Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 02: (a) X-OR Gate by IC

(b) by circuit

Note: Submit this lab hand-out in the next lab

Submission Profile

Name: Submission date (dd/mm/yy):

Marks obtained: Receiving authority name and signature:

Comments:

Instructor Signature

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify **X-OR** gate operations using 74LS86 IC & Module KL-33001
- To implement and verify **X-OR gate by using circuit**

Lab Hardware and Software Required:

1. 74LS86 IC (X-OR gate)
2. 7408, 7432,7404
3. Module KL-33001
4. Breadboard
5. Connecting Wires

Background Theory:

X-OR gate:

Exclusive OR gate, commonly abbreviated as X-OR is a special type of OR gate. Logic symbol for the XOR gate is shown in Fig 3.1 (a & b). The output of an X-OR is high only when the two inputs are at opposite logic levels. Alternatively, X-OR gate provides a high output if one input or the other is high, but not both. The X-OR gate has only two inputs. Unlike the others. The X-OR output expression is written as X=A⊕B, where the symbol" ⊕" represented X-OR operation.

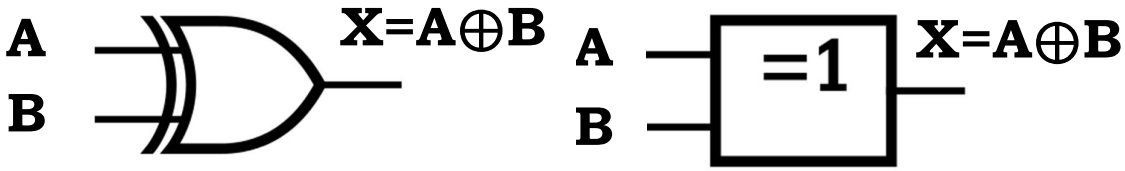


Fig 3.1: (a) Distinctive symbol of X-OR gate (b) Standard X-OR gate symbol

74LS86 2-Input X-OR gate IC:

In order to implement the X-OR operation using IC, the TTL 74LS86 2-input X-OR gate IC can be used. This IC has 14 pin Dual Inline Package (DIP) configuration as shown in fig 3.2. The power supply connections are made to pin 7 and 14. Pin 1 is identified by a small indented circle next to it or by a notch cut out between pin 1 and 14.

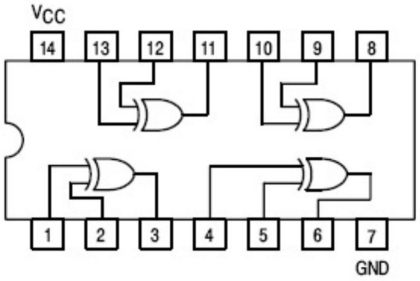


Fig 3.2: 74LS86 2-input X-OR gate IC pin configuration

Lab Examples:

Implementation of X-OR gate

- In order to implement the X-OR gate operations with the help of ICs, take the **74LS86**
- Pin Assignments for the ICs have been shown in Fig 3.2
- Take 74LS86 IC and insert it in the breadboard present on the **KL-33001** training kit
- Connect its pin 14 to +5V and pin 7 to ground
- Use first gate (any one can be taken) from IC by connecting input pin 1 & input pin 2 to SW0 & SW1 respectively, and output pin3to LED and observe the AND gate operation
- Fill the following observation table

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Table 3.1: Observation table of 2-input X-OR gate

Lab Activity:

Implementation of X-OR gate by using 7404, 7432, 7408 ICs with verification of equation.

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 02: (a) X-OR Gate by IC

(b) by circuit

Note: Submit this lab hand-out in the next lab

Submission Profile

Name: Submission date (dd/mm/yy):

Marks obtained: Receiving authority name and signature:

Comments:

Instructor Signature

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify **X-OR** gate operations using 74LS86 IC & Module KL-33001
- To implement and verify **X-OR gate by using circuit**

Lab Hardware and Software Required:

1. 74LS86 IC (X-OR gate)
2. 7408, 7432,7404
3. Module KL-33001
4. Breadboard
5. Connecting Wires

Background Theory:

X-OR gate:

Exclusive OR gate, commonly abbreviated as X-OR is a special type of OR gate. Logic symbol for the XOR gate is shown in Fig 3.1 (a & b). The output of an X-OR is high only when the two inputs are at opposite logic levels. Alternatively, X-OR gate provides a high output if one input or the other is high, but not both. The X-OR gate has only two inputs. Unlike the others. The X-OR output expression is written as X=A⊕B, where the symbol" ⊕" represented X-OR operation.

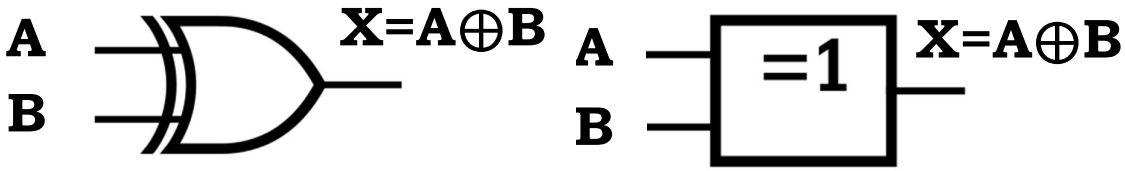


Fig 3.1: (a) Distinctive symbol of X-OR gate (b) Standard X-OR gate symbol

74LS86 2-Input X-OR gate IC:

In order to implement the X-OR operation using IC, the TTL 74LS86 2-input X-OR gate IC can be used. This IC has 14 pin Dual Inline Package (DIP) configuration as shown in fig 3.2. The power supply connections are made to pin 7 and 14. Pin 1 is identified by a small indented circle next to it or by a notch cut out between pin 1 and 14.

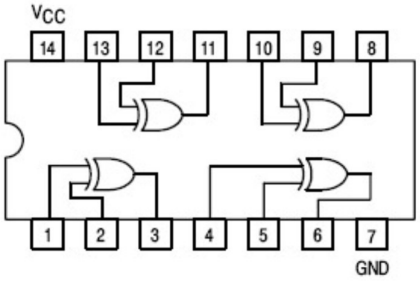


Fig 3.2: 74LS86 2-input X-OR gate IC pin configuration

Lab Examples:

Implementation of X-OR gate

- In order to implement the X-OR gate operations with the help of ICs, take the **74LS86**
- Pin Assignments for the ICs have been shown in Fig 3.2
- Take 74LS86 IC and insert it in the breadboard present on the **KL-33001** training kit
- Connect its pin 14 to +5V and pin 7 to ground
- Use first gate (any one can be taken) from IC by connecting input pin 1 & input pin 2 to SW0 & SW1 respectively, and output pin3to LED and observe the AND gate operation
- Fill the following observation table

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Table 3.1: Observation table of 2-input X-OR gate

Lab Activity:

Implementation of X-OR gate by using 7404, 7432, 7408 ICs with verification of equation.

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 03: Universal Gate

(NAND & NOR Gates)

Note: Submit this lab hand-out in the next lab

Submission Profile

Name: Submission date (dd/mm/yy):

Marks obtained: Receiving authority name and signature:

Comments:

Instructor Signature

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify **NAND** gate operations using 74LS00 IC & Module KL-33001
- To implement and verify **NOR** gate operations using 74LS02 IC & Module KL-33001

Lab Hardware and Software Required:

1. 74LS00 IC (NAND gate)
2. 74LS02 IC(NOR gate)
3. Module KL-33001
4. Breadboard
5. Connecting Wires

Background Theory:

NAND gate:

The logic symbols for NAND gate is shown in Fig 2.1 (a & b). It consists of AND symbol with inverter symbol added to output. The operation of NAND gate is same as the AND gate except that its output is inverted. NAND gates can also have two or more than two inputs and a single output. NAND gate produces a LOW output only when all the inputs are HIGH. When any of the inputs is LOW, the output will be HIGH.

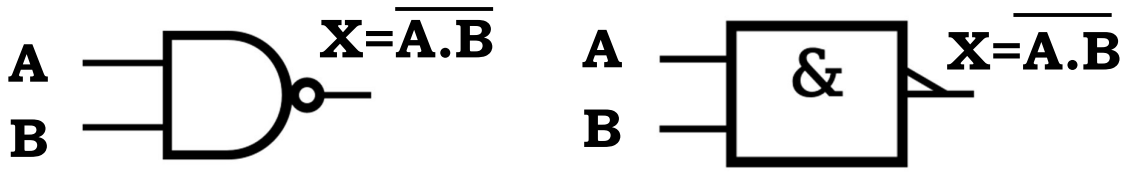


Fig 2.1: (a) Distinctive symbol of NAND gate (b) Standard NAND gate symbol

74LS00 2-Input NAND gate IC:

In order to implement the NAND gate operation using IC, the TTL 74LS00 2-input NAND gate IC can be used. This IC has 14 pin Dual Inline Package (DIP) configuration as shown in Fig 2.2. The power supply connections are made to pin 7 and 14. Pin 1 is identified by a small indented circle next to it or by a notch cut out between pin 1 and 14.

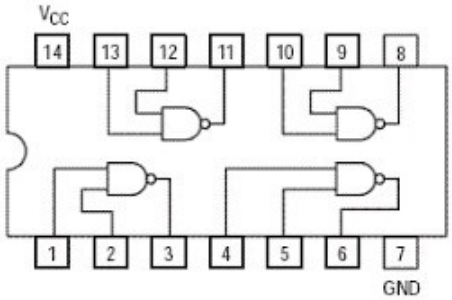


Fig 2.2: 74LS00 2-input NAND gate IC pin configuration

NOR gate:

The operation of the NOR gate is same as that of the OR gate except that its output is inverted. You can think of a NOR gate as an OR gate with an inverter at its output. The logic symbol for NOR gate is shown in Fig 2.3 (a & b). NOR gates can also have two or more than two inputs and a single output. The NOR gate produces a low output, When any one of input is high and produces a high output, when all inputs are low.

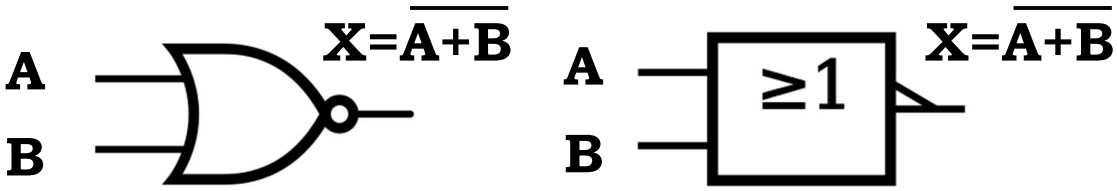


Fig 2.3: (a) Distinctive symbol of 2-input NOR gate (b) Standard 2-input NOR gate symbol

74LS02 2-input NOR gate IC:

In order to implement the NOR gate operation using IC, the TTL 74LS02 2-input NOR gate IC can be used. This IC has 14 pin Dual Inline Package (DIP) configuration as shown in Fig 2.4. The power supply connections are made to pin 7 and 14. Pin 1 is identified by a small indented circle next to it or by a notch cut out between pin 1 and 14.

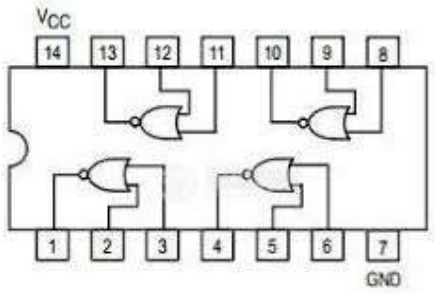


Fig 2.4: 74LS02 2-input NOR gate IC pin configuration

Lab Examples:

Implementation of NAND gate

- In order to implement the NAND gate operations with the help of ICs, take the **74LS00**
- Pin Assignments for the ICs have been shown in Fig 2.2
- Take 74LS00 IC and insert it in the breadboard present on the **KL-33001** training kit
- Connect its pin 14 to +5V and pin 7 to ground
- Use first gate (any one can be taken) from IC by connecting input pin 1 & input pin 2 to SW0 & SW1 respectively, and output pin3to LED and observe the AND gate operation
- Fill the following observation table

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Table 2.1: Observation table of NAND gate

Lab Activities:

Implementation of NOR gate

- In order to implement the AND operations with the help of IC, take the **74LS02** IC
- Pin Assignments for the ICs have been shown in Fig 2.4
- Take 74LS02 IC and insert it in the breadboard present on the **KL-33001** training kit
- Connect its pin 14 to +5V and pin 7 to ground
- Use first gate (any one can be taken) from IC by connecting input pin 1 & input pin 2 to SW0 & SW1 respectively, and output pin3to LED and observe the AND gate operation
- Fill the following observation table

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Table 2.2: Observation Table of NOR gate

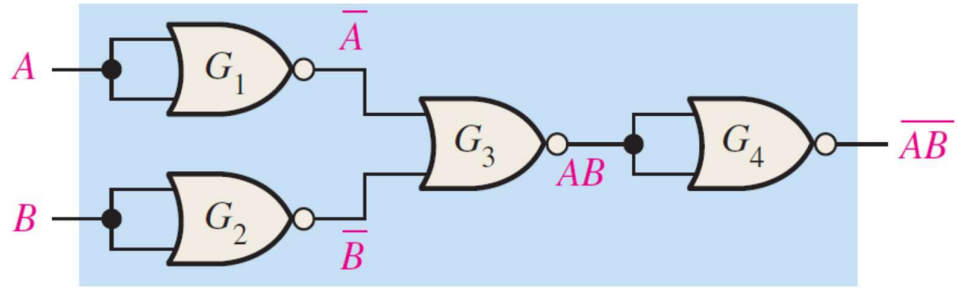
Lab Exercises:

NAND-NOR Universal Gates

NAND and NOR gates are known as Universal Gates as they can be used to construct an AND gate, an OR gate, an INVERTER, or any combination of these functions. The NAND Universal Gate can also be used to implement a NOR gate. Similarly, a NOR gate can be used to implement a NAND gate.

1. **Implement NAND gate using NOR gates**
   - Draw the schematic Diagram
   - Write the Boolean Expression
   - Fill the Observation Table

Schematic Diagram:



Boolean Expression:

$$X = \overline{\overline{A} + \overline{A} + \overline{B} + \overline{B}} + \overline{\overline{A} + \overline{A} + \overline{B} + \overline{B}}$$

as $A + A = A$

$$X = \overline{\overline{A} + \overline{B}} + \overline{\overline{A} + \overline{B}}$$

as $\overline{\overline{A} + \overline{B}} + \overline{\overline{A} + \overline{B}} = \overline{\overline{A} + \overline{B}}$

$$X = \overline{\overline{A} + \overline{B}}$$

as $\overline{\overline{A}} = A$

$$X = \overline{A} + \overline{B}$$

Demorgans law $\overline{A} + \overline{B} = \overline{A.B}$

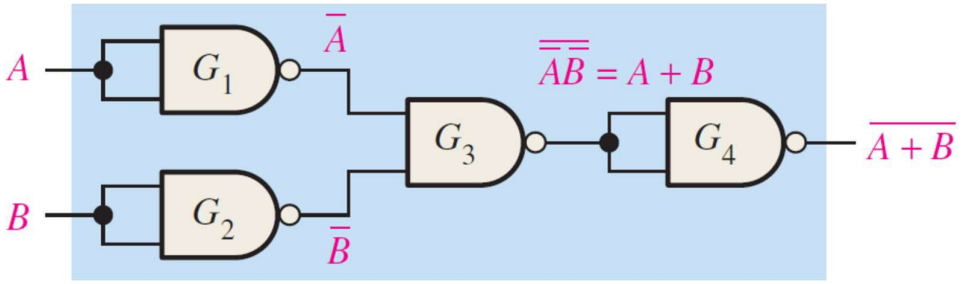
$$X = \overline{A.B}$$

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Fig 2.3: Observation table for 2-input NAND gate using NOR gates

2. **Implement NOR gate using NAND gates**
   - Draw the schematic Diagram
   - Write the Boolean Expression
   - Fill the Observation Table

Schematic Diagram:



Boolean Expression:

$$X = \overline{\overline{A.A.B}.\overline{B.A.A.B}.\overline{B.B}}$$

as $A.A = A$

$$X = \overline{\overline{A.B}.\overline{A.B}}$$

as $\overline{A.B}.\overline{A.B} = \overline{A.B}$

$$X = \overline{\overline{A.B}}$$

as $\overline{\overline{A}} = A$

$$X = \overline{A}.\overline{B}$$

Demorgans law $\overline{A}.\overline{B} = \overline{A + B}$

$$X = \overline{A + B}$$

Input		Output	
A	**B**	**LED (on / off)**	**Level (1 / 0)**
0	0		
0	1		
1	0		
1	1		

Table 2.4: Observation table for 2-input NOR gate using NAND gates

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 04: Half Adder & Full Adder

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify Half adder circuit using XOR and AND Gate
- To implement and verify Full adder circuit using XOR, AND and OR Gate

Lab Hardware and Software Required:

1. 7486, 7408, 7432
2. Module KL-33001
3. Breadboard
4. Connecting Wires

Background Theory:

Half Adder:

Many logic circuits must be supplied with devices which can carry out the sum between two numbers, for this purpose the adder circuit is used. The basic rules for binary addition are:

Binary Addition:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 and carry 1

These operations are performed by a logic circuit called a half adder. A half adder is a binary adder which adds on two bits, it accepts two binary digits on its inputs and produces two binary digits on its output, a sum bit and a carry bit. From the logical operation of the half adder as expressed by basic binary addition rules, we observe that the sum output is a1 only if the input bits are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables. Notice that the carry output is 1 only when input bits are 1, therefore carry can be expressed as the AND of the input variables. A half adder is represented by the logic symbol and logic diagram as shown in Fig 4.1 (a & b).

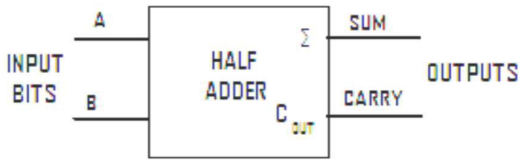
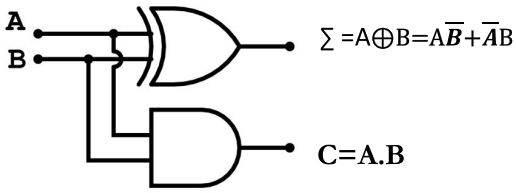


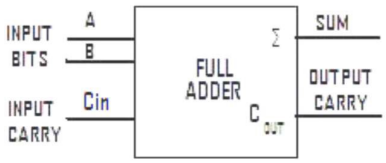
Fig 4.1: (a) Half Adder logic symbol



(b) Half Adder logic diagram

FULL ADDER:

The second basic category of adder is the full adder. The full adder accepts three inputs including an input carry and generates a sum output and an output carry. The basic difference between a full adder and half adder is that the full adder accepts an input carry. The full adder must add the two input bits and the input carry. From the half adder we know the sum of the bits A and B is the X-OR of those two variables A B. For the input carry (Cin) to be added to the input bits., it must be exclusive ORed of those variables AB , yielding the equation for the sum output of the full adder as Σ = (A B) Cin. This means that to implement the full adder sum functions, the X-OR gates can be used. The output carry is a 1 when both inputs to the second X-OR gate are 1. The equation for the carry output can be developed as Cout=AB+(A B) Cin. The logic symbol and logic diagram for a full adder are given in Fig 4.2(a & b).



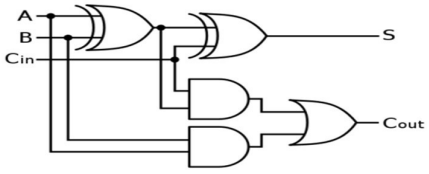


Fig 4.2: (a) Full Adder logic symbol (b) Full Adder logic diagram

Input		Output (Carry)		Output (Sum ∑)	
A	**B**	**LED (on / off)**	**Level (1 / 0)**	**LED (on / off)**	**Level (1 / 0)**

Table 4.1: Observation table of Half Adder

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 05: Implementation of 4 bit Parallel Adder by IC 74283

Lab Learning Objectives:

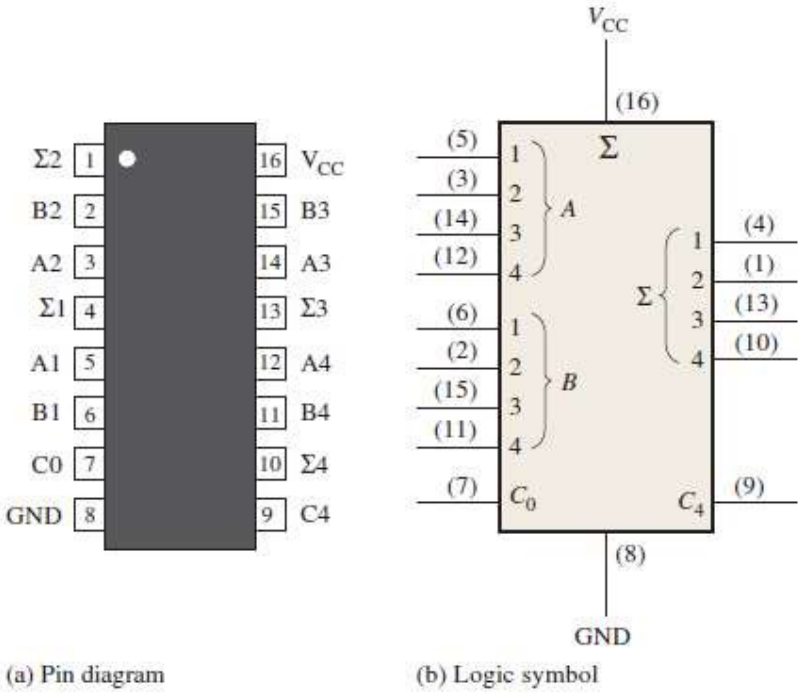
Upon successful completion of this experiment, the student will be able:

- To implement and verify 4 bit Parallel Adder by IC 74283

Lab Hardware and Software Required:

1. 74283 IC
2. Module KL-33001 / Elvis II
3. Breadboard
4. Connecting Wires

IMPLEMENTATION OF 4 BIT PARALLEL ADDER (PIN DIAGRAM AND LOGIC SYMBOL of 74283 IC):



(a) Pin diagram (b) Logic symbol

EXAMPLES:

Take 3 different examples of data A and B and perform parallel addition by using IC 74283 (show inputs and outputs on logic symbol)

Input			Output (Carry)		Output (Sum ∑)	
A	**B**	**Cin**	**LED (on / off)**	**Level (1 / 0)**	**LED (on / off)**	**Level (1 / 0)**

Table 4.2: Observation table of 3-bit Full Adder

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 05: Comparator by IC 7485

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Lab Learning Objective:

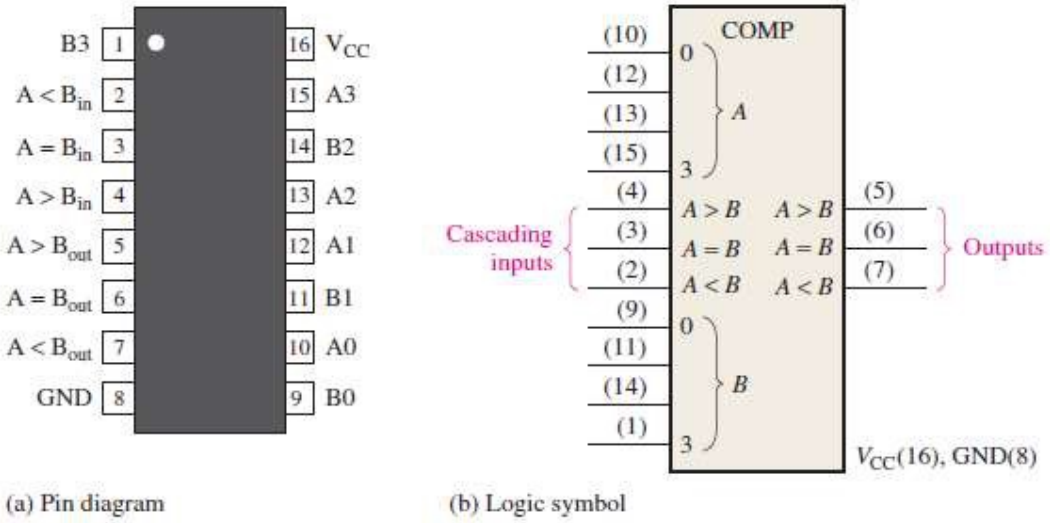
Upon successful completion of this experiment, the student will be able:

- To implement and verify Comparator operations using 74LS85 IC

Lab Hardware and Software Required:

1. 74LS85IC (Comparator)
2. Module KL-33001/ Elvis II
3. Breadboard
4. Connecting Wires

IMPLEMENTATION OF 4 BIT MAGNITUDE COMPARATOR (PIN DIAGRAM AND LOGIC SYMBOL OF IC 7485):



(a) Pin diagram (b) Logic symbol

METHODOLOGY:

Write down the methodology of comparator by using IC 7485 in your own words with practical applications.

EXAMPLES:

Take 3 different examples and show input and output of comparator using 7485 IC.

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 07 : Decoder by IC 7442

Note: Complete lab work and upload on e learning

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify Decoder operation using **74LS42** IC

Lab Hardware and Software Required:

1. 74LS42 IC (BCD-to-Decimal Decoder)
2. Module KL-33001/ Elvis II
3. Breadboard
4. Connecting Wires

Background Theory:

Decoder:

Decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to that input number. In other words, a decoder circuit looks at its inputs, determines which number is present there, and activates one output that corresponds to that number; all other outputs remain inactive. Before the design of decoder, we must decide that whether we want an active high level output or an active low level output to indicate the value selected. For an active-high indication the required output is high and all other outputs are low; while in case of active-low, the required output is low and all other outputs are high. Fig 1 (a & b) shows the logic symbol and logic diagram for 2x 4 decoder. It uses all AND gates, so the outputs are active high. For a given input code, the only output that is active (high), is one corresponding to decimal equivalent of the binary input code.

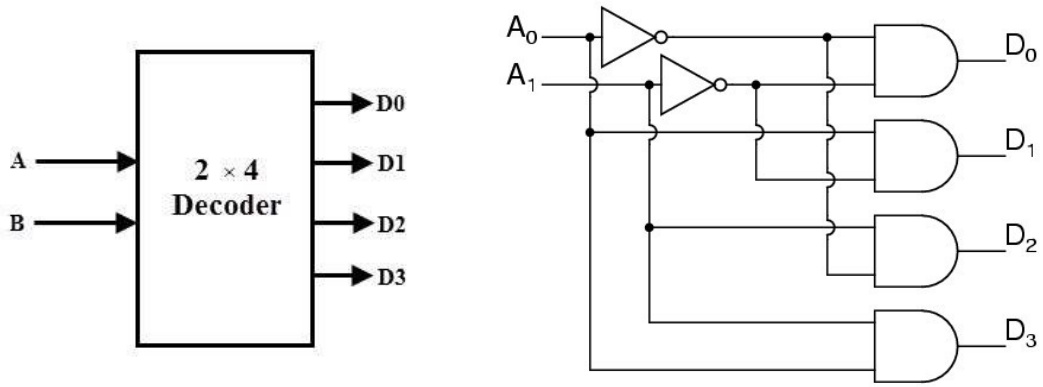


Fig 1: (a) Logic Symbol of 2 x 4 Decoder (b) Logic Diagram of 2 x 4 Decoder

74LS42 Decoder IC:

The TTL 74LS42 accepts four lines of input data and it has ten active-low outputs. The inputs must be supplied by a coder type BCD, which causes the activation of the output line corresponding to the applied number. Each output goes low only when its corresponding BCD input is applied, while all other inputs remain high. Fig 2 shows the pin-out of 74LS42 IC.

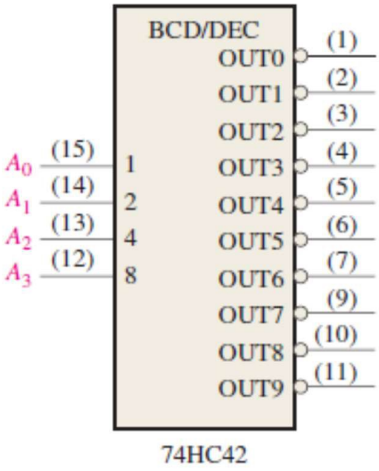


Fig 2: 74LS42 Decoder IC pin configuration

Lab Activity:

Implementation of BCD-to-Decimal Decoder using 74LS42:

- Connect the inputs A, B, C and D to the switches
- Connect outputs 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 to the LEDs
- Compose all binary numbers from 0 to 9 and analyse the state of LEDs

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

Table : Observation table of BCD-to-Decimal Decoder

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab #08 : Multiplexer by IC 74151

Note: Complete the lab work and upload on e learning

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement and verify Multiplexer operation using 74LS151 IC

Lab Hardware and Software Required:

1. 74LS151 IC
2. Module KL-33001/ Elvis II
3. Breadboard
4. Connecting Wires

Background Theory:

Multiplexer:

A Multiplexer (MUX) is a device that allows digital information from several sources to route on to single line common to destination. The basic multiplexer has several data inputs and a single output line. It has also a data select input, which allows digital information on the one input to be switched to be output line. Multiplexers are also called data selectors. The applications of multiplexer are many, and range from parallel to serial converters, register shifting, data transmission etc. Logic symbol and diagram for 2-input multiplexer is shown in Fig 1 (a & b):

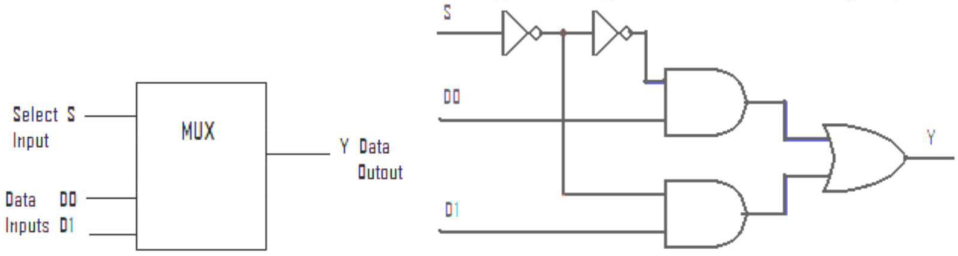
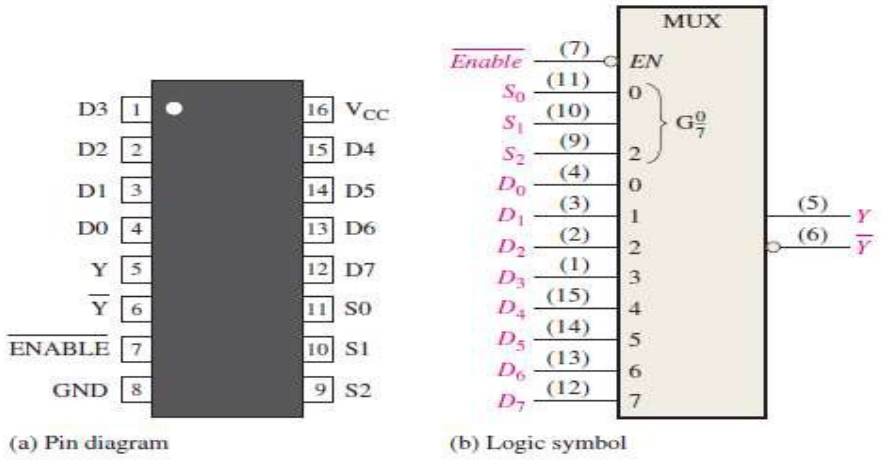


Fig 1: (a) Logic Symbol for 2-input MUX (b) Logic Diagram for 2-input MUX

74151 MULTIPLEXER IC:

The TTL 74151 Multiplexer IC has eight data inputs multiplexers. It has three selection inputs, one enable input which is active low, and two outputs. Fig 2 shows the pin configuration of IC.



Fig,2: pin configuration of 74LS151 MUX IC

Lab Activity:

Implement the Multiplexer IC 74151 on Breadboard and complete the truth table and show all possible states.

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab #9 : S R Latch

Submission Profile

Name: Submission date (dd/mm/yy):

Marks obtained: Receiving authority name and signature:

Comments:

Instructor Signature

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

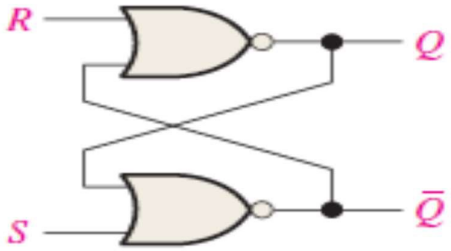
- To implement and verify R S latch operation by using two NOR gates.

Lab Hardware and Software Required:

1. 74LS02 IC
2. Module KL-33001/ ELVIS II
3. Breadboard
4. Connecting Wires

Background Theory:

S R Latch: A Latch is a type of bistable logic device or multivibrator, An active high input S R (SET, RESET) Latch is formed with two cross coupled NOR gates as shown in figure.



Lab Example:

Implementation and verification of active high inputs S R Latch using NOR gates:

- Carry out the circuit of Fig above by using IC of NOR gates 7402.
- Make the circuit on breadboard present on the **KL-33001** training kit / ELVIS-II
- Apply power supply to it
- Connect the inputs S and R to the TTL switches
- Connect the outputs Q and $\overline{Q}$ to the LEDs
- Analyze different states by carrying out different combinations with the switches and fill the observation tables

R	S	Q	$\overline{Q}$	Comments

Observation table of active high S R latch

Lab Exercise: Implement the active low S R Latch and verify the conditions.

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab # 10 : JK-Flip Flop with $\overline{preset}$ and $\overline{clear}$

Note: Complete the lab and upload on e learning

Lab Learning Objectives:

Upon successful completion of this experiment, the students will be able:

- To implement and verify JK flip flop with preset and clear operations using **74LS76 IC** .

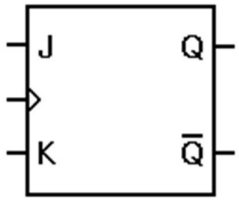
Lab Hardware and Software Required:

1. 74LS76 IC
2. Module KL-33001 / Elvis II
3. Breadboard
4. Connecting Wires

Background Theory:

JK-FLIP FLOP:

The JK-flip flop is very versatile and most widely used flip flop. The functioning of the JK flip flop is identical to that of S R flip flops in the SET, RESET and no change condition of operation. The difference is that the JK flip flop has no invalid state as does the SR flip flop. J K Flip Flop has toggle state if both inputs are 1.Therefore the JK flip flop is a very versatile device that finds application in digital system. Fig (a & b) show a logic symbol and logic diagram for the JK flip flop.



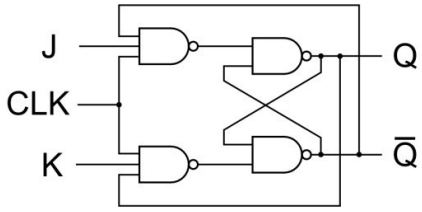


Fig : (a) logic symbol of JK-flipflop (b) logic diagram of JK-flip flop

The data at the J and K inputs of the JK flip flop controls its output. Notice that it differs from the RS flip flop in the output of the flip flop is coupled back to input. J and K inputs control the state of the flip flop in the same way as the S and R inputs do for the clocked RS flip flop except for one major difference; the J=K=1 condition does not result in ambiguous output. For this 1, 1 condition, the flip flop always go to its opposite state upon the arrival of the clock signal. This is called the toggle mode of states (toggle) for each of the clock pulse.

74LS76 JK Flip Flop with preset and clear IC:

The 74LS76 is a popular JK flip flop IC; having dual flip flops (two flip flops in each IC package). Its pin configuration is given in Fig below. It has asynchronous inputs (active low preset and clear) as well as synchronous inputs (J, K) and is negative-edge triggered flip flop. The asynchronous inputs preset and clear are active low, that is low on preset (set) will set (preset) the flip flop (Q=1) and low on clear will reset (clear) the flip flop (Q=0). The asynchronous inputs will cause the flip flop to respond immediately regardless to the clock . For synchronous operation using J,K, and Clock, the asynchronous inputs must be disabled by putting a high level on both.

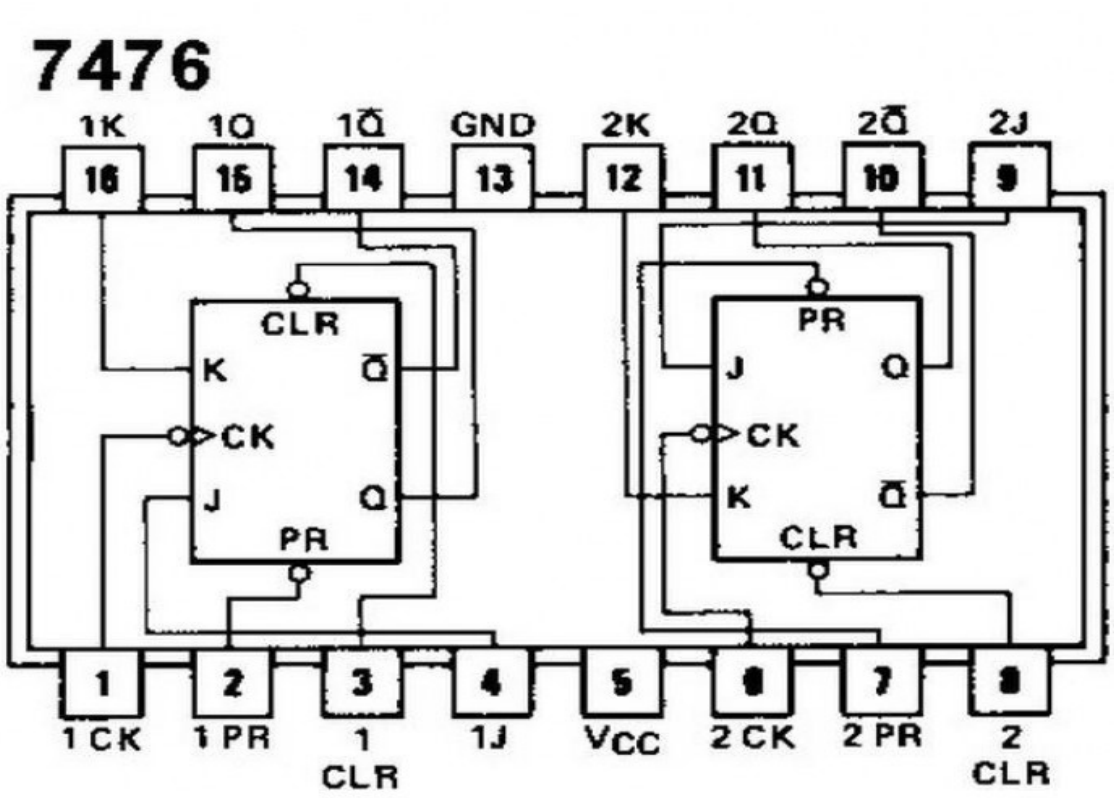


Fig .2 pin configuration of 7476 JK flip-flop with preset and clear IC

Lab Examples:

Implementation of JK flip flop with preset and clear using 7476 IC:

- Carry out the circuits of Fig 2 by using IC 74LS76
- Make the circuit on breadboard present on the **KL-33001** training kit
- Apply power supply to it
- Connect the inputs J & K of circuit to the switches
- Connect the terminal of clock to the input CK of the flip flop
- Connect the outputs Q and $\overline{\mathbf{Q}}$ of both circuits to the LEDs
- Analyze the circuit's behaviour by carrying out different combinations with the switches and fill the observation tables

Inputs					Outputs		Comments
CLK (negative edge triggered)	**J**	**K**	$\overline{pre}$	$\overline{clear}$	**Q**	$\overline{\mathbf{Q}}$	

Table : Observation table of JK flip flop with active low preset and clear

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab #11 : Introduction of VHDL in Muli-Sim

Note: Complete the lab work and upload on e learning

Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To use basic tabs of Model Sim and process of writing and compiling a code
- Basic of Structure Modelling using VHDL

Background Theory:

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system, or anything in between.

The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description. The VHDL language can be regarded as an integrated amalgamation of the following languages:

- Sequential language
- Concurrent language
- Net-list language
- Timing specifications
- Waveform generation language.

Therefore, the language has constructs that enable you to express the concurrent or sequential behavior of a digital system with or without timing. It also allows you to model the system as an interconnection of components. Test waveforms can also be generated using the same constructs. All the above constructs may be combined to provide a comprehensive description of the system in a single model. The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator. It is a strongly typed language and is often verbose to write. It inherits many of its features, especially the sequential language part, from the Ada programming language. Because VHDL provides an extensive range of modeling capabilities, it is often difficult to understand. Fortunately, it is possible to quickly assimilate a core subset of the language that is both easy and simple to understand without learning the more complex features. This subset is usually sufficient to model most applications. The complete language, however, has sufficient power to capture the descriptions of the most complex chips to a complete electronic system.

Lab Activity:

Task 1.

```
--library IEEE; --std_logic
use IEEE.std_logic_1164.all;


entity ANDgate is
port (
      A: in bit;
      B: in bit;
      X: out bit
    ) ;
end entity ANDgate ;

architecture ANDfunction of ANDgate is
begin
X<= A and B;
end architecture ANDfunction;
```

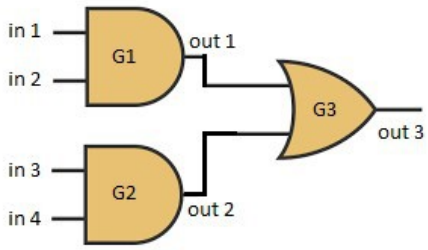
Task 2.

```
-----  OR Circuit
--library IEEE; --std_logic
use IEEE.std_logic_1164.all;

entity ORgate is
port (
      A: in std_logic;
      B: in std_logic;
      X: out std_logic
    ) ;
end entity ORgate;

architecture ORfunction of ORgate is
begin
X<= A or B;
end architecture ORfunction;
```

Task 2.

```
----Fuller Adder
library IEEE;
use IEEE.std_logic_1164.all;

entity FullAdder is
port (A, B, CIN: in std_logic;
    SUM, COUT: out std_logic);
end entity FullAdder;
architecture LogicOperation of FullAdder is
begin
SUM <= (A xor B) xor CIN;
COUT<= ((A xor B) and CIN) or (A and B);
end architecture LogicOperation;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
```




```
library IEEE;
use IEEE.std_logic_1164.all;
entity AND_OR_Logic is
port ( IN1 , IN2 , IN3 , IN4 : in std_logic; OUT3: out std_logic );
end entity AND_OR_Logic;
architecture LogicOperation of AND_OR_Logic is
component ANDgate is
port (A, B: in std_logic ; X: out std_logic) ;
end component ANDgate ;
component ORgate is
port (A,B: in std_logic ; X: out std_logic ) ;
end component ORgate ;
signal OUT1, OUT2: std_logic ;
begin
G1 : ANDgate port map(A=> IN1 ,B=>IN2 , X=>OUT1);
G2 : ANDgate port map(A=> IN3 ,B=>IN4 , X=>OUT2);
G3 : ORgate port map(A=> OUT1, B=>OUT2, X=>OUT3);
end architecture LogicOperation;
```

Additional Activities

Implement the circuit of XOR

Sukkur IBA University

Department of Computer Science

ESE-201: Digital Logic Design Lab

Lab #12 : Combinatorial Circuits in VHDL

Note: Complete the lab work and upload on e learning

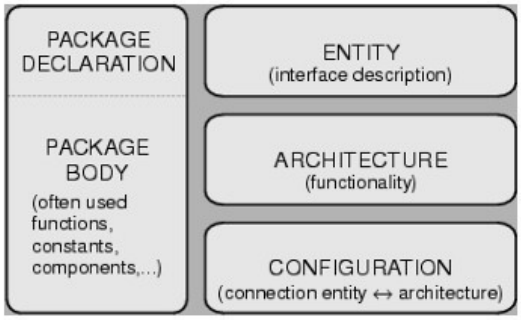
Lab Learning Objectives:

Upon successful completion of this experiment, the student will be able:

- To implement various combinatorial circuits using VHDL.
- Simulate and verify the behaviour of each circuit.

Background Theory:

A VHDL design begins with an **ENTITY** block that describes the interface for the design. The interface defines the input and output l1ogic signals of the circuit being designed. The **ARCHITECTURE** block describes the internal operation of the design. Within these blocks are numerous other functional blocks used to build the design elements of the logic circuit being created.



entity entity-name is
[port(interface-signal-declaration);]
end [entity] [entity-name];
architecture architecture-name of entity-name is [declarations]
begin
architecture body
end [architecture] [architecture-name];

ENTITY BLOCK

An entity block is the beginning building block of a VHDL design. Each design has only one entity block which describes the interface signals into and out of the design unit. The syntax for an entity declaration is:

entity entity_name is
port (signal_name,signal_name : mode type;
signal_name,signal_name : mode type);
end entity_name;

An entity block starts with the reserve word **entity** followed by the entity_name. Names and identifiers can contain letters, numbers, and the under score character, but must begin with an alpha character. Next is the reserved word **is** and then the port declarations. The indenting shown in the entity block syntax is used for documentation purposes only and is not required since VHDL is insensitive to white spaces. A single **PORT** declaration is used to declare the interface signals for the entity and to assign MODE and data TYPE to them. If more than one signal of the same type is declared, each identifier name is separated by a comma. Identifiers are followed by a colon (:), mode and data type selections.

The entity declaration is completed by using an **end** operator and the entity name. Optionally, you can also use an **end** entity statement. In VHDL, all statements are terminated by a **semicolon**.

Example declaration of SR Latch

```
entity latch is
port (s,r : in bit;
      q,nq : out bit);
end latch;
```

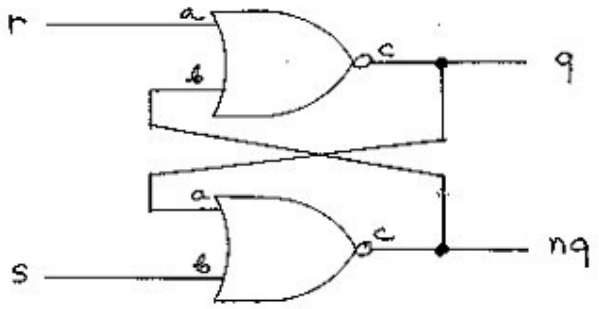
ARCHITECTURE BLOCK

The architecture block defines how the entity operates. This may be described in many ways, two of which are most prevalent: STRUCTURE and DATA FLOW or BEHAVIOR formats. The BEHAVIOR approach describes the actual logic behavior of the circuit. This is generally in the form of a Boolean expression or process. The STRUCTURE approach defines how the entity is structured - what logic devices make up the circuit or design. The general syntax for the architecture block is:

```
architecture arch_name of entity_name is
declarations;
begin
statements defining operation;
end arch_name;
```

We have described the entity of SR Latch, and now we construct the architecture according to circuit diagram

```
library ieee;
use ieee.std_logic_1164.all;
-- entity block
 entity latch is
-- interface signal declarations
     port (s,r : in std_logic;
           q,nq : out std_logic);
end latch;
-- architecture block
architecture flipflop of latch is
begin
-- assignment statements
     q <= r nor nq;
    nq <= s nor q;
end flipflop;
```



Lab Activity:

In this lab we will implement several combinatorial circuits and simulate them.

1. Full Adder Implementation

A single bit full-adder was implemented in the previous lab, for convenience the code is provided below

```
entity FullAdder is
port (A, B, CIN: in bit; SUM, COUT: out bit );
end entity FullAdder ;
architecture LogicOperation of FullAdder is
begin
SUM <= (A xor B) xor CIN;
COUT<= ( (A xor B) and CIN) or (A and B) ;
end architecture LogicOperation;
entity BitFullAdder is
port (A1,A2,A3,A4,B1,B2,B3,B4,C0: in bit ; S1,S2,S3,S4,C4: out bit );
end entity BitFullAdder;
architecture LogicOperation of BitFullAdder is
component FullAdder is
port (A, B, CIN: in bit ; SUM, COUT: out bit) ;
end component FullAdder;
signal X,Y,Z: bit;
begin
FA1 : FullAdder port map (A => A1 , B => B1 , CIN => C0 , SUM => S1 , COUT => X);
FA2 : FullAdder port map (A => A2 , B => B2 , CIN => X , SUM => S2 , COUT => Y);
FA3 : FullAdder port map (A => A3 , B => B3 , CIN => Y , SUM => S3 , COUT => Z);
FA4 : FullAdder port map (A => A4 , B => B4 , CIN => Z , SUM => S4 , COUT => C4);
end architecture LogicOperation;
```

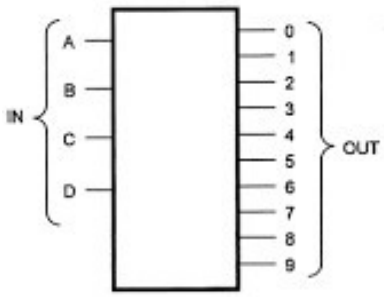
2. 4-Bit Comparator

```
entity 4BitComparator is
port (A0 , A1 , A2 , A3 , B0 , B1 , B2 , B3 : in bit ; AequalB : out bit);
end entity 4BitComparator ;
architecture LogicOperation of 4BitComparator is
begin
AequalB <= (A0 xnor B0) and (A1 xnor B1) and (A2 xnor B2) and (A3 xnor B) ;
end architecture LogicOperation ;
```

3. BCD Decoder

The block diagram of a BCD to Decimal decoder is illustrated below, simulate the code and verify the results

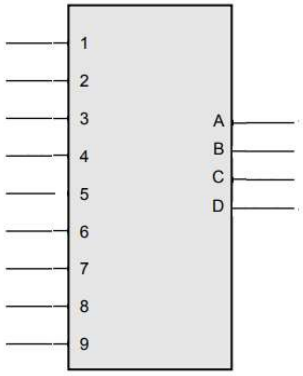
```
entity BCDdecoder is
port (A0 , A1 , A2 , A3 : in bit ; OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7, OUT8, OUT9: out bit ) ;
end entity BCDdecoder ;
architecture LogicOperation of BCDdecoder is
begin
OUT0 <= not ( not A0 and not A1 and not A2 and not A3) ;
OUT1 <= not (A0 and not A1 and not A2 and not A3) ;
OUT2 <= not ( not A0 and A1 and not A2 and not A3) ;
OUT3 <= not (A0 and A1 and not A2 and not A3) ;
OUT4 <= not ( not A0 and not A1 and A2 and not A3) ;
OUT5 <= not (A0 and not A1 and A2 and not A3) ;
OUT6 <= not ( not A0 and A1 and A2 and not A3) ;
OUT7 <= not (A0 and A1 and A2 and not A3) ;
OUT8 <= not ( not A0 and not A1 and not A2 and A3) ;
OUT9 <= not (A0 and not A1 and not A2 and A3) ;
end architecture LogicOperation ;
```



4. Decimal to BCD Encoder

The block diagram of the circuit

```
entity DecBCDencoder is
port (D1, D2, D3, D4, D5, D6, D7, D8, D9 :in bit ; A0, A1, A2, A3 : out bit ) ;
end entity DecBCDencoder ;
architecture LogicFunction of DecBCDencoder is
begin
A0 <= (D1 or D3 or D5 or D7 or D9) ;
A1 <= (D2 or D3 or D6 or D7) ;
A2 <= (D4 or D5 or D6 or D7) ;
A3 <= (D8 or D9) ;
end architecture LogicFunction;
```



5. 8:1 Multiplexor

```
entity EightInputMUX is
port (S0, S1, S2, D0, D1, D2, D3, D4, D5, D6, D7,
EN: in bit; Y: inout bit; YI: out bit);
end entity EightInputMUX;

architecture LogicOperation of EightInputMUX is
signal AND0, AND1, AND2, AND3, AND4, AND5, AND6, AND7: bit;
begin
AND0 <= not S0 and not S1 and not S2 and D0 and not EN;
AND1 <= S0 and not S1 and not S2 and D1 and not EN;
AND2 <= not S0 and S1 and not S2 and D2 and not EN;
AND3 <= S0 and S1 and not S2 and D3 and not EN;
AND4 <= not S0 and not S1 and S2 and D4 and not EN;
AND5 <= S0 and not S1 and S2 and D5 and not EN;
AND6 <= not S0 and S1 and S2 and D6 and not EN;
AND7 <= S0 and S1 and S2 and D7 and not EN;
Y <= AND0 or AND1 or AND2 or AND3 or AND4 or AND5 or AND6 or AND7;
YI <= not Y;
end architecture LogicOperation;
```