

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

Contents

1. Experiment No.1 - Combinational Logic Circuits	3
2. Experiment No. 2 - Comparators, Adders and Subtractors.....	17
3. Experiment No. 3 - Encoders, Decoders, Multiplexers and Demultiplexers	32
4. Experiment No. 4 - Digital Circuits Implementation using Breadboard.....	44
5. Experiment No. 5 -Sequential Logic Circuits.....	49
6. Experiment No. 6 - Sequential Logic Circuits using Breadboard and IC's	66
7. Experiment No. 7 - Constructing Memory Circuits Using Flip-Flops.....	74
8. Experiment No. 8 - Introduction to QUARTUSII Software	79
9. Experiment No. 9 - A Simple Security System Using FPGA.....	90
10. Experiment. No. 10 - Simple Computer Simulation.....	98
11. Experiment No. 11 - Arithmetic Elements	106

1. Revision History

Title	Digital Lab Manual			
Date Created	January, 2018			
Maintained By	Dr.Khader Mohammed			
Version Number	Modified By	Modifications Made	Date Modified	Status
1	Khader Mohammad* Dalal Hamdan Rana Alqaisi	All Experiments	Oct., 2017-Feb 1., 2018	Done

*Thanks to all committee members who participated in reviewing the experiments: Dr. Adnan Y., Abualseoud H. Hanna Balta and Jamal Siam

2. Time Estimate for Experiments

This time was estimated based on previous semesters. Also, we tested the new experiments using the new equipment's to get the exact time needed.

Experiment Number	Expected Time (Hours)
Exp1.	1.5
Exp2.	2.5
Exp3.	2.5
Exp4.	2.0
Exp5.	2.5
Exp6.	2.0
Exp7.	2.0
Exp8.	2.5
Exp9.	2.5
Exp10.	2.5
Exp11.	2.5

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

1. Experiment No.1 - Combinational Logic Circuits

1.1 OBJECTIVES

1. To become familiar with AND, OR, NOT, NAND, NOR, XOR operations and their implementation.
2. To construct NOT, AND, OR and XOR gates using NAND gates.
3. To become familiar with concept of Truth table.
4. To implement different Boolean function using NAND gate only.
5. To learn techniques of solution of logic design problems.
6. To become familiar with minimization techniques and with use of Karnaugh maps.
7. To construct AOI gate with basic gates.

1.2 EQUIPMENT REQUIRED

1. IT-3000 Basic Electricity Circuit Lab
2. IT-3002 Basic Gates.

1.3 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 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.

1.4PROCEDURE

1.4.1 NOR Gate Circuit

1. Set module IT-3002 and locate block NOR gate (Fig. 1.1). Connect +5V of module IT-3002 to the +5V output of fixed power supply.

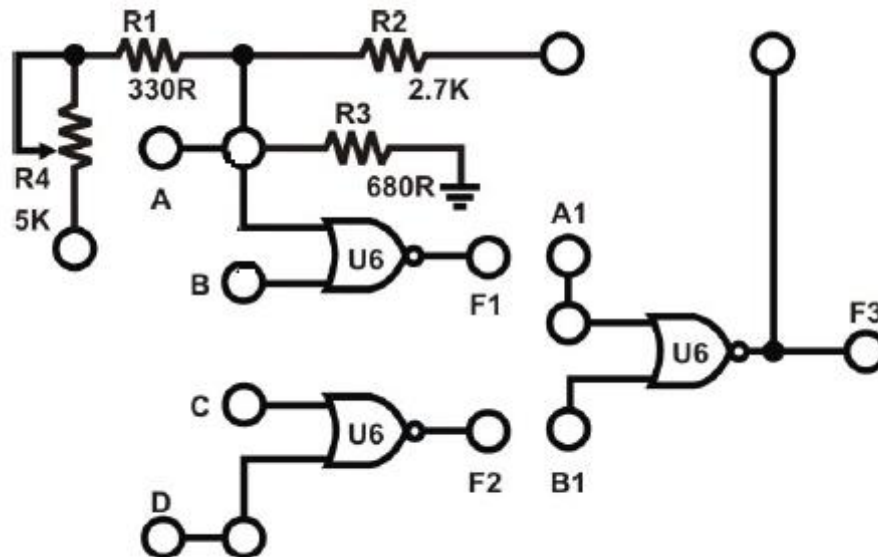


Fig. 1.1 IT-3003 NOR Gate Block

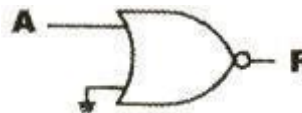


Fig. 1.2 NOR gate used as NOT gate

2. Connect inputs A, B to Data Switches SW0, SW1 and output F1 to Logic Indicator L1. Set SW0 to "0", observe states of F1 at SW1= "0" and SW1= "1".

When SW1="0". F1= _____

When SW1="1", F1= _____

Does the circuit act as a NOT gate? (as shown in Fig. 1.2)

3. Insert a connection clip between A and B as shown in Fig. 1.3 below. Connect A to SW0 and F1 to L1. What is the state of F1 when SW0=0 and SW0=1?

When SW0="0", F1= _____
 When SW0="1", F1= _____

Does the circuit act as a NOT gate?

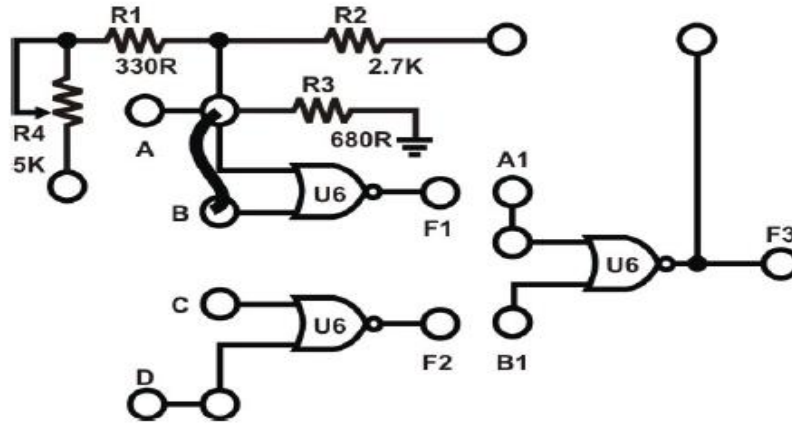


Fig. 1.3 NOR gate used as NOT gate

4. Use U6 to construct a buffer shown on the left side of Fig. 1.4. Insert connection clips between A~B, F1~A1, A1~B1. Connect inputs A to SW0 and output F3 to L1. What is the state of F3 when SW0=0 and SW0=1?

When SW0= "0", F3 = _____
 When SW0= "1", F3 = _____

Does the circuit act as a buffer?

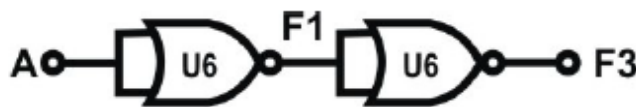


Fig. 1.4 NOR gate used as Buffer

5. Use U6 to construct an OR gate shown on the right side of Fig. 1.5. 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 1.1.

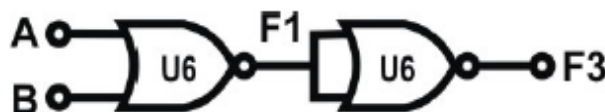


Fig. 1.5 NOR gate used as OR gate

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

Table 1.1

6. Insert connection clips according to the Fig. 1.6. The circuit will act as an AND gate.

- (1) Connect A to SW0, D to SW1, F1 to A1; F2 to B1 and F3 to L1.
- (2) Follow the input sequences given below; record the output states in Table 1.2.

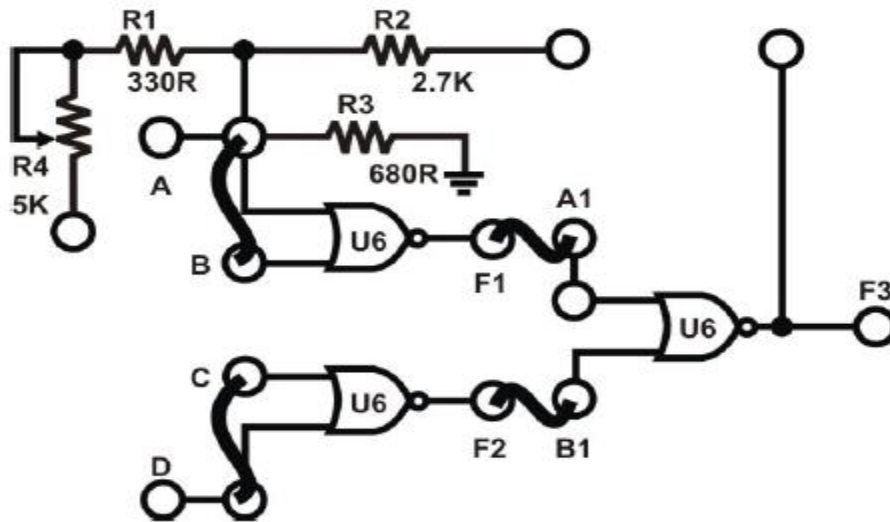


Fig. 1.6 NOR gate used as AND gate

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

Table 1.2

1.4.2 NAND Gate Circuit

The symbol of a NAND gate is shown in Fig. 1.7. The Boolean expression for a NAND gate is $F = (AB)'$; in De Morgan's theorem, $(AB)' = A' + B'$.

When $A=B$, $F = (AB)' = A'$. When $B=1$, $F = (AB)' = (A \cdot 1)' = A'$. Like the NOR gates, NAND gates can be used to construct just about any basic logic gates. We will attempt to construct various basic gates in this experiment by connecting NAND gates in different ways.

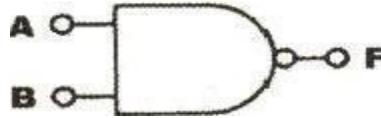
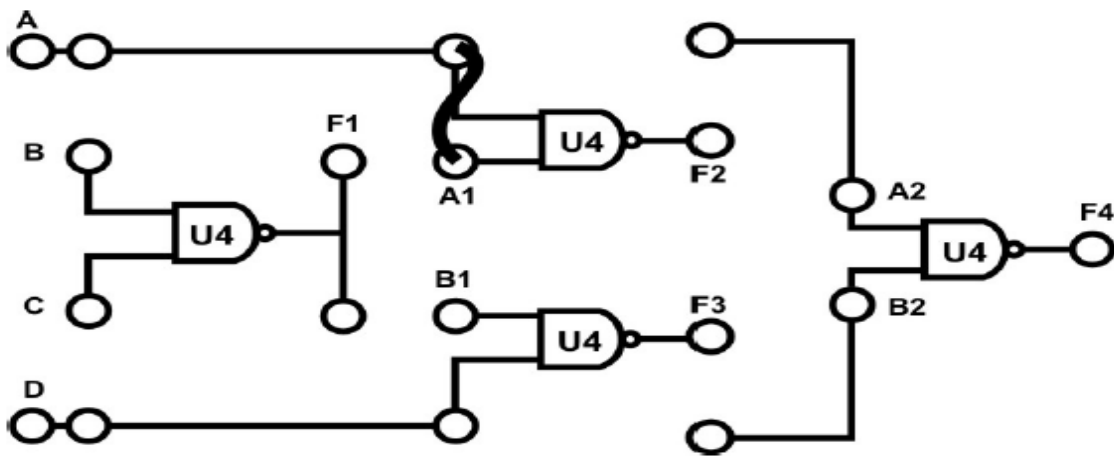
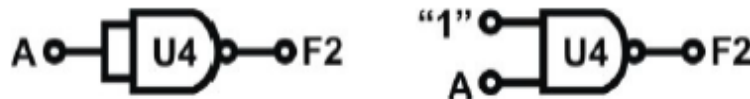


Fig. 1.7 Symbol of NAND gate

1. Set module IT-3002 and locate block NAND gate. Insert connection clips according to Fig. 1.8 (a), using U4 to construct the NOT gate as shown on left side of Fig. 1.8 (b).



(a) Wiring diagram (KL-26001 block b)



(a) NOT gate constructed with NAND gate

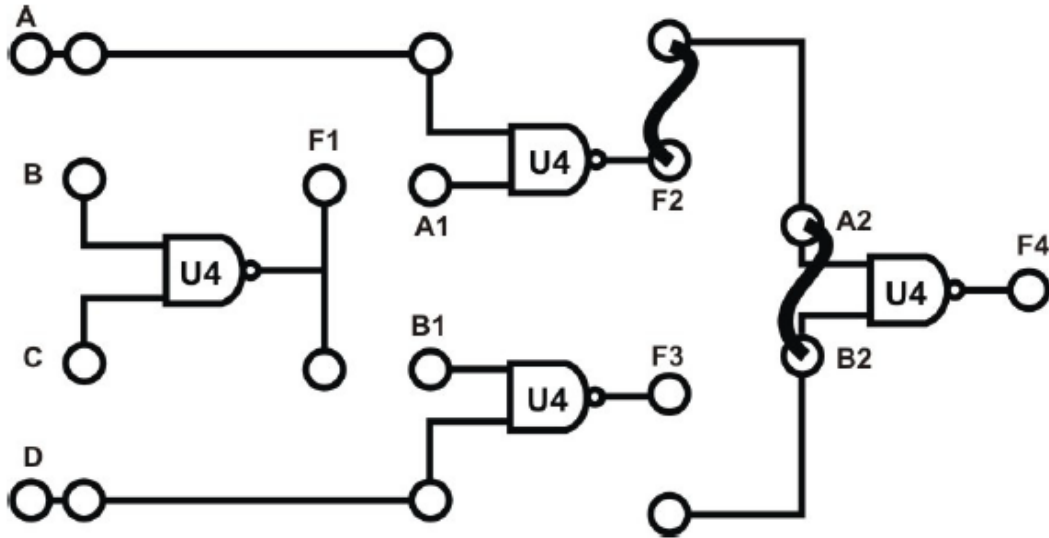
Fig.1.8 NOT gate constructed with NAND gate

2. Connect input A to A1 and Data Switch SW1 and output F2 to Logic Indicator L1. Observe the output states.
 When SW1= "0", F2= _____
 When SW1= "1", F2= _____

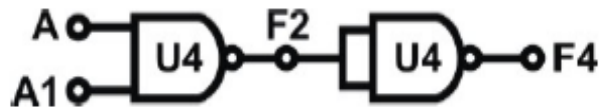
Does the circuit act as a NOT gate?

- Connect input A to +5V("1") and remove the connection clip between A and A1 to create the NOT gate shown on the right side of Fig. 1.2.2 (b). Connections remain the same. Observe the output states.
 When SW1= "0", F2= _____
 When SW1= "1", F2= _____
- Remove connection clips and insert them again according to Fig. 1.9 (a) to construct the AND gate shown in Fig. 1.9 (b). Connect A to SW1, A1 to SW2 and F4 to L1.

Does the circuit act as an AND gate?



(a) Wiring diagram (KL-26001 block b)



(b) Equivalent to an AND gate

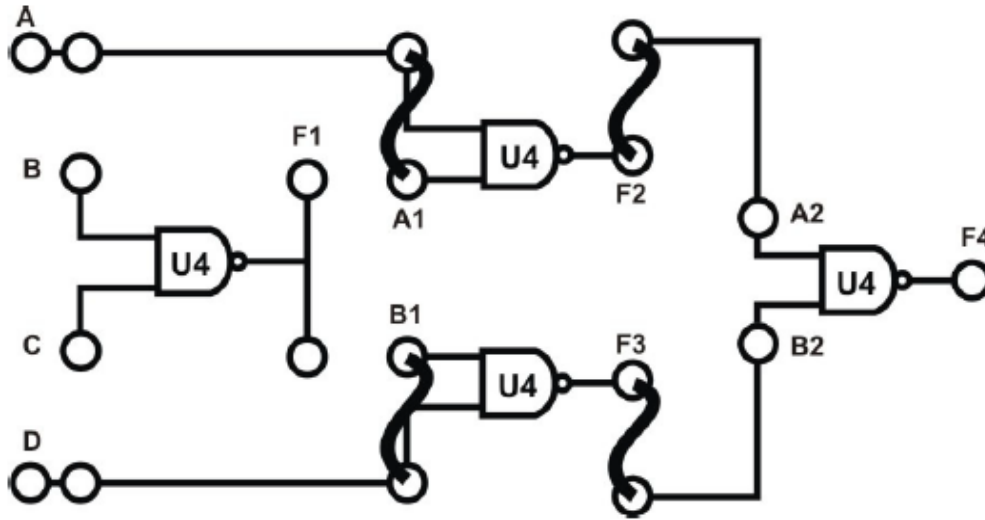
Fig. 1.9 AND gate constructed with NAND gates

- Follow the input sequences given below and record the outputs in Table 1.3. Does the circuit act as a NOT gate?

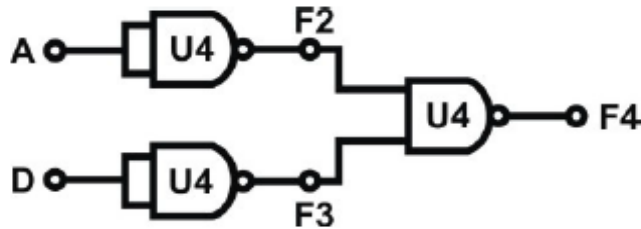
SW2(A1)	SW1(A)	F4
0	0	
0	1	
1	0	
1	1	

Table 1.3

6. Insert connection clips according to Fig. 1.10 (a) to construct the circuit of Fig. 1.10 (b). Connect A to A1 and SW1; F2 to A2; D to B1 and SW2; F3 to B2; F4 to L1.



(a) Wiring diagram (KL-26001 block b)



(b) Equivalent to an OR gate

Fig 1.10 OR gate constructed with NAND gates.

7. Follow the input sequences in Table 1.4 and record the outputs. Does the circuit act as an OR gate ($F=A+B$)?.

SW2(D)	SW1(A)	F4
0	0	
0	1	
1	0	
1	1	

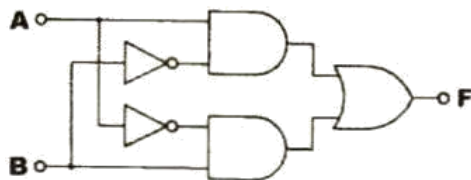
Table 1.4.

1.4.3 XOR Gate Circuit

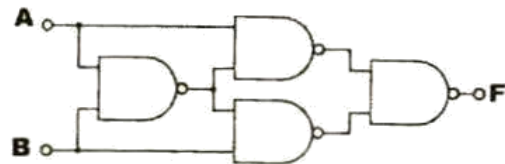
The symbol of an XOR gate is shown in Fig. 1.11. The output F is equal to $A \oplus B = A'B + AB'$. XOR gates can be constructed using NOT, OR, AND, NOR or NAND gates or by using four NAND gates, as shown in Fig. 1.12 (a) and (b).



Fig. 1.11 Symbol of XOR gate



(a) Constructed with basic gates



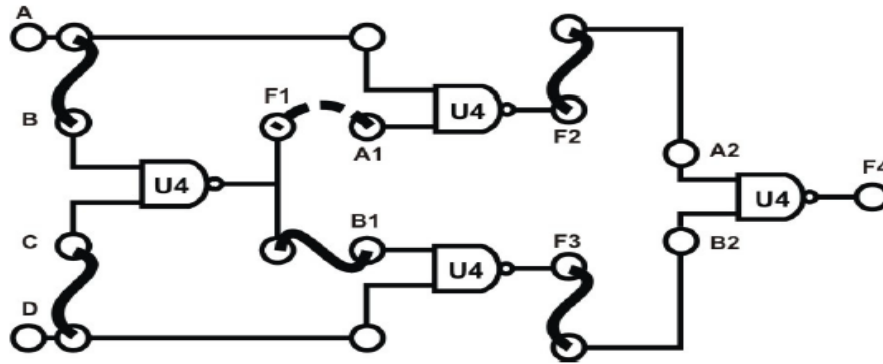
(b) Constructed with NAND gates

Fig. 1.12 XOR gate circuits

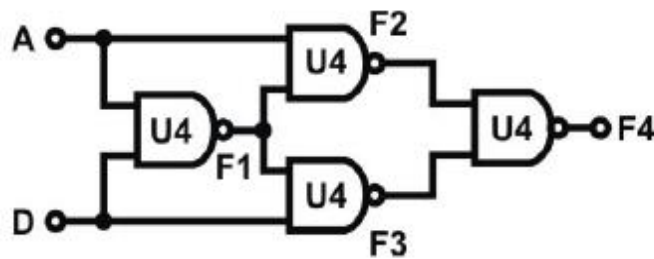
Since $F = A'B + AB'$, when $B=0$, $F = A' \cdot 0 + A \cdot 0' = A \cdot 1 = 1$ and the circuit act as buffer. When $B=1$, $F = A' \cdot 1 + A \cdot 1' = A' \cdot 1 = A'$, the circuit act as an inverter. In other words, the input state of an XOR gate determines whether it will act as a buffer or an inverter. In this experiment, we will use basic logic gates to construct XOR gates and study the relationship between the inputs and outputs.

(A) Constructing XOR gate with NAND gate (Module IT-3002 block NAND gates)

1. Insert connection clips according to Fig. 1.13 (a) to construct the circuit of Fig. 1.13 (b). Connect inputs A to SW1, D to SW2, outputs F1 to L1, F2 to L2, F3 to L3 and F4 to L4.



(a)



(b) Equivalent Circuit

Fig. 1.13 XOR gate Using NAND gates

2. Follow the input sequences for A and D in Table 1.5 and record the outputs.

INPUT		OUTPUT			
D	A	F1	F2	F3	F4
0	0				
0	1				
1	0				
1	1				

Table 1.5

3. Determine the Boolean expression for F1, F2, F3 and F4.

(B) Constructing XOR Gate with Basic Gate (Module IT-3002 block Comparator1)

1. Insert connection clips according to Fig. 1.14 (a) to construct the equivalent circuit of Fig. 1.14 (b).

2. Connect inputs A, B to SW1, SW2 outputs F1, F2, F3 to L1, L2, L3.

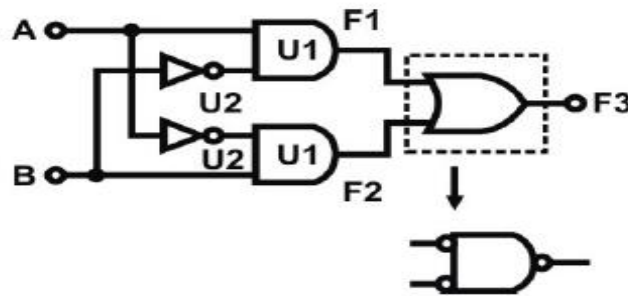
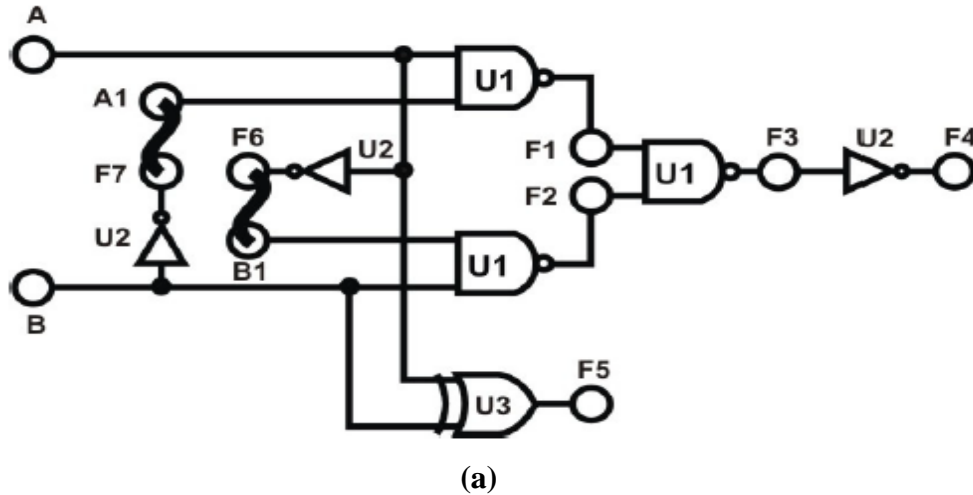


Fig. 1.14 XOR gate Using Basic gates

3. Follow the input sequences for A and B in Table 1.6 and record the outputs.

INPUT		OUTPUT		
SW2(B)	SW1(A)	F1	F2	F3
0	0			
0	1			
1	0			
1	1			

Table 1.6

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 Fig. 1.15. The Boolean expression for the output F is:

$$F = (AB+CD)' \dots\dots\dots (1)$$

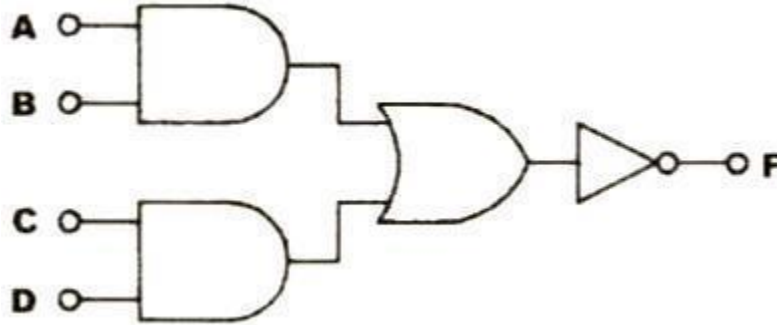


Fig. 1.15 AOI gate

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

$$F = (A'+B')(C'+D') \dots\dots\dots (2)$$

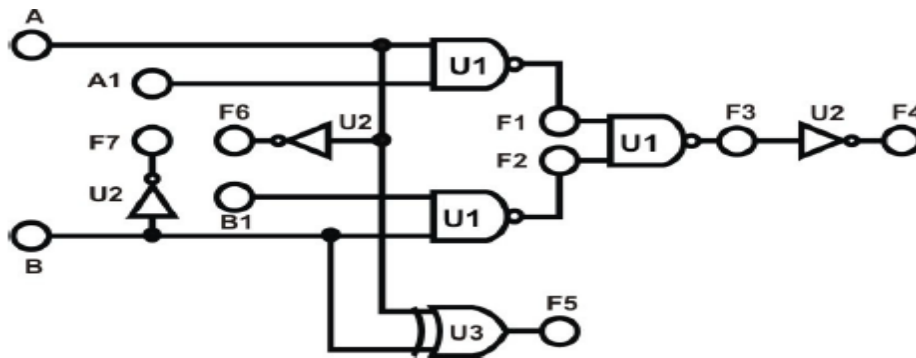
Eq. (1) is also referred to as "Sum of Products".

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

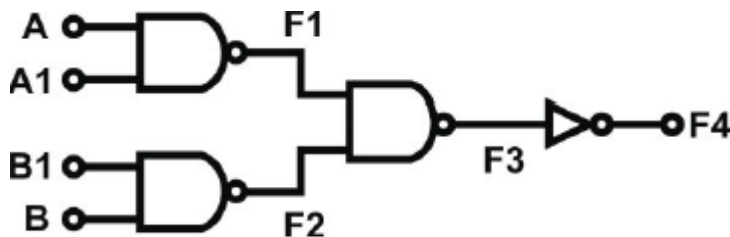
Basically, the A-Q-I gate is a "Sum of Products" logic combination.

PROCEDURE

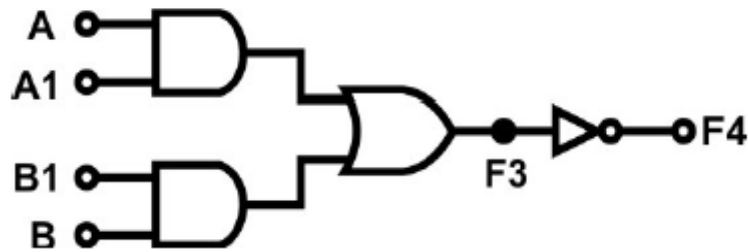
1. Use U1 and U2 on block Comparator 1 of module IT-3002, shown in Fig. 1.16 (a), to construct the A-O-I gate of Fig. 1.16 (b). Fig. 1.16 (c) is the equivalent A-O-I circuit.



(a) Wiring diagram



(b) Actual circuit



(c)Equivalent circuit

Fig. 1.16: AOI circuit.

2. Connect inputs A, A1, B, B1 to Date Switches SW0, SW1, SW2 and SW3 respectively. Connect outputs F3, F4 to Logic Indicators L1 and L2.
3. Set $B \times B1$ to "0", follow the input sequences for A, A1 in Table 1.7 and record the outputs.

A1	A	F3	F4
0	0		
0	1		
1	0		
1	1		

Table 1.7

Does F3 act as an AND gate between A and A1?

4. When $B \times B1$ is "0", does F3 act as an AND gate between A and A1? ($F3 = A \times A1$)
5. When $A1 \times A$ is "0", follow the input sequences for B, B1 in Table 1.8 and record the outputs.

B1	B	F3	F4
0	0		
0	1		
1	0		
1	1		

Table 1.8

Does F3 act as an AND gate between B and B1?

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

1.5 Logic problem

- Given the logic system is shown in Fig. 1.17; output of the system F is in "1" state in each of the following condition
 C and D are in the "1" state.
 A, B and D are in the "1" state, C is in "0" state.
 B and D are in the "1" state A and C are in "0" state.
 C and B are in the "1" state A and D are in "0" state.
 C is in the "1" state A, B and D are in the "0" state. Complete the truth table of the above logic system.



Fig. 1.17 Block diagram

- Write the Boolean equation as canonical sum.
- Minimize the expression that you obtained using laws of Boolean algebra. Write down reduced equation.
- Draw the Karnaugh map of logic problem and indicate columns and rows circle the sub-cubes and write down the equation that you obtained is the equation identical to the one that you obtained in paragraph C.
- Implements the reduced expression that you obtained in the previous paragraph using NAND gates only.

1.6 Post-lab:

Please solve the following problems. If a report is required, include the solutions in the report. Otherwise, submit the solutions separately.

- Draw the logic diagram showing the implementation of the following Boolean equation using "AND" gates $F = AB(CA)$.
- Draw the logic diagram of the following Boolean equations
 - $F_2 = (A+B)(CD+A)$
 - $F_3 = (ABC+D)C$
- Implement the OR operation using AND, NOT gate.
- Implement the AND gate using OR, NOT gate. Draw the logic diagram used in both cases and write 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 Demorgarn's theorem).
- Suggest a logic diagram which will give a NAND gate with four inputs using two input NAND gates, Implement suggested network.
- Show how is it possible to reduce Boolean expressions by means of karnaugh map
 $F_1 = A'BCD + ABCD' + A'BCD' + ABCD'$
 $F_2 = A'B'C'D' + AB'CD' + A'B'CD' + A'BC'D'$

Implement the minimal expressions using NAND gates.

ENCS 211 Digital Electronics and Computer Organization

Lab



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization

Lab

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

1.1 Objectives

1. To understand the construction and operating principle of digital comparators.
2. To construct comparators with basic gates and IC.
3. To implement half- and full adders using basic logic gates and IC.
4. To understand the theory of complements.
5. To construct half- and full- subtractor circuits.

1.2 Apparatus

1. IT-3000 Basic Circuit Lab.
2. IT-3002 Basic Gates Circuit.
3. IT-3003 Adder/Subtractor circuits

1.3 Pre Lab

Prepare all sections and work out all the required designs.

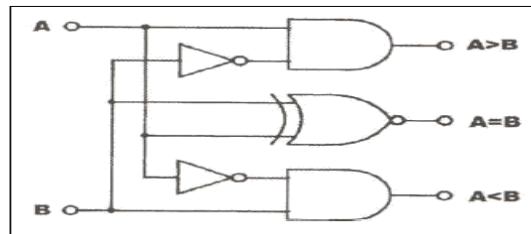
ENCS 211 Digital Electronics and Computer Organization

Lab

1.4 Theory

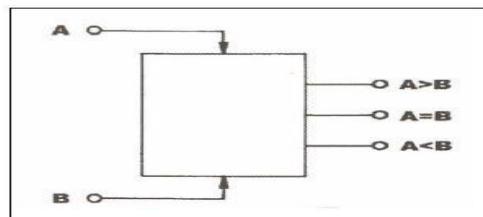
1.4.1. Comparator Circuit

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, there are three possible outputs: $A > B$, $A = B$, and $A < B$. **Fig 2.1** shows the schematic and symbol of a simple



comparator.

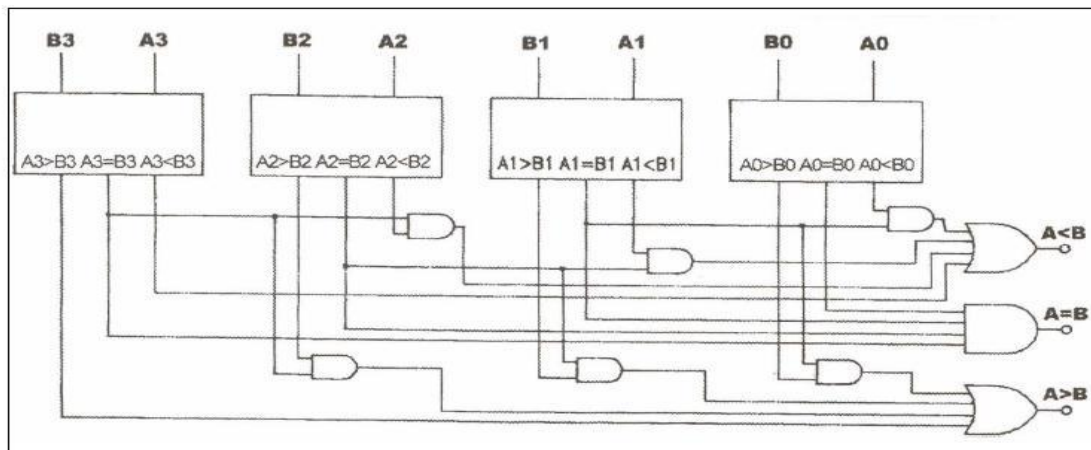
(a) Logic Diagram



(b) Circuit symbol

Fig2.1: Comparator

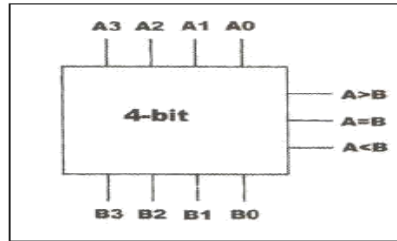
In actual applications 4-bit comparators are used most often. In a 4-bit comparator, each bit represents $2^0, 2^1, 2^2$ and 2^3 . Comparison will start from the most significant bit (2^3), if input A is greater than input B at the 2^3 bit, the “ $A > B$ ” output will be in high state. **Fig 2.2** shows the schematic and symbol of 4 bit comparator



ENCS 211 Digital Electronics and Computer Organization

Lab

(a) Constructed with four 1-bit Comparators



(b) Circuit Symbol

Fig 2.2: 4-bit Comparator

1.4.2. Half- and Full- Adder Circuits

Digital computers perform a variety of information processing tasks. Among the functions encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. Combinational circuit that performs the addition of two bits is called a half adder. One that performs the addition of three bits (two significant bits and previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

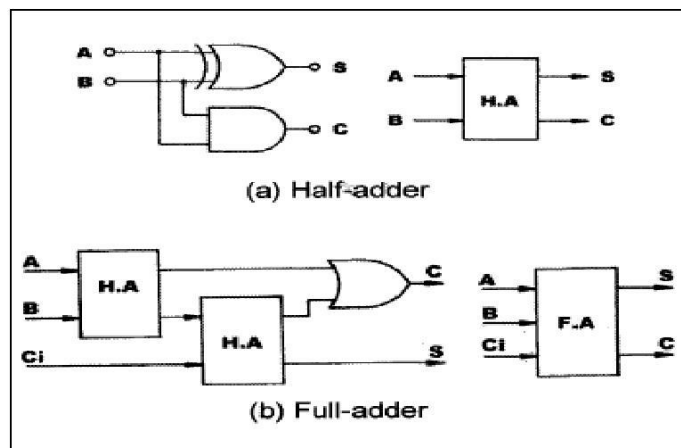
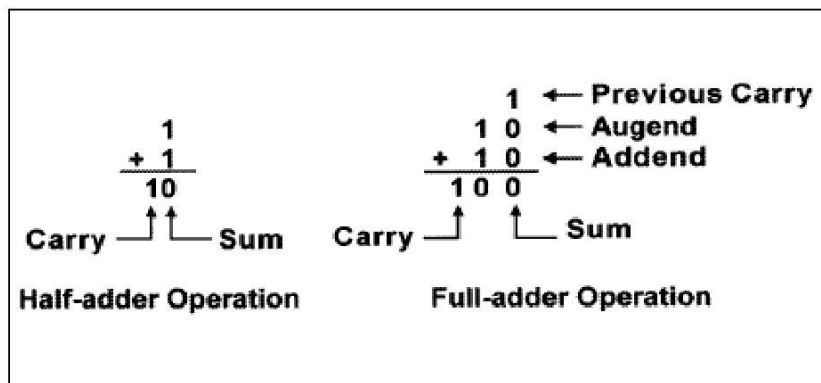


Fig 2.3: Half – and full- adders

1.4.3. Half- and Full-Subtractor Circuits

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 of 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.

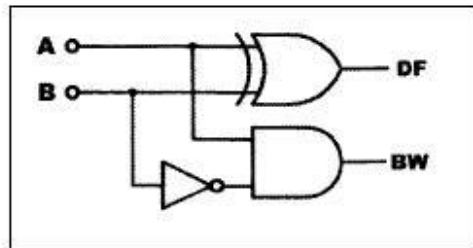


Fig 2.4: Half-Subtractor

The full-subtractor has to consider borrow(s) from previous stages.

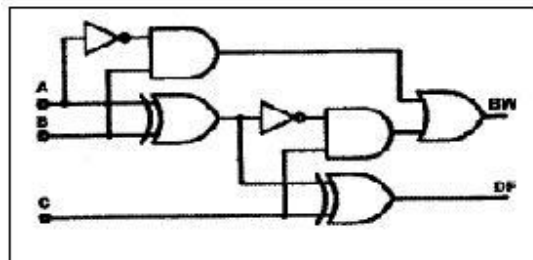


FIG 2.5: Full-Subtractor

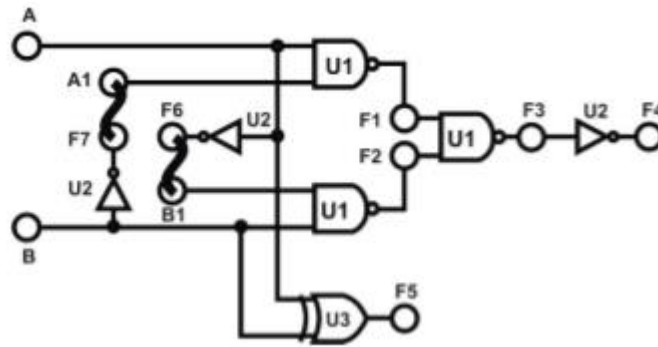
1.5 Procedure

1.5.1. Comparator Circuits

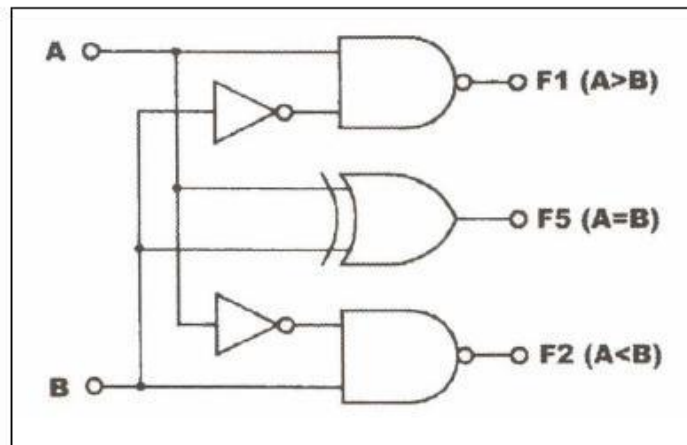
A. Constructing Comparator with Basic Logic Gates

Use Module IT-3002 block Comparator 1:

1. Insert connection clips according to Fig. 2.6 (a). U1, U2 and U3 will be used to construct the 1-bit comparator shown in Fig. 2.6 (b).



(a) Wiring diagram (IT-3002 Comparator 1 block)



(b) Logic Diagram

Fig 2.6: 1-bit comparator

2. The inputs are triggered by high state voltage. Connect inputs A and B to Data Switch SW1 and SW2. The outputs are triggered by low state voltage. Connect outputs F1, F2, F5 to Logic Indicators L1, L2 and L3 respectively.

3. Follow the input sequences in Table 2.1. Measure and record the outputs.

INPUTS			OUTPUTS		
B (SW2)	A (SW1)		F1 (L1)	F2 (L2)	F5 (L3)
0	0	A=B			
0	1	A>B			
1	0	A<B			
1	1	A=B			

Table 2.1

(B) Constructing Comparator with TTL IC

1. Block (Comparator 2) of module IT-3002 will be used in this section. U5 is a 74LS85 4-bit Comparator IC.

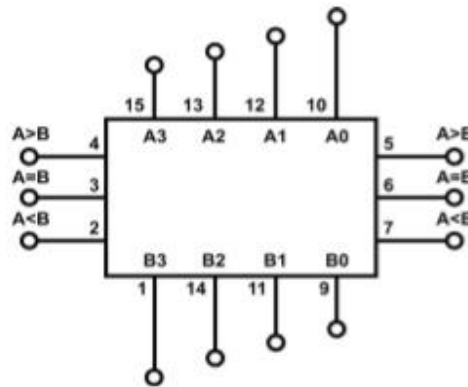


Fig 2.7: IT-3002 block Comparator 2

2. Connect input A<B to SW1, A=B to SW2, A>B to SW3. The inputs A1~A4 and B1~B4 of the 74LS85 also connected to the BCD rotary switch.

3. Set comparing inputs A1~A4=As, B1~B4=Bs and As=Bs from rotary switch, follow cascading inputs sequences in Table 2.2 and record the outputs

INPUT			OUTPUT		
A>B	A=B	A<B	A>B	A=B	A<B
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	1			

Table 2.2

4. Set SW3 to “0”; SW2 to “1”; SW1 to “0”. Observe and record the outputs under the following conditions:

- (1) $A_s > B_s$
- (2) $A_s = B_s$
- (3) $A_s < B_s$

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

1.5.2. Half- and Full-Adder Circuits

A. Constructing Half- and Full-Adders with Basic logic Gates

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

Use Module IT-3003 block Half-Adder:

1. Insert connection clips according to Fig 2.8, using U5 and U6 to assemble the half-adder circuit of Fig. 2.9. Connect +5V of module IT-3003 to the +5V output of fixed power supply.

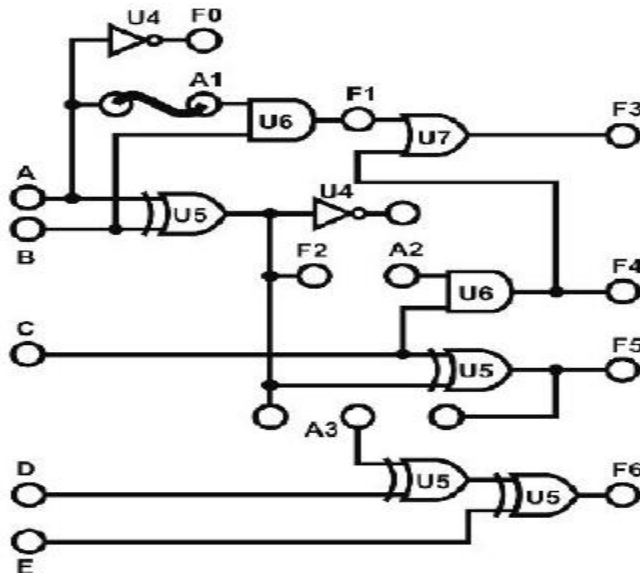


Fig 2.8: Wiring Diagram (IT-3003 Half Adder block)

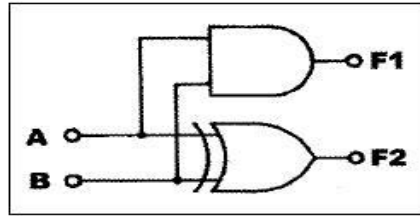


Fig 2.9: Half-Adder Circuit

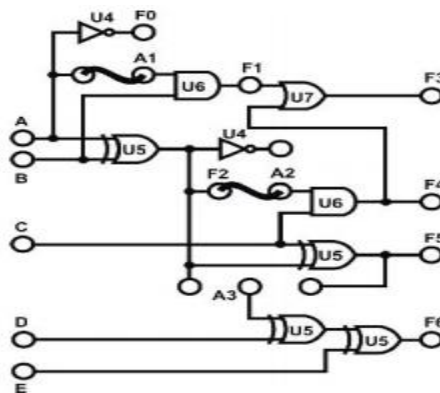
2. Connect inputs A, B to Date Switches SW0, SW1 and connect outputs F1, F2 to logic indicator L1 and L2. Follow the input sequences for A and B in Table 2.3 and record the output states.

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

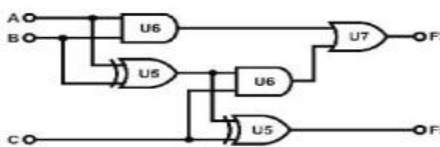
Table 2.3

3. Reassemble the circuit according to Fig. 2.10 (a) to construct the full-adder circuit shown in Fig. 2.10 (b).

4. 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 2.4 and record output states.



(a)



(b)

Fig 2.10: Full-Adder Circuit

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

Table 2.4

B. Constructing 4-Bit Full-Adder with IC

1. U9 on block Full-Adder of module IT-3003 is used as a 4-bit adder. Connect input Y5 to SW0, so the XOR gates U8, which are connected to Y0~Y3, will act as buffers. Connect input X0~X3 (addends), Y0 ~Y3 (augends) to DIP switches DIP2.0~2.3 and DIP1.0~1.3 respectively. Connect F1, Σ0, Σ1, Σ2, Σ3 to L1~L5. Follow input sequences in Table 2.5 and set SW0 to “0”; record F1 and Σ in binary numbers.

$$X = X_3X_2X_1X_0$$

$$Y = Y_3Y_2Y_1Y_0$$

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

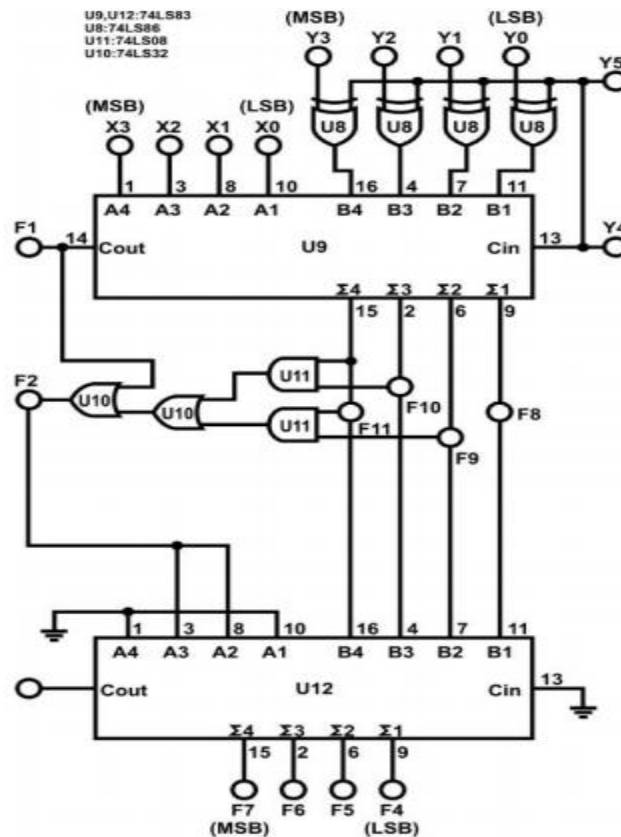


Fig 2.11: Wiring Diagram (IT-3003 Full-Adder block)

2. Connect inputs X0~X3 to DIP 1.0~1.3; Y0~Y3 to DIP 2.0~2.3; Y5 to “0”. U9 and U12 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 U12 to another 7-Segment display SEG-3 and F3 to L10.

3. 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 2.6 and record the output states.

INPUT								OUTPUT (U9)					LAST (U12)					
X3	X2	X1	X0	Y3	Y2	Y1	Y0	F1	F11	F10	F9	F8	F2	F3	F7	F6	F5	F4
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											

Table 2.6

1.5.3. Half- and Full Subtractor Circuits.

A. Constructing Half-/Full Subtractors with basic logic Gates.

Design the Logic Diagram, Boolean function, and truth table of a half- and full-Subtractor.(Pre lab)

Use Module IT-3003 block Half-Adder:

1. Insert connection clips according to Fig. 2.13.

2. Connect inputs A~C to Data Switches SW0~SW2 outputs F2 to Logic Indicator L1; F1 to L2; F3 to L3; F5 to L4. When C=0 the circuit is a half-subtractor. F1 is the borrow output; F2 is the difference and $F5=F2$; $F4=0$; $F3=F1$. When C=1 the circuit is a full-subtractor. F3 is the borrow output and F5 is the difference output.

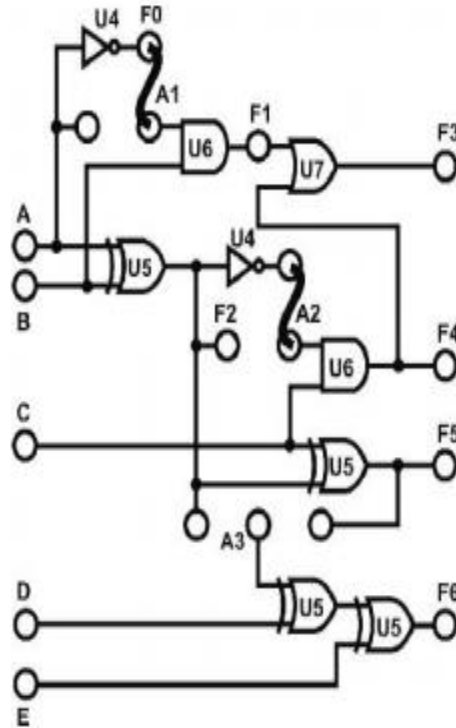


Fig 2.13: Wiring Diagram (Half-Subtractor)

3. Follow the input sequences in Table 2.7 and record output states.

Inputs			Outputs			
C	A	B	F1	F2	F3	F5
0	0	1				
0	0	0				
0	1	1				
0	1	0				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Table 2.7

B. Constructing 4-Bit Full-Subtractor with IC

Use Module IT-3003 block Full Adder (Fig. 2.14):

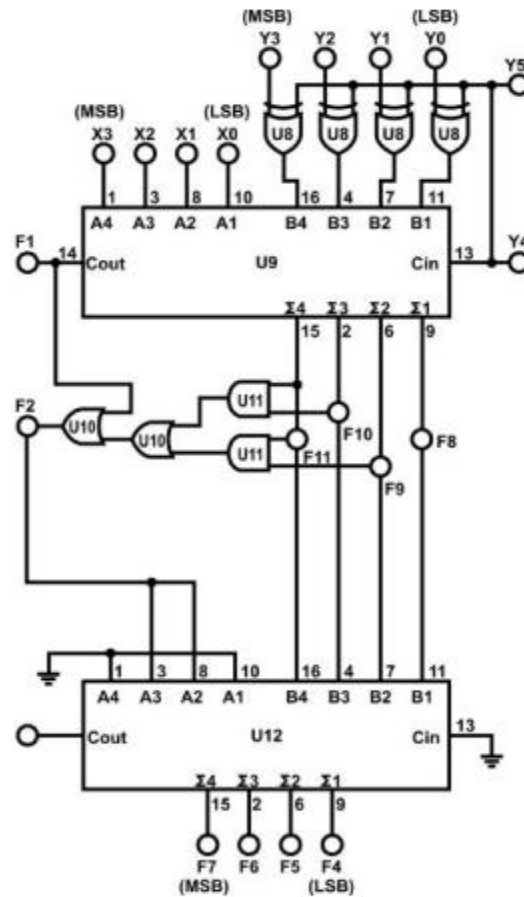


Fig 2.14

2. Connect inputs X3~X0 (minuend) to DIP Switch 1.3~1.0; Y3~Y0 (subtrahend) to DIP 2.3~DIP2.0; 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 U9=1). Follow the input sequences below and record the output states in Table 2.8.

INPUT								OUTPUT				
X3	X2	X1	X0	Y3	Y2	Y1	Y0	F1	F11	F10	F9	F8
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					

Table 2.8

1.6 Problem: Building Comparator Circuit

A 4-input, 3-output circuit that compares 2-bit unsigned numbers and output 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 211

Digital Electronics and Computer Organization Lab

3. 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 IC.
- To construct multiplexers and demultiplexers using basic gates and IC

3.2 EQUIPMENT REQUIRED

1. IT-3000 Basic Electricity Circuit Lab.
2. IT-3004 Encoder/Decoder Circuits.
3. IT-3005 Multiplexer/Demultiplexer Circuits.

3.3 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 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.

3.4PRE-LAB

1. Prepare all sections and Handout all the required design to your teaching assistant.
2. 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. $F(A,B,C) = \sum(\dots)$):

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

(b) Sketch on figure 3.1 the input connections necessary to implement the function in (a).

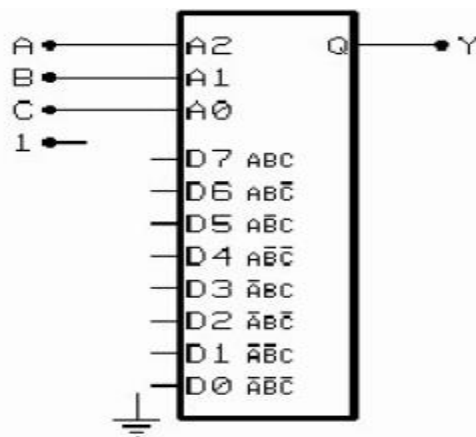


Figure 3.1: 8-to-1 Multiplexer

3. Design a circuit which uses an SN74138 demultiplexer to implement a sum-of-products expression, as follows:

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

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

(b) The demultiplexer output is selected, and will go low, by the address on inputs A, B, and C when the IC is enabled. Therefore, we can create the output function Y by summing together the outputs indicated by the summation form of the expression. Since the outputs of the demultiplexer are active-low, this is done with a NAND gate. Connect each of the TRUE minterm outputs of the demultiplexer in Figure 3.2 (indicated by the summation equation) to an input of the NAND gate. Connect all unused NAND inputs to logic 1.

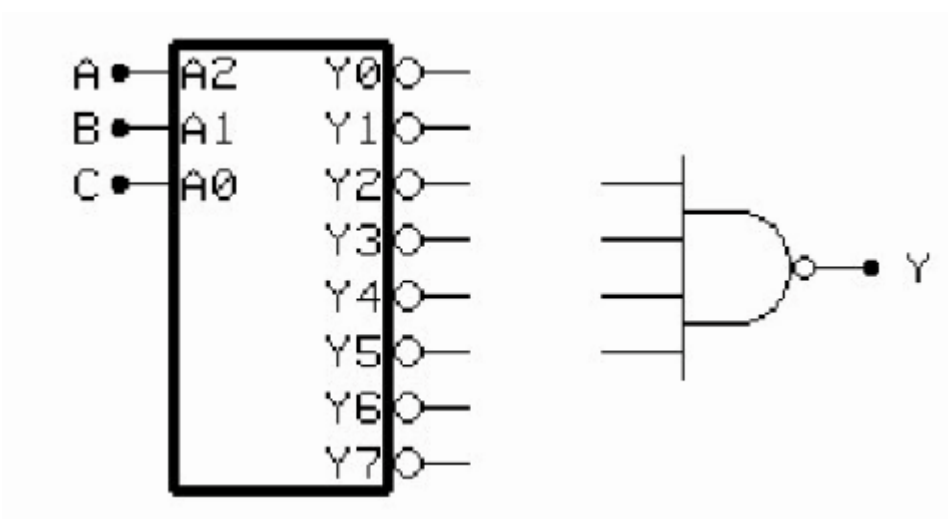


Figure 3.2:3-to-8 Demultiplexer

3.5 PROCEDURE:

A. Constructing 4-to-2-Line Encoder with Basic Gates

1. Constructing a 4-to-2 Encoder with Basic Gates (Module IT-3004 block Encoder 1)

1. Insert connection clips according to Figure 3.3.

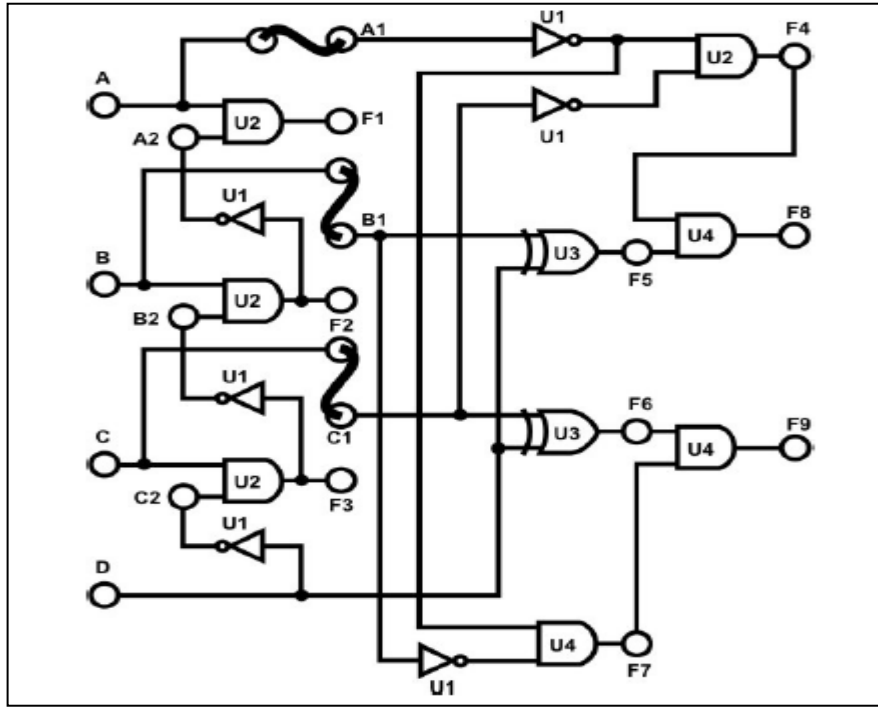


Figure 3.3: wiring diagram of 4-to-2 line Encoder

2. Connect +5V of module IT-3004 to the +5V output of fixed power supply section of IT-3000.

3. Connect inputs A~D to Date Switches SW0~SW3 respectively; outputs F8 and F9 to Logic Indicator L0 and L1.

4. Follow the input sequences for D, C, B, A; in Table 3.1 and record the output states.

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		

Table 3.1

B. Constructing 9-to-4-Line Encoder with TTL IC

1. The 74147 (U5) on block Encoder 2 of module IT-3004 is used in this section of the experiment. Connect +5V of module IT-3004 to the +5V output of fixed power supply.

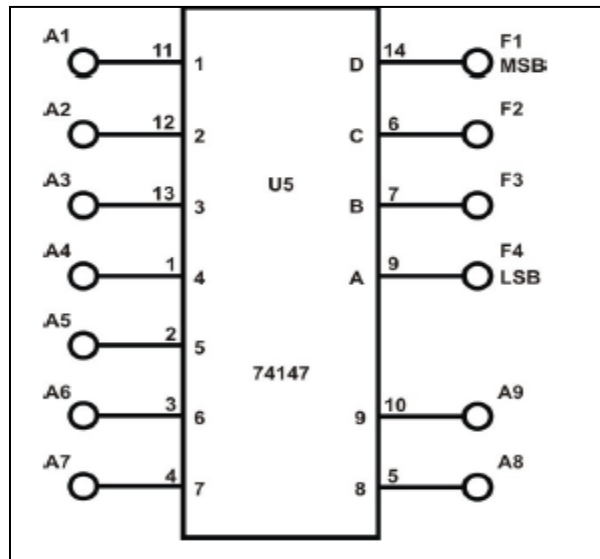


Figure 3.4:(74147)BCDPriorityEncoder

2. Connect inputs A1~A8 to DIP Switches 1.0~1.7 and A9 to 2.0. Connect outputs F1~F4 to Logic indicators L1~L4. Follow the input sequences given in Table 3.2 and record output states.

A9	A8	A7	A6	A5	A4	A3	A2	A1	F4	F3	F2	F1
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	0	1	1	1				

Table 3.2

C. Constructing 2-to-4 Line Decoder with Basic Gates

1. Block Decoder 1 of module IT-3004 will be used in this section of the experiment. Connect +5V of module IT-3004 to the +5V output of fixed power supply.

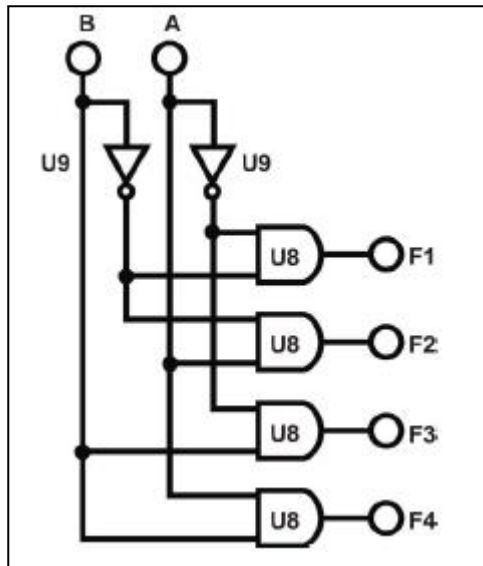


Figure 3.5: 2-to-4 Decoder

2. Connect inputs A, B to Data Switches SW0 and SW1. Connect outputs F1~F4 to Logic Indicators L0~L3 respectively.
3. Follow the input sequences for A and B in Table 3.3 and record output states.

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

Table 3.3

D. Constructing 4-to-10 Line Decoder with TTL IC

1. U6 (7442) on block Decoder 2 of module IT-3004 will be used in this section of the experiment. 7442 is a BCD-to-Decimal decoder IC.

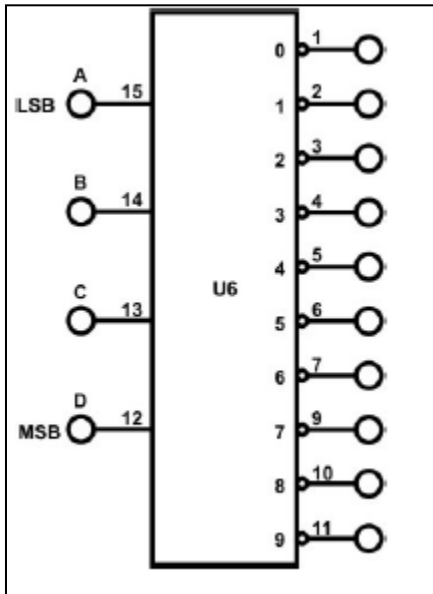


Figure 3.6:4-to-10lineDecoder

2. Connect inputs A, B, C and D to the Data Switches SW0, SW1, SW2 and SW3. Connect outputs to Indicator L0~L9 respectively.
3. Adjust the switches according to Table 2.2. Observe the output states at L0~L9. Record input and output logic states in Table 3.4.

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

Table 3.4

E. Constructing 2-to-1-Line Multiplexer with basic Gates

1. Block Multiplexer 1 of module IT-3005 will be used as a 2-to-1 MUX. Connect +5V of module IT-3005 to the +5V output of fixed power supply.

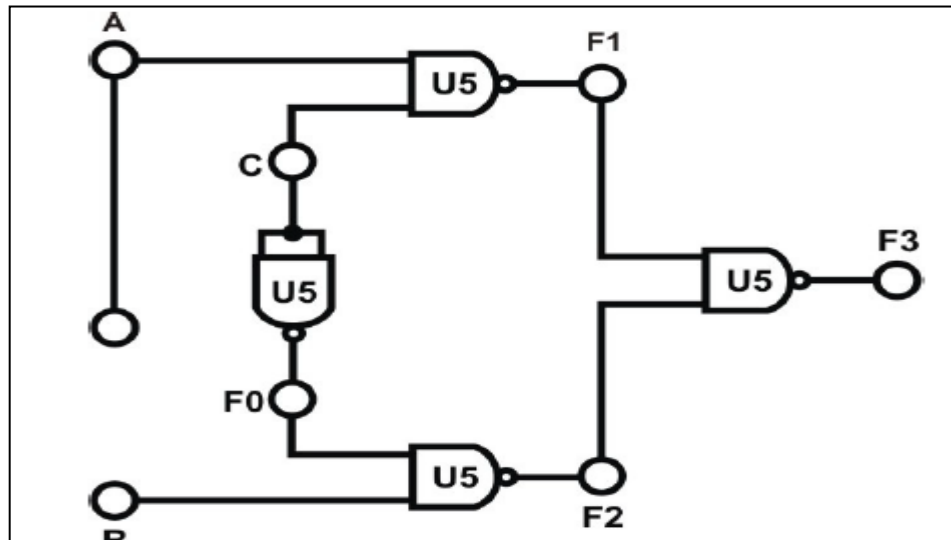


Figure 3.7:2-to-1 Multiplexer

2. Connect inputs A, B to Data Switches SW0, SW1; selector C to SW2. Connect output F3 to Logic Indicator L0.
3. Follow the input sequences in Table 3.5 and record states of F3. Which input (A or B) determines the output?

C	A	B	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	

Table 3.5

F. Constructing 8-to-1 Line Multiplexer with IC

1. U3 (74LS151) on block Multiplexer 2 of module IT-3005 will be used in section of the experiment.

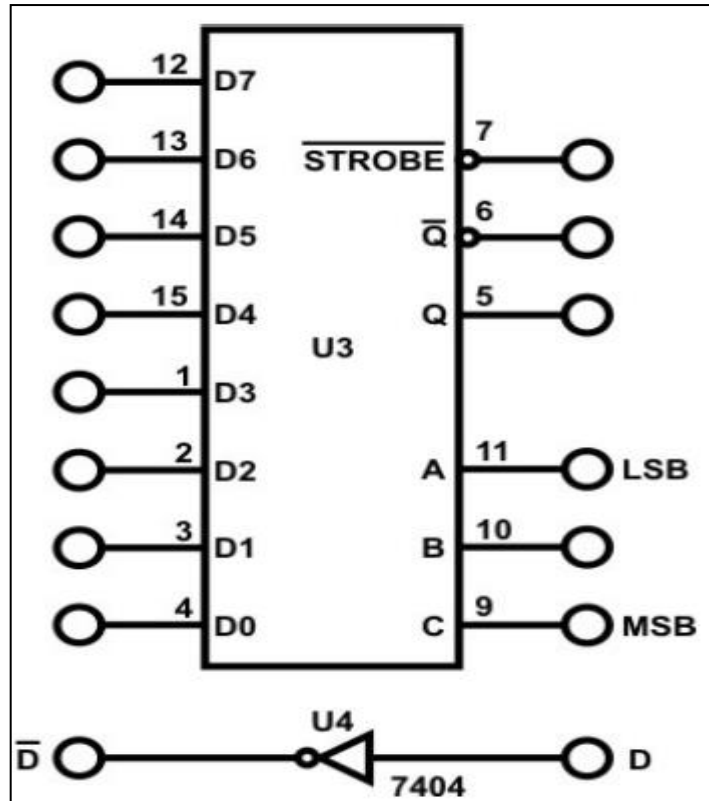


Figure 3.8:8-to-1MUX

2. Refer to the data sheet for specifications of the 74LS151.
 When CBA = "000", data at D0 is send to output Q.
 When CBA = "010", data at D2 is send to output Q.
 When CBA = "111", data at D7 is send 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".
3. 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 record output states. Determine on which input among D0~D7 does Q depend on.

C	B	A	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	

Table 3.6

G. Using Multiplexer to Create a Logic Function

Given the following function:

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

1. Implement the function using the **8-to-1 multiplexer**.
2. Connect inputs D, C, B, A to Data Switches SW3, SW2, SW1, and SW0 respectively. Connect output Y to Logic Indicator L0. Record output states in Table 3.7.

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	

Table 3.7

H. Constructing 1-to-2 Line Demultiplexer with Basic Logic Gates

- Block Multiplexer 1 of module IT-3005 will be used in this section. Insert connection clip according to Fig. 10. Connect A to Data Switch SW0; C to SW3; F1 and F2 to Logic Indicators L0 and L1 respectively.

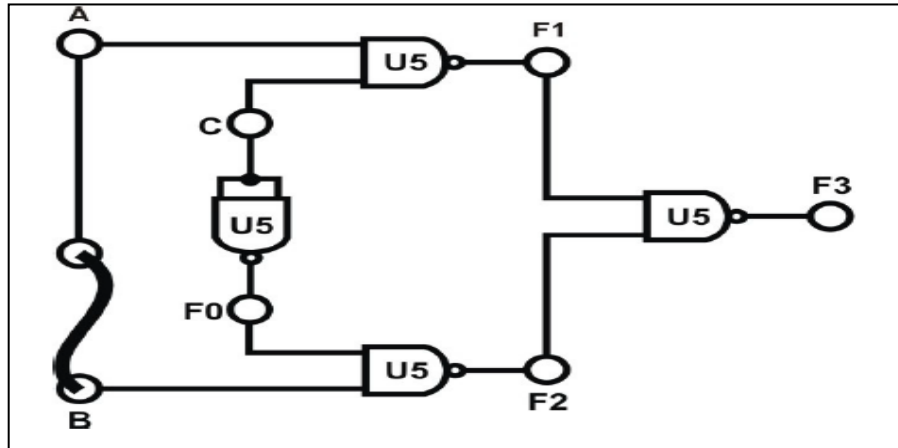


Figure 3.10:1-to-2 Demultiplexer

- 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.

I. Constructing 1-to-8-Line Demultiplexer with CMOS IC

- U6 (4051) on block Demultiplexer of module IT-3005 is used in this section of the experiment. Connect +5V, -5V of module IT-3005 to the +5V and -5V output of fixed power supply respectively.

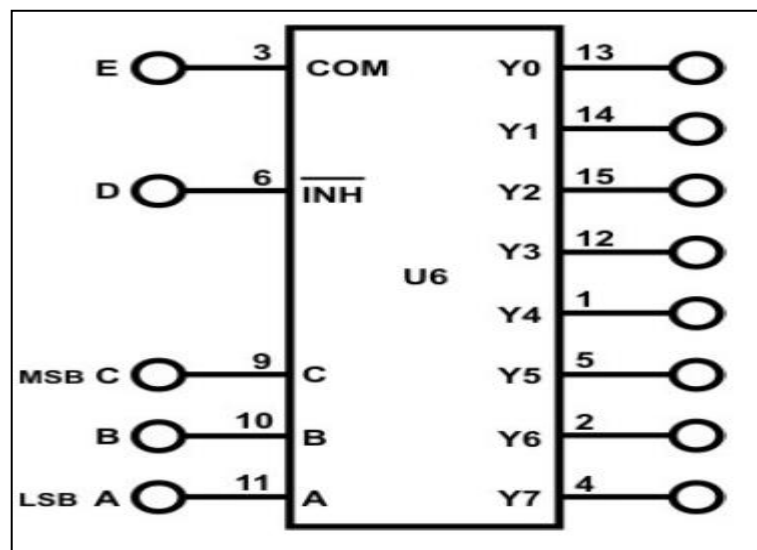


Figure 3.11:1-to-8 Demultiplexer

2. 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.
3. At D=0, apply the input sequence 1-0-1-0 to the common input E and observe outputs Y0~Y7. Did the outputs change as the input sequence is applied?
4. At D=1, apply the input sequence 1-0-1-0 to the common input E and observe outputs Y0~Y7. Did the outputs change as the input sequence is applied?
5. Using the same sequence for E (1-0-1-0) with D=0, follow the sequence for A, B and C given in Table 3.8. Record output states.

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								

Table 3.8

3.6 POST-LAB PROBLEM

Solve the following problem in your report.

Design a Majority Circuit; a circuit that takes 4 inputs and 1 output, its output equals 1 when 3 or 4 of the inputs are 1. You can only use two 4×1 multiplexers.



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

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

4.1 OBJECTIVES

The objective of this laboratory experiment is to continue to experiment with simple digital devices and their operations using breadboard.

4.2 EQUIPMENT REQUIRED

1. KL-22001 Basic Electricity Circuit Lab
2. Breadboard (you may bring your own breadboard, if you have one)
3. Integrated circuits (Chips): IC 7404 (inverter), IC 7408 (2-input AND), IC 7432 (2-input OR), IC 7400 (2-input NAND) and IC 7486 (2-input XOR)

4.3 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 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.
16. Make sure that the power of the board is switched off when you are moving ICs and connecting wires. Switch it on once the circuit is ready.

4.4 PRE-LAB

1. Review how the breadboard works and the way components, including chips are connected to the breadboard as was done in experiment 1. You can watch the following videos for that:
<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 a **Full Adder** using the gates on the chips. Your final circuit must include the IC's, their pin numbers, and the connections between the pins.
4. Design and Implement a 4x1 multiplexer using the gates on the chips. Your final circuit must include the IC's, their pin numbers, and the connections between the pins.

5. Design and Implement a 2-4 “active-low” decoder using the gates on the chips. Your final circuit must include the IC’s, their pin numbers, and the connections between the pins.
6. 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.
7. (Optional??) Use the 2-4 decoder to implement a 2 input function that acts like an equivalence gate: gives 1 on the output if both inputs are equal.

The designs need to be detailed and you need to show all of your designs to the Instructor/TA.

Figure 4.1 shows some digital gates with identification numbers and pin assignment.

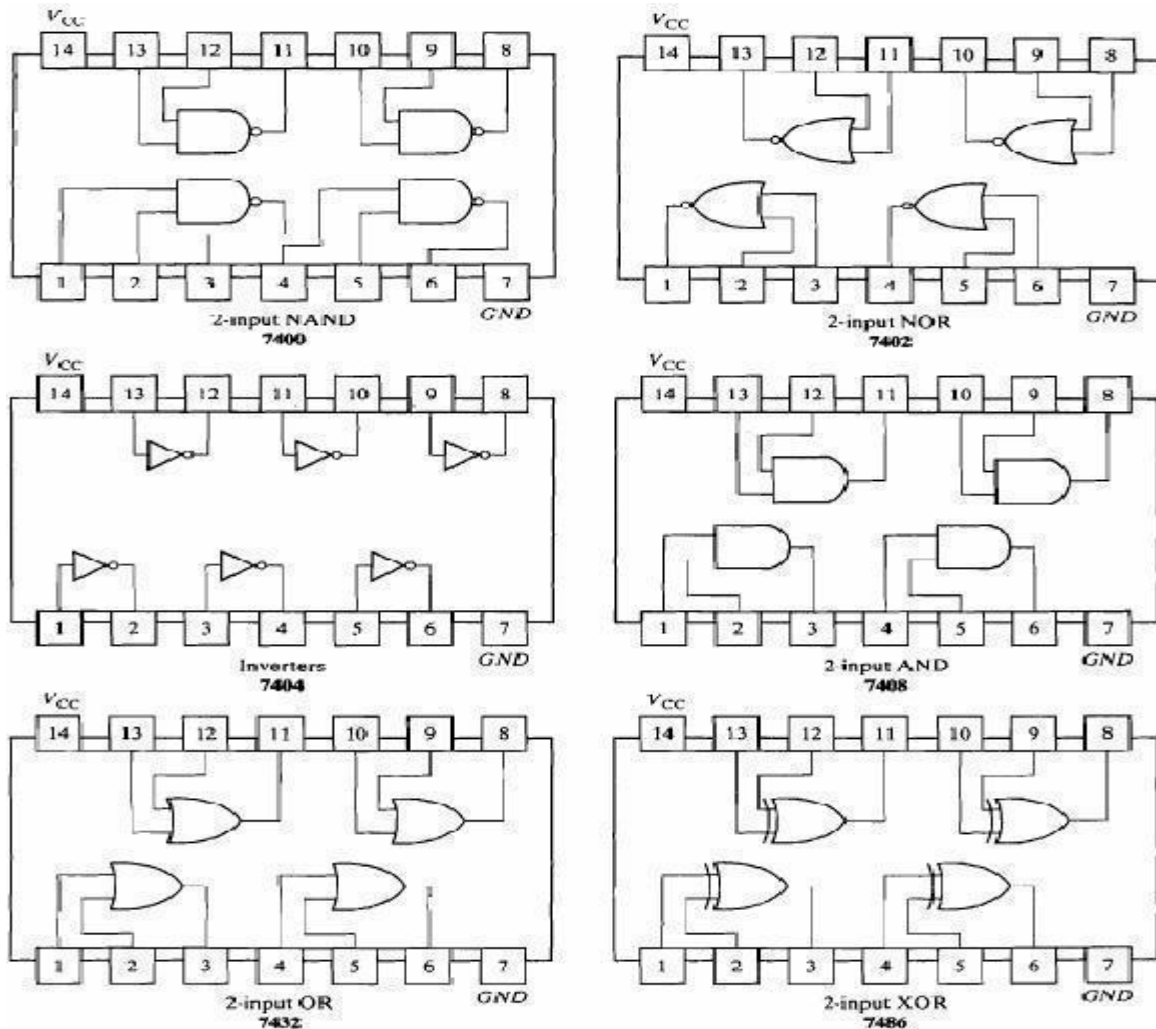


Figure 4.1: DIGITAL GATES IN IC PACKAGES

4.5 PROCEDURE

4.5.1 Verification of basic logic gates

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

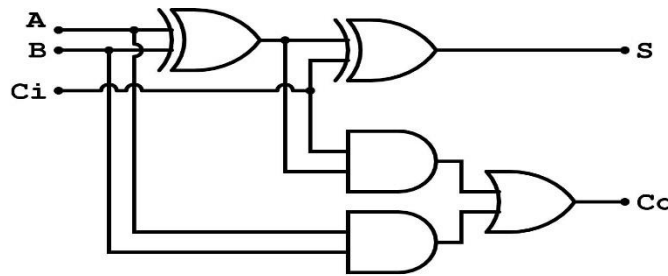
1. Test each chip using the IC tester and make sure that it is functioning properly.
2. Place each chip shown in Figure 4.1 on the breadboard in such a way that its pins are not short-circuited. Make sure power is off while you place IC's and connect wires.
3. Connect GND and +5V for each chip you want 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) chip by observing how the output of the Gates changes in response to input changes.

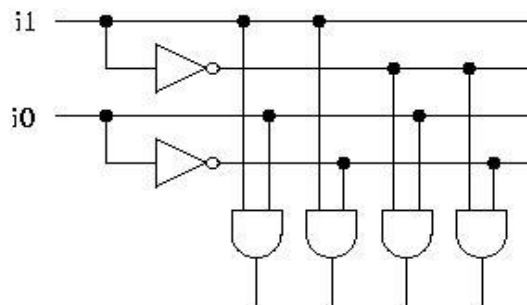
- 4a. how does the gate act if one of its two input is held at “1”?
- 4b. how does the gate act if its two input are connected together?

4.5.2 Full-Adder, Multiplexer and Decoder Implementations

1. (**Adder**) Use any needed gates shown in figure 4.1 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.

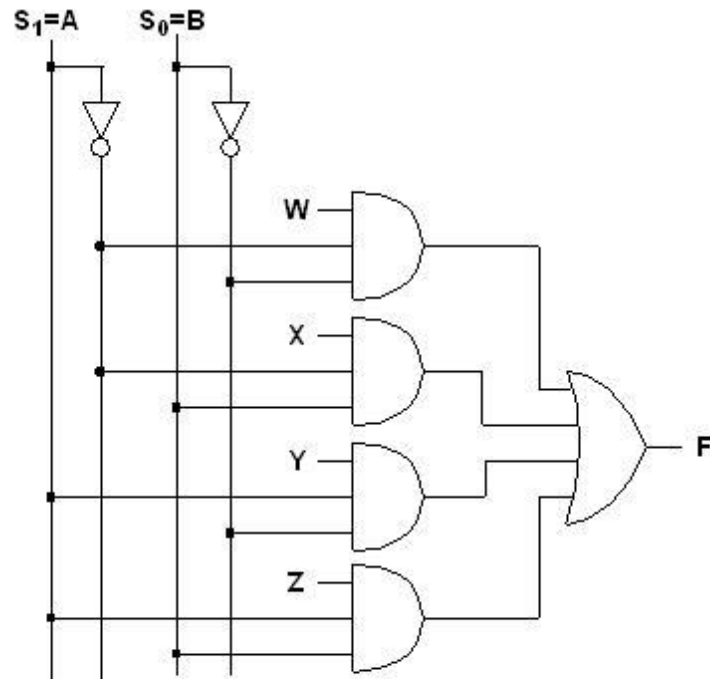


(b) (**Decoder**) Use any needed gates shown in figure 4.1 and the designs you prepared as a pre-lab to implement the 2-4 decoder. Test your design by verifying the truth table of the DECODER.



- a. How do you go about adding an Enable (E) signal to the decoder. Modify the implementation to show that. (Design Only)
- b. How to use that to implement a 3-8 decoder. Show all work in your post lab.

(c) **(Multiplexer)** Use any needed gates shown in figure 4.1 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.



$$F = \bar{A}\bar{B}W + \bar{A}BX + \bar{A}BY + ABZ$$

- a. 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).



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

5. 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

1. IT-3000 Basic Electricity Circuit Lab.
2. IT-3007 J-K Flip-Flop Circuits.
3. IT-3008 Flip-Flop Circuits.

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 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.

5.4 PRE LAB

Prepare all sections and be ready for a quiz!

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 past 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*.

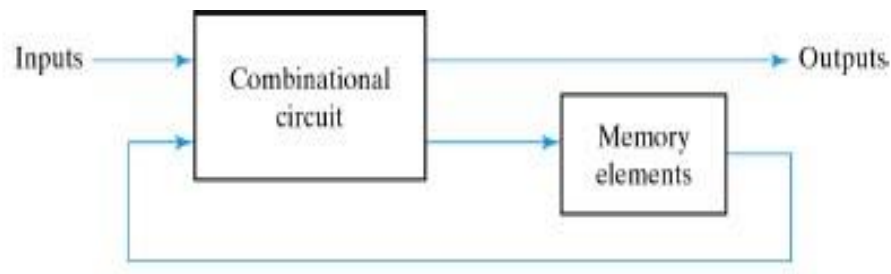


Figure 5.1: Sequential Circuit Block Diagram

5.5.2 Latches

Latches form one class of flip-flops. This class is characterized by the fact that the timing of the output changes 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. 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.

The condition that is undefined is when both inputs are equal to 0 at the same time.

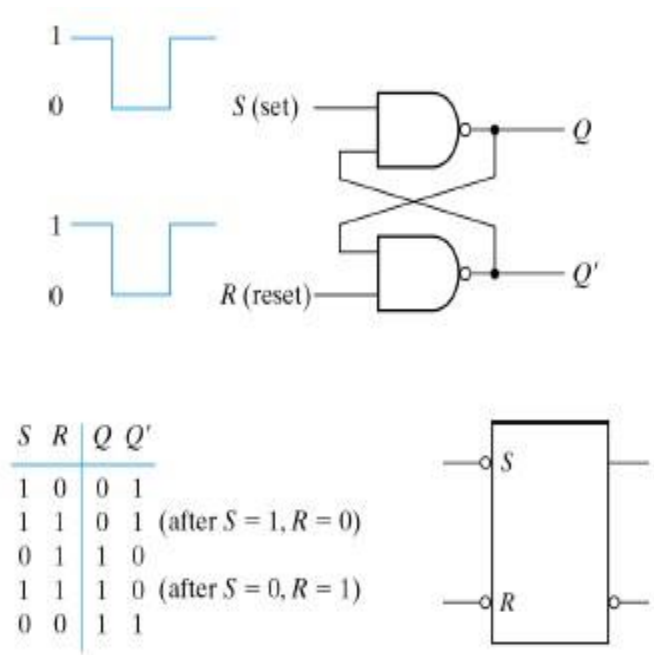


Figure 5.2: SR latch with NAND gate

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

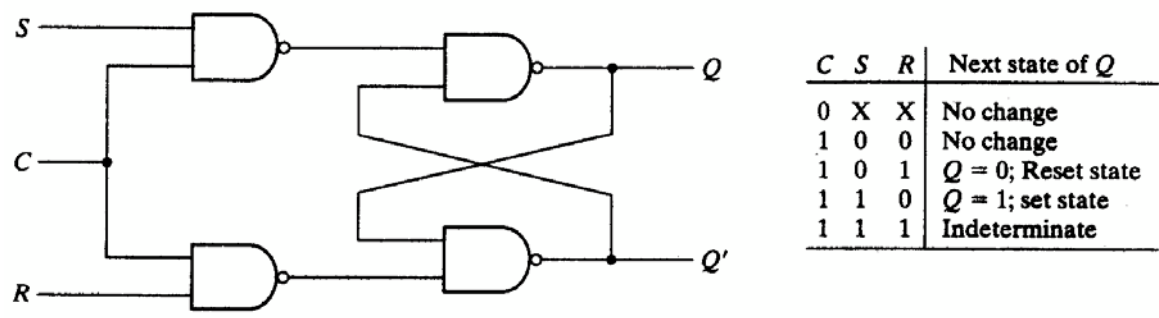


Figure 5.3: RS latch with control input

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

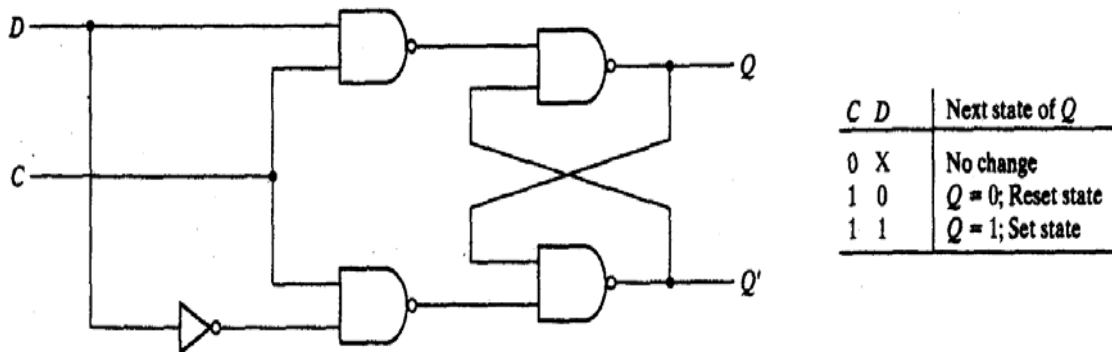


Figure 5.4: D-Latch

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.

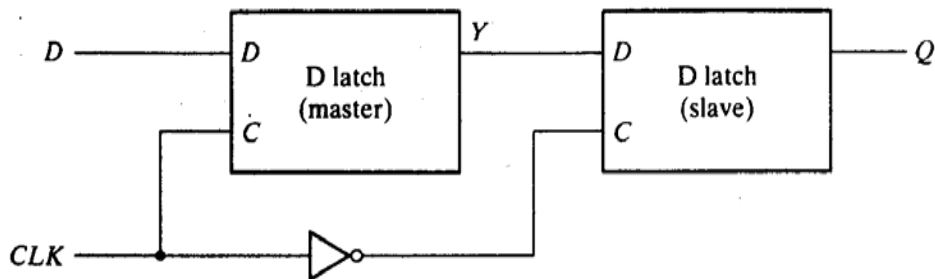


Figure 5.5: D flip flop implemented with two D latches

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.

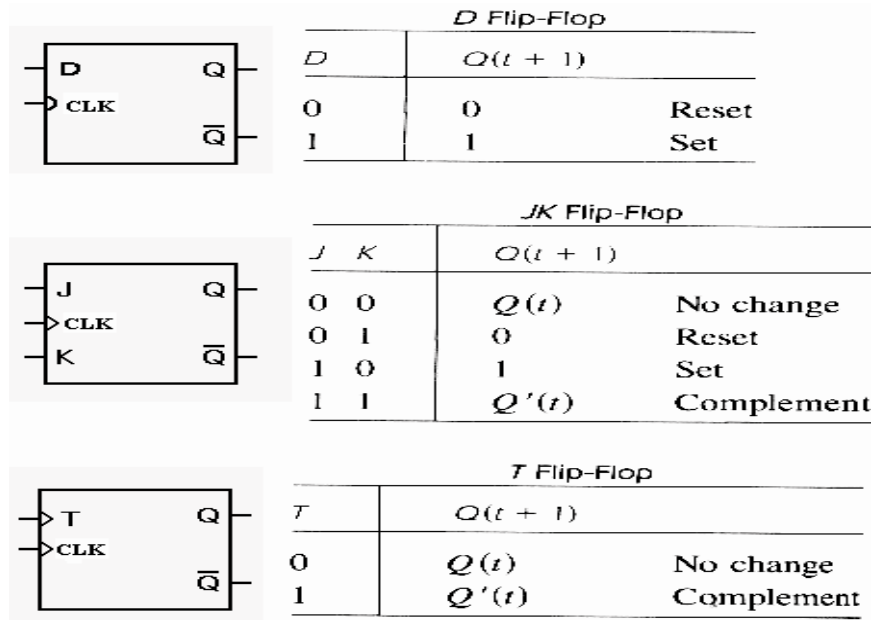


Figure 5.6: D, JK, and T flip flops

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

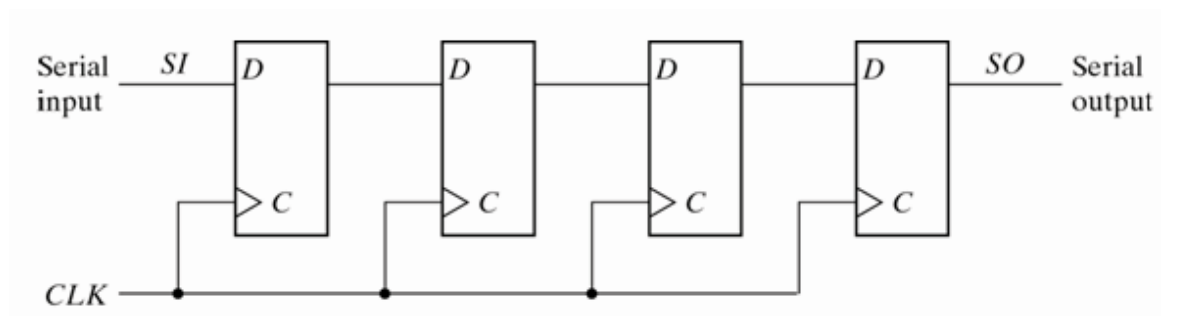


Figure 5.7: 4-bit Register

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.

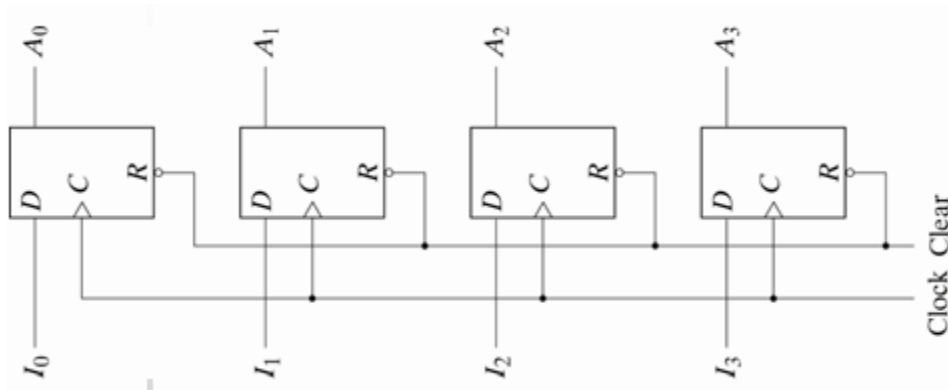


Figure 5.8: 4-bit shift- right register

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

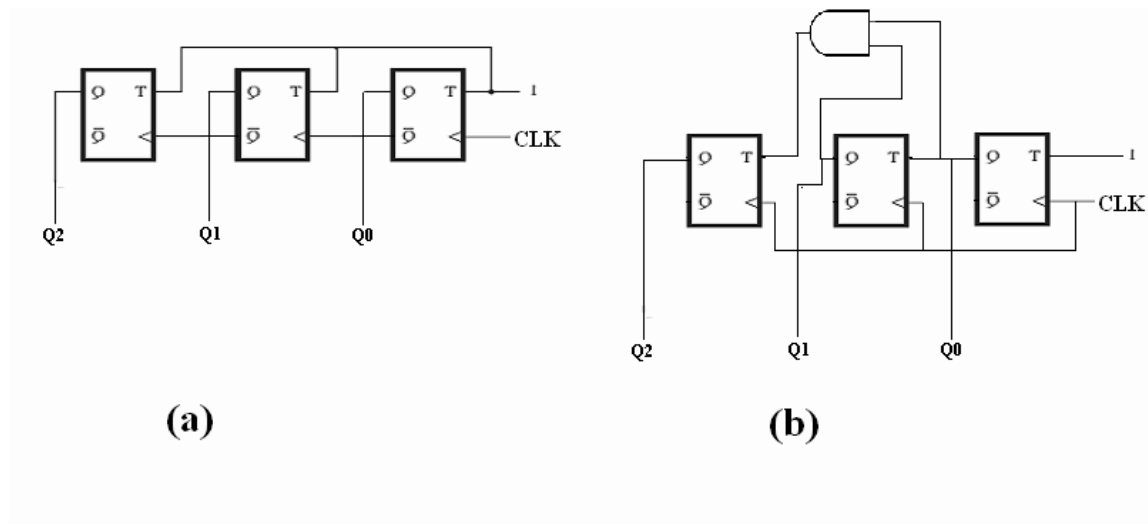


Figure 5.9: (a) 3-bit ripple counters, (b) 3-bit synchronous counter

5.6 PROCEDURE

5.6.1 Latches and Flip flops:

A) Constructing RS latch with Basic Logic Gates

Use IT-3008 module to construct the circuit shown in Figure 5.10. Connect inputs A3, A4 to Pulser Switches SWA **A**(TTL), SWB **B** (TTL). Connect outputs F6 and F7 to Logic Indicators L1, L2. Follow these sequences in Table 5.1

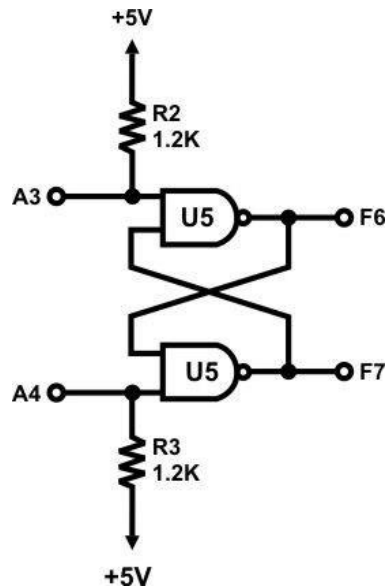


Figure 5.10: RS latch

A3	A4	F6	F7
0	0		
0	1		
1	0		
1	1		

Table 5.1

B) Constructing RS latch with control input

Use IT-3008 module to connect the circuit shown in Figure 5.11. Connect inputs A1, A5 to Pulser Switches SWA **A**, SWB **B** and follow the input sequence in Table 5.2.

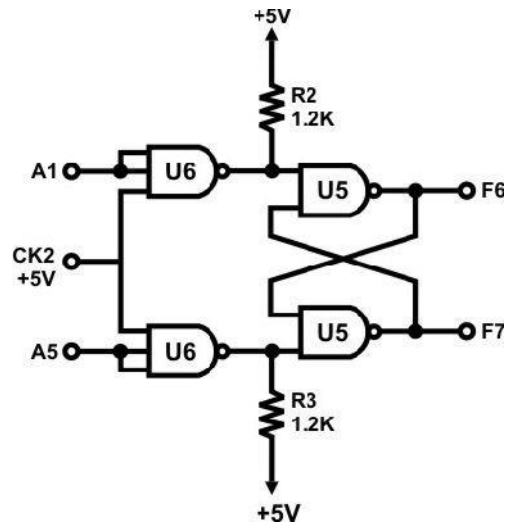


Figure 5.11: RS Latch with control input

A1	A5	F6	F7
0	0		
0	1		
1	0		
1	1		

Table 5.2

C) Constructing D latch with RS latch

Use IT-3008 module to construct the circuit shown in Figure 5.12. Connect A1 to SW1; CK2 to SWA A and F6 to L1. Follow the sequences in Table 5.3

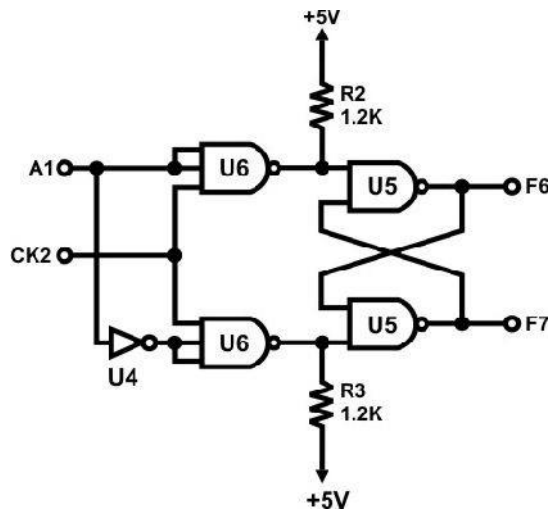


Figure 5.12: D Latch



CK2	A1	F6
0	0	
0	1	
	0	
	1	

Table 5.3

D) Constructing JK latch with RS latch

Use IT-3008 module to construct the circuit shown in Figure 5.13. Connect CK2 to SWB B output; A1 to SW0; A5 to SW1; F6 to L1. Follow the sequences in Table 5.4

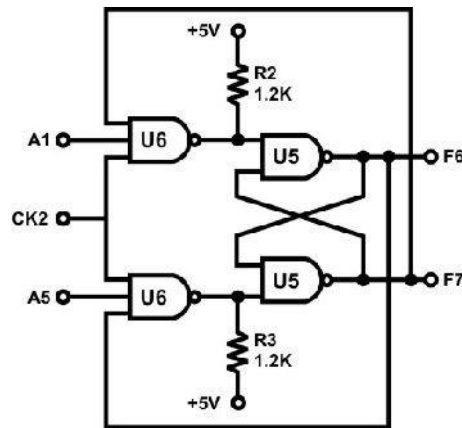


Figure 5.13: JK Latch





CK	A1	A5	F6
	0	0	
	0	1	
	1	0	
	1	1	

Table 5.4

E) Constructing JK Flip-flop with master- slave RS latches

Use IT-3008 module to construct the circuit shown in Figure 5.14. Connect CK2 to **Pulsar switch**. Connect CK1 to SWA A output; J to SW1; K to SW0; F1, F2, F6, F7 to L3, L2, L1 and L0 respectively. Follow the sequences in Table 5.5

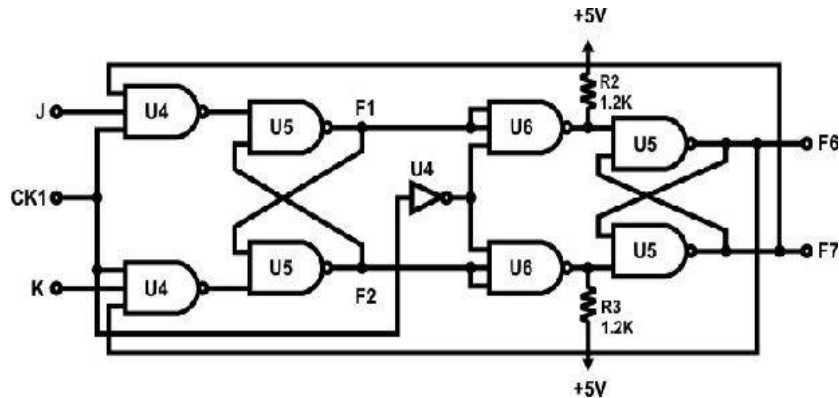


Figure 5.14: JK Flip-Flop

CK	K	J	F1	F2	F6	F7
⌊	0	0				
⌊	0	1				
⌊	1	0				
⌊	1	1				
⌊	1	1				

Table 5.5

5.6.2 Registers

A) Constructing Shift Register with D Flip-Flops

1. Block Shift Register 1 of module IT-3008 will be used to construct the circuit shown in Figure 5.15

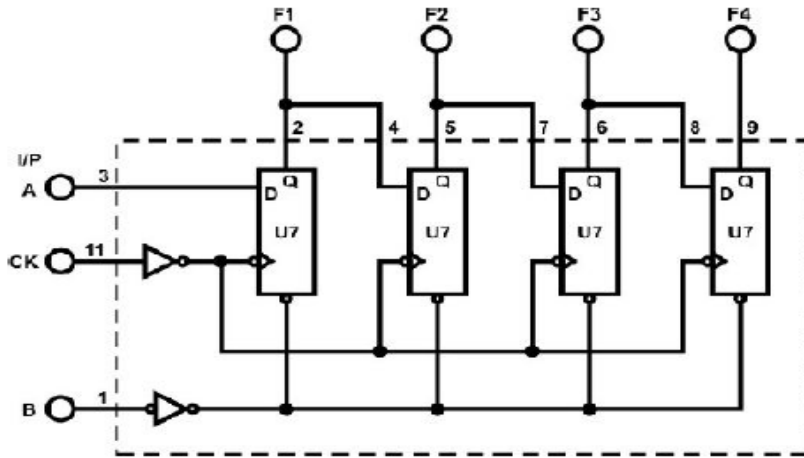


Figure 5.15:ShiftRightRegister

1. Connect B (clear) to SW0; A (I/P) to SW1; CK to SWA A output; F1, F2, F3, F4to L1, L2, L3, L4 respectively.
2. Set SW0 to “0” to clear B and then set SW0 to “1”.
3. Follow the input sequence for A(I/P) below, and observe output display at F1, F2, F3 and F4:
 - 1) at A= “1”, send in a CK signal from SWA
 - 2) at A= “0”, send in a CK signal from SWA
 - 3) at A= “0”, send in a CK signal from SWA
 - 4) at A= “1”, send in a CK signal from SWA

B) 4-Bit Shift Register withserial and parallel load

Use Shift Register 2 module in IT-3008 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.

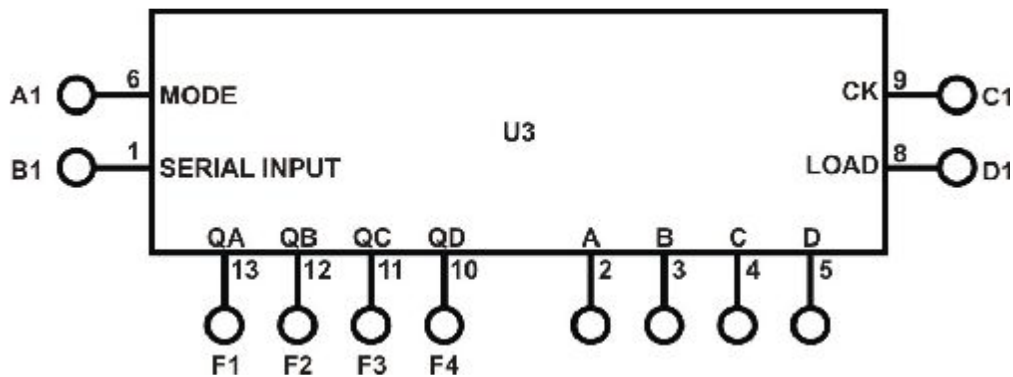


Figure 5.16: shift register with serial andparallel load

- Complete the following connections:
 Inputs A, B, C, D to SW0, SW1, SW2, SW3 Outputs F1, F2, F3, F4 to L0, L1, L2, L3
 B1 (I/P) to DIP2.0
 A1 (MODE) to DIP2.1



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

Table 5.6

- Connect CK (C1) to the clock generator TTL level output at 1Hz and change dataat B1 with DIP2.0. Follow the input sequences for A1 in Table 5.7. Observe and record the outputs.





Input		Output			
A1	C1	L3	L2	L1	L0
0					
0					
0					
1					

Table 5.7

- 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 5.8. Observe and record the outputs.






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				

Table 5.8

5.6.3 Counters

A) 2-bit Synchronous Counter

1. Use IT-3007 module to implement the 2-bit synchronous counter shown in Figure 5.17.
2. Connect CLK input to pulser switch.
3. Connect counter outputs Q1 and Q0 to indication lamps.
4. Apply clock pulses to CLK input. Observe and record the outputs in Table 5.6 (a)
5. Apply counter outputs Q1 and Q0 to **seven segment** display. Observe and record the outputs in Table 5.6 (b).

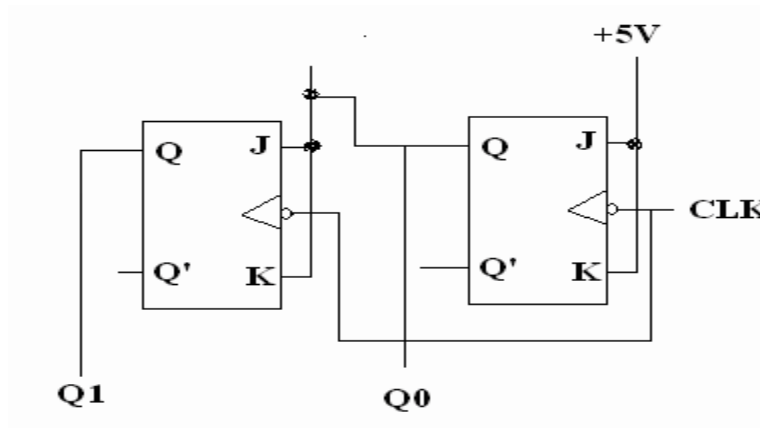


Figure 5.17: 2-bit Synchronous Counter

CLK	Q1	Q0	CLK	D1

(a) (b)

Table 5.6

B) 3-bit (divide-by-eight) Ripple Counter

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

1. Use the IT-3007 module to implement the 3-bit (divide by eight) Ripple counter shown in Figure 5.18.
2. Connect CLK input to 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 5.7 (a).
5. Apply counter outputs Q2, Q1 and Q0 to **seven segment** display. Observe and record the outputs in Table 5.7 (b).

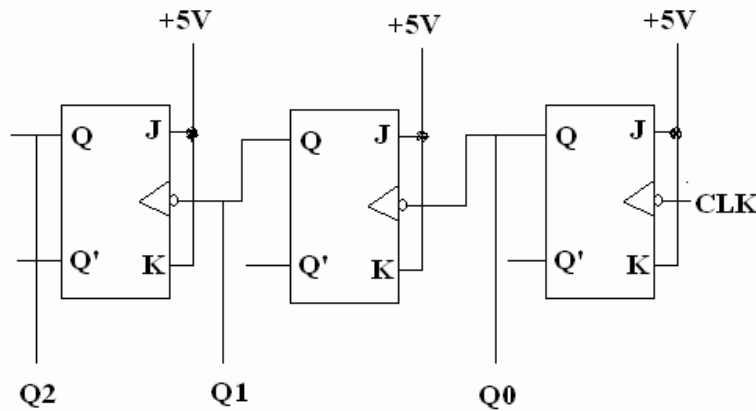


Figure 5.18: 3-bit Ripple Counter

CLK	Q2	Q1	Q0	CLK	D1
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	
⌊				⌊	

(a)
(b)

Table 5.7

Task2: Modify the circuit in Figure 5.16 to be 3-bit Synchronous Counter. Attach the design with this experiment report.

C) BCD Counter

Locate the BCD counter (IC 7490) on IT-3008 module, which is shown in Figure 5.19. Functional block diagram of U1 is shown in Figure 5.20.

1. Connect C3, C4 to SW0 and SW1; D1, D2 to SW2 and SW3; F1~F4 to L1~L4, A2 to SWAA output; B2 to SWBB output.
2. Connect F1 to B2, set C3, C4, D1 and D2 to ground and A2 to SWAA pulse. Measure and record the outputs F1, F2, F3, F4.

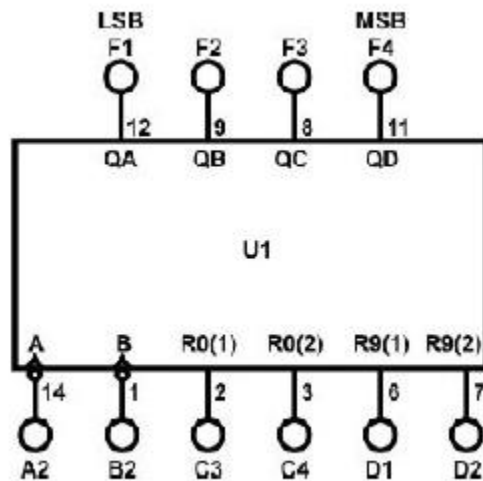


Figure 5.19: IC 7490 BCD Counter

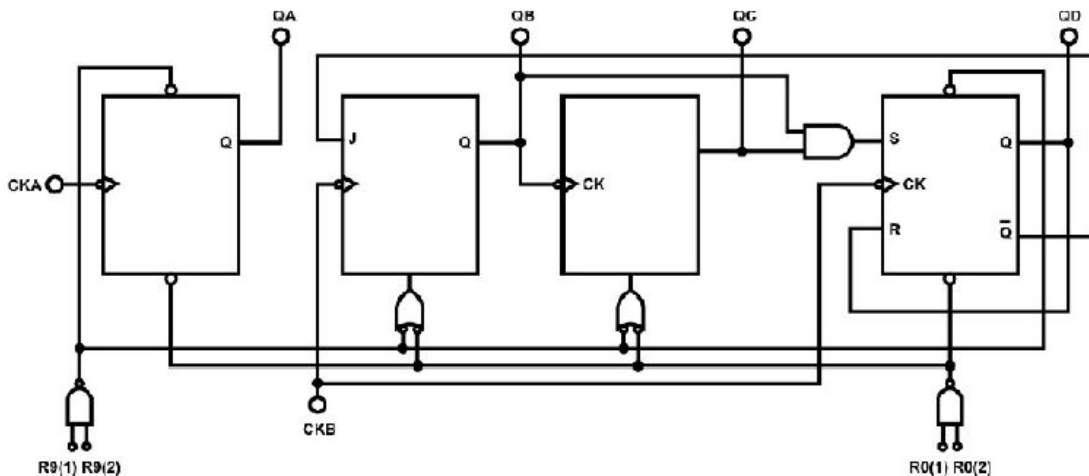


Figure 5.20: IC 7490 BCD Counter

D) Divide-by-8 counter using BCD chip counter:

1. Connect R0(2) (pin3) to +5V, and connect R0(1) (pin2) to QD (pin11) output. This will make 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 counter in Figure.19 to count from:

- **0-to-5**
- **0-to4**

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 211

Digital Electronics and Computer Organization Lab

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

6.1 Objectives:

1. To learn how to use some integrated circuits (ICs) such as counters (IC7490) and 7-segment display driver/decoder (IC7447).
2. To understand the function of the 7-segment display and how to find its pin assignment.
3. To build one, two or more decade counters with 7-segment displays.

6.2 Introduction

6.2.1 Seven-Segment Display

The seven-segment LED (Light Emitting Diode) display is a common device in consumer electronics, from calculator to clock to microwave ovens. In this lab, you will learn the basic principles of operation of the seven-segment display and the process of converting BCD values to the proper signals to drive this display.

The display has seven separate bar-shaped LED's arranged as shown in figure 6.1. In addition, many seven-segment displays have one (or two) circular LED used as a decimal point.

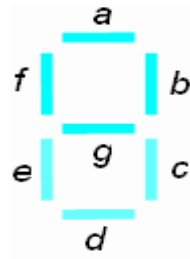


Figure 6.1: seven-segment display

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

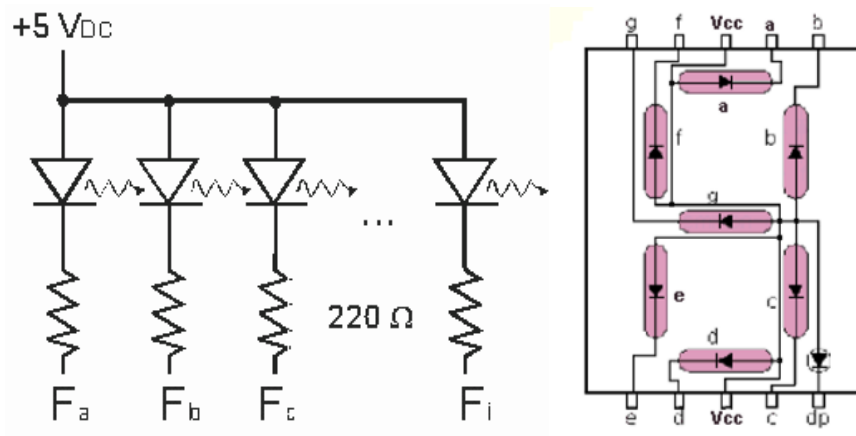


Figure 6.2: common anode display

If your seven-segment display is designed to have the common connection tied to the ground, 0V, it is called a **common cathode** configuration as shown in figure 6.3. To turn on these LED segments, the inputs must be **logic high**.

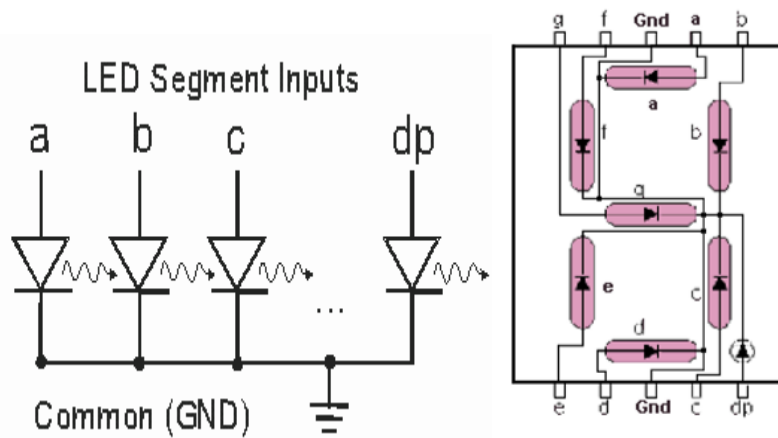


Figure 6.3: common cathode display

In both common-anode and common cathode configurations, current-limiting resistors are used to lower the amount of current that the driver sends into the LED's. This achieves two goals:

1. Prevent over-current which may burn the LED's.
2. Control the brightness of the LED's.

6.2.2 BCD-to-seven-segment decoder

A BCD-to-seven-segment decoder is a logic circuit that allows the display of a binary-coded decimal on a seven-segment display.

In this lab the IC type 7447 decoder will be used. The 7447 pin assignment is shown in Figure 6.4. Its pin description is shown in Table 6.1

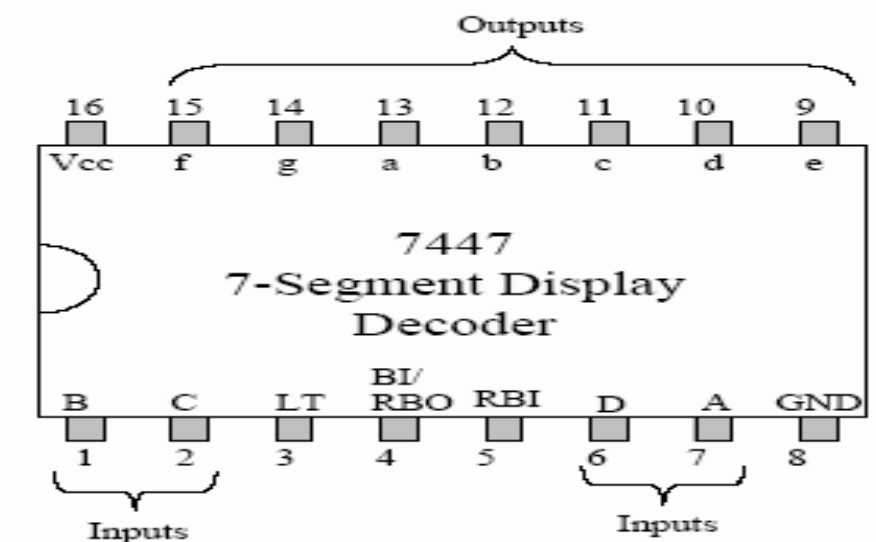


Figure 6.4: 7447 pin assignment

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

Table 6.1: 7447 pin description

LT is a lamp-test feature, active low. This line should be high for normal operation and, when pulled low, all 7-segments will be turned on so you can see all the 'lamps' are good.

RBI is the ripple blanking input, also active low. It allows you to hide a '0'. When this line is high, and there's a 0 at the inputs, a '0' will be displayed on the display. When the line is low with a 0 at the inputs, nothing will be displayed.

BI/RBO can be used as an input or output. **BI**, or blanking input, active low, is used to turn off all these segments. **RBO**, or ripple blanking output, active low, is used with **RBI**. When **RBI** is activated and the input of DCBA is 0000, **RBO** will be active to indicate that the segments are blanked. This allows you to drive the **RBI** input of another 7-segment display.

For **normal operation without blanking**, the three inputs: **LT**, **RBI**, and **BI/RBO** should be connected to +5V.

6.2.3 Counter

In this lab the IC type 7490 counter will be used. The 7490 pin assignment is shown in Figure 6.5 and its logic diagram is shown in Figure 6.6.

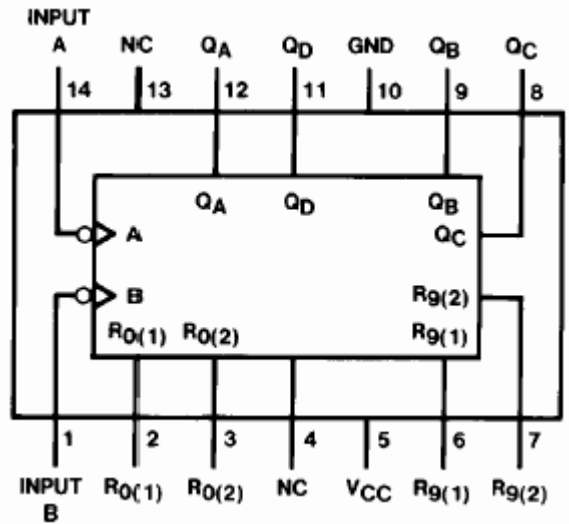


Figure 6.5: 7490 counter pin assignment

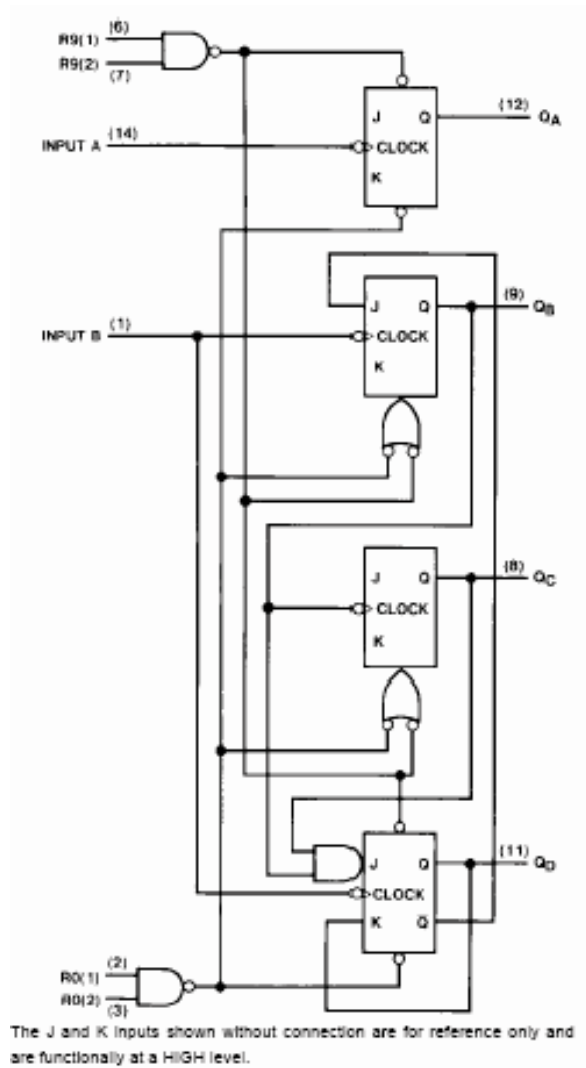


Figure 6.6: 7490 counter logic diagram

Reset Inputs				Outputs			
R0(1)	R0(2)	R9(1)	R9(2)	Q _D	Q _C	Q _B	Q _A
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			

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

Table 6.2: Reset/count function table

6.3 Pre Lab:

- 1- What is the appropriate display type (common anode/common cathode) that must be used with 7447 display decoder? Why?
- 2- **We would like to limit the current in the LED segments to 10mA.** Assuming that the turn-on voltage for the LED's is 1.7V, what is the proper value of the resistors to be connected between the 7447 decoder and the 7-segment display?
- 3- **Assume that the resistors provided in the lab are 220Ω. What would the current flowing into the LED's be?**
- 4- **Design a decade counter circuit using the 7490 counter, the 7447 decoder, and a 7-segment display. Show the pin numbers on the IC's in your design**

6.4 Procedure

6.4.1 Seven- Segment display

a. Testing lamps in the display

1. Place the display and the 7447 chips on the breadboard.
2. Implement the circuit shown in Figure 6.7.
3. Connect pin 4 and pin 5 of the 7447 decoder to the +5V
4. Connect pin 3 (LT) of the 7447 decoder to the ground. All 7 segments must be turned on. This to verify that all segments in the display are working correctly.

b. Blanking all segments

1. Connect pin 4 (BI) of the decoder to the ground. All 7 segments must be turned off.
2. Keep this circuit connected. You may use it as part of the next step.

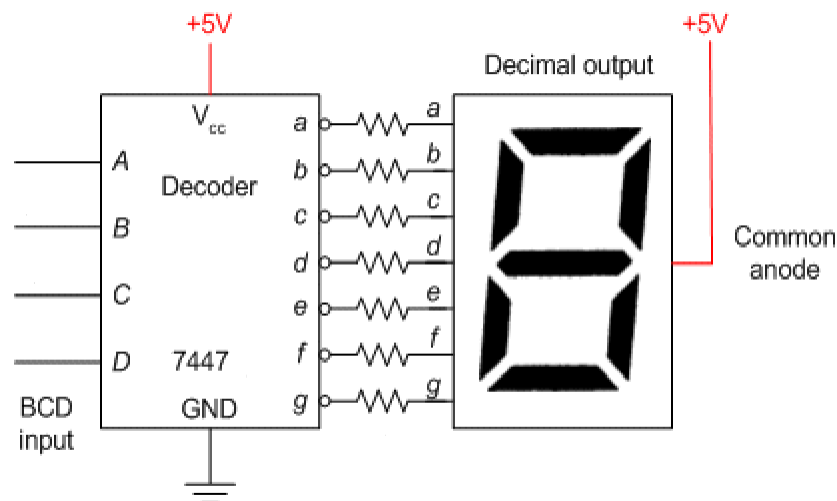


Figure 6.7: display-decoder connection

6.4.2. 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 the instructor before turning on the power.

7. Experiment No. 7 - Constructing Memory Circuits Using Flip-Flops

7.1 OBJECTIVES

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

7.2 Theory

1. CONSTRUCTING RANDOM ACCESS MEMORY (RAM) WITH D FLIP-FLOP

Two different types of basic structure for RAM 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.

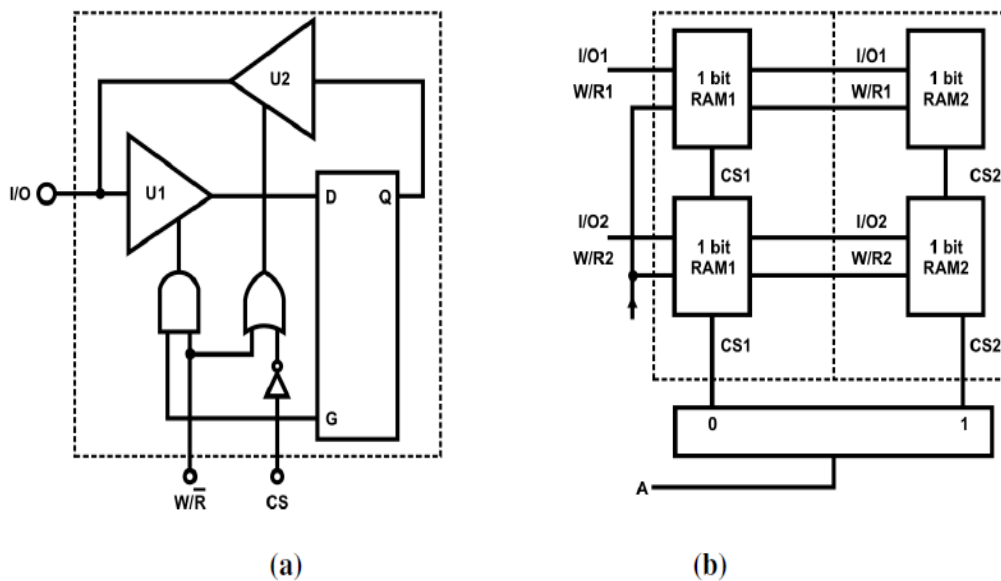


Figure 7.1

When $CS=0$, tri-state gates U1 and U2 do not operate, so data input is not possible, the flip-flop output Q is not send to the I/O terminal.

When $CS=1$, W/R controls the D flip-flop. When $W/R' = 1$, U1 opens but U2 does not, I/O will accept data input. If $W/R'=0$, the exact opposite will happen and I/O act as the data output.

Figure 7.1 (b) shows another connection that will increase the RAM capacity. When CS1=1, RAM1 I/O1 and I/O2 are selected. Address line A is used to select between RAM1 and RAM2. Since there is only one address line, we can only select from 2 RAMs. In Figure 7.1 (b), each CS can only selected a 2-bit RAM so the total capacity is 2×2 .

- 2) In Figure 7.2, a 2-address (2-bit) RAM circuit has independent input and output. When Address=0, input D1 is enabled and the content of D1 will be made available at the output. When Address=1, input D2 is enabled and the content of D2 will be made available at the output. The ENABLE terminal must be triggered in order for the output to correspond with the constantly changing inputs.

When Address=1, input D2 is enabled and the content of D2 will be made available at the output. The ENABLE terminal must be triggered in order for the output to correspond with the constantly changing inputs.

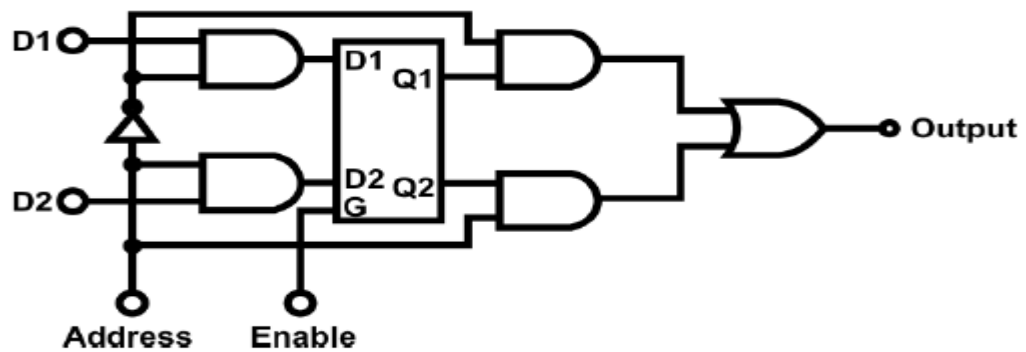


Figure 7.2.

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 16×4 , where the 4 is the number of data while 16 is the number of address lines.

Fig. 7.3 shows the 7489 IC, which is a 16×4 memory with 64 memory capacities. Also, its function table.

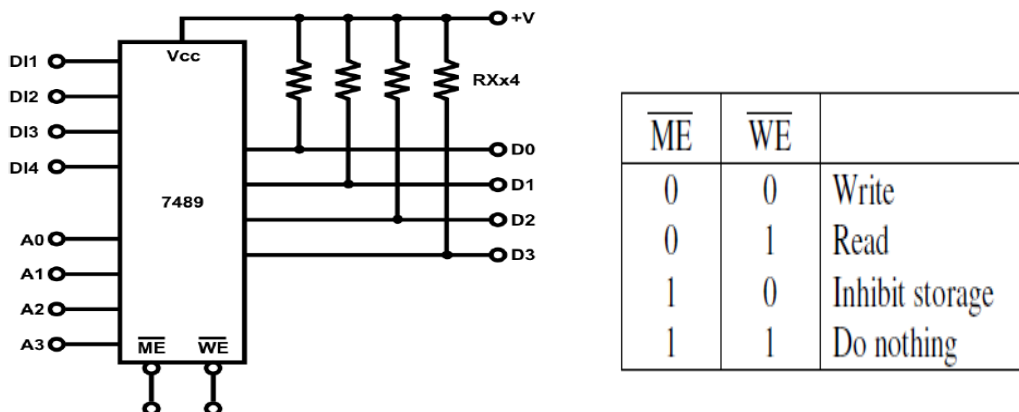


Figure 7.3.

When $ME' = 0$ and $WE' = 0$, 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 “RXx4 ” 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 Fig. 7.4. The operating sequence will be controlled by ME' and WE' .

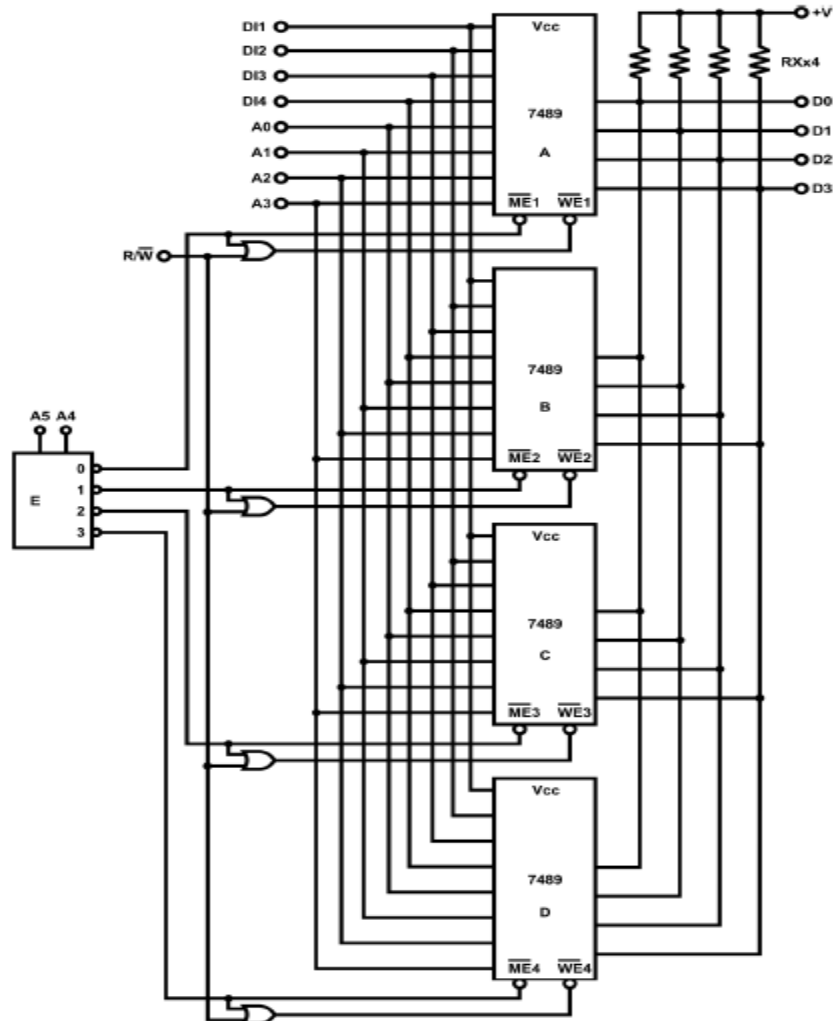


Figure 7.4

When A4A5=00, A is selected, ME' and WE' of B, C and D all equal to "1". Similarly, when A4A5=01, B is selected, ME' and 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 ME' and WE' are both "1", each R/W' control of 7489 are connected to an OR gate to ensure that when ME' = "1", WE' will be equal to "1" too.

When ME' = "0", WE' is controlled by external R/W' control so that the "READ" operation is performed if R/W'="1". The "WRITE" operation is performed when R/W'="0".

The 7488 is a 256-bit open-collector ROM which has similar structure as the 7489. Their methods of expansion are similar as well.

7.3 Procedure

- a. RAM block with D Flip-Flop of module IT-3011 which is shown in Figure 7.5 will be used in this part.

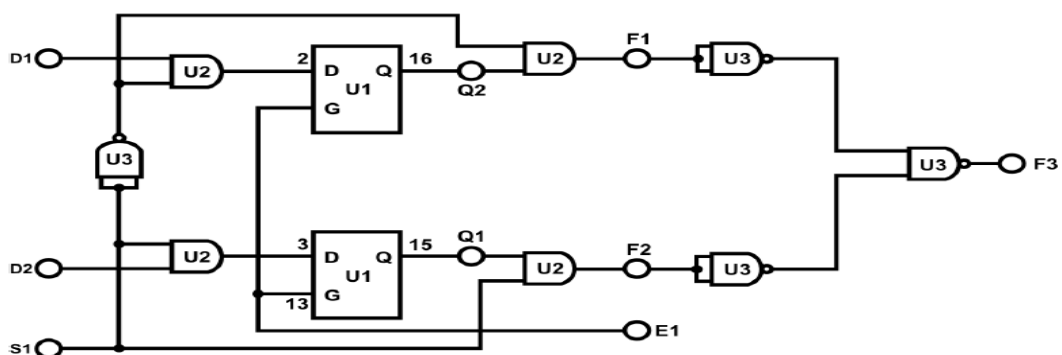


Figure 7.5

- 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.

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			

Table 7.1

Task 1: Discuss and explain the results to your TA.

- c. Then, set module IT-3011 and locate block RAM Circuit. Insert connection clip according to Figure 7.5, connect +5V, +15V of module IT-3011 to the +5V, 15V output of fixed power supply respectively.

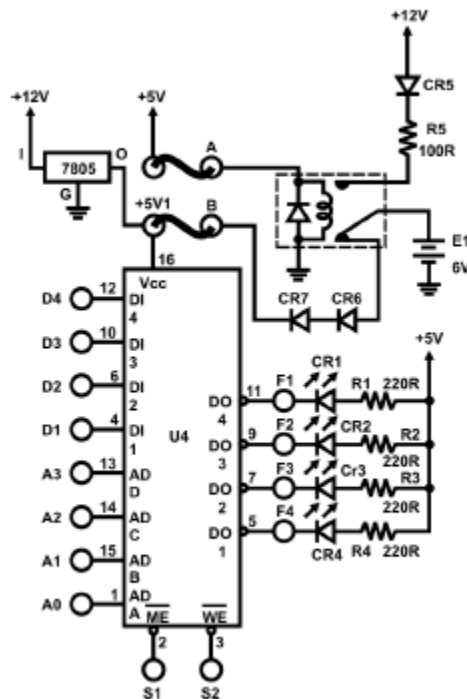


Figure 7.5

- 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, 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.
- f. Now set SW0 (WE') to “1” for the “READ” task and connect S1 (ME') to Pulser Switch SWA A. Observe states of CR1~CR4 and record under the “READ” column in Table 7.2.

- 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 address and pressSWA then attempt to read the data. Are they still stored in the RAM?
- h. Disconnect “B” clip, VCC disappears. Repeat Step 5 to see if the data are still stored in the RAM.

Address				Write					Read						
A3	A2	A1	A0	\overline{ME}	\overline{WE}	D4	D3	D2	D1	\overline{ME}	\overline{WE}	F4	F3	F2	F1
0	0	0	0	\downarrow	0					\uparrow	1				
0	0	0	1	\downarrow	0					\uparrow	1				
0	0	1	0	\downarrow	0					\uparrow	1				
0	0	1	1	\downarrow	0					\uparrow	1				
0	1	0	0	\downarrow	0					\uparrow	1				
0	1	0	1	\downarrow	0					\uparrow	1				
0	1	1	0	\downarrow	0					\uparrow	1				
0	1	1	1	\downarrow	0					\uparrow	1				
1	0	0	0	\downarrow	0					\uparrow	1				
1	0	0	1	\downarrow	0					\uparrow	1				
1	0	1	0	\downarrow	0					\uparrow	1				
1	0	1	1	\downarrow	0					\uparrow	1				
1	1	0	0	\downarrow	0					\uparrow	1				
1	1	0	1	\downarrow	0					\uparrow	1				
1	1	1	0	\downarrow	0					\uparrow	1				
1	1	1	1	\downarrow	0					\uparrow	1				

Table 7.2



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

8. Experiment No. 8 - Introduction to QUARTUSII Software

8.1 Objective:

Quartus II software is dedicated to program PLDs or FPGAs, in this experiment; we will study the schematic capture and HDL on Quartus II and then download simple examples on the FPGA using theKit DE1.

8.2 Pre-lab

For students who are not familiar with Quartus II, see **Appendix A**. Also, you have to watch the videos under the following link to be familiar with Verilog and Quartus:

<https://www.youtube.com/playlist?list=PLnyw1IVZpaTukmt80aNs7gT74U3vboDYr>

Then, you have to do the following:

- Using Quartus II, create a Verilog file for the 4-bit full-adder and simulate it.
- Using Quartus II, create a Verilog file for a 4-bit comparator and simulate it.
- Using Quartus II, create a Verilog file for a 2 by 1 mux and simulate it.

8.3 Procedure:

79

For each Verilog file, press the right click on its name and choose **Set as Top-Level Entity**, then from **File > Create/ Update > Create symbol Files for Current Files**. This step is necessary to create a high level blocks for your designs.

Now, if you try to create new Block Diagram/ Schematic File, Project libraries will be appeared that contain your designs as shown in Figure 8.1.

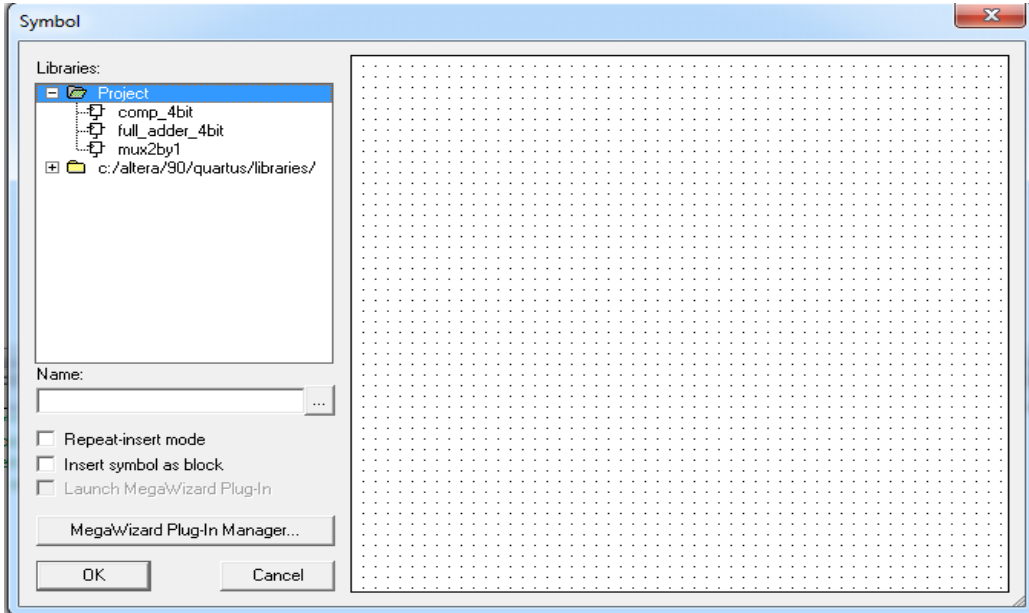


Figure 8.1.

Complete the connection to achieve the design which appears in Figure 7.2, you have to be careful and use suitable wires, and name the input and output to reflect its size, e.g. bus of 4 bits A[3..0], while a one bit signal is named as A.

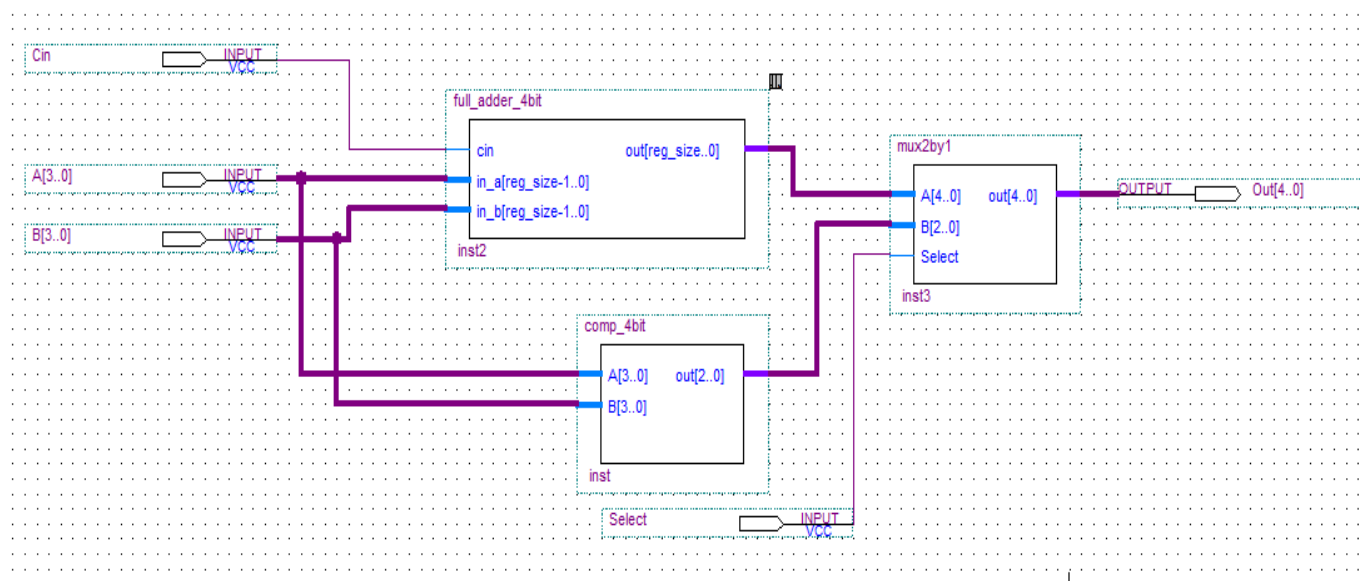


Figure 8.2

Routing the Project on the FPGA:

After verifying that the design works as expected we can install the code on the hardware (FPGA).

- To check that the correct device is selected, this can be done using Assignments > Device
- We need 4 switches for A, 4 for B, a switch for the select, and one for Cin. 5 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.

Signal Name	FPGA Pin No.	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]

Table 8.1: Pin Assignment for Switches.

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]

Table 8.2: Pin Assignment for LEDs.

- 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:

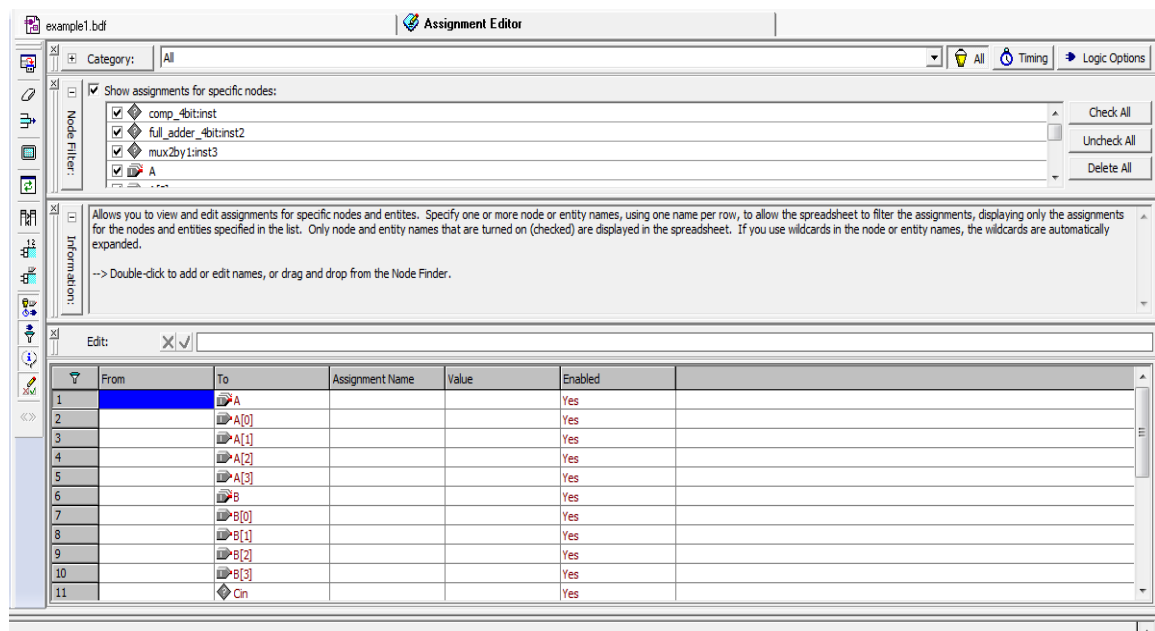


Figure 8.3.

- Select Pin in the category and select a switch for input.

Category: Pin

Show assignments for specific nodes:

- comp_4bitinst
- full_adder_4bitinst2
- mux2by1inst3
- A
- A[3]
- A[2]
- A[1]

Edit:

To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	Enabled
A[0]	PIN_L22	5	3.3-V LVTTTL	Dedicated Clock	CLK4, LVDSCLK2p, In...		Yes
A[1]	PIN_L21	5	3.3-V LVTTTL	Dedicated Clock	CLK5, LVDSCLK2n, In...		Yes
A[2]	PIN_M22	6	3.3-V LVTTTL	Dedicated Clock	CLK6, LVDSCLK3p, In...		Yes
A[3]	PIN_V12	7	3.3-V LVTTTL	Dedicated Clock	CLK12, LVDSCLK6n, I...		Yes
B[0]	PIN_W12	7	3.3-V LVTTTL	Dedicated Clock	CLK13, LVDSCLK6p, I...		Yes
B[1]	PIN_U12	8	3.3-V LVTTTL	Dedicated Clock	CLK14, LVDSCLK7n, I...		Yes
B[2]	PIN_U11	8	3.3-V LVTTTL	Dedicated Clock	CLK15, LVDSCLK7p, I...		Yes
B[3]	PIN_M2	1	3.3-V LVTTTL	Dedicated Clock	CLK3, LVDSCLK0n, In...		Yes
Cin	PIN_M1	1	3.3-V LVTTTL	Dedicated Clock	CLK2, LVDSCLK1p, In...		Yes
Select	PIN_L2	2	3.3-V LVTTTL	Dedicated Clock	CLK1, LVDSCLK0n, In...		Yes
Out[0]	PIN_R20	6	3.3-V LVTTTL	Row I/O	VREFB6N0		Yes
Out[1]	PIN_R19	6	3.3-V LVTTTL	Row I/O	LVDS84p		Yes
Out[2]	PIN_U19	6	3.3-V LVTTTL	Row I/O	LVDS89n		Yes
Out[3]	PIN_Y19	6	3.3-V LVTTTL	Row I/O	LVDS90n		Yes
Out[4]	PIN_T18	6	3.3-V LVTTTL	Row I/O	LVDS91n		Yes

Compilation at Mon Nov 20 21:01:49 2017 Jerusalem Standard Time

Figure 8.4

The Figure below shows the circuit with PIN assignments.

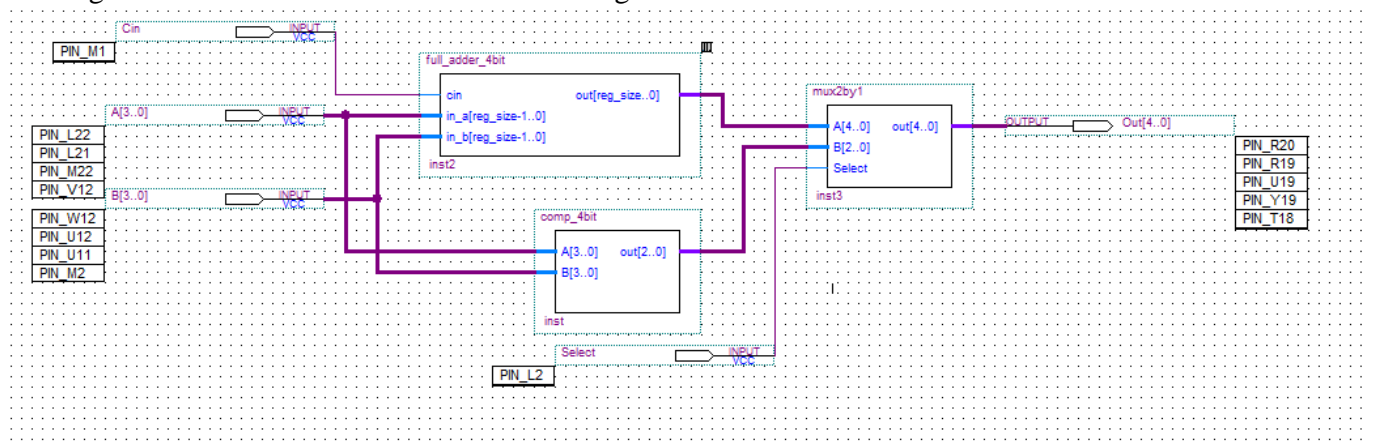
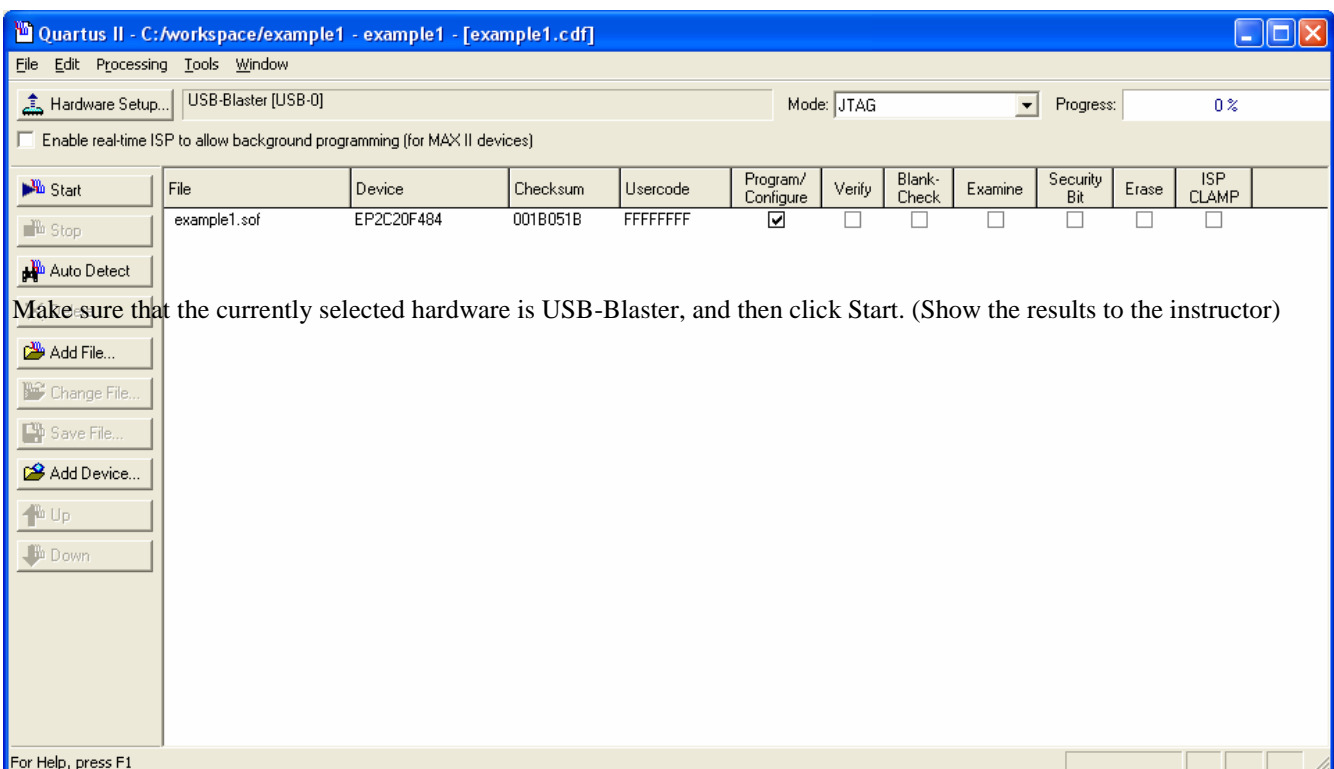


Figure 8.5.

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



Make sure that the currently selected hardware is USB-Blaster, and then click Start. (Show the results to the instructor)

Appendix

1. Appendix A-New Project

- Run the QUARTUSII software: double click on the QUARTUS II item in the desktop.
- Start new project :**File > New Project Wizard**, press next for the first window.
- Follow the windows as shown below:

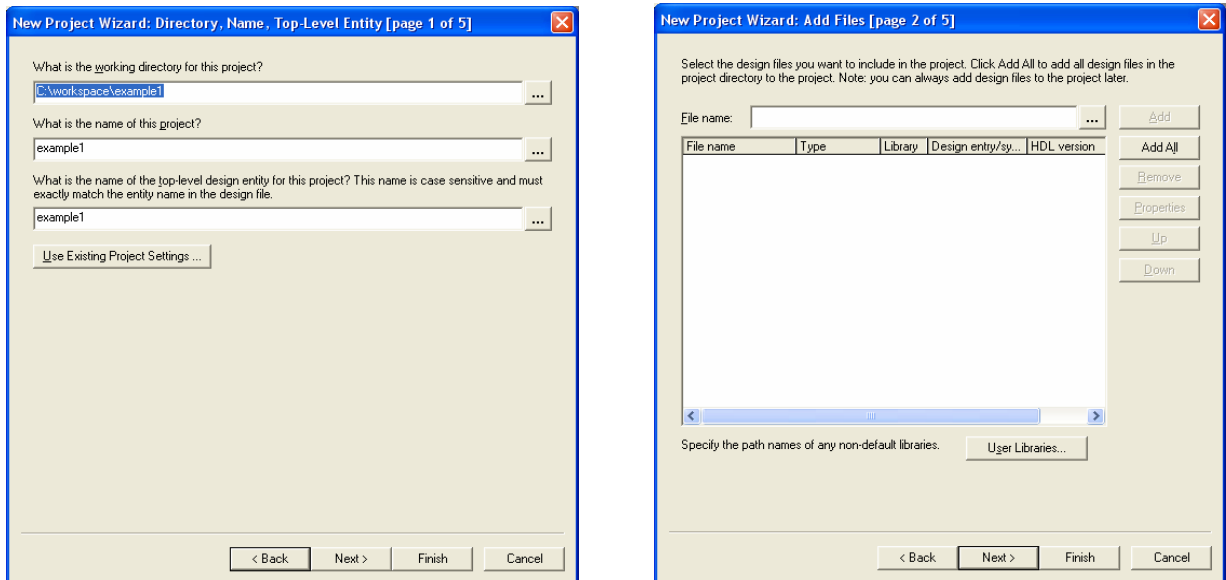


Figure 1: a) Project Folder, Project Name, Top-Level-Entity b) Add existing files.

- Change the working directory to indicate your folder that will contain your project.
- Then name this project and the top level design entity, you must notice that the name of the project and the high level design entity must have the same name as the module in your program (in the case of HDL programming), then press next.
- If you have already files to add, you can add using window in Figure 1-b, otherwise (if you want to build the design now) press next.
- The next window (Figure 2) give us a chance to use a specific device (which we will use later), choose the Family cyclone II and the device EP2C20F484C7.
- Since our project is simple, and no need to add any other Tools, we just press next.

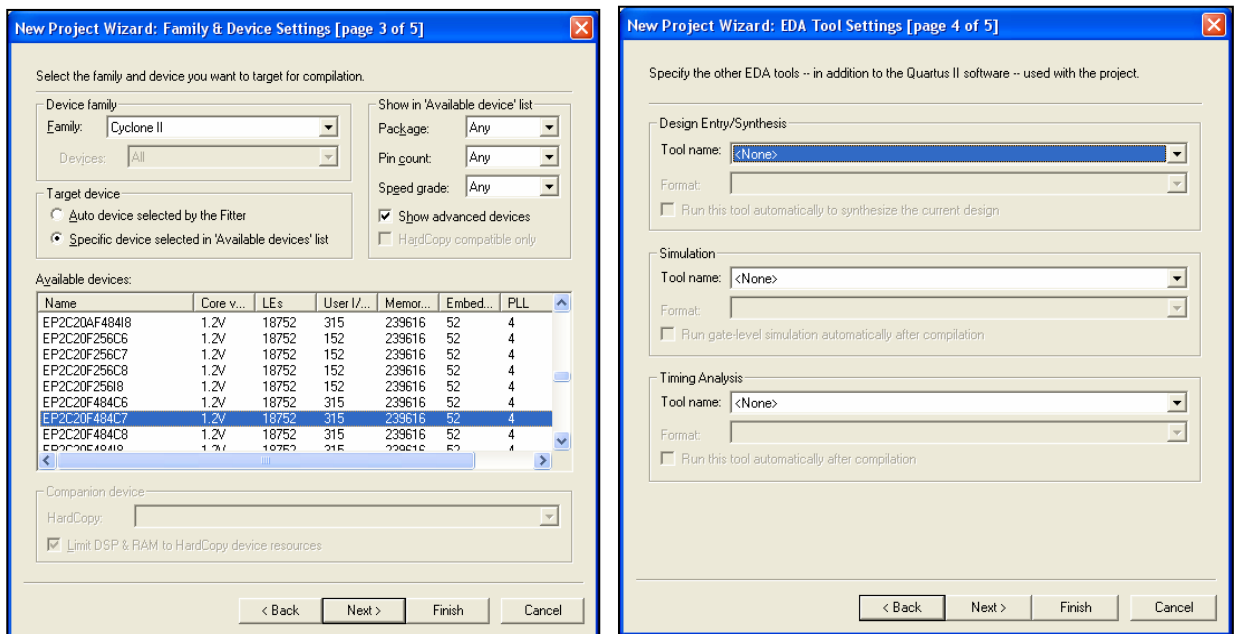


Figure 2

- After that the following window appears, finally press finish.

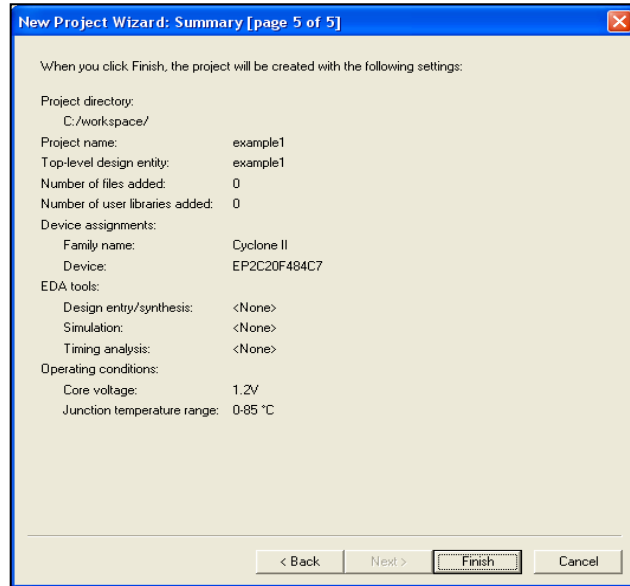


Figure 3.

2. Appendix B- New File

- To start new File, press **File > New**, the window in Figure 4 will appear.

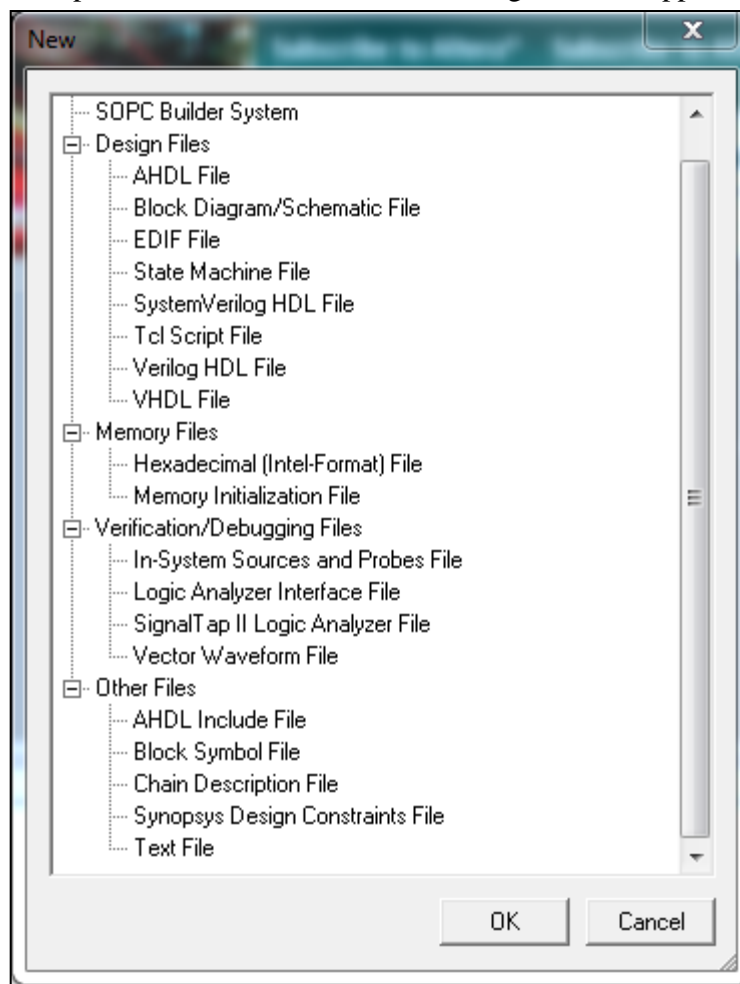


Figure 4.

Three common files will be used:

1. Block Diagram/ Schematic File.

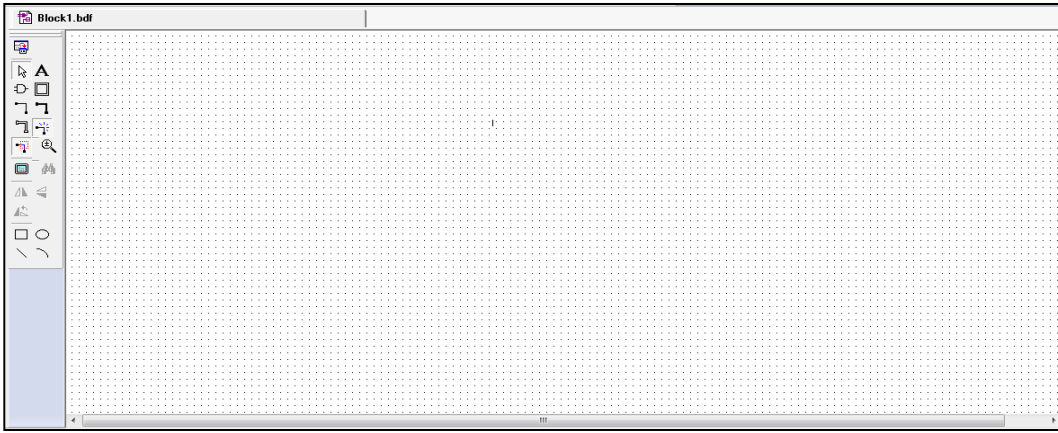


Figure 5.

- To enter components of our design double click anywhere on the schematic window, or select the symbol tool (The little and gate) as shown in Figure 5.
- 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

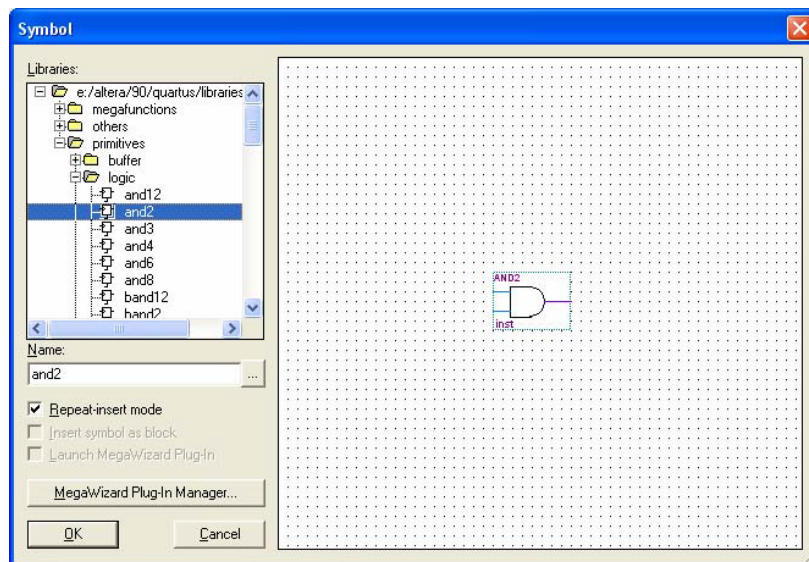


Figure 6.

- 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:

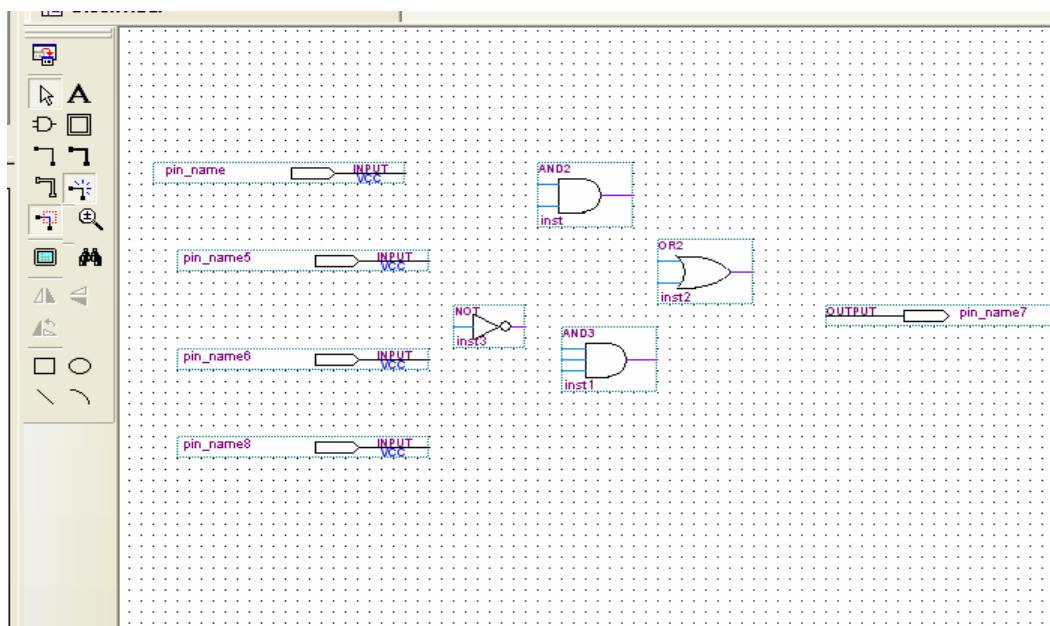


Figure 7.

- 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 a wiring tool that can be used to connect your circuit. When done, disable the wiring tool by clicking the arrow on the tool bar.
- 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.
- 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.

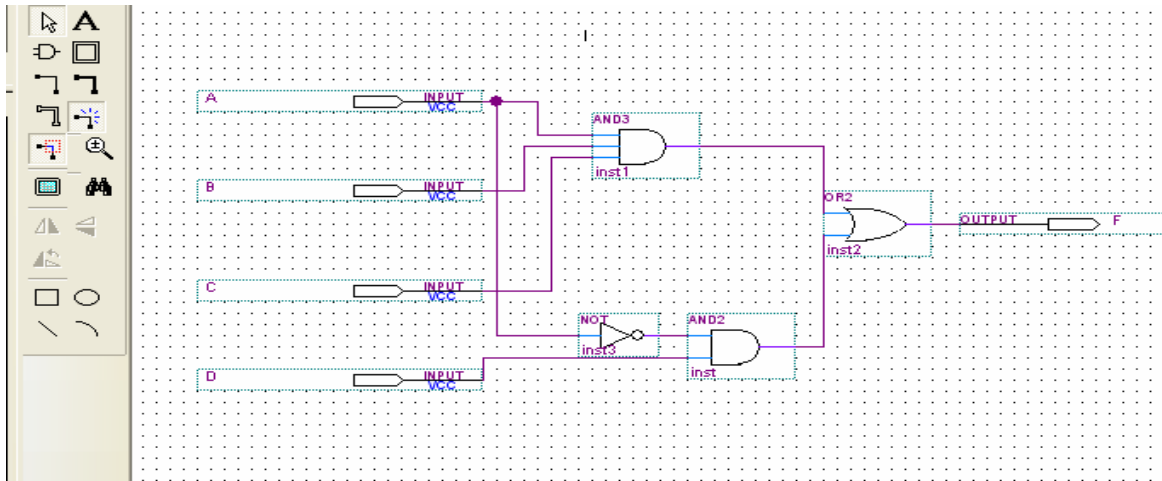


Figure 8.

2. Verilog HDL File.

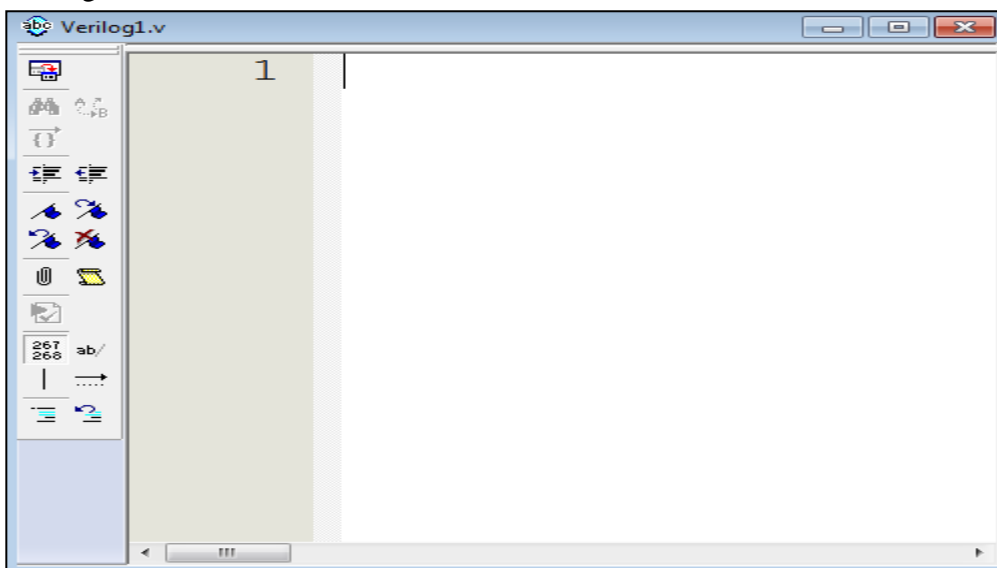



Figure 9.

This file is used to enter your Verilog code, you have to save the file as the name of the module.

Compilation

Using the right click over the name of the file, choose **Set as Top-Level Entity**.

After that we compile our design by either clicking on processing > start compilation or click on . If there are errors fix them before proceeding.

3. Vector waveform File.

This File is used to check the functionality of the design works as expected or not. Figure 10 shows a Vector waveform File.

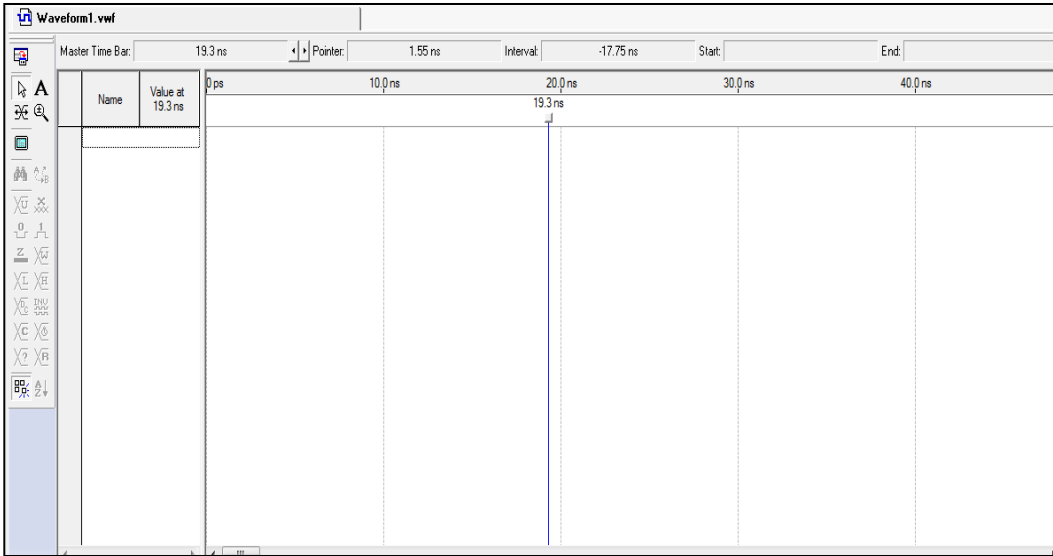


Figure 10.

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

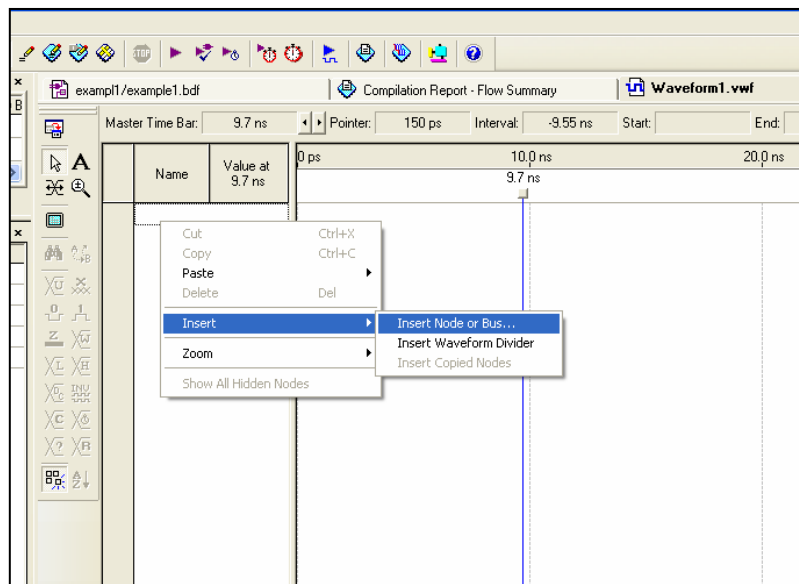


Figure 11.

Click on **Node Finder** and then on List (using the Filter pins: all)

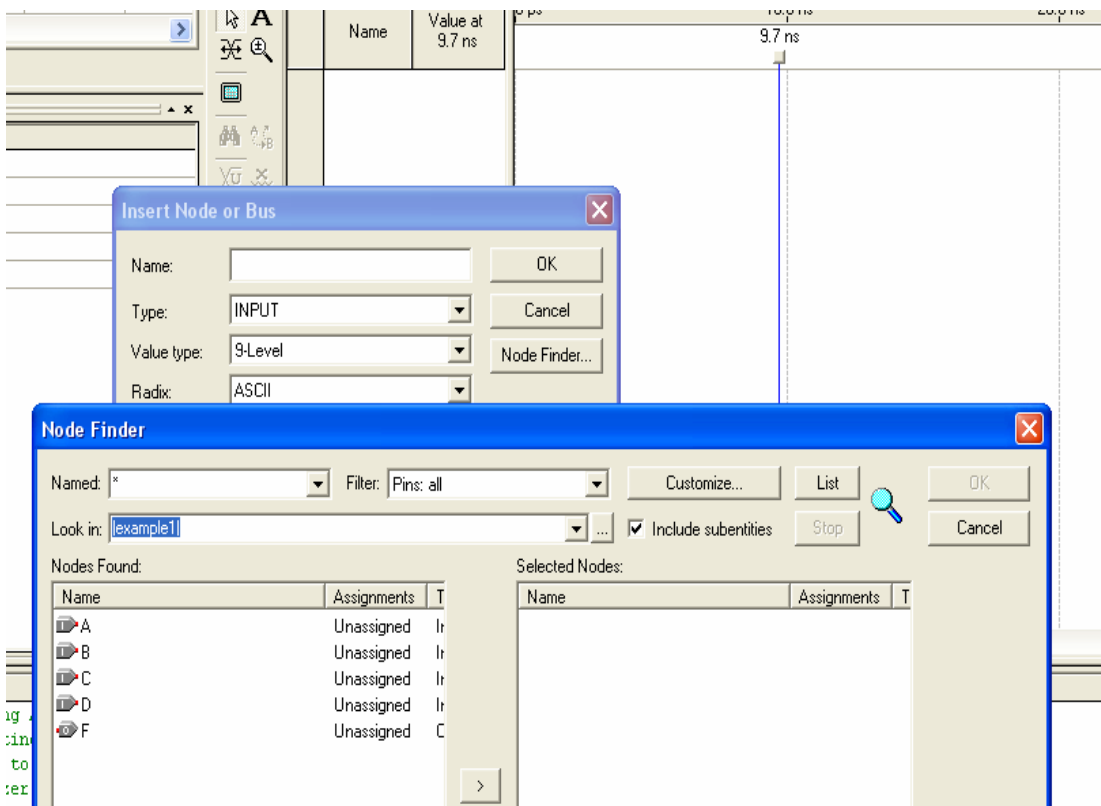


Figure 12.

- Select all inputs and the output by clicking on (>>) (you can select one by one).
- 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:

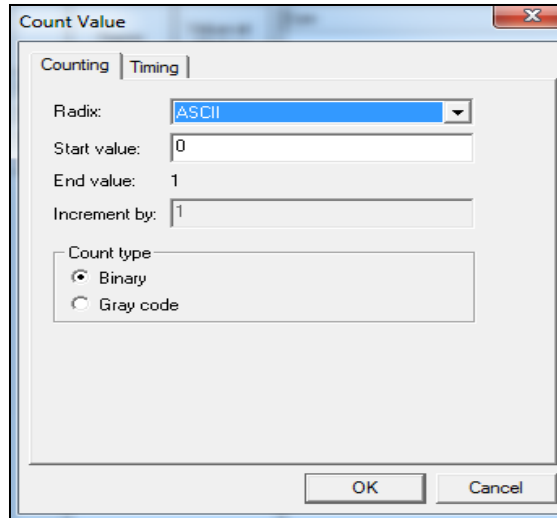


Figure 13.

- 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 14 will appear:

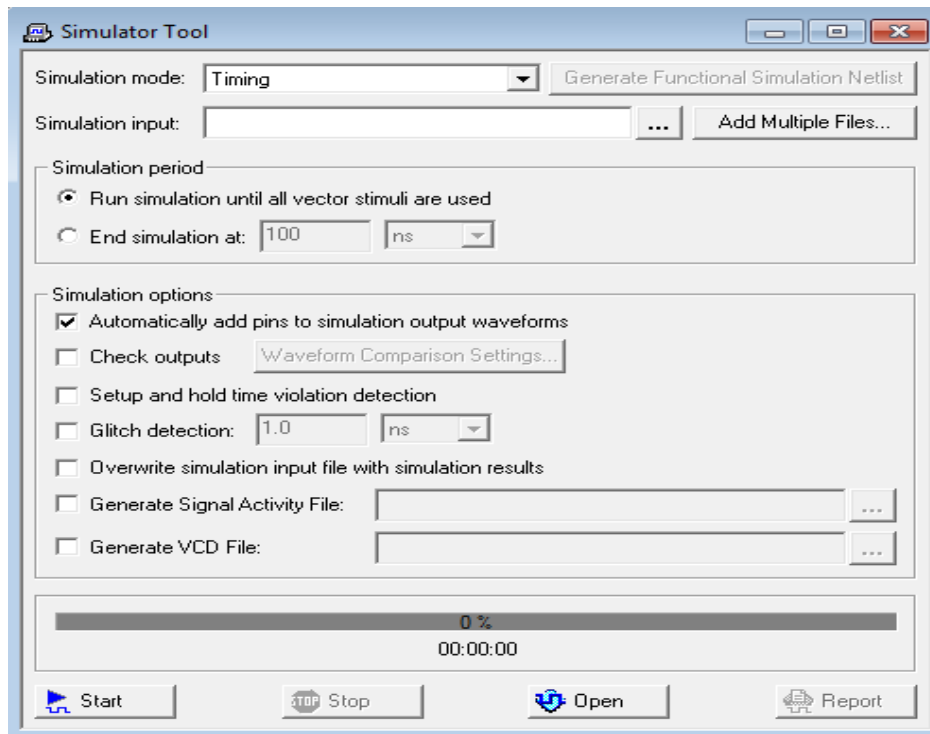


Figure 14.

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.



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

9. Experiment No. 9 - A Simple Security System Using FPGA

9.1 Objectives:

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

9.2 Apparatus:

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

9.3Pre Lab: (Bring a soft copy of your pre lab with you to the lab)

Prepare each part of the procedure section where it says **(Pre Lab)**. **NOTE:** It's **important that you 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 our system to DE1 Board (FPGA board).

Our security system is simply a 2 digit digital lock, User enter a number of 2 digits (digit range: 0 to 3, so every digit will has a lower limit of 0 and an upper limit of 3) using a keypad (using the

switch keys build 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. Else a red led is always on; blocking us from passing. The following computer based design will explain the architecture of our system:

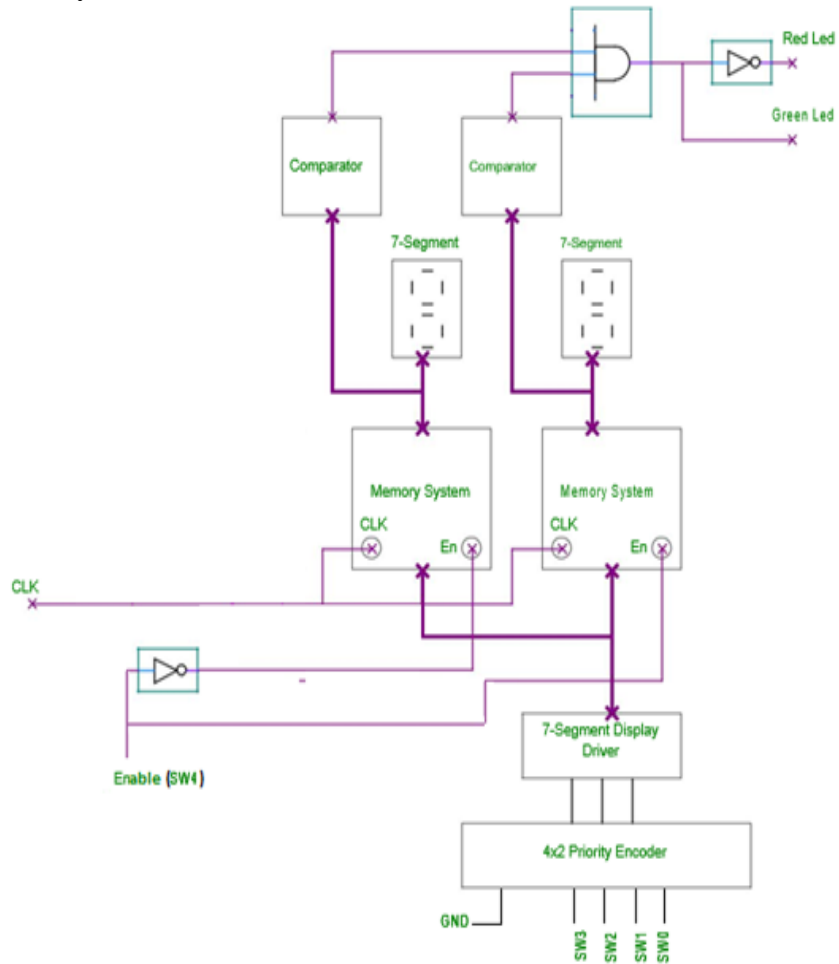


Figure 9.1: The architecture of the Security System.

As we can see in the above figure (FIG 9.1) the system consist of the following components:

1) 4x2 Priority Encoder:

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'001.

2) Enable Port:

The purpose of this port is to let the user select which memory system is active thus which 7-segment display to use, for example if SW4 is high then the En pin of the 1st memory system is enabled and ready to read user input on the 4x2 priority encoder.

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

3) 7-segment display driver:

This driver is used to convert the output of the priority encoder to the proper input for the 7-segment displays, the output of the driver is first stored in a memory unit before transferring to a 7- segment (depends on which memory system is enabled using the 2x4 decoder).

4) Memory System:

The purpose of such system is to ensure that the value selected by user to display on a certain 7-segment is kept there when the user switch to select another 7-segment. Each memory system is consisting of seven D- flip flops and 2x1 MUXs as seen in the following figure (FIG 9.2):

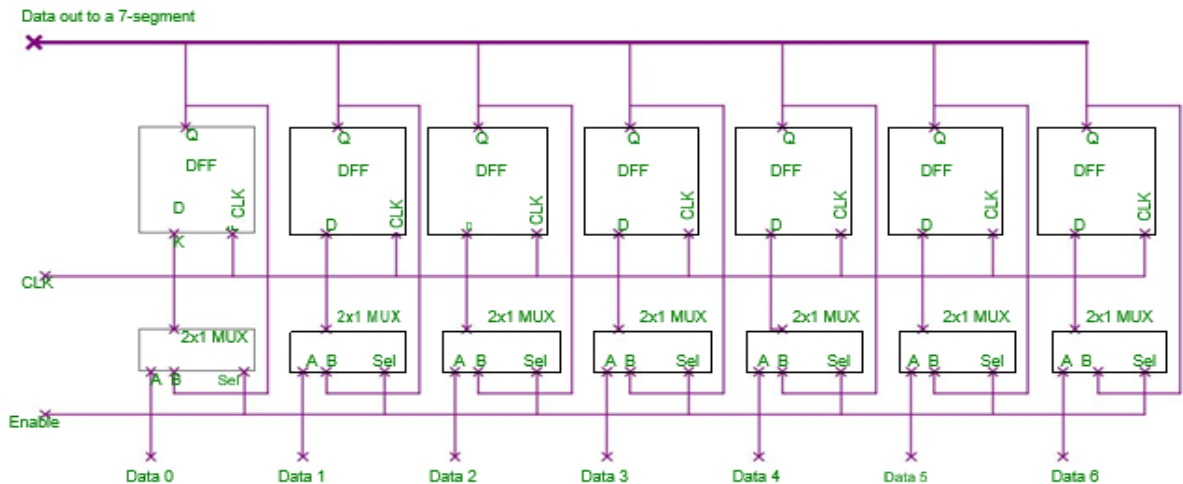


FIG 2: Memory System.

Figure 9.2

When the Enable pin =0, the output of each DFF becomes its input at every clock cycle, when the Enable pin becomes 1 the data coming from the 7-segment driver is then stored in the each DFF. The output of each DFF is sent as a data bus to a 7-segment display. Note: For each 7-segment display we need a memory system block.

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 are equal then the output of the comparator is 1 else the output will be 0; for example if one of the comparators has a reference value = 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 7-segment display). The purpose of the comparator is to lock/unlock our security system.

6) 2-input AND gate:

This AND gate will make sure that the two 7-segment displays have the correct combination; if each comparator output = 1, then the AND gate output will be 1, thus a green light is on, else a red light will be always on.

9.5 Procedure:

After we understood the architecture of our security system it's time to start programming and designing it (get our feet wet 😊).

NOTE: *Create a Symbol for each component you build.*

1. 4x2 priority encoder.

Write down the following code, compile and simulate it (**Pre Lab**).

```
3  module MyEncoder(i0,i1,i2,i3,en_out);
4  input i0,i1,i2,i3;
5  output [1:0] en_out;
6  reg [1:0] en_out;
7
8  always @ (i0 or i1 or i2 or i3 )
9
10 if (i3==1'b1)
11   en_out <= 2'b11;
12 else if (i2==1'b1)
13   en_out <= 2'b10;
14 else if (i1==1'b1)
15   en_out <= 2'b01;
16 else
17   en_out <= 2'b00;
18
19 endmodule
```

2. 7-segement driver:

Write down the following code (Complete to include the cases of 4,5, and 6), compile and simulate it (**Pre Lab**).

```
2  module MyDriver(in_v,out_v);
3  input[1:0]in_v;
4  wire[1:0]in_v;
5  output[6:0]out_v;
6  reg[6:0] out_v;
7  always@(in_v)
8  begin
9  case(in_v)
10 0:out_v = 7'b00000001;
11 1:out_v = 7'b10011111;
12 2:out_v = 7'b00100010;
13 3:out_v = 7'b00000110;
14  endcase
15  end
16  endmodule
```

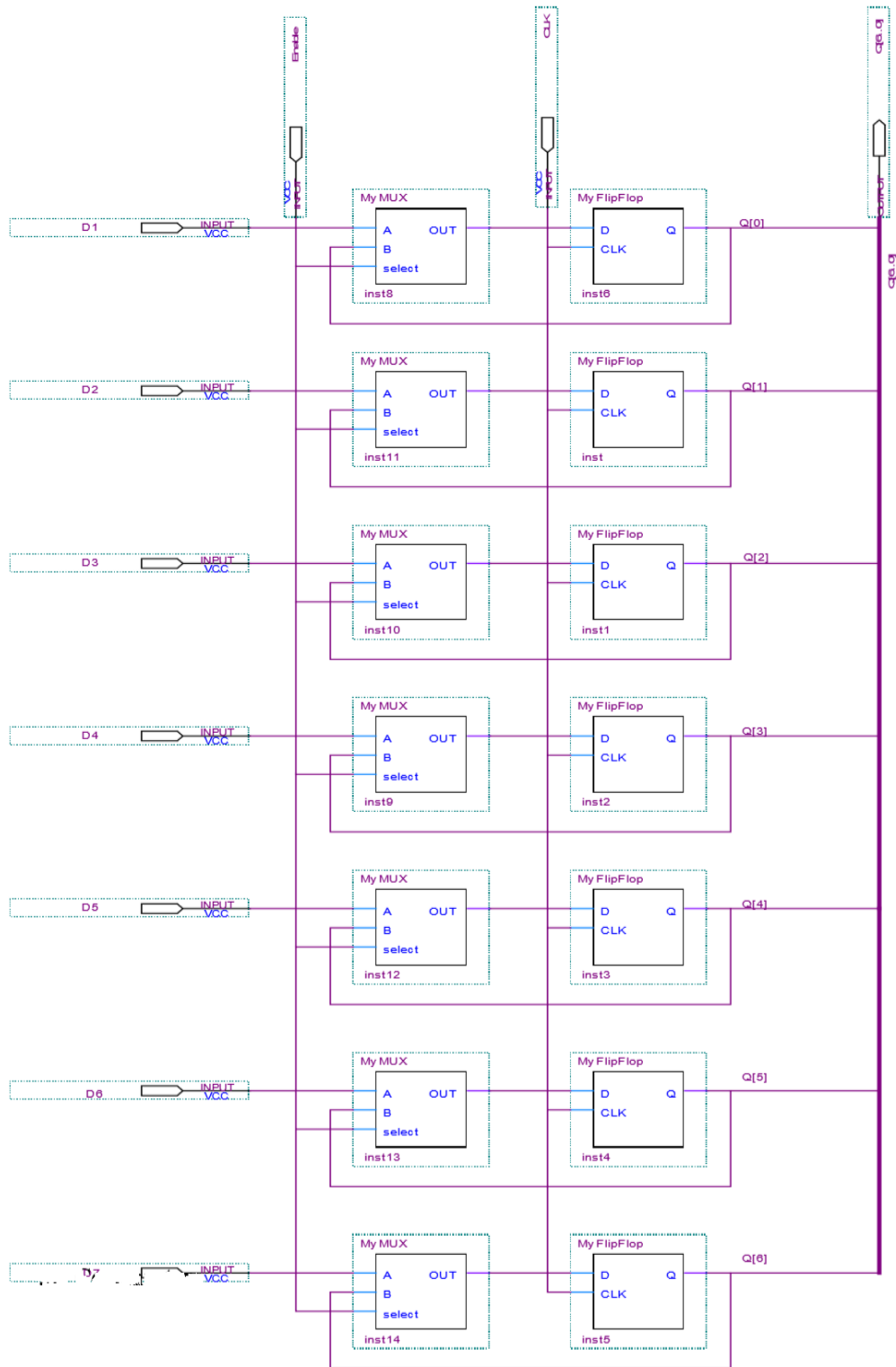
3. Memory System:

1-Write the code of a D- Flip Flop, compile and simulate it (**Pre Lab**).

2-Write the code of a 2x1 MUX, compile and simulate it (**Pre Lab**).

Note: It's important that your MUX behaves as explained in the memory system section (check back the theory).

4. Then, use a block diagram to build the following design:



5. Comparator:

Write down the following code, compile and simulate it (**Pre Lab**).

```
3   module MyComp (CData, out) ;
4   input  [6:0] CData;
5   wire  [6:0] CData;
6   output out;
7   reg  out;
8   always @(CData)
9   begin
10  if(CData == 7'b0100100)
11  out <= 1'b1;
12  else
13  out <= 1'b0;
14  end
15  endmodule
```

Note: For simplifying reasons we will build one comparator based on a reference value X (in this example X = 5), you can build four different comparator with four different values to compare with.

Putting everything together:

Build and design the security system using the components you build during the previous sections. The final block design should look as the one in FIG 9.3 (check the next page please).

Assign pins values to the security system design you just build and then download the system to the FPGA board.

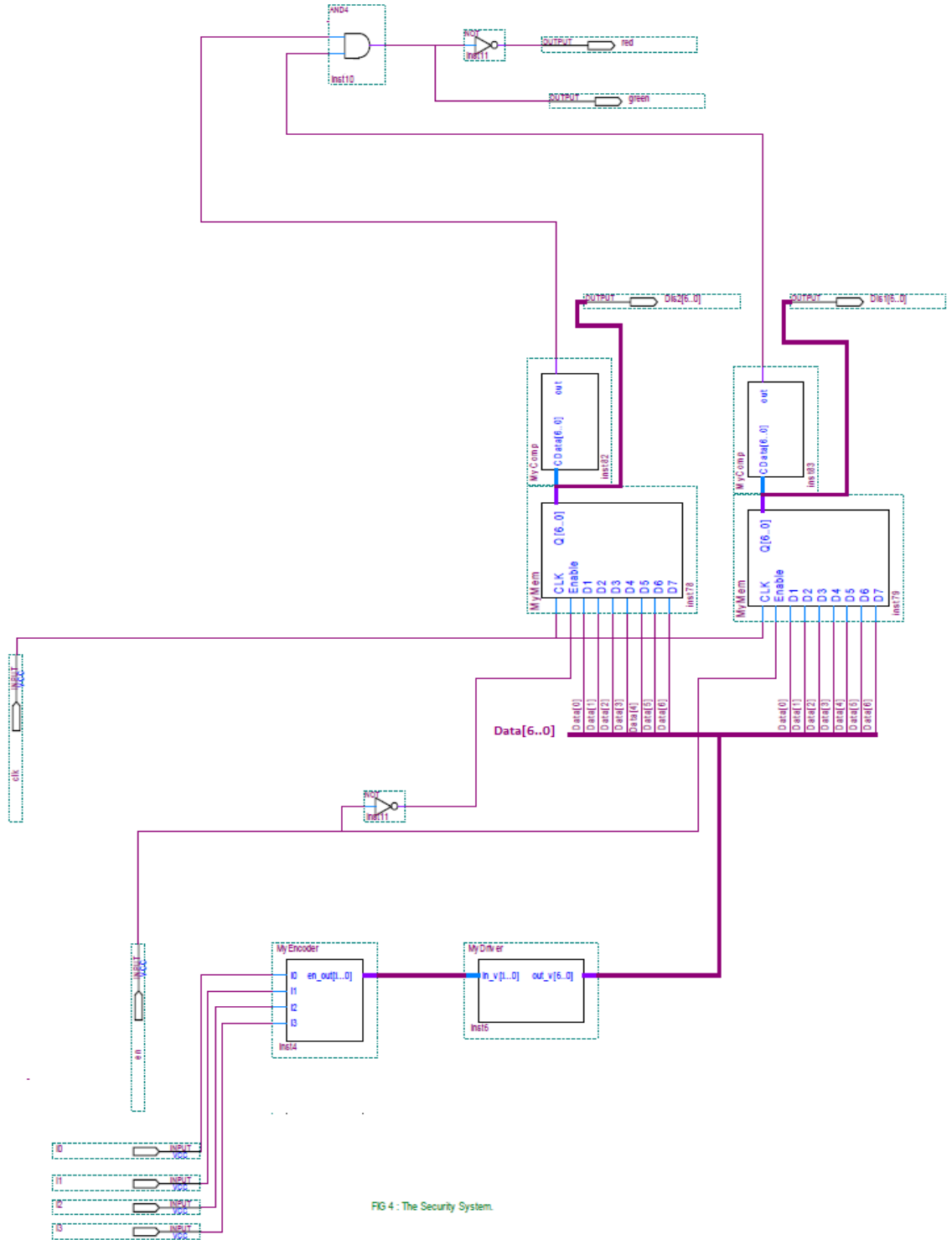


FIG 4 : The Security System.

Figure 9.3



Faculty of Engineering & Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

10. Experiment. No. 10- Simple Computer Simulation

Objective:

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 you practice in the ideas of designing a simple CPU with the Verilog HDL notation.

Background

- **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:**

1. **Memory:** Storage of information (data/program)
2. **Processing Unit:** Computation/Processing of Information
3. **Input:** Means of getting information into the computer. e.g. keyboard, mouse
4. **Output:** Means of getting information out of the computer. e.g. printer, monitor
5. **Control Unit:** Makes sure that all the other parts perform their tasks correctly and at the correct time.

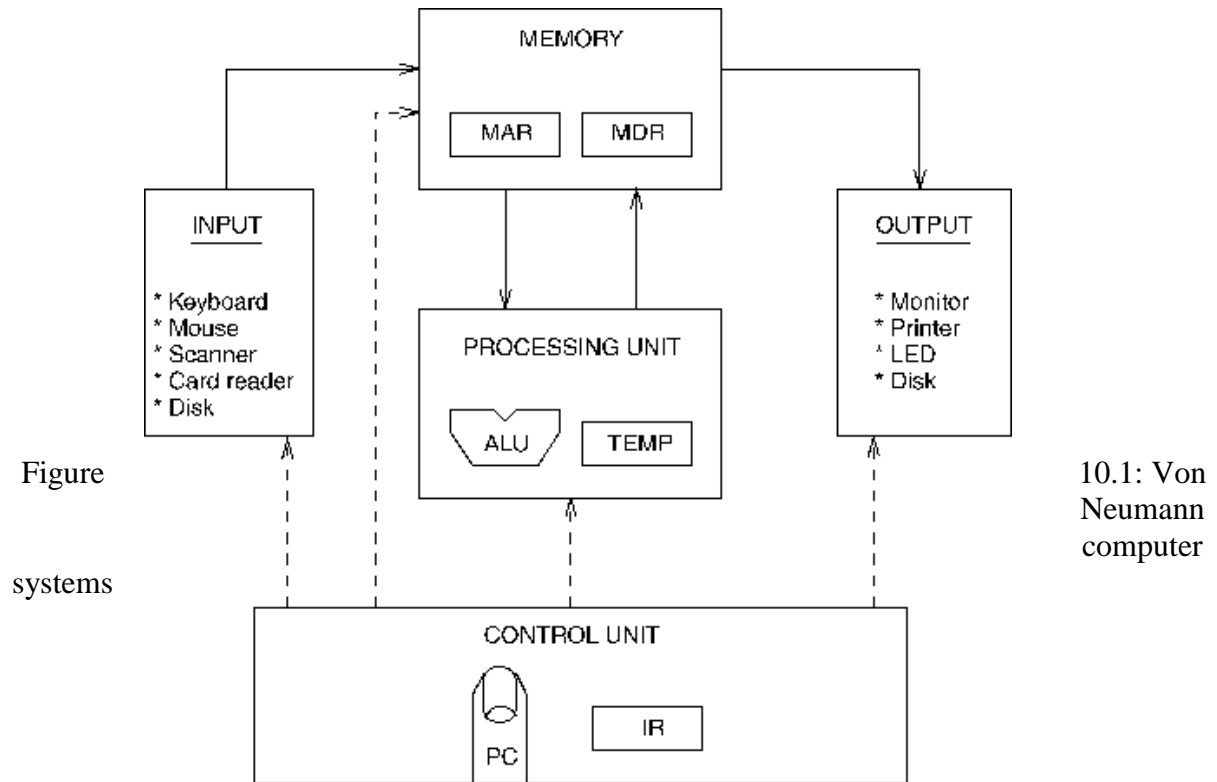


Figure
systems

10.1: Von
Neumann
computer

- **General Registers:**

The number of registers in a processor unit may vary from one processor to another. Below are the general registers used by most processor:

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 for 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 purpose, the resulting computer is said to have general register organization. In the case of processor registers, a registers is selected by the multiplexers that form the buses.

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.

- 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. Details on how this is achieved is a topic for a hardware-focused course in computer architecture.

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.

mull x, y, product

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

- **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. Following are the main addressing modes that are used on various platforms and architectures.

Our Simple Computer

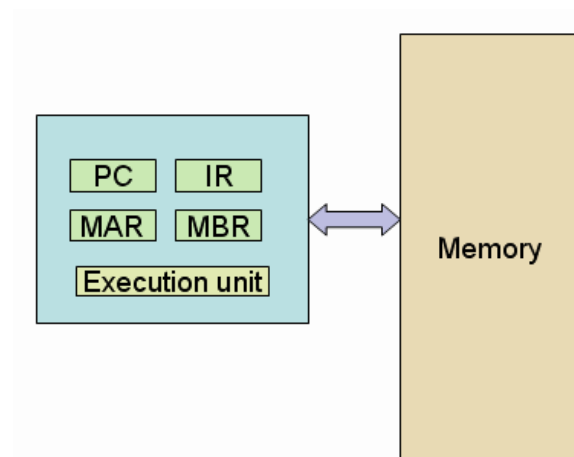
SIMCOMP has a two byte-addressable memory with size of 128byte. The memory is synchronous to the CPU, and the CPU can read or write a word in single clock period. The memory can only be accessed through the memory address register(MAR) and the memory buffer register(MBR). To read from memory, you use:

MBR <= Memory[MAR];

And to write to memory, you use:

Memory[MA] <= MBR;

The CPU has three registers -- an accumulator (**AC**), a program counter (**PC**) and an instruction register (**IR**). In addition, The SIMCOMP has only three instructions: **Load**, **Store**, and **Add**. The size of all instructions is 16 bits; all the instructions are single address instructions and access a word in memory.





InstructionFormat

The opcodes are:

- **0011 LOAD M**// loads the contents of memory location **M** into the accumulator.
- **1011 STORE M**// stores the contents of the accumulator in memory location **M**.
- **0111 ADD M**// adds the contents of memory location **M** to the contents of the accumulator.

Procedure:

1- The Verilog program described by the following table is shown in next page, study, and simulate the code.

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
32	Data	7	Memory[32]=16'd7
33	Data	5	Memory[32]=16'd5

```

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 the program
22     PC = 10; state = 0;
23 end
24
25
26 always @(posedge clock) begin
27     case (state)
28     0: begin
29         MAR <= PC;
30         state=1;
31     end
32     1: begin // fetch the instruction from
33         IR <= Memory[MAR];
34         PC <= PC + 1;
35         state=2; //next state
36     end
37     2: begin //Instruction decode
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

```

Task1:

Modify the code to include the jump instruction.

1. 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.

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'h100B
32	Data	7	Memory[32]=16'd7
33	Data	5	Memory[32]=16'd5

2-SIMCOMP2: Add register file

Modify the instruction format so that SIMCOMP2 can handle four addressing modes and four registers. To this end, SIMCOMP is an **accumulator** machine which you can think of as a machine with one general-purpose register. Historically, many old computers were accumulator machines.

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];
```

And, 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 MEM.* 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.
```

Task2:

Modify the four instructions of the old SIMCOMP2 to the following new form:

LOAD R[i],M loads the contents of memory location **M** into R[i].

STORE R[i],M stores the contents of R[i] in memory location **M**.

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 **PC starts at 10**, and suppose **IR[9:8]** is used to specify the register number, with add instruction **IR[11:10]** destination register, **IR[9:8]**, **IR[7:6]** source1, source2 respectively.

3	DATA	A	Memory[3]=16'hA
4	DATA	6	Memory[4]=16'h6
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

3-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 an LOAD, the operand is loaded into the register.

LoadI R1,8 R1 <- 8

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

PC = 10

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

Memory [11] LoadI R2,-4 //Use 2's complement to represent (-4)

Memory [12] Add R1,R1,R1

Memory [13] Store R1,5

3	DATA	A	Memory[3]=16'hA
4	DATA	6	Memory[4]=16'h6
10	LOADI	R1,3	0011-1001-0000-0011b=16'h3903
11	LOADI	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

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

MBR <= ~(~(IR[7:0])+1);



Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 211

Digital Electronics and Computer Organization Lab

11. Experiment No. 11 - Arithmetic Elements

OBJECTIVES

- 1.To understand functions and applications of the ALU (arithmetic logic unit).
- 2.To perform arithmetic and logic operations using the74181ALU IC.
- 3.To understand the construction and applications of parity generators.
- 4.To generate parity bit using XOR gates and parity generator IC.

EQUIPMENT REQUIRED

1. IT-3000Basic Electricity Circuit Lab.
2. IT-3003 Module.

11.1 ARITHMETIC LOGIC UNIT (ALU) CIRCUIT

Theory

The logic diagram of the ALU is shown in Fig. 11.1:

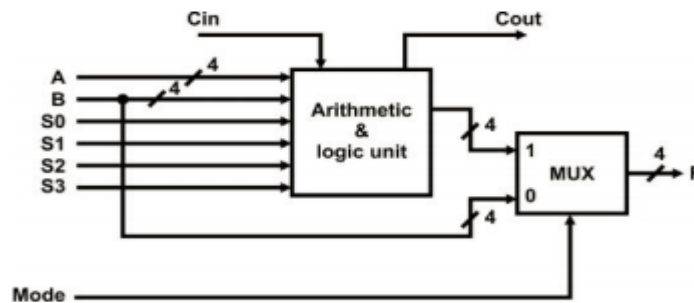


Fig 11.1

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. Fig. 11.2 shows the pin assignment:

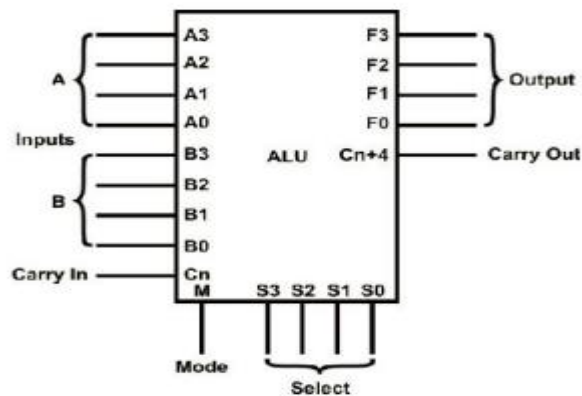


Fig 11.2

The circuit has two 4-bit inputs A and B, as well as a “carry-in” (C_n) 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 “ C_{n+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.

Procedure

1. Connect function-select lines S3~S0 to DIP2.7~2.4 respectively. Connect M to Data Switch SW0 to select arithmetic and logic operation. When M= “0” input B3~B0 is displayed at output. When M= “1” arithmetic and logic function is performed.

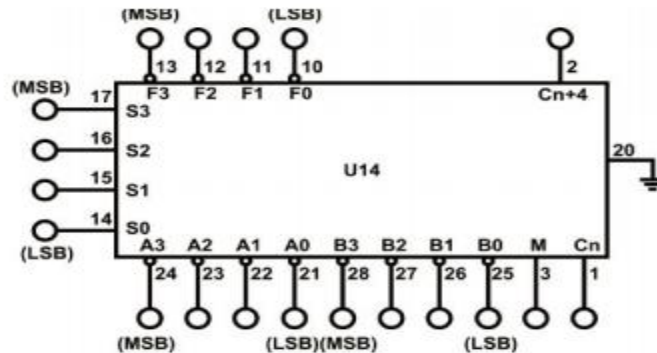


Fig 11.3

2. 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.

3. Set M to “1” to perform the following arithmetic functions.

a. 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= _____

b. Set Cn to “1” and add 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= _____

c. Set Cn to “0”. When S3S2S1S0=0001 perform the subtraction.

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= _____

4. Again set M to “1” to perform the following arithmetic and logic functions according to Table 11.1. Set inputs sequence $A_0 \sim A_3 = A$, $B_0 \sim B_3 = B$ from DIP switches.

Input selection				M=H Cn=L	Output			
S3	S2	S1	S0		F3	F2	F1	F0
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×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	$(\sim A) \times (\sim B)$				

Table 11.1

11.2 BIT PARITY GENERATOR CIRCUIT

Theory

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 a “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 Fig. 11.4 will be 0 if the inputs ABCDEFGH is equal to 10111011.

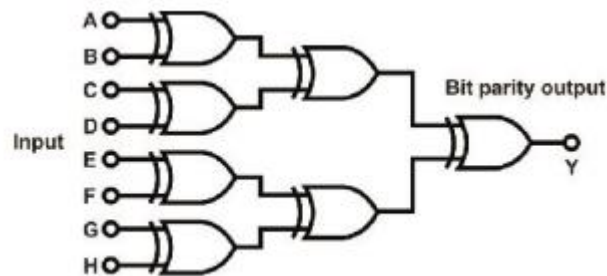


Fig 11.4 “Even” Bit Parity Generator Circuit

Procedure

(a) Bit Parity Generator Construct with XOR Gates (Module IT-3003 block Half-Adder)

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

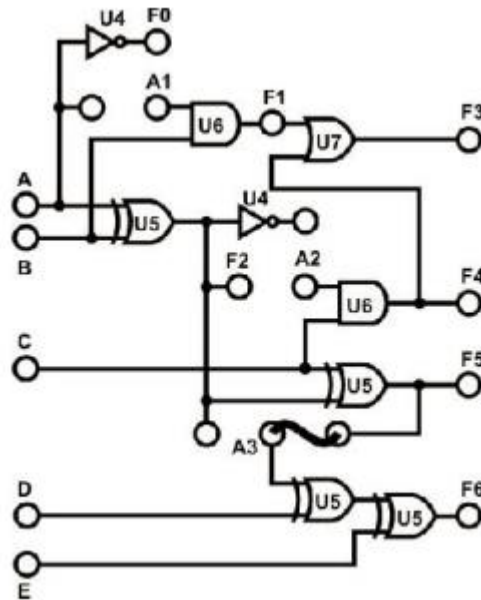


Fig 11.5

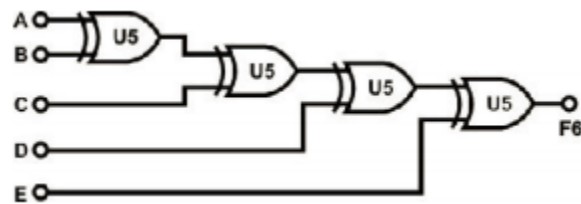


Fig 11.6 "Even" Bit Parity Generator Circuit

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

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	

Table 11.2

(b) Bit Parity Generator IC

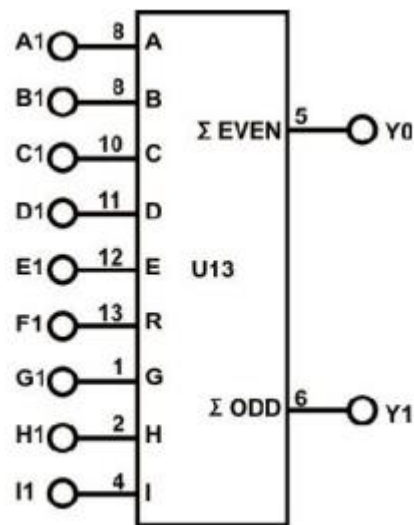


Fig 11.7

1. U13 on block Bit Parity Generator of module IT-3003 is a bit parity generator IC. Connect inputs A1, B1, C1, D1, E1, F1, G1, H1 and I1 to DIP Switches 1.0~1.7 and 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.

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		

Table 11.3