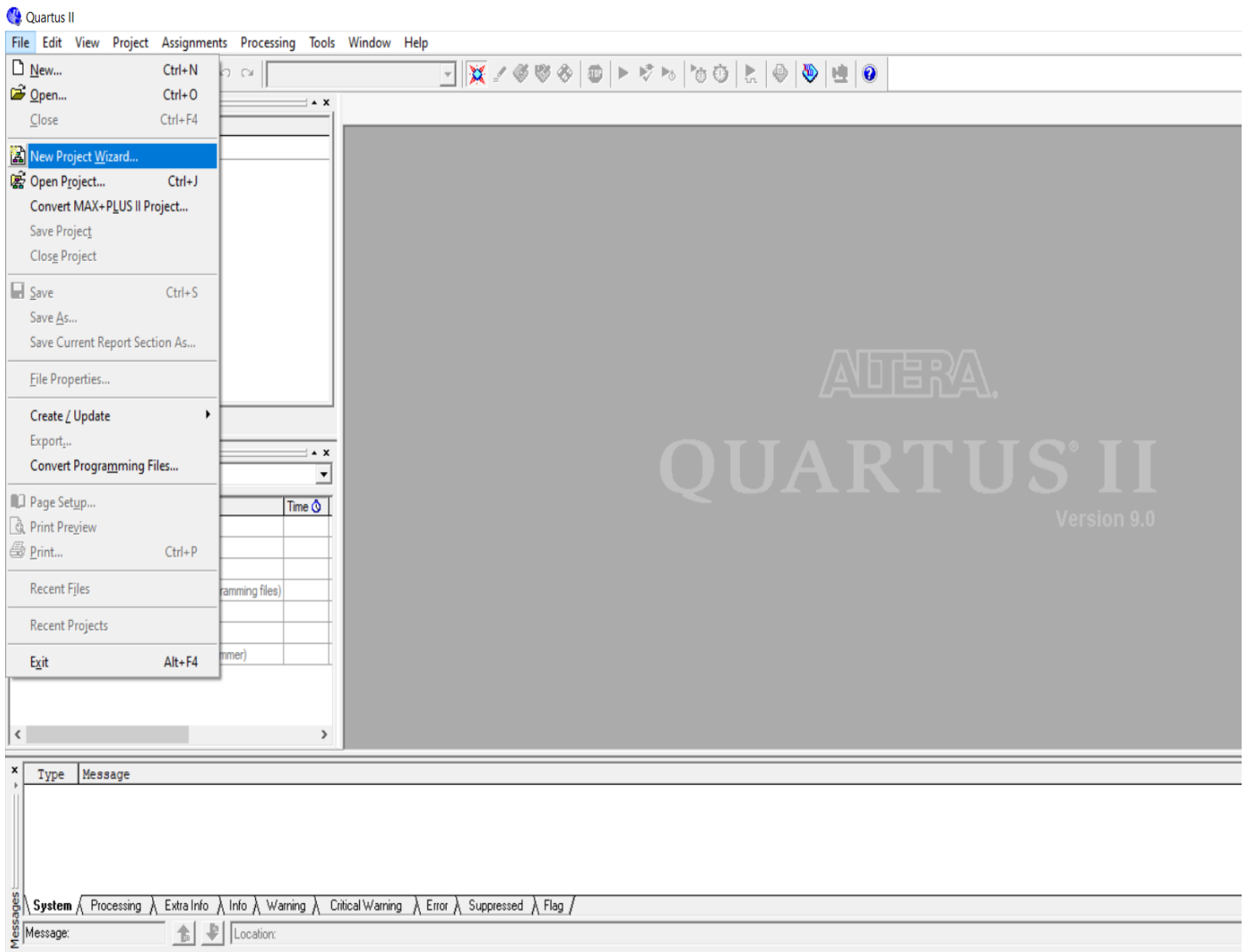


Figure 8.4: How to create new project (second way).

- **Step3:** Then the window in Figure 8.5 will be shown, press next for First window.

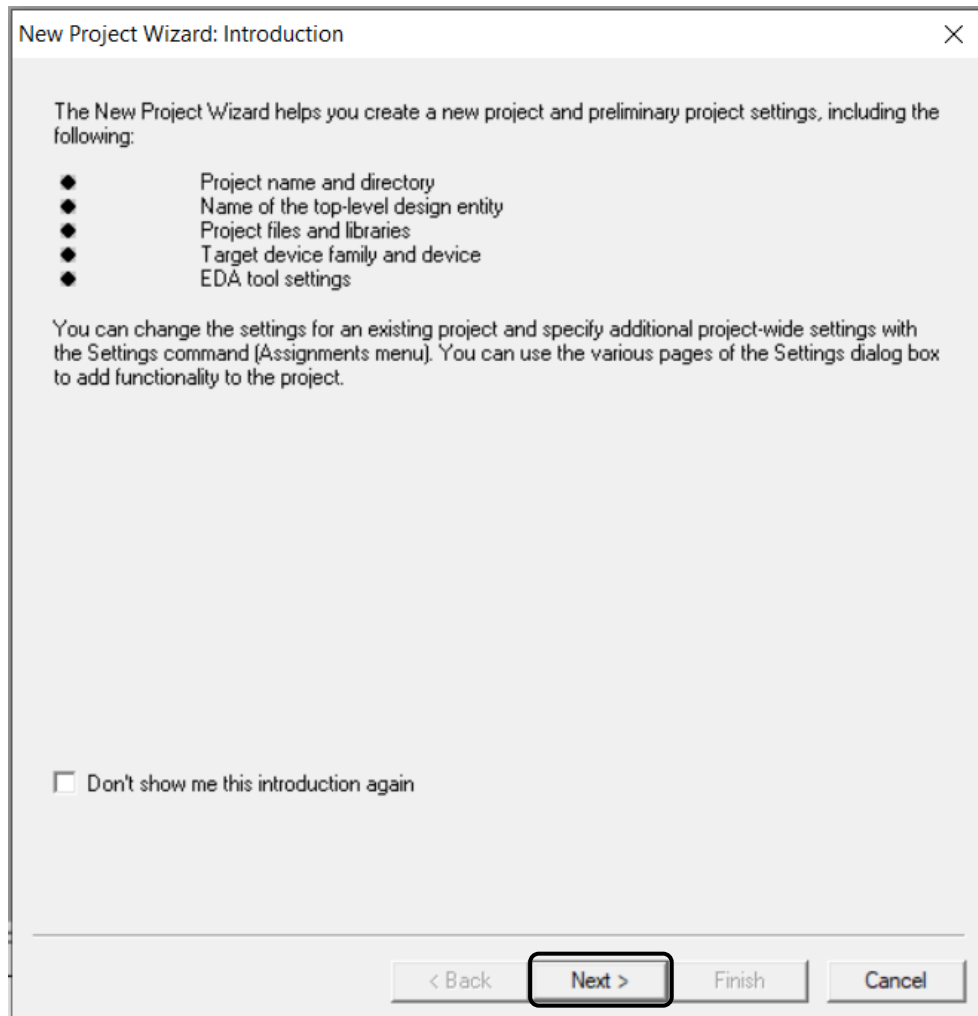


Figure 8.5: First Windows.

- **Step4:** Then Later, a window will appear asking you to enter the project storage location and the name of the project. After filling out the information, you will be clicked on the next option at the bottom of the screen. As shown in Figure 8.6. Then another window will appear asking to enter previously existing files. If there are no files, click on the Next option.as shown in Figure 8.7.



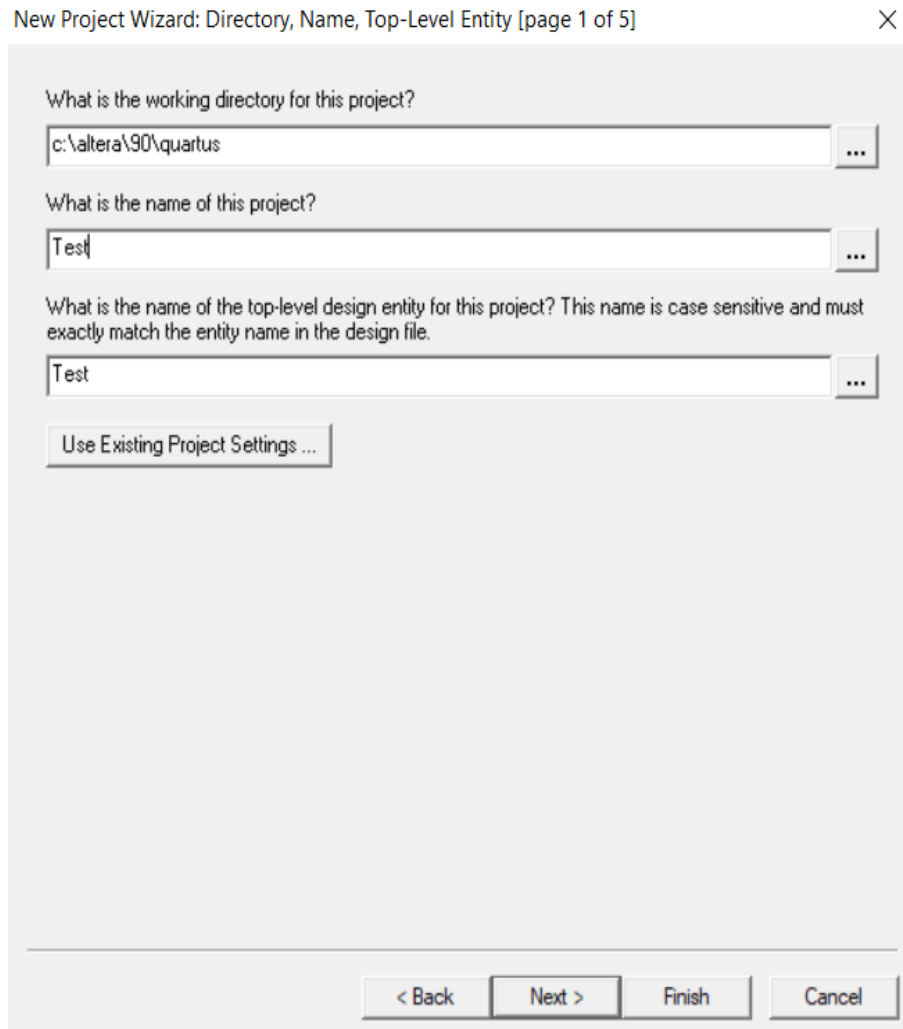


Figure 8.6: Project Folder, Project Name, Top-Level-Entity.

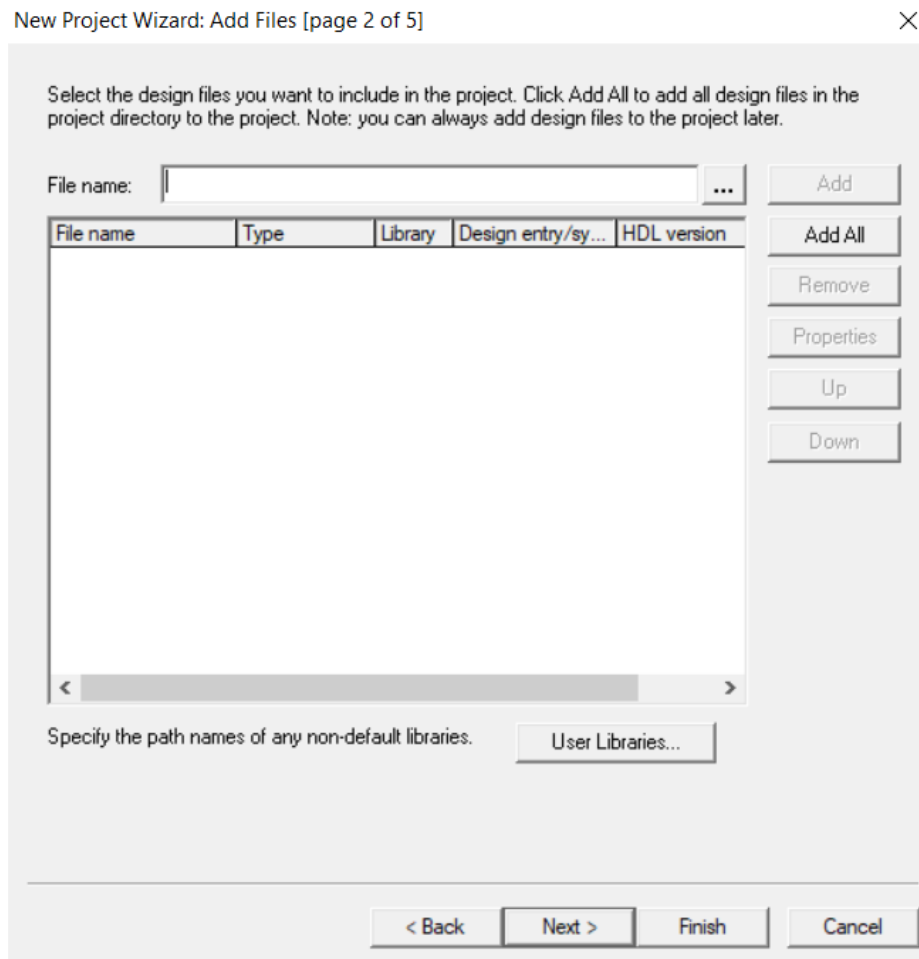


Figure 8.7: Add existing files.

- **Step5:** In Figure 8.8, show the windows where we can choose the family of FPGA we went to use and the number of FPGA, then select Finish.

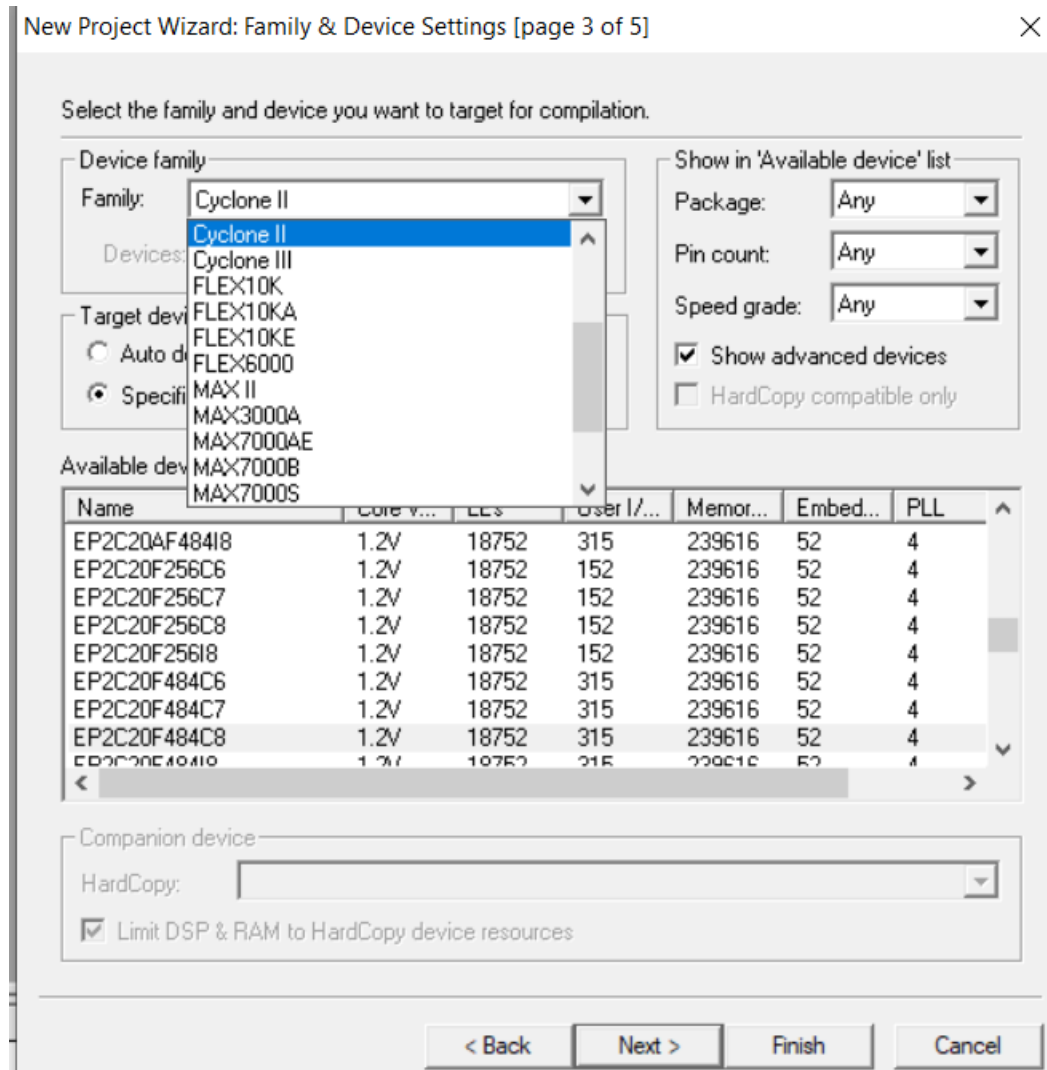


Figure 8.8: choose family and number for FPGA.

## 8.5.2 How to write code using Verilog HDL

1. To make new File, press File > New, the window in Figure 8.9 will appear.

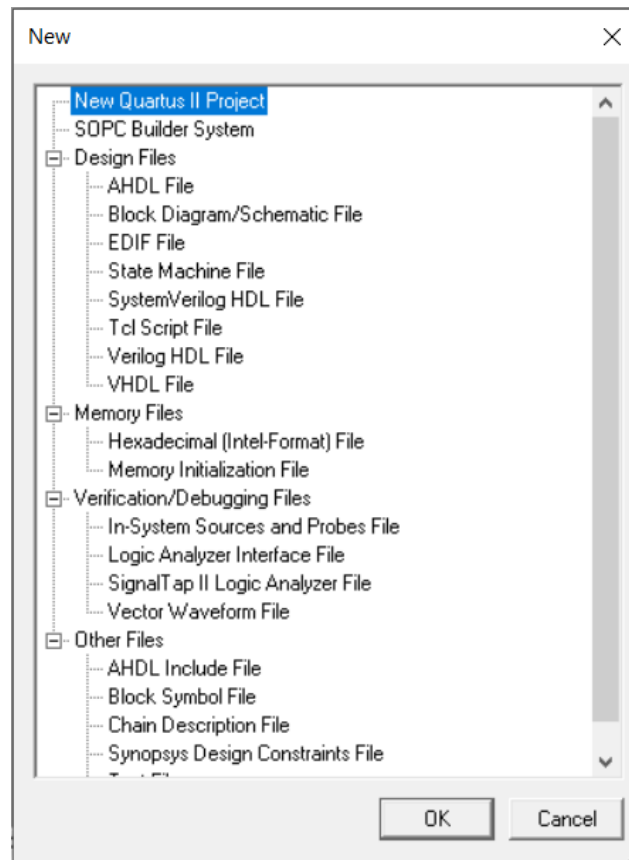


Figure 8.9: to create new files.

- **There are three choice in this window:**
    - A) Verilog HDL File:** use to write the code Verilog HDL.
    - B) Vector Waveform File:** use to test code.
    - C) Block Diagram /Schematic File:** to make symbol from code.
2. For this lap we choose **Verilog HDL** File from Figure 8.9. Then a white screen will appear on which we will write the code. the Figure 8.10 show simple code for half adder.

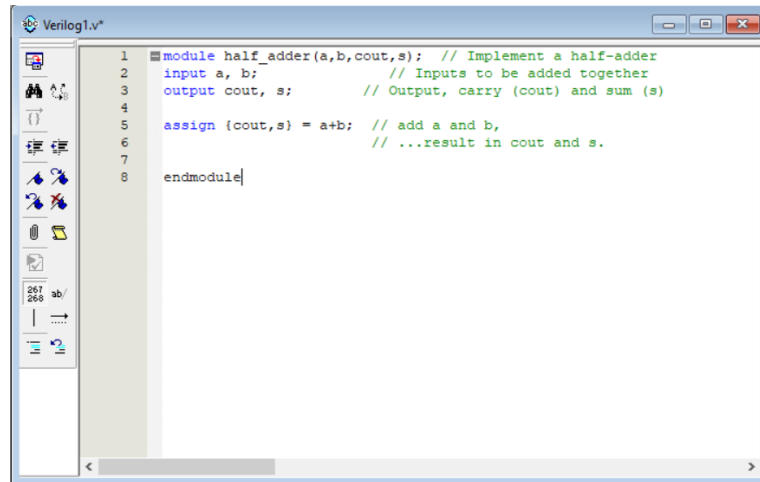


Figure 8.10: simple code for half adder.

**Important note: This file is used to enter your Verilog code; you must save the file as the name of the module. Look to Figure 8.11.**

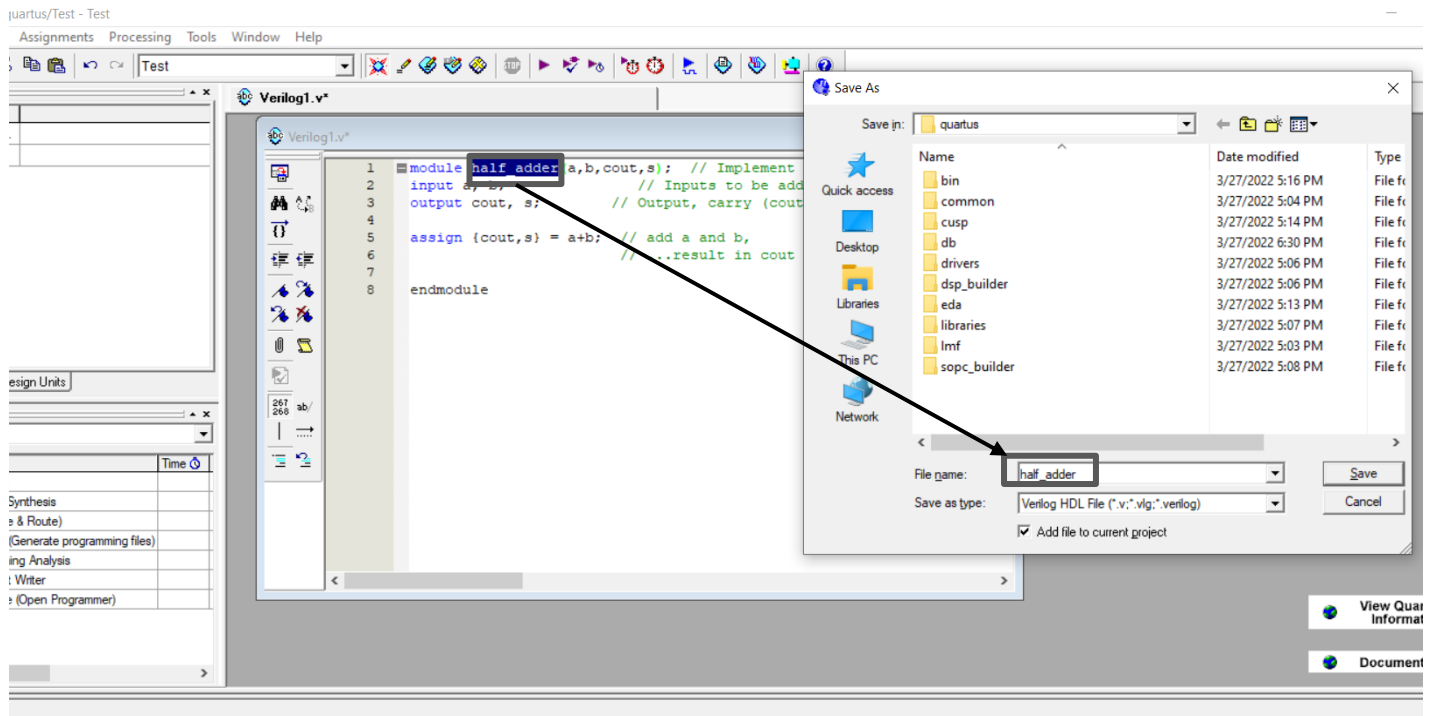



Figure 8.11: how to save file.

- To run code Right click on the file name and select option **Set as Top-Level Entity** then click (start compilation) select this icon  in top of the screen or clicking on processing> start compilation. If there is any error, you can see in white area in bottom of screen.

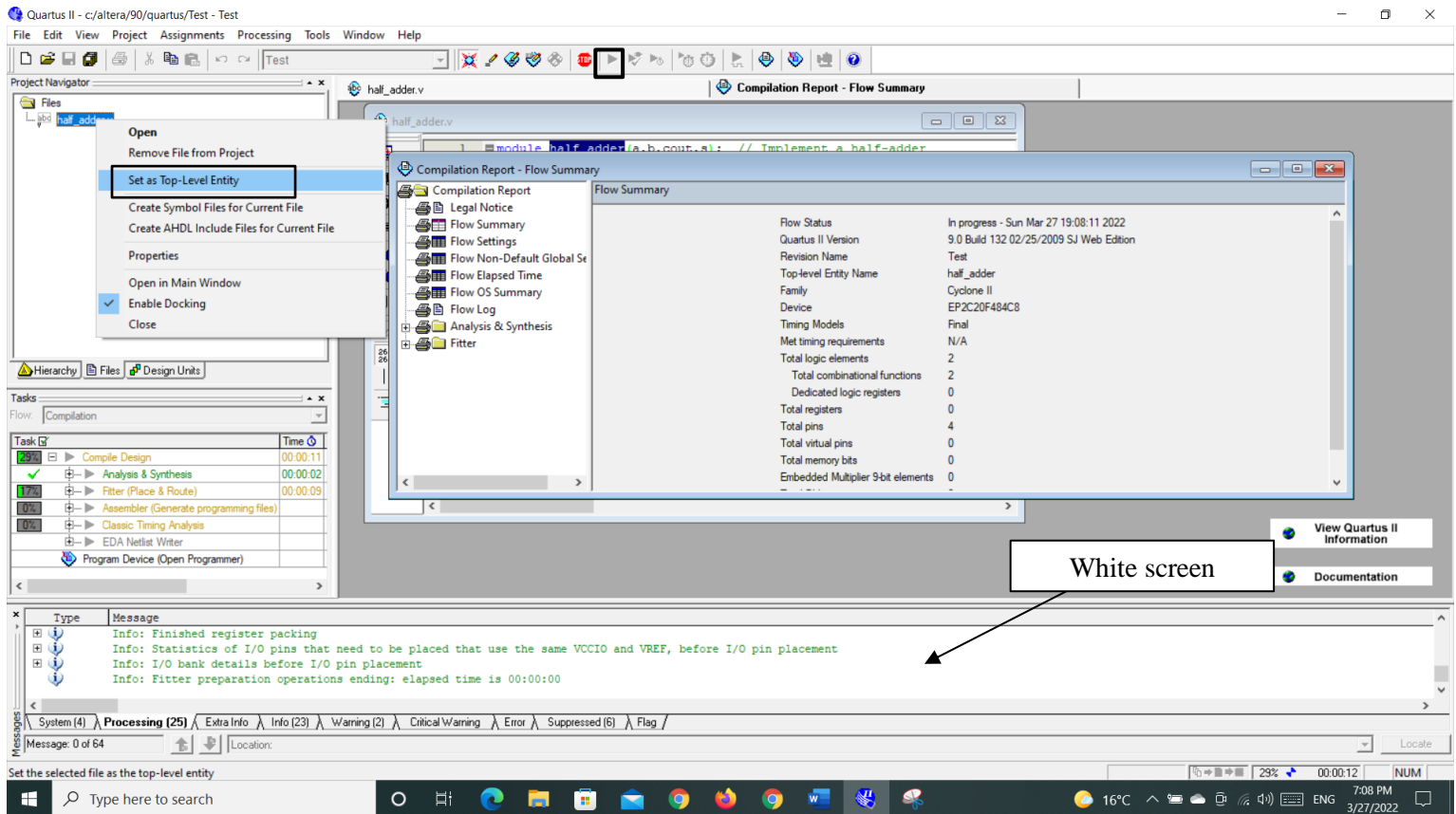


Figure 8.12: how to run code.

### 8.5.3 How to test code

- To test code, we choose **Vector Waveform File** from Figure 8.9. This File is used to check the functionality of the design works as expected or not. Figure 8.13 shows a Vector waveform File.

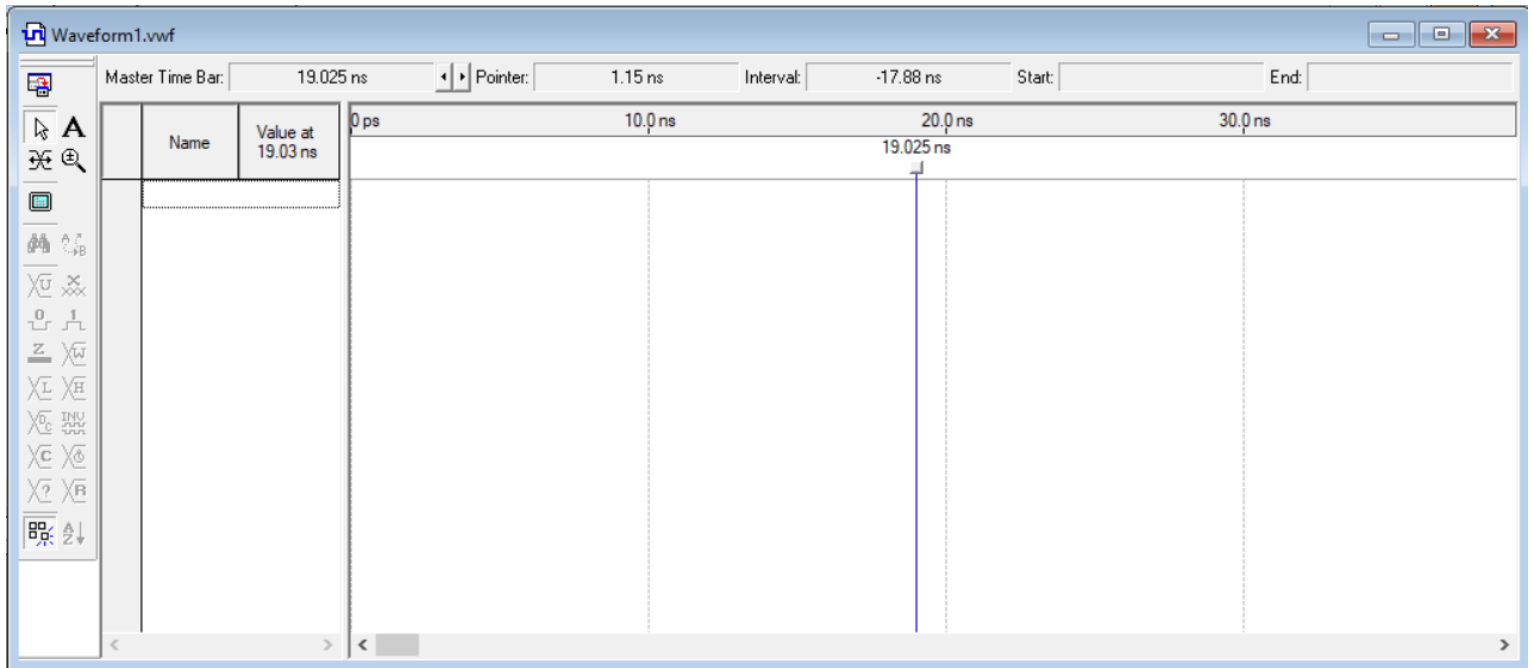


Figure 8.13: Vector waveform File.

2. Then right-click on the left most side of the window (under Name), then click insert node or bus, as shown below

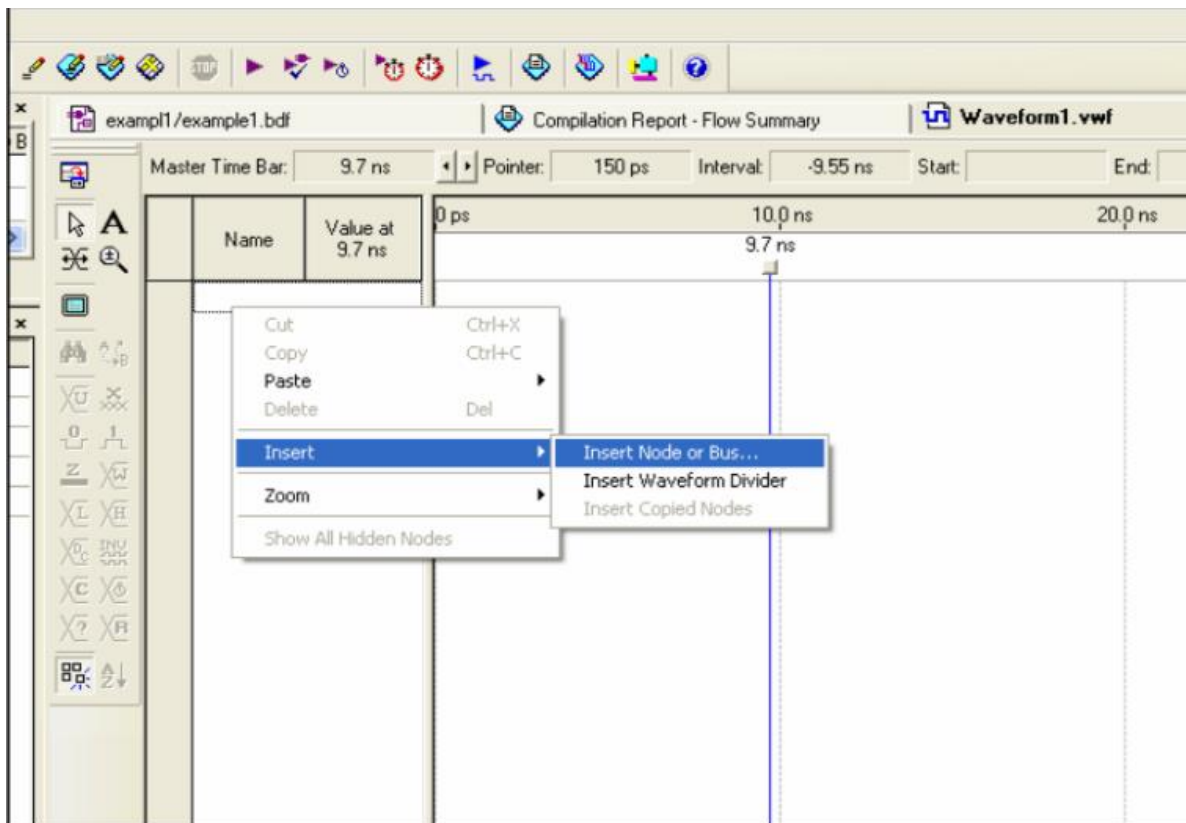


Figure 8.14: insert node or bus.

- Click on **Node Finder** and then on List (using the Filter pins: **all**) and note that file name in look in option as shown in Figure below. Select all inputs and the output by clicking on (>>) (you can select one by one).

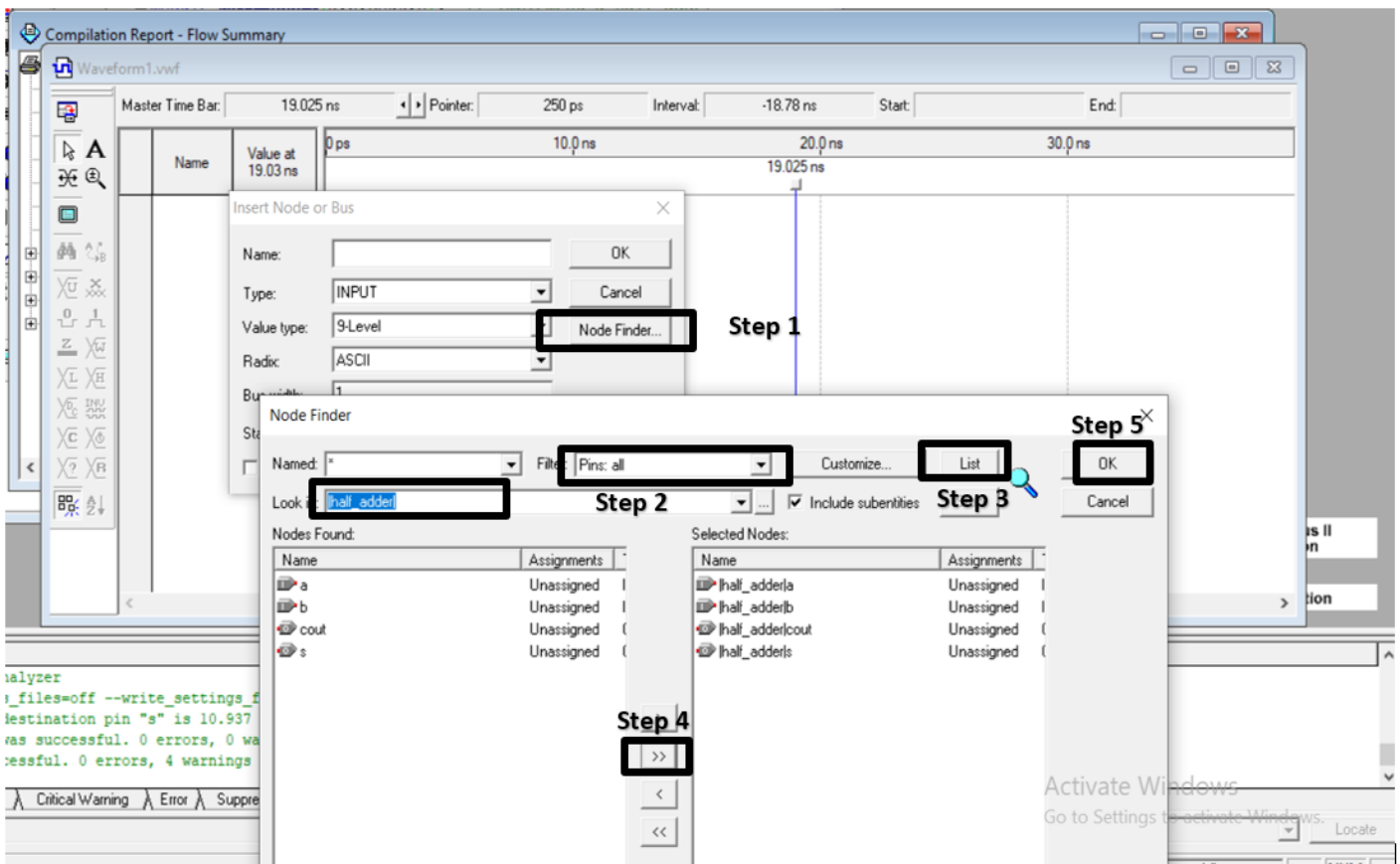


Figure 8.15: Select inputs and outputs.

- You can select the intervals when you want the inputs to be one or zero, either by shadowing the interval, then press the one level in the tool bar, or by write clicking the name, then select value > count value, the change the start value, the end value, and the radix as shown in the following figure:



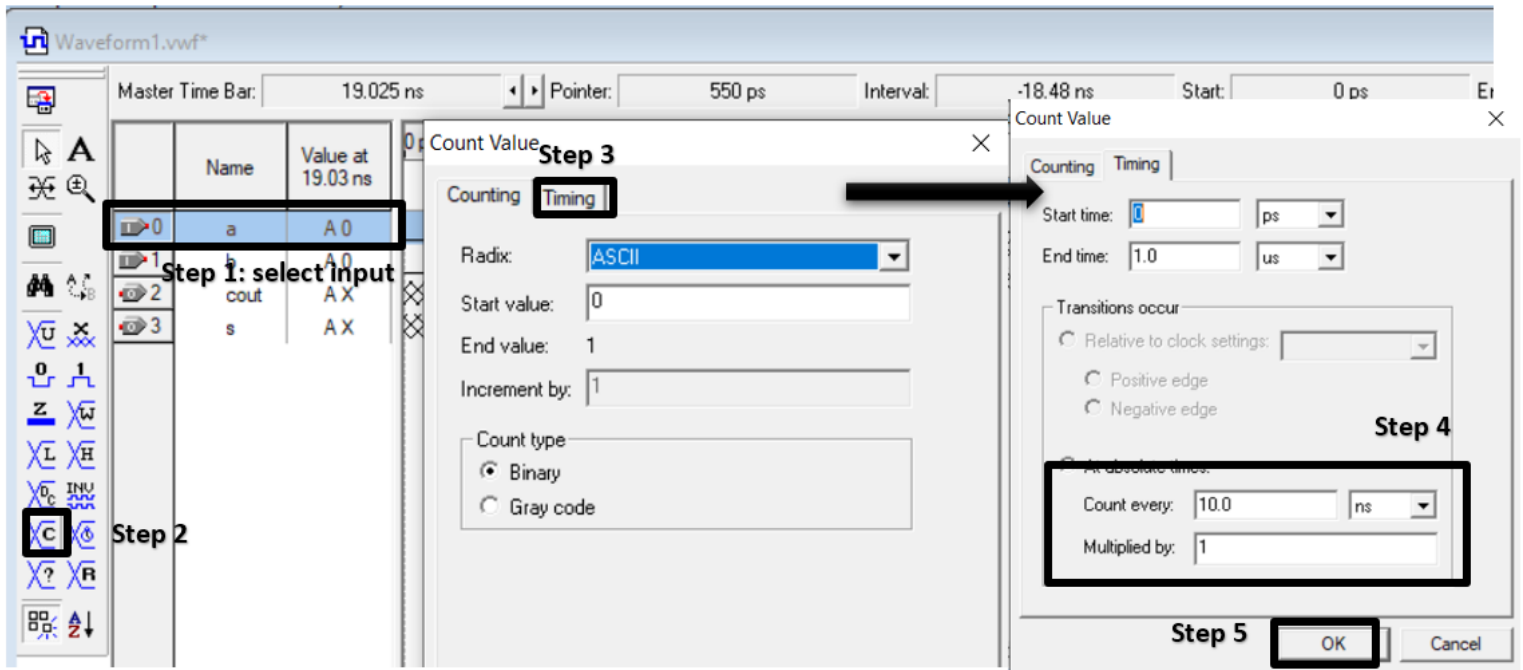


Figure 8.16: put value for inputs.

- Now save the file with the same name as your project and in the same folder.
- Then, from Processing > Simulator Tool, the window in Figure 8.14 will appear:

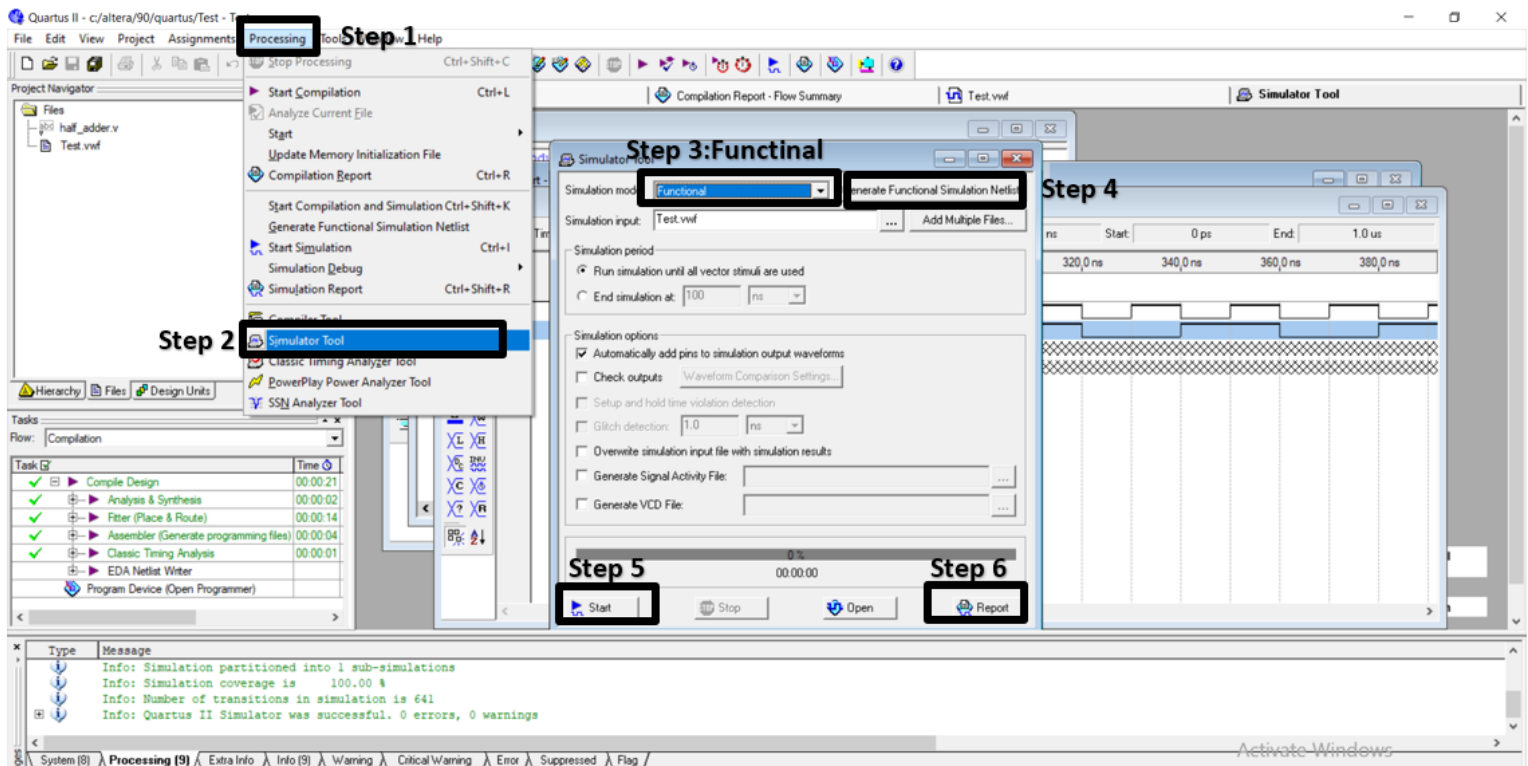


Figure 8.17: to show result.

7. The generated waveform is inserted as simulation input, then press on Generate Functional simulation Netlist. Then Start after the simulation finish press on Report to see the output.

#### 8.5.4 How to make symbol from code, make diagram and Schematic File

1. To start new File, press File > New, the window in Figure 8.9 will appear press **Block Diagram /Schematic File**.
2. The window in Figure 8.18 will appear, to enter components of our design double click anywhere on the schematic window or select the symbol tool (The little and gate).

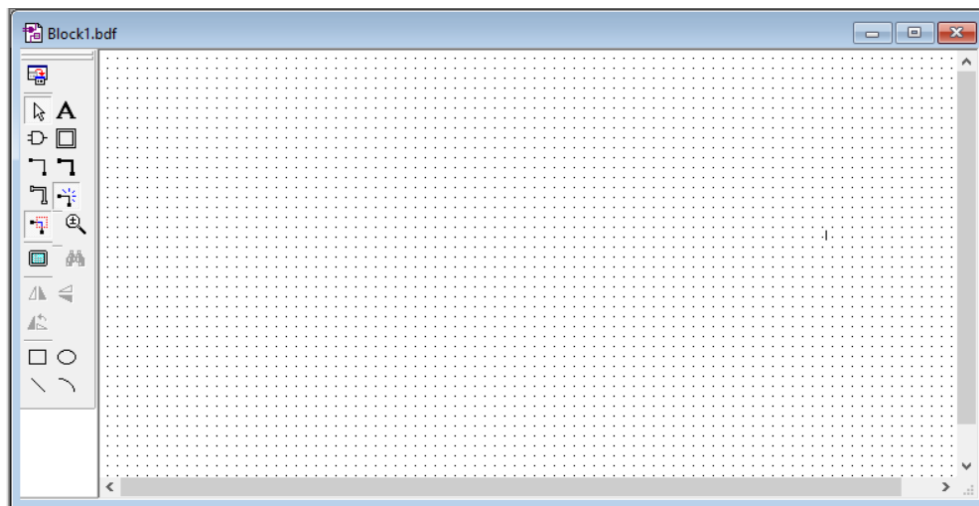


Figure 8.18: diagram window.

3. This opens the symbol window in which available libraries, including the standard QUARTUSII library, open this library by clicking on the little plus sign next to it, then select primitives, and then select logic, then select the gate you want in your implementation.as shown in Figure 8.19.

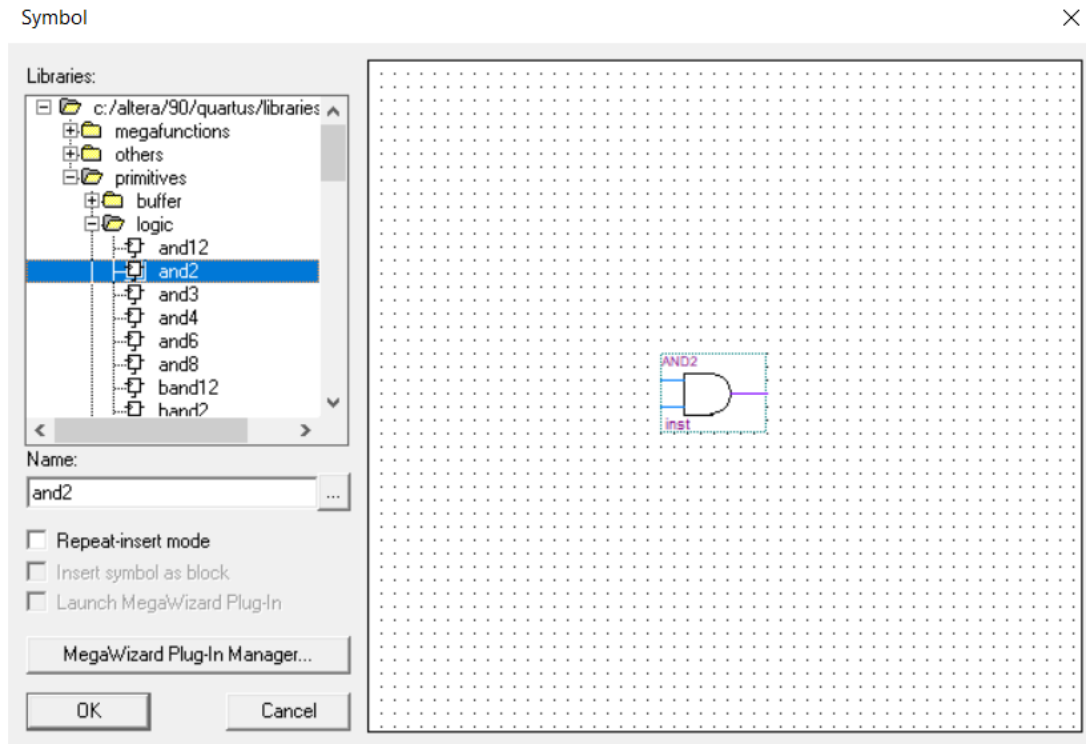


Figure 8.19: choose components.

4. You have to define the inputs and outputs of your implementation, and you do so by opening the symbol **window> primitive> pin**, the following figure shows all the design components that must now be connected, we will build half adder using gates:

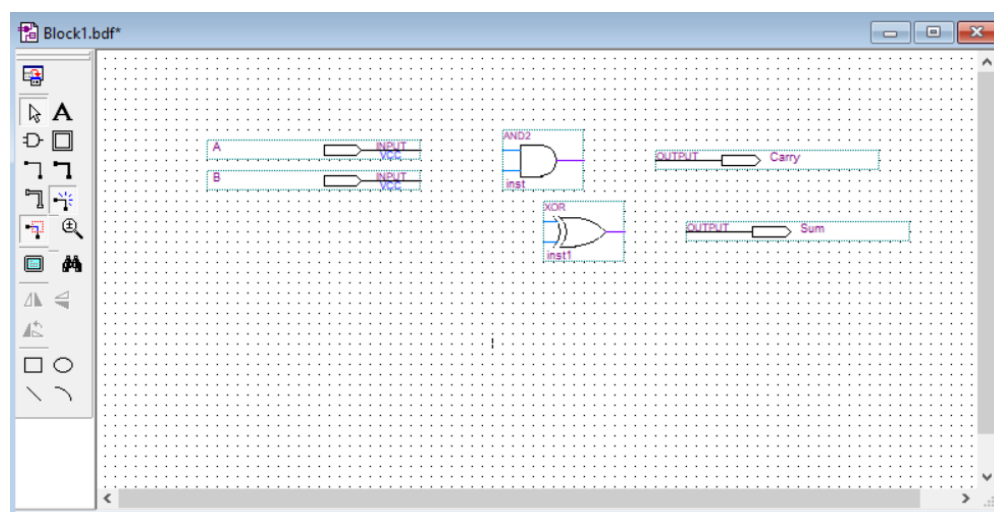


Figure 8.20: build half adder using gates.

5. To connect the components of the previous figure, select the 90° thin line with the dots on its ends on the tool bar, this makes your cursor wiring tool that can be

used to connect your circuit. When done, disable the wiring tool by clicking the arrow on the tool bar.

6. Rename input and output ports to the variable names of our design, to name a pin, either double click it to open its pin properties window, or right-click it, and select properties from the pull-down menu that shows up.
7. Save your design, and make sure it is named the same as your project, the following figure shows the completed block diagram of our design.

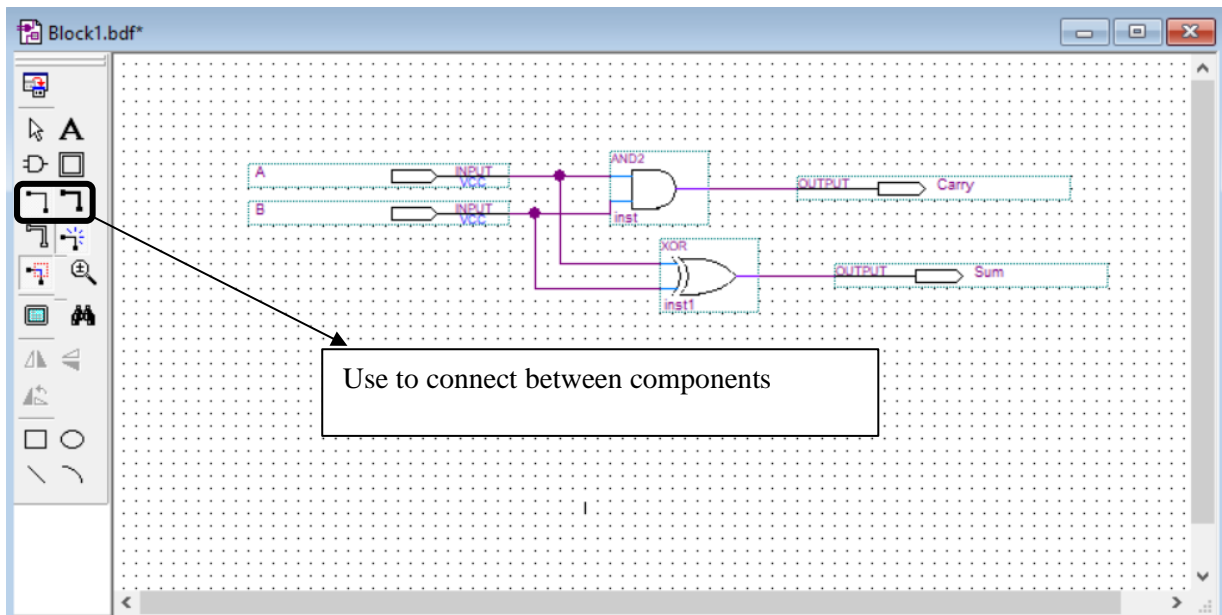



Figure 8.21: connect components.

8. To run first Right, click on the file name and select option **Set as Top-Level Entity** then click (start compilation) select this icon  in top of the screen or clicking on processing> start compilation. If there is any error, you can see in white area in bottom of screen.
9. To test diagram, we repeat the same steps in previous section.

10. We can make symbol from code first Right click on the file name and select option **Set as Top-Level Entity** then right click in name of code file and choose **Create Symbol File for Current File**. as shown in Figure 8.22.

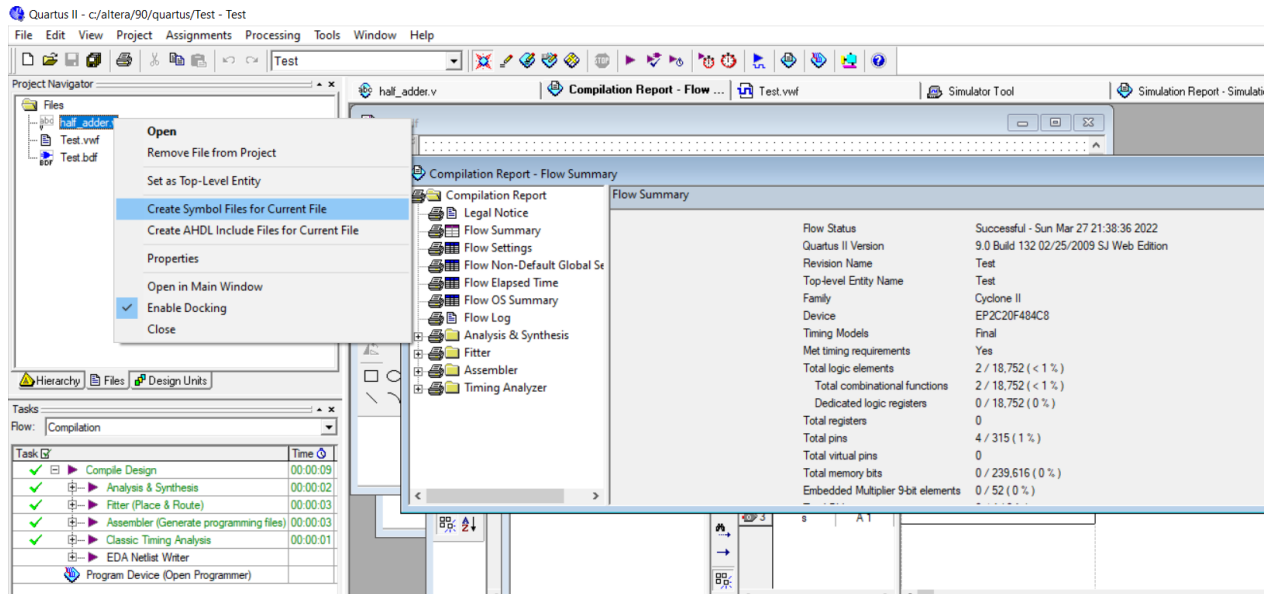


Figure 8.22: create symbol.

11. we want to use this symbol in diagram we search name of symbol (code name).as shown in Figure 8.23.

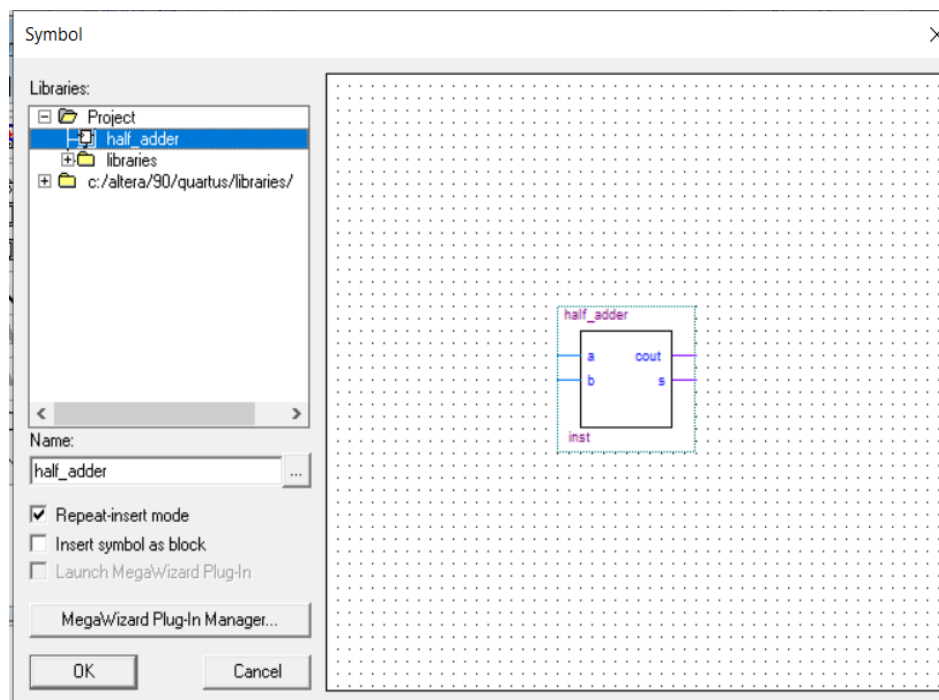


Figure 8.23: use symbol in diagram.

## 8.5.5 How to Download code in FPGA and choose inputs and outputs in board

1. After verifying that the design works as expected we can install the code on the hardware (FPGA).
2. To check that the correct device is selected, this can be done using **Assignments > Device**.
3. We need 2 switches for A and B (Inputs). 2 LEDs to see the output are needed. We have to use the user manual of DE1 to figure out the location of the switches and the LEDs. The Tables below show the location of the switches.

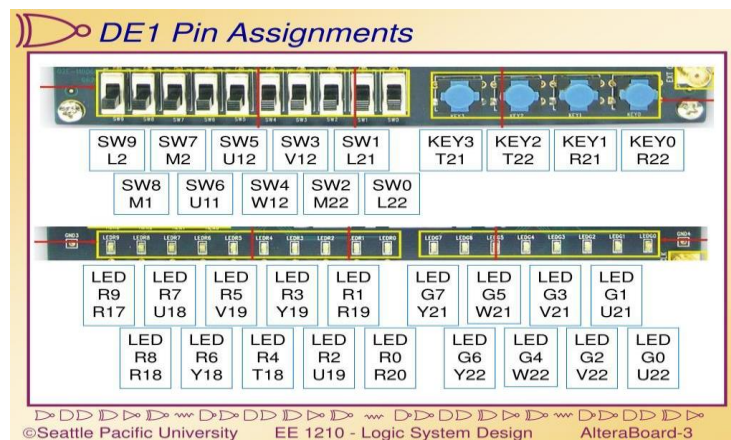


Figure 8.24: pin assignment.

*Table 2-11. Toggle Switch FPGA Pin Connections*

Switch	FPGA Pin	Description
SW[0]	PIN_L22	Toggle Switch(0)
SW[1]	PIN_L21	Toggle Switch(1)
SW[2]	PIN_M22	Toggle Switch(2)
SW[3]	PIN_V12	Toggle Switch(3)
SW[4]	PIN_W12	Toggle Switch(4)
SW[5]	PIN_U12	Toggle Switch(5)
SW[6]	PIN_U11	Toggle Switch(6)
SW[7]	PIN_M2	Toggle Switch(7)
SW[8]	PIN_M1	Toggle Switch(8)
SW[9]	PIN_L2	Toggle Switch(9)

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_R20	LED Red[0]
LEDR[1]	PIN_R19	LED Red[1]
LEDR[2]	PIN_U19	LED Red[2]
LEDR[3]	PIN_Y19	LED Red[3]
LEDR[4]	PIN_T18	LED Red[4]
LEDR[5]	PIN_V19	LED Red[5]
LEDR[6]	PIN_Y18	LED Red[6]
LEDR[7]	PIN_U18	LED Red[7]
LEDR[8]	PIN_R18	LED Red[8]
LEDR[9]	PIN_R17	LED Red[9]
LEDG[0]	PIN_U22	LED Green[0]
LEDG[1]	PIN_U21	LED Green[1]
LEDG[2]	PIN_V22	LED Green[2]
LEDG[3]	PIN_V21	LED Green[3]
LEDG[4]	PIN_W22	LED Green[4]
LEDG[5]	PIN_W21	LED Green[5]
LEDG[6]	PIN_Y22	LED Green[6]
LEDG[7]	PIN_Y21	LED Green[7]

Figure 8.25: Pin Assignment for Switches.

Figure 8.26: Pin Assignment for LEDs.

4. To select which inputs and outputs we use we select Assignments > Assignments Editor then the window in below will appear.

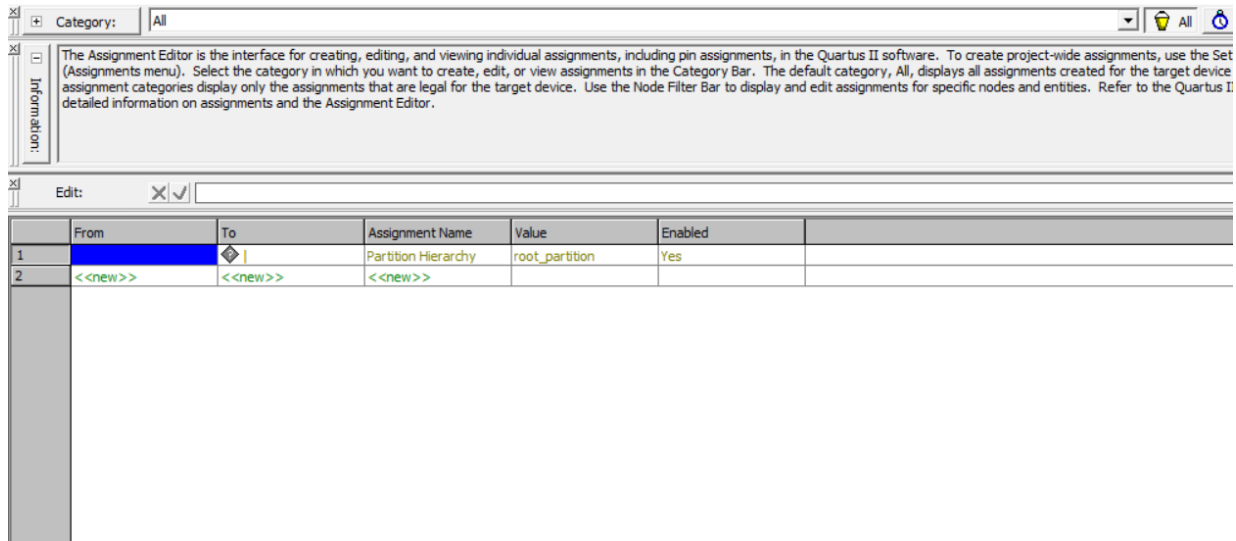


Figure 8.27: Pin Assignment Editor.

5. Assign PINs for the Inputs and Outputs. You can assign them by selecting all the schematic file then right click Locate > Locate in assignment editor, the following Figure appears:

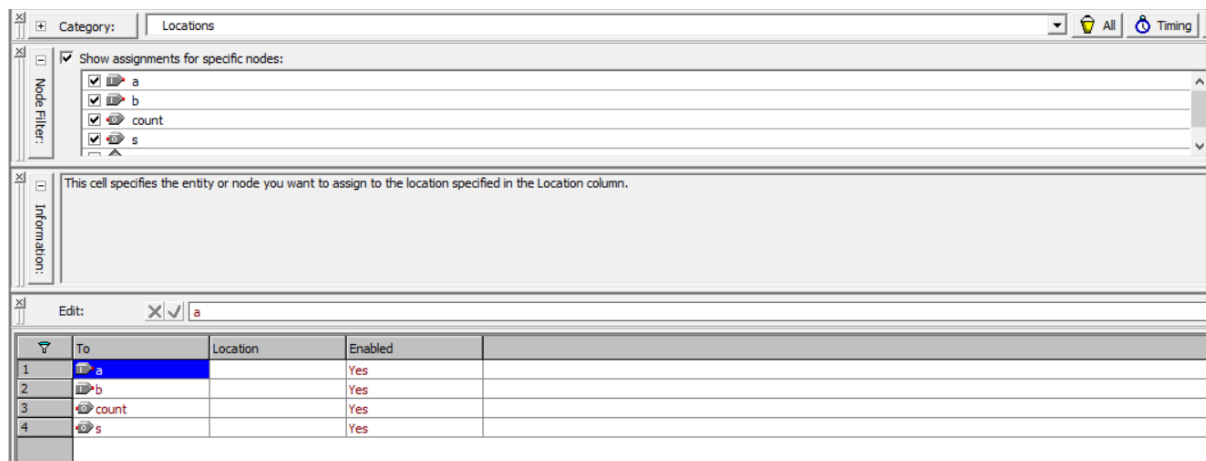


Figure 8.28: choose inputs and outputs.

6. Select Pin in the category and select a switch for input. After that save the pin assignment. Then the pins will appear in diagram as shown in Figure 8.29.

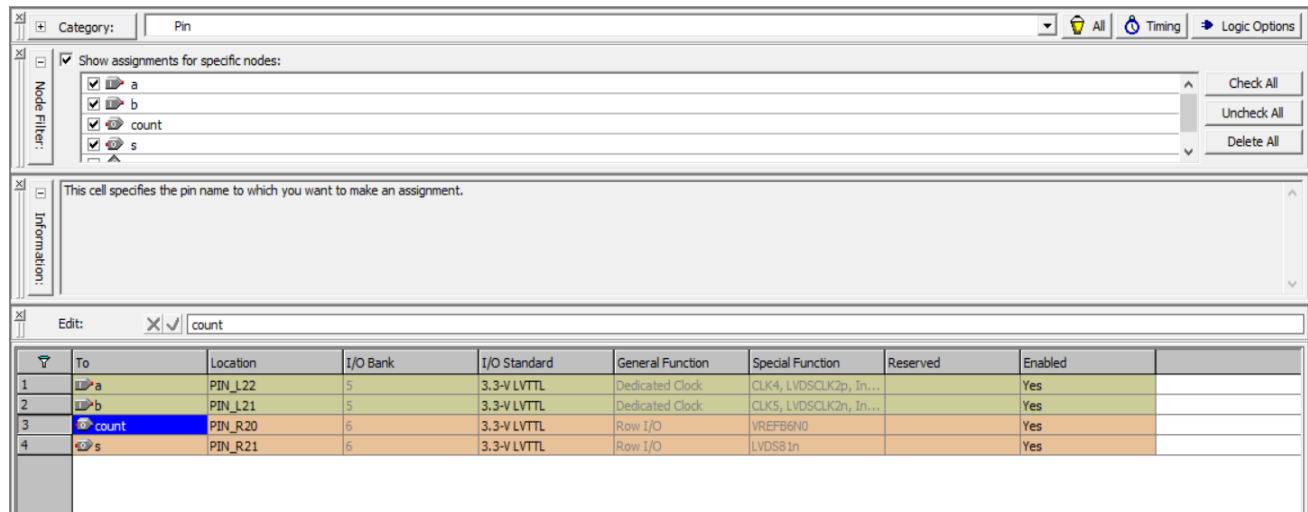


Figure 8.29: choose inputs and outputs in board.

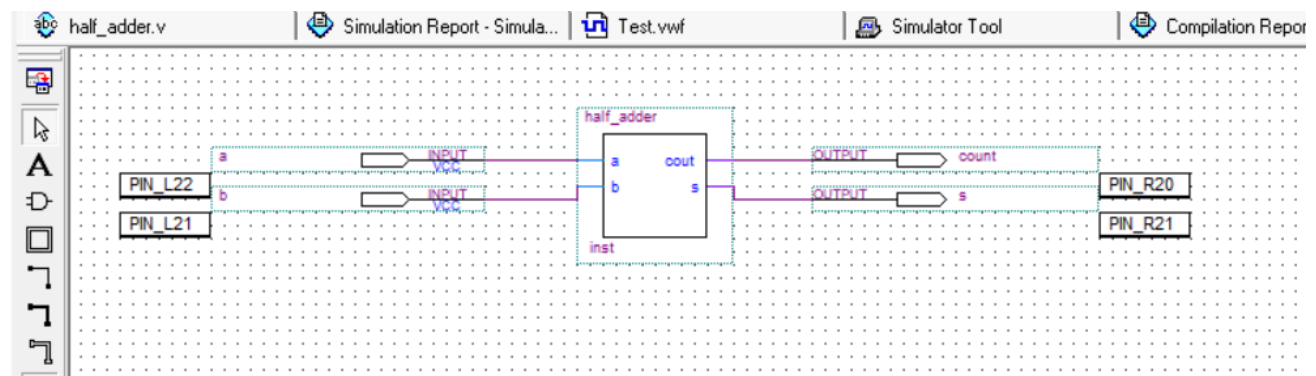


Figure 8.30: show the pins in diagram.

7. Re-compile the Project so that the new changes in the PINs take place. If the project name differs from you block diagram file, then you have to set the file as top-level entity as mentioned before. Download the Program on the FPGA Tools >Programmer.



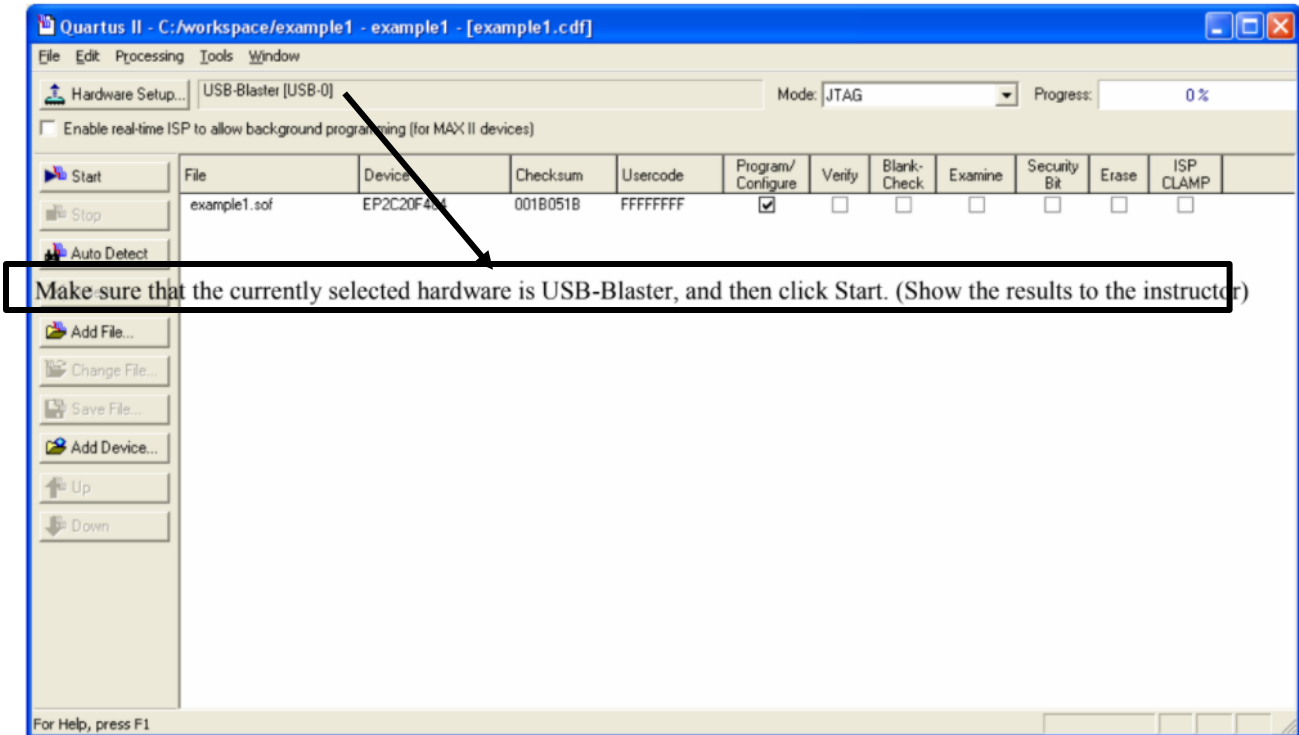
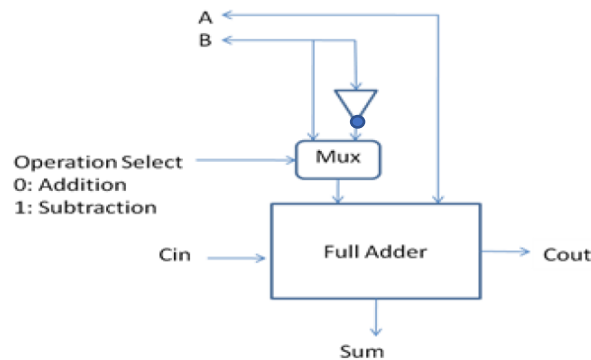


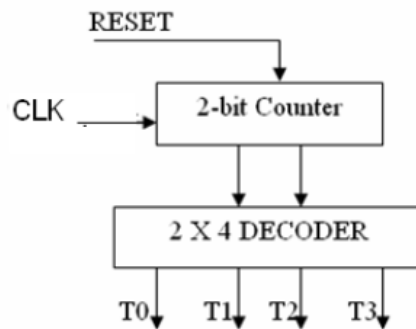
Figure 8.31: download code in FPGA.

## 8.6 Task in the lab

1. Use Verilog HDL to implement a 2-to-1 MUX. Use Verilog HDL to implement a Full Adder. Create schematic symbols for both the MUX and the Full adder, then connect them as shown in the figure. Run a meaningful simulation for this circuit.

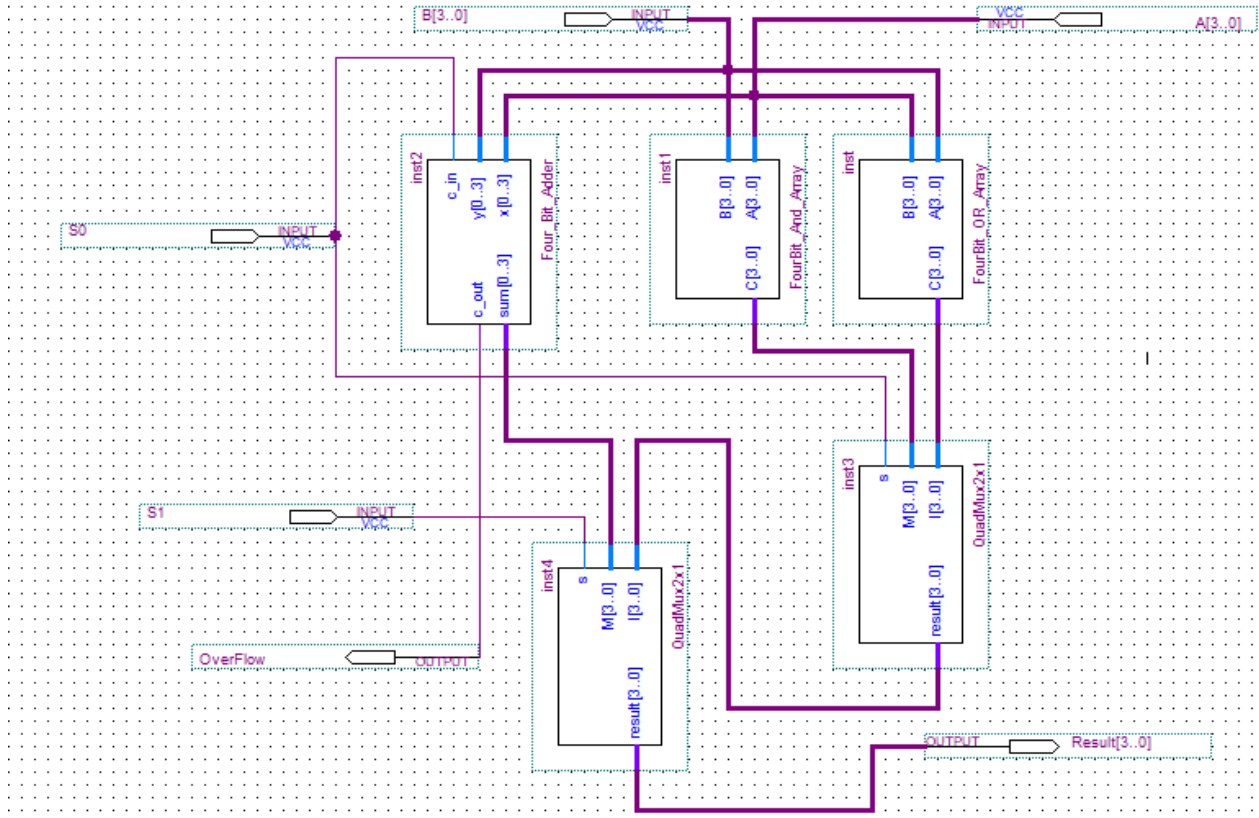


2. Use Verilog HDL to implement a 2-bit counter with direct reset input (RESET). Use Verilog HDL to implement a 2-to-4 Decoder. Create schematic symbols for both the counter and the decoder, then connect them as shown in figure below. Run a meaningful simulation for this circuit.



## 8.7 Post Lab

- Design the following circuit:





**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 9 - A Simple Security System Using FPGA**

### **9.1 Objectives**

- To practice building different digital components using Quartus either by building Verilog codes or Block diagrams.
- Learning how to put some of the digital components, you have studied and build in pervious lab sessions, together to build useful systems.
- To become more familiar with FPGA programming.

### **9.2 Equipment Required**

- A computer with Quartus II (7.2 +) and USB driver installed
- Altera DE1 system with its datasheets. (For FPGA pins map)

### **9.3 Pre-Lab (Bring a soft copy of your prelab with you to the lab)**

- 1) Prepare each part of the procedure section where it says (Pre-Lab).
- 2) **NOTE:** You must come prepared, as this will reflect your work time during the lab plus it will be a critical variable in the evaluation of your lab report.

## 9.4 Theory

In this experiment, we are going to build a simple security system using Altera Quartus software. Then we will program and download this system on the FPGA board. This security system is simply a 2-digit digital lock. The user enters a number of two digits, such that, the digit ranges from 0 to 3. Thus, every digit has a lower limit of 0 and an upper limit of 3. The number is entered using a keypad (using the 91 switch keys built in our FPGAs). Each digit is represented by a 7-segment display and if the total number entered on the displays equals to XX a green led is on; allowing us to pass. Otherwise, a red LED is always on, blocking us from passing.

Figure 9.1 depicts this security system architecture.

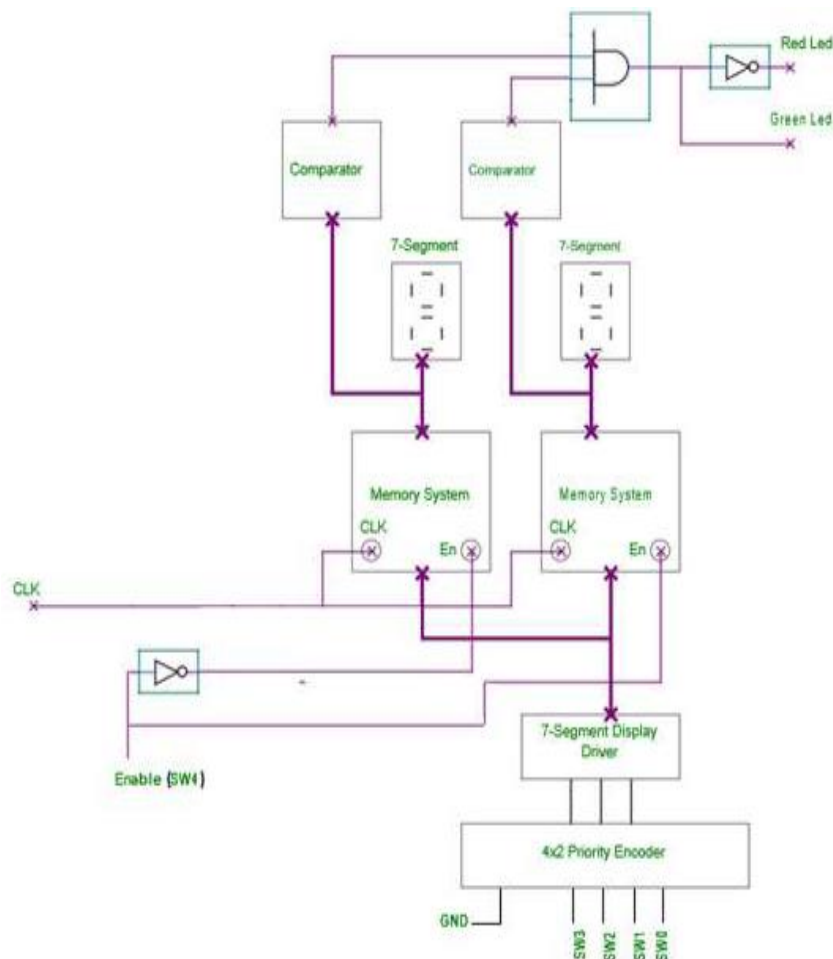


Figure 9.1: The Security System Architecture.



Figure 9.2: High level view of the system.

As Figure 9.1 shows, this security system comprises the following components:

#### 9.4.1 4x2 Priority Encoder

The normal digital encoder is a combinational circuit that encodes  $2^n$  input lines by  $n$  output lines. In other words, it generates the binary code equivalent of the input line, which is active high. However, this kind of encoders has a problem. It only works when only one of the inputs is active. In other words, if there is more than one input line active, the encoder will generate the wrong code.

This issue can be resolved using the priority encoder. This kind of encoders prioritizes the level of each input. If multiple input lines are active, the output code will correspond to the input line with the highest priority as shown in Figure 9.3.

The user will use this priority encoder to choose what value to view on a 7-segment display (values range from 0 to 3 in decimal), for example, if the user switches SW1 to high and keeps SW2 and SW3 low then the output of the encoder will be b'01.

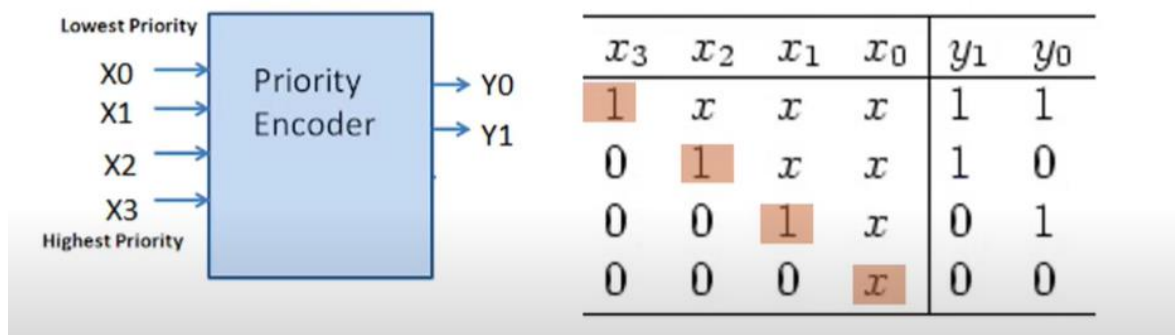


Figure 9.3: 4x2 Priority Encoder.

### 9.4.2 Enable Port

The purpose of this port is to allow the user to select which memory system is active, and hence which 7- 7-segment display to use, for example, if SW4 is high then the En pin of the first memory system is enabled, and ready to read the user input on the 4x2 priority encoder.

**Note:** The enable pin of the decoder must be active low while switching between the selection lines of the decoder.

### 9.4.3 segment display driver

This driver is used to convert the output of the priority encoder to the proper input for the 7- 7-segment displays, the output of the driver is first stored in a memory unit before it is transferred to a 7- 7-segment (depending on which memory system is enabled using the 2x4 decoder). Figure 9.4 show a 4x7 7-segment decoder in this experiment we will use a 2x7 7-segment decoder.

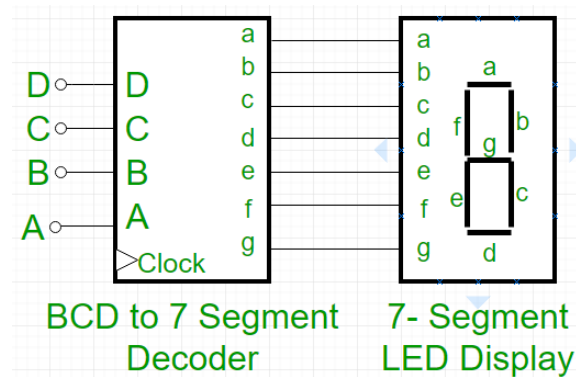


Figure 9.4: 7 -segment decoder.

### 9.4.4 Memory System

The purpose of such a system is to ensure that the value selected by the user to display on a certain 7- 7-segment is kept there when the user switches to select another 7-segment. Each memory system consists of seven D- flip-flops and 2x1 MUXs as shown in Figure 9.5.

When the enable pin equals 0, the output of each DFF becomes its input at every clock cycle. On the other hand, when the Enable pin becomes 1, the data coming from the 7-segment driver is stored in each DFF. The output of each DFF is sent on a data bus to a 7-segment display.

**Note:** For each 7- segment display, we need a memory system block

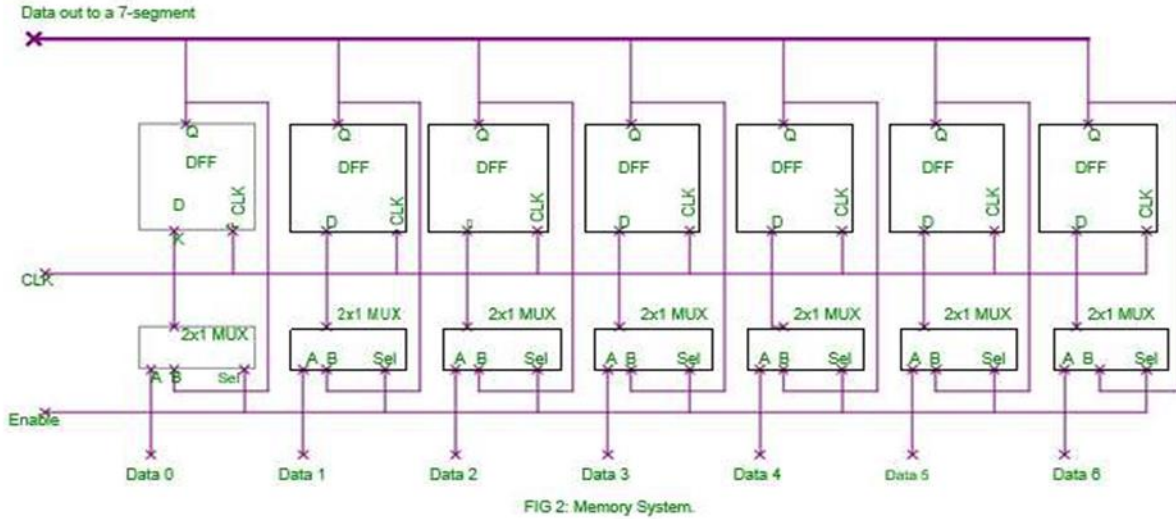


Figure 9.5: Memory System.

### 9.4.5 Comparator

The input of each 7-segment display is connected also to a comparator. every comparator has a built-in value (reference) which is compared with the value of the 7-segment display. If both values are equal, then the output of the comparator is 1, and it is 0, otherwise. For example, if one of the comparators has a reference value equals 5, then its output will be 1 if and only if the input is equal to=7'b0100100 (which is the value of 5 in the 7-segment display). The purpose of the comparator is to lock/unlock the security system.

### 9.4.6 2-input AND gate

This AND gate will make sure that the two 7-segment displays have the correct combination. In other words, if each comparator output is “1”, then the AND gate output will be “1”, and the green light is ON. Otherwise, the red light will be always ON.



## 9.5 Procedure

1. Design a 4 x 2 priority encoder by writing and simulating the Verilog code shown in Figure 9.6 using Quartus. **(Pre-LAB)**.

```
//4 x 2 Priority encoder
module priority_encoder(out, in);

    input [3:0] in;
    output reg [1:0] out;
    always @ (in)

begin

    casex(in)
        4'b0001:out = 2'b00;
        4'b001x:out = 2'b01;
        4'b01xx:out = 2'b10;
        4'b1xxx:out = 2'b11;
        default:out = 2'b00;
    endcase

end

endmodule
```

Figure 9.6: 4 x 2 Priority Encoder Verilog Code.

2. Design the 7-segment display driver by writing and simulating the Verilog code shown in Figure 7 using Quartus. **(Pre-Lab)**.

```

//this will convert binary input
// to 7-segment input
module sevenSegmentDriver(in_v , out_v ) ;
input [1:0] in_v ;
output [6:0] out_v ;
wire[1:0] in_v ;
reg [6:0] out_v ;
always @ (in_v)
begin
    case(in_v)
        0 : out_v = 7'b1000000;
        1 : out_v = 7'b1111001;
        2 : out_v = 7'b0100100;
        3 : out_v = 7'b0110000;
    endcase
end
endmodule

```

Figure 9.7: 7-Segment Display Driver Verilog Code.

3. Write and simulate the Verilog code of a D- Flip Flop using Quartus **(Pre Lab)**
4. Write and simulate the Verilog code of a 2x1 MUX using Quartus **(Pre Lab)**.

**Note:** The MUX should behave as explained in the memory system section (check back the theory).

5. Use a block diagram to build the design shown in Figure 9.8.
6. Write and simulate the Verilog code of the comparator shown in Figure 9.9 using Quartus

**Note:** for simplification, we will build one comparator based on the reference value X (in this case, it is 5). You can build four different comparators with four different values to compare with.

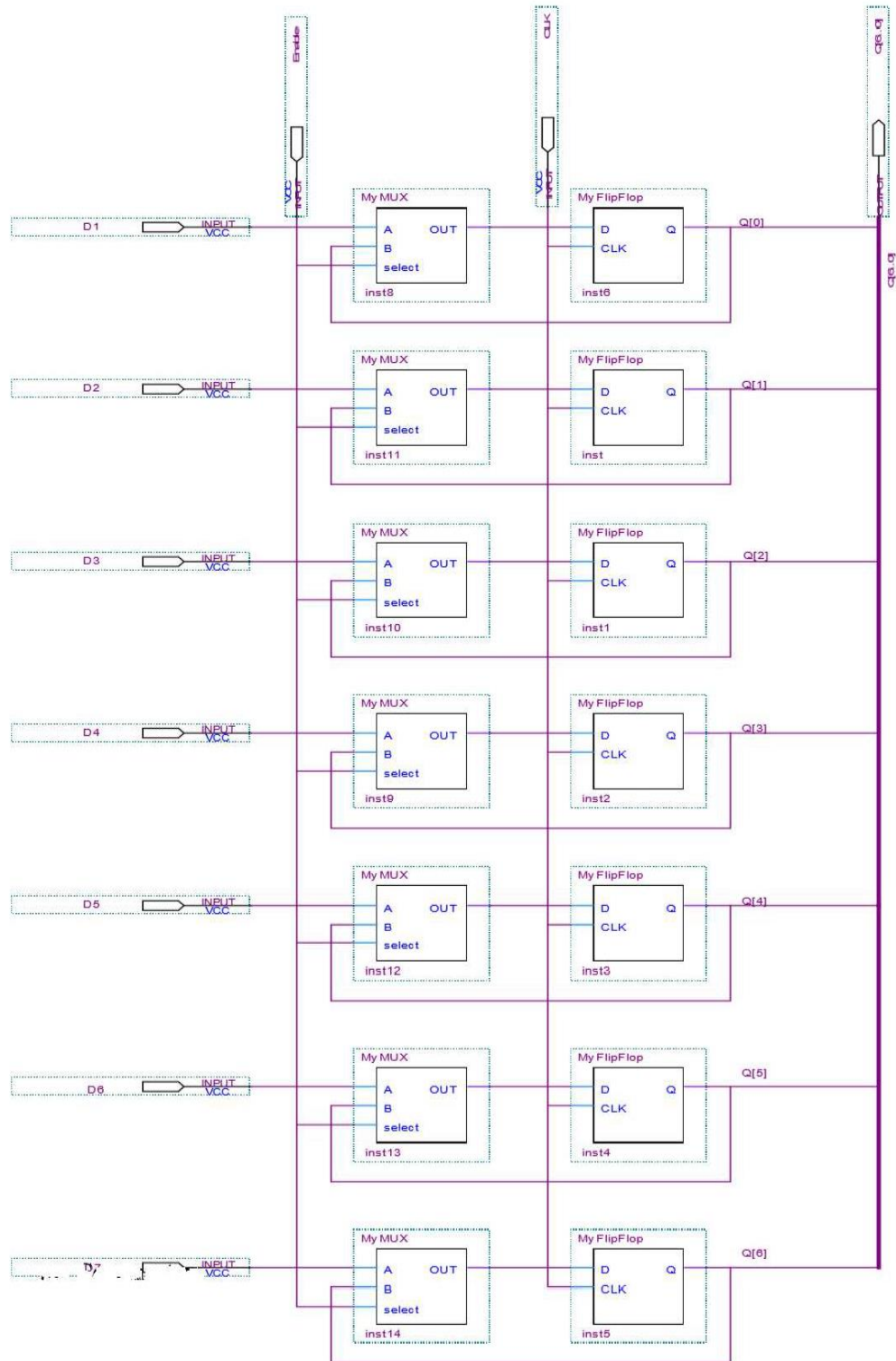


Figure 9.8: Memory System Block Diagram.

```

module MyComp(CData,out);
input [6:0] CData;
wire [6:0] CData;
output out;
reg out;
always @ (CData)
begin
    //compare value in register with 2
    if(CData == 7'b0100100)
        out <=1'b1;
    else
        out <=1'b0;
    end
endmodule

```

Figure (9): Comparator Verilog Code.

7. Build and design the security system using the components you built in the previous steps. The final block design should look like the one in Figure 9.10. Assign pins values to the security system design you just built and then download the system on the FPGA board.

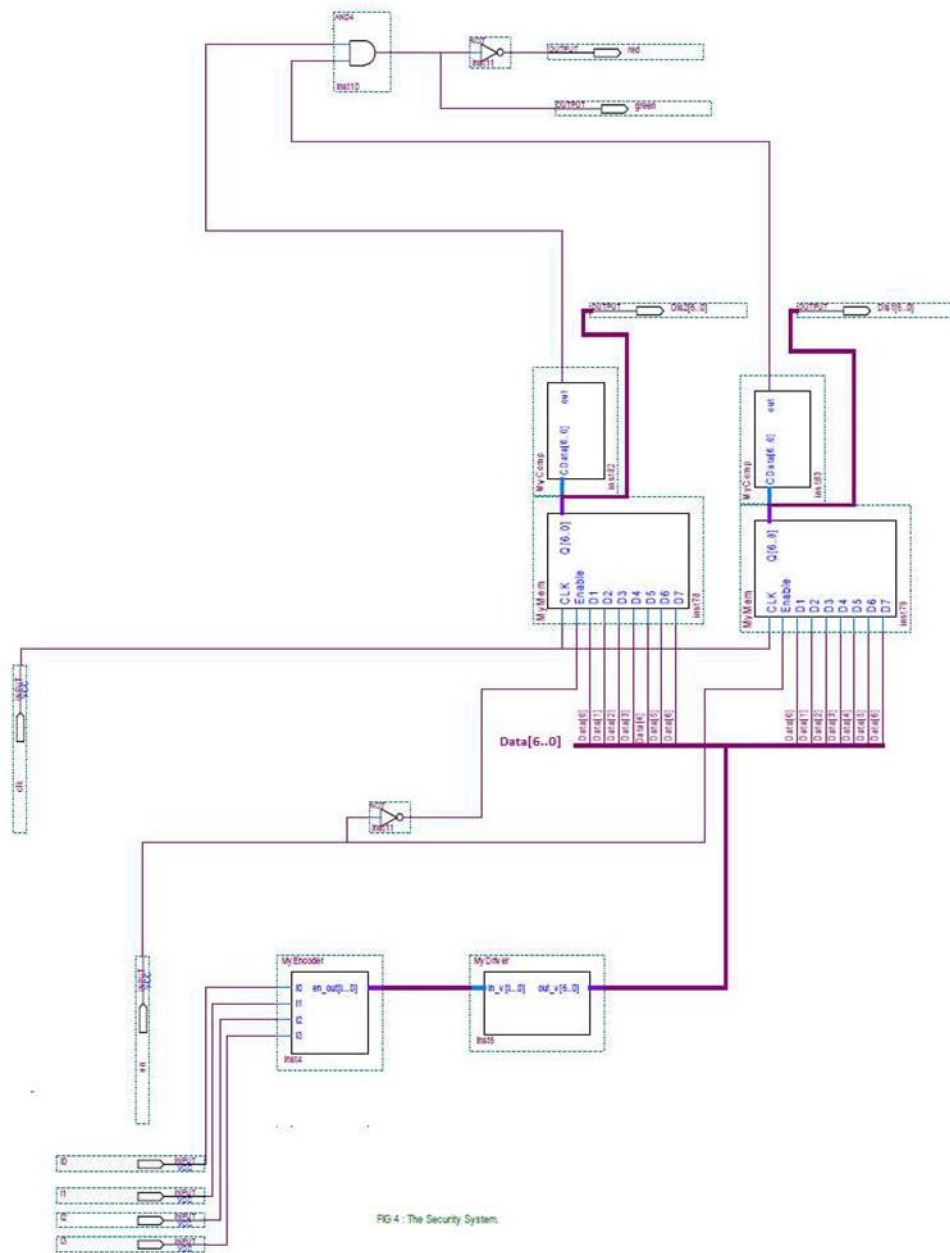


Figure 9.10: The Security System Final Block Diagram.

## 9.6 Do in lab

1. Change in code to make user enter number from 0-7.
2. Make your system take four different password numbers.



**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 10 - Simple Computer Simulation**

### **10.1 Objectives**

In this experiment, we are going to design the Verilog HDL control sequence for a simple computer (SIMCOMP). The SIMCOMP is a very small computer to give the students practice in the ideas of designing a simple CPU with the Verilog HDL notation.

### **10.2 Equipment Required**

- A computer with Quartus II

### **10.3 Pre-Lab**

- 1) Read the experiment to find the prelab.

## 10.4 Theory

### 10.4.1 Basic Computer Model - Von Neumann Model

Von Neumann computer systems contain three main building blocks: the central processing unit (CPU), memory, and input/output devices (I/O). These three components are connected together using the system bus. The most prominent items within the CPU are the registers: they can be manipulated directly by a computer program, See Figure 10.1:

#### Function of the Von Neumann Component:

**Memory:** Storage of information (data/program)

**Processing Unit:** Computation/Processing of Information

**Input:** Means of getting information into the computer. e.g., keyboard, mouse

**Output:** Means of getting information out of the computer. e.g., printer, monitor

**Control Unit:** Makes sure that all the other parts perform their tasks correctly and at the correct time.

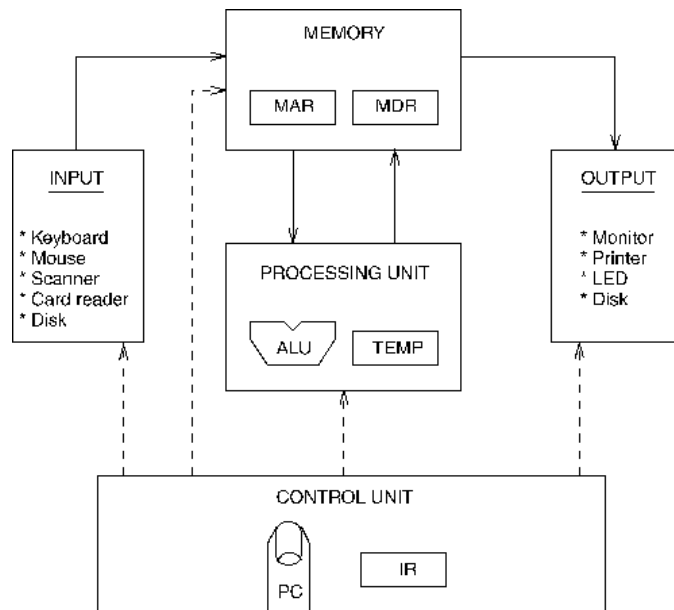


Figure 10.1: Von Neumann computer systems.

### 10.4.2 General Registers

- 1) One of the CPU registers is called as an accumulator **AC** or 'A' register. It is the main operand register of the ALU it is used to store the result generated by ALU.
- 2) The data register (**MDR**) acts as a buffer between the CPU and main memory. It is used as an input operand register with the accumulator.
- 3) The instruction register (**IR**) holds the opcode of the current instruction.
- 4) The address register (**MAR**) holds the address of the memory in which the operand resides.
- 5) The program counter (**PC**) holds the address of the next instruction to be fetched/execution.

Additional addressable registers can be provided for storing operands and address. This can be viewed as replacing the single accumulator by a set of registers. If the registers are used for many purposes, the resulting computer is said to have general register organization. In the case of processor registers, a register is selected by the multiplexers that form the buses.

### 10.4.3 Communication Between Memory and Processing Unit

Communication between memory and processing unit consists of two registers:

Memory Address Register (MAR).

Memory Data Register (MDR).

**To read,**

- 1- The address of the location is put in MAR.
- 2- The memory is enabled for a read.
- 3- The value is put in MDR by the memory.

**To write,**

- 1- The address of the location is put in MAR.
- 2- The data is put in MDR.
- 3- The **Write Enable** signal is asserted.
- 4- The value in MDR is written to the location specified.

### 10.4.4 Generic CPU Instruction Cycle



The generic instruction cycle for an unspecified CPU consists of the following stages:

1) **Fetch instruction:**

Read instruction code from address in PC and place in IR. ( $IR \leftarrow \text{Memory}[PC]$ )

2) **Decode instruction:**

Hardware determines what the opcode/function is and determines which registers or memory addresses contain the operands.

3) **Fetch operands from memory if necessary:**

If any operands are memory addresses, initiate memory read cycles to read them into CPU registers. If an operand is in memory, not a register, then the memory address of the operand is known as the *effective address*, or EA for short. The fetching of an operand can therefore be denoted as  $\text{Register} \leftarrow \text{Memory}[EA]$ . On today's computers, CPUs are much faster than memory, so operand fetching usually takes multiple CPU clock cycles to complete.

4) **Execute:**

Perform the function of the instruction. If arithmetic or logic instruction, utilize the ALU circuits to carry out the operation on data in registers. This is the only stage of the instruction cycle that is useful from the perspective of the end user. Everything else is overhead required to make the execute stage happen. One of the major goals of CPU design is to eliminate overhead and spend a higher percentage of the time in the execute stage.

5) **Store result in memory if necessary:**

If destination is a memory address, initiate a memory write cycle to transfer the result from the CPU to memory. Depending on the situation, the CPU may or may not have to wait until this operation completes. If the next instruction does not need to access the memory chip where the result is stored, it can proceed with the next instruction while the memory unit is carrying out the write operation.

**Below is an example of a full instruction cycle which uses memory addresses for all**

**three operands:**

- 1- **Mull product, x, y**
- 2- Fetch the instruction code from Memory [PC]
- 3- Decode the instruction. This reveals that it's a multiply instruction, and that the operands are memory locations x, y, and product.
- 4- Fetch x and y from memory.
- 5- Multiply x and y, storing the result in a CPU register.
- 6- Save the result from the CPU to memory location product.

### 10.4.5 Addressing Modes

The term **addressing modes** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand.

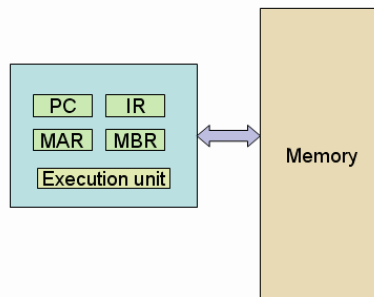


Figure 10.2: connection between memory and registers.

## 10.5 Our Simple Computer

SIMCOMP has a two byte-addressable memory with size of 128byte. The memory is synchronous to the CPU, and the CPU can read or write a word (or cell) in single clock period.

The memory can only be accessed through the memory address register (**MAR**) and the memory buffer register (**MBR**). To read from memory, you use:

- `MBR <= Memory [MAR];`

And to write to memory, you use:

- `Memory [MA] <= MBR;`

1. The CPU has three registers: an accumulator (AC), a program counter (PC) and an instruction

register (IR).

2. The SIMCOMP has only three instructions: Load, Store, and Add.
3. The size of all instructions is 16 bits; all the instructions are single address instructions and access a word in memory.



Figure 10.3: Instruction format.

### Instruction Format

The opcodes are:

Op-code	Instruction	Description
0011	Load M	Loads the contents of memory location <b>M</b> into the accumulator.
1011	Store M	Stores the contents of the accumulator in memory location <b>M</b> .
0111	Add M	Adds the contents of memory location <b>M</b> to the contents of the accumulator.

Table 1

### Prelab

- 1) You must write the basic code shown in Fig 1 at the last page of this experiment in your own Quartus II file.
- 2) You have to trace the basic code manually so that you can understand what does the code do. Use the following table:

Line #	Code	Description of what is done
6	reg [15:0] Memory [0:63]	
7	reg [2:0] state	
13	Memory [10] = 16'h3020	
14	Memory [11] = 16'h7021	
15	Memory [12] = 16'hB014	
18	Memory [32] = 16'd7	
19	Memory [32] = 16'd7	

22	PC = 10; state = 0	
29	MAR <= PC	
30	state = 1	
33	IR <= Memory [MAR]	
34	PC <= PC + 1	
35	state = 2	
38	MAR <= IR [11:0]	
39	state = 3	
42	state = 4	
43	case (IR [15:12])	
44	load: MBR <= Memory [MAR]	
45	add: MBR <= Memory [MAR]	
46	store: MBR <= AC	
52	AC <= AC + MBR	
56	AC <= MBR	
60	Memory [MAR] <= MBR	

Table 2

3) You have to summarize the objective of Program #1 above, then repeat for Program #2

## 10.6 Accumulator Based Simple Computer

The **Verilog program** described by the following table is shown in Fig 1, study, and simulate the code.

Instructions				
Memory location	Instruction assembly		Instruction machine code	in Hex
10	Load [32]		0011-0000-0010-0000b	16'h3020
11	Add [33]		0111-0000-0010-0001b	16'h7021h
12	Store [20]		1011-0000-0001-0100b	16'hB014h
Data				
32	Data	7	Memory [32]	16'h7
33	Data	5	Memory [32]	16'd5

Table 3

**Task1: Modify the code to include the jump instruction**

Choose any opcode e.g., jump=4'b0001, you have to include the execution of jump which changes the PC to the specified address in the instruction.

Instructions				
Memory location	Instruction assembly		Instruction machine code	in Hex
10	Load [32]		0011-0000-0010-0000b	16'h3020
11	Add [33]		0111-0000-0010-0001b	16'h7021h
12	Store [20]		1011-0000-0001-0100b	16'hB014h
13	Jump 11		0001-0000-0000-1011b	16'h100Bh
Data				
32	Data	7	Memory [32]	16'h7
33	Data	5	Memory [32]	16'd5

Table 4

**Task 2: Write the General form of the instruction set for Prog #1**

**Task 3: Trace the Modified code as was done in the prelab but this time using the waveforms**

**10.7 Register Based Simple Computer [SIMCOMP2]**

**Modify the instruction format so that SIMCOMP can handle four addressing modes and four registers.**

This new SIMCOMP2 has four 16-bit general purpose registers, R[0], R[1], R[2] and R[3] which replace the AC. In Verilog, you declare R as a bank of registers much like we do Memory.

```
reg [15:0] R [0:3]; // declaration of registers bank
```

Since registers are usually on the CPU chip, we have no modeling limitations as we do with Memory - **with Memory we have to use the MAR and MBR registers to access the memory.** Therefore, in a load

you could use R as follows:

**R [IR [9:8]] <= MBR; //where the 2 bits in the IR specify which R register to set.**

#### Task 4: Modify the four instructions of the old SIMCOMP

The new instruction should follow the new form:

1. **LOAD R[i], M** loads the contents of memory location M into R[i].
2. **STORE R[i], M** stores the contents of R[i] in memory location M.
3. **ADD R[i], R [j], R[k]** adds contents of R[j] and R[k] and places result in R[i].

To test your SIMCOMP2 design, perform the following program where:

1. **PC starts at 10,**
2. Suppose **IR [9:8]** is used to specify the register number.
3. In the “add instruction” **IR [11:10]** destination register, **IR [9:8], IR [7:6]** source1, source2 respectively.

Instructions					
Memory location	Instruction	Destination register	Source Register/Memory	Instruction set code in binary	in Hex
10	Load	R1	3	0011-0001-0000-0011b	16'h3103
11	Load	R2	4	0011-0010-0000-0100b	16'h3204
12	Add	R1	R1, R2	0111-0101-1000-0000b	16'h7580
13	Store	R1	5	1011-0001-0000-0101b	16'hB105
Data					
3	Data	A	Memory [3]		16'hA
4	Data	6	Memory [4]		16'd6

Table 5

#### Task 5: Write the General form of the instruction set for SIMCOMP2.

#### Task 6: Trace SIMCOMP2 code using the waveforms.

#### Task 7: Add immediate addressing to the SIMCOMP2:

If bit (IR [11]) is a one in a Load, the last eight bits are not an address but an operand. The operand is in the range -128 to 127.

If immediate addressing is used in a LOAD, the operand is loaded into the register.

**Load R1, 8**

**R1 ← 8**

Simulate the following test **with handwritten comments** explaining what you are doing.

PC = 10

Memory [10]: Load R1,3 // Load immediate

Memory [11]: Load R2, -4 //Use 2's complement to represent (-4) Memory [12] Add R1, R1, R1

Memory [13]: Store R1,5

Instructions					
Memory location	Instruction	Destination register	Source Register/Memory	Instruction set code in binary	in Hex
10	Load Immediate	R1	3	0011-1001-0000-0011b	16'h3903
11	Load Immediate	R2	-4	0011-1010-1111-1100b	16'h3AFC
12	Add	R1	R1, R2	0111-0101-1000-0000b	16'h7580
13	Store	R1	5	1011-0001-0000-0101b	16'hB105
Data					
3	Data	A	Memory [3]		16'hA
4	Data	6	Memory [4]		16'd6

Table 6

**Hint: How to read the 2's complement (8 bit) in Verilog:**

$MBR \leq -(\sim(IR[7:0]) + 1)$

**Task 8: Write the General form of the instruction set for Prog #2 with immediate load**

```

1  module SIMCOMP(clock, PC, IR, MBR, AC, MAR);
2      input clock;
3      output PC, IR, MBR, AC, MAR;
4      reg [15:0] IR, MBR, AC;
5      reg [11:0] PC, MAR;
6      reg [15:0] Memory [0:63];
7      reg [2:0] state;
8
9      parameter load = 4'b0011, store = 4'b1011, add=4'b0111;
10
11  initial begin
12      // program
13      Memory [10] = 16'h3020;
14      Memory [11] = 16'h7021;
15      Memory [12] = 16'hB014;
16
17      // data at byte address
18      Memory [32] = 16'd7;
19      Memory [33] = 16'd5;
20
21      //set the program counter to the start of t
22      PC = 10; state = 0;
23  end
24
25  always @(posedge clock) begin
26      case (state)
27      0: begin
28          MAR <= PC;
29          state=1;
30          end
31      1: begin // fetch the instruction from memory
32          IR <= Memory[MAR];
33          PC <= PC + 1;
34          state=2; //next state
35          end
36      2: begin //Instruction decode
37
38          MAR <= IR[11:0];
39          state= 3;
40          end
41      3: begin // Operand fetch
42          state =4;
43          case (IR[15:12])
44              load : MBR <= Memory[MAR];
45              add : MBR <= Memory[MAR];
46              store: MBR<=AC;
47          endcase
48          end
49      4: begin //execute
50          if (IR[15:12]==4'h7) begin
51              AC<= AC+MBR;
52              state =0;
53          end
54          else if (IR[15:12] == 4'h3) begin
55              AC <= MBR;
56              state =0; // next state
57          end
58          else if (IR[15:12] == 4'hB) begin
59              Memory[MAR] <= MBR;
60              state = 0;
61          end
62          end
63      endcase
64      end
65  end

```

Figure 10.4: Basic code.



**At the end of this experiment, you should have two files:**

- 1- An Accumulator Based Simple Computer with 4 instructions as in program 1
- 2- A register based simple computer with 3 opcodes (noting that the load opcode can be immediate or normal) as in program 2.

## 10.8 Post Lab

1. Modify Program #2 so that it finds the sum of three elements and then traces the process (using waveform).
2. Modify the program in the first part of the post-lab to include the jump instruction with opcode (jump=4'b1001) and change code to run this code below. (Write the instruction format)
3. Modify the previous program making it sum 10 elements using a jump (loop) opcode to sum the 10 elements.

Instructions					
Memory location	Instruction	Destination register	Source Register/Memory	Instruction set code in binary	in Hex
10	Load	R1	3		
11	Load	R2	4		
12	Load	R3	-9		
13	Add	R1	R1, R2, R3		
14	Store	R1	5		

Table 7



**Faculty of Engineering and Technology**  
**Department of Electrical and Computer Engineering**  
**ENCS 2110**  
**Digital Electronics and Computer Organization Lab**  
**Experiment No. 11 - Arithmetic Elements**

### **11.1 Objectives**

- To understand functions and applications of the ALU (arithmetic logic unit).
- To perform arithmetic and logic operations using the 74181 ALU IC.
- To understand the construction and applications of parity generators.
- To generate parity bit using XOR gates and parity generator IC.

### **11.2 Equipment Required**

- KL-31001 Basic Electricity Circuit Lab
- KL-33003 Combinational Logic Circuit Experiment Model (2).
- KL-33004 Combinational Logic Circuit Experiment Model (3).

## 11.3 Theory

### 11.3.1 ARITHMETIC LOGIC UNIT (ALU) CIRCUIT

The logic diagram of the ALU is shown in Figure 11.1:

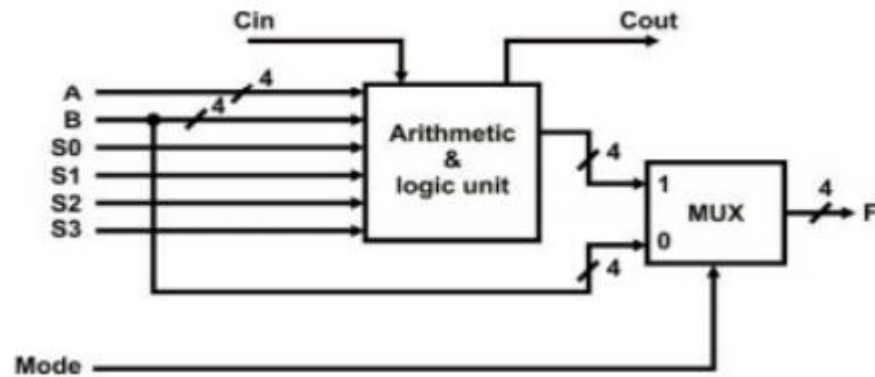


Figure 11.1: logic diagram of the ALU.

It consists of two major parts: the arithmetic unit and the logic unit. The output, either arithmetic or logic which is selected by the selection switches. Figure 11.2 shows the pin assignment:

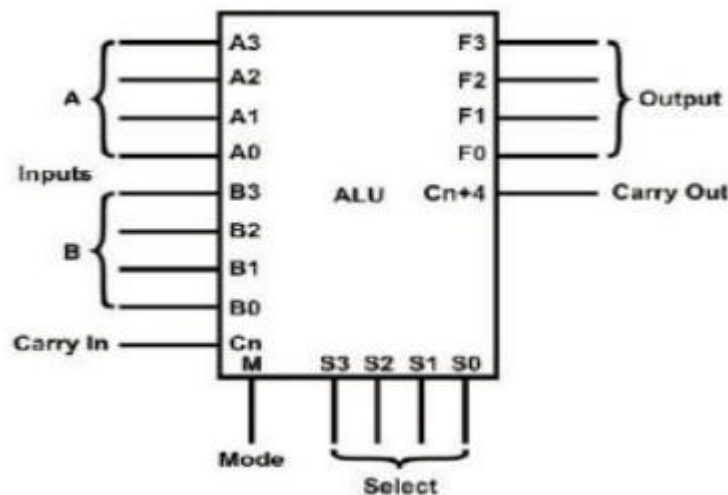


Figure 11.2: pin assignment.

The circuit has two 4-bit inputs A and B, as well as a “carry-in” (Cn) input. There is a mode control input (M) and 4 function-select lines S0, S1, S2, S3 forming logic or arithmetic operations.

Also, it has a 4-bit output (F3~F0); a “carry-out” or “Cn+4” output. The biggest advantage of the design is its ability to perform arithmetic functions such as addition, subtraction, multiplication; and logic functions such as AND, XOR functions.

The mode control input (M) and function-select lines (S0~S3) determines which function it will perform.

### 11.3.2 BIT PARITY GENERATOR CIRCUIT

A bit parity generated by the bit parity generator, usually accompanies the data transmission process. The bit parity provides as a reference point and allows us to compare and check whether the transmission process and the data transmitted are correct or not.

There are two types of bit parity generators: The “Odd” bit parity generator will generate “1” if the data contains an even number of “1” s. For example, the data “10111011” has six “1s”. When the bit parity is added to the end of this data, the number of “1s” in the data will become an “ODD” number, hence the name “Odd Parity Generator”. On the other hand, an “Even” bit parity generator will add a “1” to data with odd number of “1s” to make the total number of “1s” even. If the data already has an even number of “1” s no bit parity is generated. Output Y of the “Even” bit parity generator shown in Figure 11.3 will be 0 if the inputs ABCDEFGH is equal to 10111011.

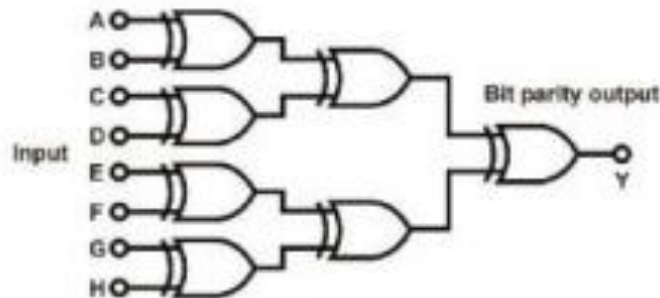


Figure 11.3: “Even” Bit Parity Generator Circuit.

## 11.4 Procedure

### 11.4.1 ARITHMETIC LOGIC UNIT (ALU) CIRCUIT

- A) Connect function-select lines S3~S0 to DIP2.7~2.4 respectively of Module KL-33003 block b. Connect M to Data Switch SW0 to select arithmetic and logic operations. When M= “0” inputB3~B0 is displayed at output. When M= “1” arithmetic and logic function is performed.

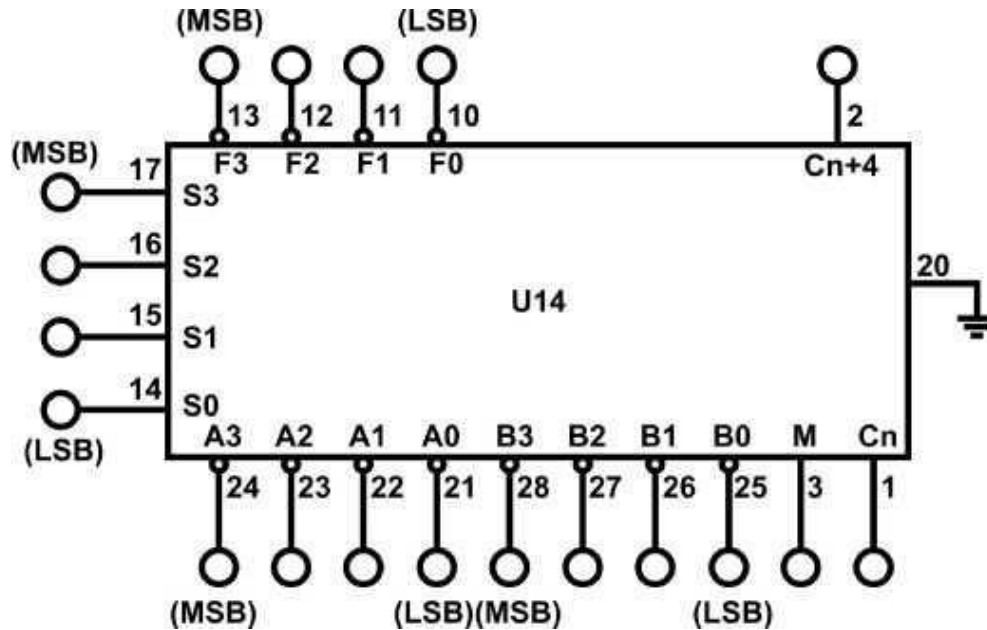


Figure 11.4: “Even” Bit Parity Generator Circuit.

- B) Connect inputs A3~A0 to DIP1.3~1.0 and B3~B0 to DIP2.3~2.0; Connect Cn to Data Switch SW1; outputs F3~F0 to Logic Indicators L3~L0 and Cn+4 to L4.
- C) Set M to “1” to perform the following arithmetic functions.
- Set Cn to “0” and ignore the previous carry.
- When S3S2S1S0=0000 perform the addition.
- What is the output when A3A2A1A0=0000 and B3B2B1B0=1111?
- F3F2F1F0= \_\_\_\_\_; Cn+4=\_\_\_\_\_
- What is the output when A3A2A1A0=1001 and B3B2B1B0=0100?
- F3F2F1F0= \_\_\_\_\_; Cn+4=\_\_\_\_\_

- Set  $C_n$  to “1” and add the previous carry.

When  $S_3S_2S_1S_0=0000$  perform the addition.

What is the output when  $A_3A_2A_1A_0=0000$  and  $B_3B_2B_1B_0=1111$ ?

$F_3F_2F_1F_0=$  \_\_\_\_\_;  $C_{n+4}=$  \_\_\_\_\_

What is the output when  $A_3A_2A_1A_0=1001$  and  $B_3B_2B_1B_0=0100$ ?

$F_3F_2F_1F_0=$  \_\_\_\_\_;  $C_{n+4}=$  \_\_\_\_\_

- Set  $C_n$  to “0”. When  $S_3S_2S_1S_0=0001$  perform the subtraction.

What is the output when  $A_3A_2A_1A_0=0000$  and  $B_3B_2B_1B_0=1111$ ?

$F_3F_2F_1F_0=$  \_\_\_\_\_;  $C_{n+4}=$  \_\_\_\_\_

What is the output when  $A_3A_2A_1A_0=1001$  and  $B_3B_2B_1B_0=0100$ ?

$F_3F_2F_1F_0=$  \_\_\_\_\_;  $C_{n+4}=$  \_\_\_\_\_

D) Again set  $M$  to “1” to perform the following arithmetic and logic functions according to Table 1.

Set inputs sequence  $A_0\sim A_3=A$ ,  $B_0\sim B_3=B$  from DIP switches.

Table 11.1: Data of part 11.4.1(D)

Input selection				$M=H$ $C_n=L$	Output			
$S_3$	$S_2$	$S_1$	$S_0$		$F_3$	$F_2$	$F_1$	$F_0$
0	0	1	0	A				
0	0	1	1	$\sim A$				
0	1	0	0	B				
0	1	0	1	$\sim B$				
0	1	1	0	$A \& B$				
0	1	1	1	$A \times B$				
1	0	0	0	$A \wedge B$				
1	0	0	1	$A \times (\sim B)$				
1	0	1	0	$(\sim A) \times B$				
1	0	1	1	$(\sim A) \times (\sim B)$				

## 11.4.2 BIT PARITY GENERATOR CIRCUIT

A) Bit Parity Generator Construct with XOR Gates (Module KL-33004 block a).

- 1) Insert connection clip according to Figure. 11.5 to construct the even bit parity generator circuit of Figure 11.6.

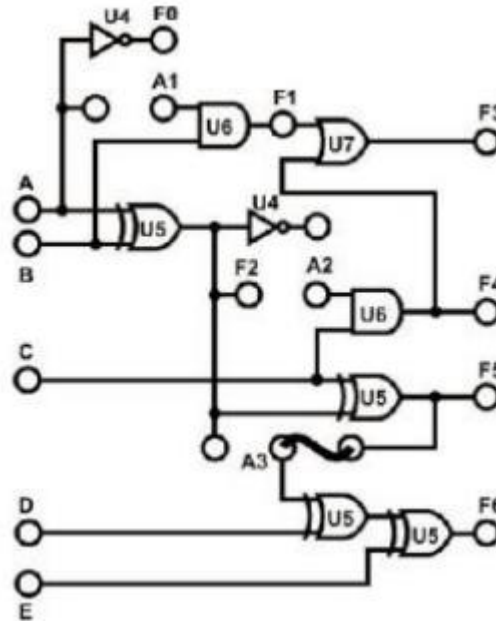


Figure 11.5: Bit Parity Generator Circuit.

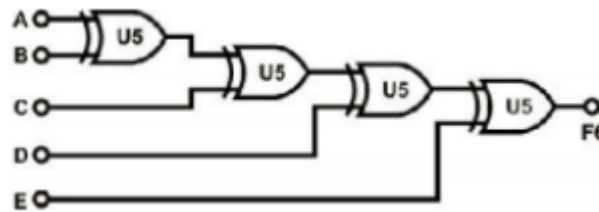


Figure 11.6: "Even" Bit Parity Generator Circuit.

- 2) Connect inputs A, B, C, D, E to DIP Switches 1.0~1.4 and output F6 to Logic Indicator L1. Follow the input sequences in Table 11.2 and record the outputs.

Table 11.2:Data of part 11.4.2 (A-2)

Input					Output
E	D	C	B	A	F6
0	0	0	0	0	
0	0	0	1	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	0	1	
0	1	1	1	0	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	1	0	
1	1	1	1	1	

B) Bit Parity Generator IC.

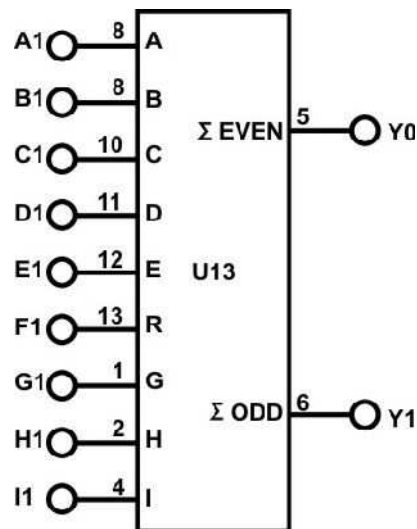


Figure 11.6: Bit Parity Generator IC.

- 1) chip on block d of module KL-33003 is a bit parity generator Connect inputs A1, B1, C1, D1, E1, F1, G1, H1 and I1 to DIP Switches 1.0~1.7and DIP 2.0 respectively. Connect outputs Y0 to L0; Y1 to L1. Follow the input sequences given in Table 11.3 and record the outputs.

Table 11.3:Data of part 11.4.2 (B-1)



Input									Output	
I	H	G	F	E	D	C	B	A	Y0 (even)	Y1 (odd)
0	0	0	0	0	0	0	0	0		
0	0	0	0	0	0	0	0	1		
0	0	0	0	0	0	0	1	1		
0	0	0	0	0	0	1	1	1		
0	0	0	0	0	1	1	1	1		
0	0	0	0	1	1	1	1	1		
0	0	0	1	1	1	1	1	1		
0	0	1	1	1	1	1	1	1		
0	1	1	1	1	1	1	1	1		
1	1	1	1	1	1	1	1	1		
1	1	1	1	1	1	1	0	1		
1	1	1	1	1	1	1	0	0		
1	1	0	0	0	1	1	0	0		