TPCT's
College of Engineering, Osmanabad

Laboratory Manual

Digital Electronics

For

Second Year Students

Manual Prepared by

Prof.R.B.Randive

Author COE, Osmanabad

TPCT's

College of Engineering, Solapur Road,
Osmanabad

Department of Computer Science & Engineering

Vision of the Department:

To achieve and evolve as a center of academic excellence and research center in the field of Computer Science and Engineering. To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society

Mission of the Department:

The department strives to continuously engage in providing the students with in-depth understanding of fundamentals and practical training related to professional skills and their applications through effective Teaching- Learning Process and state of the art laboratories pertaining to CSE and interdisciplinary areas. Preparing students in developing research, design, entrepreneurial skills and employability capabilities

College of Engineering

Technical Document

This technical document is a series of Laboratory manuals of Computer Science & Engineering Department and is a certified document of College of engineering, Osmanabad. The care has been taken to make the document error-free. But still if any error is found, kindly bring it to the notice of subject teacher and HOD.

Recommended by,

HOD

Approved by,

Principal

FOREWORD

It is my great pleasure to present this laboratory manual for second year engineering students for the subject of Digital Electronics to understand and visualize the basic concepts of various circuits using electronic components. Digital Electronics cover basic concepts of electronics. This being a core subject, it becomes very essential to have clear theoretical and numerical aspects.

This lab manual provides a platform to the students for understanding the basic concepts of decimal to binary conversion and vice-versa, different types of gates with their IC circuit diagram and connection. This practical background will help students to gain confidence in qualitative and quantitative approach to electronic circuits.

H.O.D
CSE Dept

LABORATORY MANUAL CONTENTS

This manual is intended for the Second Year students of CSE branches in the subject of Digital Electronics. This manual typically contains practical/ Lab Sessions related to Digital Electronics covering various aspects related to the subject for enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

SUBJECT INDEX:

1. Do's & Don'ts in Laboratory.

2. Lab Exercises
    1.
    2.
        .
        .
        .
        .

3. Quiz
4. Conduction of viva voce examination
5. Evaluation & marking scheme

Dos and Don'ts in Laboratory :-

1. Do not handle any equipment before reading the instructions /Instruction manuals.
2. Read carefully the power ratings of the equipment before it is switched ON, whether ratings 230 V/50 Hz or 115V/60 Hz. For Indian equipment, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50Hz., which will damage the equipment.
3. Observe type of sockets of equipment power to avoid mechanical damage.
4. Do not forcefully place connectors to avoid the damage.
5. Strictly observe the instructions given by the Teacher/ Lab Instructor.

Instruction for Laboratory Teachers:-

1. Submission related to whatever lab work has been completed should be done during the next lab session.
2. Students should be instructed to switch on the power supply after getting the checked by the lab assistant / teacher. After the experiment is over, the students must hand over the circuit board, wires, CRO probe to the lab assistant/teacher.
3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

2. Lab Exercises

1. Study logic gates IC's & verify the Truth Table of logic gates .
2. Study & verification of operation of Half Adder & Full Adder.
3. Study & verification of operation of Half Adder & Full Substractor.
4. Study & verification of operation of Binary to Gray converter.
5. Study & verification of operation of Gray to Binary converter.
6. Simulation of verilog module for Boolean Expression.
7. Simulation of verilog module for half adder.
8. Simulation of verilog module for full adder.
9. Simulation of verilog module for 4:1 multiplexer.
10. Simulation of verilog module for 8:1 multiplexer.

EXPERIMENT NO: 01

STUDY OF LOGIC GATES

AIM : Study logic gates IC's & verify the Truth Table of logic gates .

OBJECTIVE:- Verifying truth table of different gates.

EQUIPMENTS : Power supply, Connecting wires, Logic Trainer Kit .

COMPONENTS : IC 7408, IC 7432, IC 7404, IC 7400, IC 7402, IC 7486, Bread board.

THEORY :

In a digital system, **basic logic gates** used are

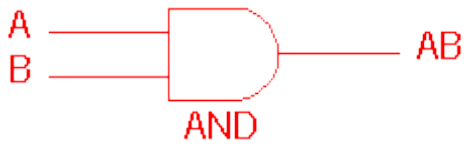
1) AND gate
2) OR gate
3) NOT gate

Universal gates

1) NAND gate
2) NOR gate
3) EX-OR gate

Basic Logic gates:

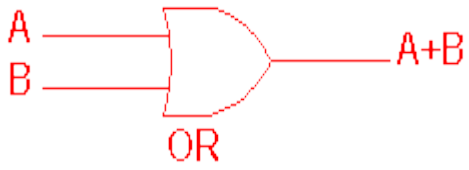
1) **AND gate** : The output of an AND gate is 1 if &only if all the inputs are Logic high level.



TRUTH TABLE

Input		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

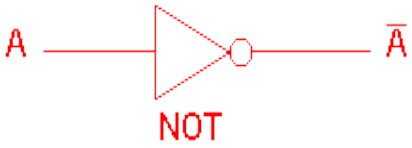
2) OR gate : The output of an OR gate is 1 if &only if one or more inputs are 1.



TRUTH TABLE

Input		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

3) NOT gate: The output of an NOT gate is complement / Inversion of input.

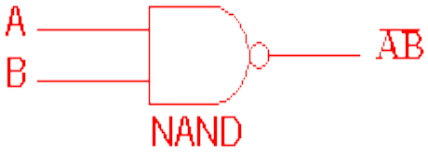


TRUTH TABLE

Input	Output
A	Y=A
0	1
1	0

<u>Universal logic gates:</u>

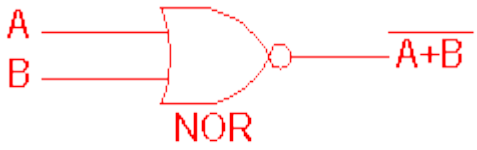
1) NAND gate : The NOT-AND gate operation is known as NAND operation



TRUTH TABLE

Input		Output
A	B	Y=A.B
0	0	1
0	1	1
1	0	1
1	1	0

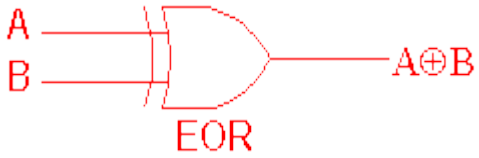
2) **NOR gate**: The NOT-OR operation is known as NOR operation.



TRUTH TABLE

Input		Output
A	B	Y=A+B
0	0	1
0	1	0
1	0	0
1	1	0

3) **EX-OR gate**: The EX-OR operation is special gate output is 1 when inputs are opposite level.



TRUTH TABLE

Input		Output
A	B	Y=AB+AB
0	0	0
0	1	1
1	0	1
1	1	0

PROCEDURE:

1) Connect the IC according to pin configuration shown in fig with logic trainer kit.
2) Apply the input to the appropriate input terminals & check the output pin (LED will glow for logic 1 output & LED will not glow for logic 1 level)
3) Repeat the step 2 for different logic gates IC's.
4) Verify the output with truth table.

OBSERVATION:

As per truth table we have checked the outputs of different gates & verified according with truth table.

RESULT:

Hence we verify the truth table of various logic gate IC's.

CONCLUSION: By analyzing the performance of logical gates, they can be implemented in different digital applications

EXPERIMENT NO: 02

OPERATION OF ADDER

AIM: Study & verification of operation of Half Adder & Full Adder.

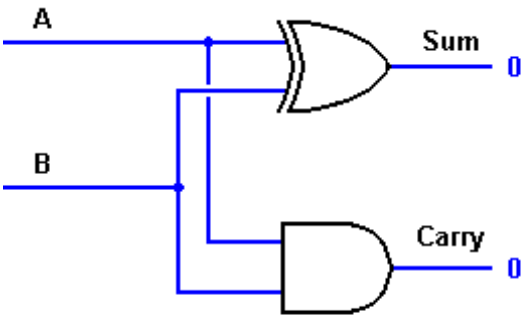
OBJECTIVE : To Design the half & full adder circuit & analyze the result of Half & Full Adder.

EQUIPMENTS: Digital trainer kit, Bread board, etc.

COMPONENTS: IC –7486, 7400, connecting wires, etc.

CIRCUIT DIAGRAM:

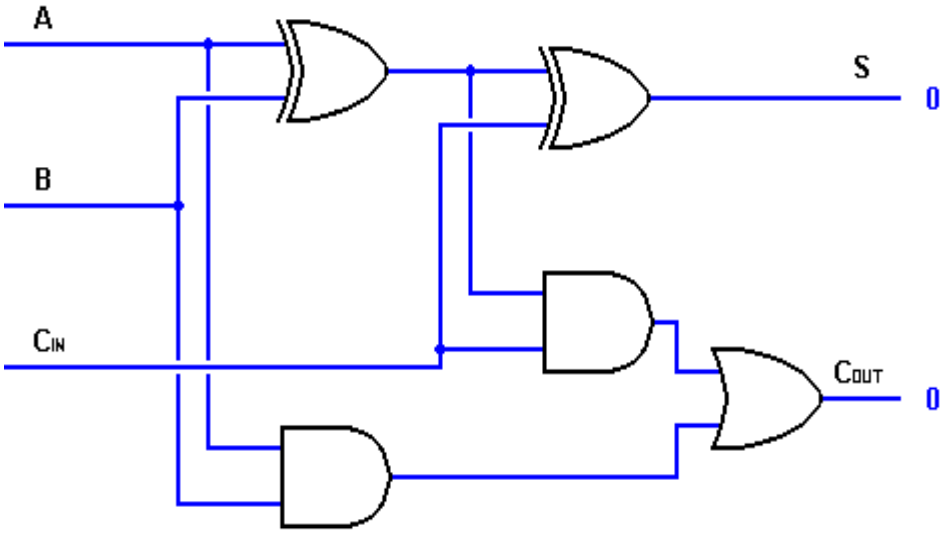
HALF ADDER:



TRUTH TABLE - 1

Inputs		Outputs	
A	B	Sum	Carry Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

FULL ADDER:



TRUTH TABLE – 2

Inputs			Outputs	
A	B	Carry In	Sum	Carry Out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

THEORY :

1.HALF –ADDER

1) Half Adder is logic circuit which can add two binary digits at a time.
2) It consists of an exclusive OR gate & AND gate.
3) The output of EX-OR gate is the sum & output of AND gate is carry

2) FULL ADDER

1) Full Adder is logic circuit which can add three binary digits at a time.
2) It consists of two Half Adders (i.e. 2 AND gates &2 Ex-OR gates) & OR gate.

PROCEDURE:

1. Wire the circuit of Half Adder using SN7400 IC's
2. Enter the four input level combinations given in table -1 & Record the resultant value of the Sum & Carry Out.
3. Construct the Full Adder as shown in fig. using three SN7400 IC's
4. Enter the level combination of A,B & Carry In as given in table 2 & record the resultant Sum & Carry Outputs.

OBSERVATION :

As per truth table we verified the output of Half –Adder & Full-Adder.

RESULT:

We have designed Half –Adder & Full-Adder ckt's & verified their outputs with truth table.

CONCLUSION:

By analyzing the performance of Half –Adder & Full-Adder & they can be used as arithmetic ckt's in digital applications.

EXPERIMENT NO: 03

OPERATION OF SUBSTRACTOR.

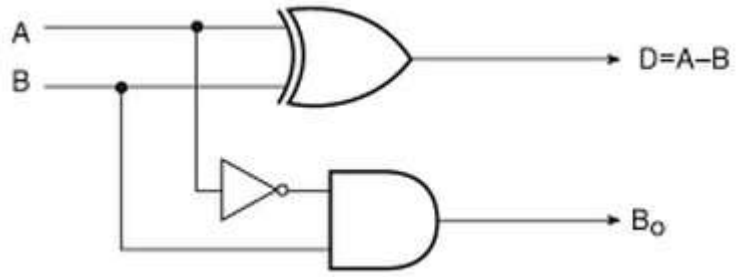
AIM: Study and verification the half substractor & full substractor circuit.

EQUIPMENTS: Digital trainer kit, Bread board, etc.

COMPONENTS: IC –7486, 7400, connecting wires, etc.

CIRCUIT DIAGRAM:

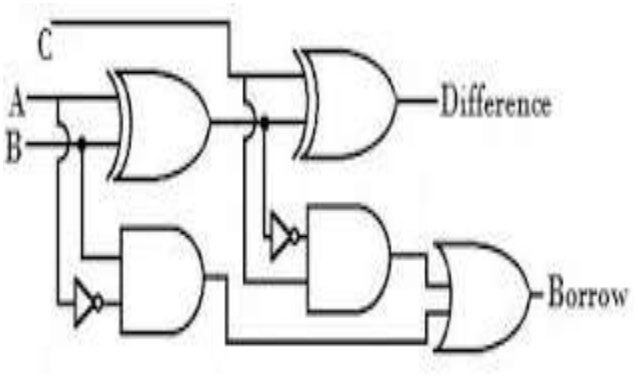
HALF SUBSTRACTOR :



TRUTH TABLE 1

Inputs		Outputs	
A	B	Difference	Barrow Out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

FULL SUBSTRACTOR:



TRUTH TABLE 2

Inputs			Outputs	
A	**B**	**Borrow In**	**Difference**	**Borrow Out**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

THEORY:

HALF-SUBTRACTOR

Half-subtractor is used to subtract one binary digit from another to give DIFFERENCE output and a BORROW output. The truth table of a half-subtractor is shown in figure.

FULL SUBTRACTOR

Full subtractor performs subtraction of two bits, one is minuend and other is subtrahend. In full subtractor '1' is borrowed by the previous adjacent lower minuend bit. Hence there are three bits are considered at the input of a full subtractor. There are two outputs, that are DIFFERENCE output D and BORROW output Bo. The BORROW output indicates that the minuend bit requires borrow '1' from the next minuend bit. Figure shows the truth table of a full subtractor.

PROCEDURE:

1. Wire the circuit of Half Substractor using SN7400 &7486 IC's as shown in fig.
2. Enter the four input level combinations given in table -2 & record the resultant value of the Difference & Borrow Outs.
3. Construct the Full Substractor shown in fig. using SN740 IC for NAND gates & an IC SN7486 as source of EX-OR gates.
4. Enter the combination of A,B & Borrow In given in table-4 & record the resultant Difference & Borrow Outs.

RESULT :

We have designed Half – Substractor & Full- Substractor circuit's & verified their outputs with truth table.

CONCLUSION :

By analyzing the performance of Half – Substractor circuit's & Full- Substractor circuit's they can be used as arithmetic circuit's in digital applications.

EXPERIMENT NO: 04

BINARY TO GRAY CODE CONVERTER.

AIM : Study & verification of operation of Binary to Gray converter.

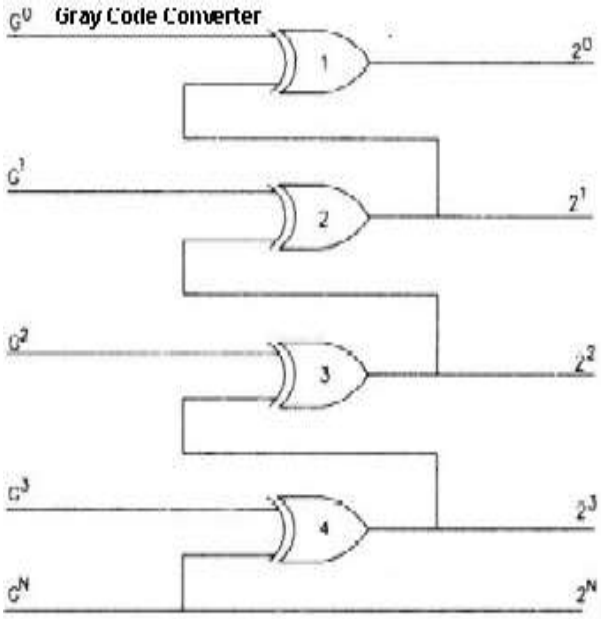
OBJECTIVE : To minimize the K-map for binary to gray code converter & find out the logic equation.

COMPONENTS :, IC 7486, connecting wires, etc.

EQUIPMENTS : Trainer Kit , Bread board.

CIRCUIT DIAGRAM :

BINARY TO GRAY CODE CONVERTER



TRUTH TABLE :

B_0	B_1	B_2	B_3	G_0	G_1	G_2	G_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

THEORY:

IC circuit used to convert 4-bit Gray code to Binary using Exclusive OR [XOR] gates. The schematic shows a 4-bit converter circuit, but more XOR gates may be added to produce a converter of any number of bits. A 7486 Quadruple 2-Input Exclusive-OR Gate [74xx86] could be used to implement the Gray code circuit above. There are no Gray code specific converter ICs.

A Binary to Gray Code converter can also be made using the same 7486 XOR gates. Instead of feeding the input of the XOR gate from the output of the lower one use its input line. For example XOR-3 has an input tied to output line 2sup3 of XOR-4, tie that input line to Gsup3 instead. So XOR gate 3 would have an input G2 and G3.

PROCEDURE :

1. Set –up the circuit of a Binary to Gray code converter using IC – 7486 Ex-OR gate.
2. Apply 4 bit binary inputs 0000 through 1111.
3. Observe the output in each case as shown in table & verify the result.

OBSERVATION :

As per truth table we have checked the output for 4 bit binary inputs.

RESULT :

Hence ,we convert the Binary code to Gray code.

CONCLUSION :

By observing the truth table of gray code we found that the successive number is change by only one bit. They can be used to design different ckts.

EXPERIMENT NO: 05

GRAY TO BINARY CODE CONVERTER

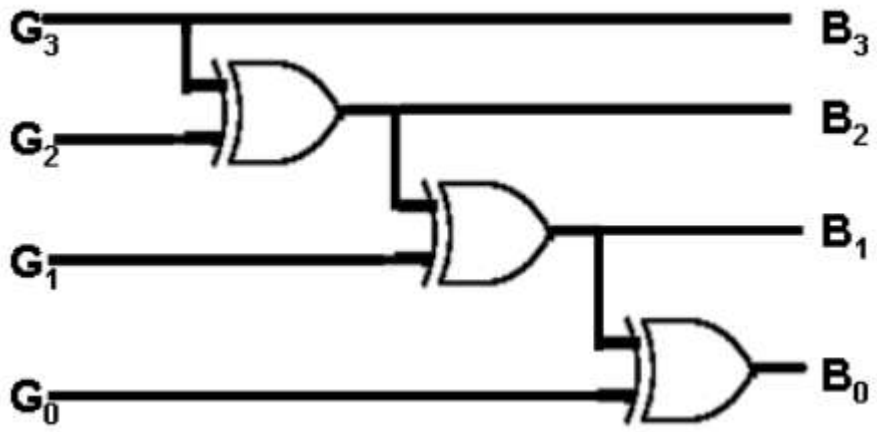
AIM : Study & verification of operation of Gray to Binary converter.

OBJECTIVE : To minimize the K-map for Gray to binary code converter & find out the logic equation.

COMPONENTS :, IC 7486, connecting wires, etc.

EQUIPMENTS : Trainer Kit , Bread board.

CIRCUIT DIAGRAM :



THEORY :

To convert 4-bit gray to binary code in which only one bit changes between successive numbers. .To convert 4-bit gray code into the binary code .Write MSB of gray number as it is then add MSB of binary to the successive bit of gray number write sum & ignore carry. Continue the process until LSB reached.

TRUTH TABLE :

G_0	G_1	G2	G3	B_0	B_1	B_2	B_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

PROCEDURE:

1. Set –up the circuit of a Gray code to binary code converter using IC – 7486 Ex-OR gate.
2. Apply 4 bit binary inputs 0000 through 1111.
3. Observe the output in each case as shown in table & verify the result.

OBSERVATION:

As per truth table we have verified the output for 4 bit gray inputs.

RESULT : Hence, we convert the Gray code to Binary code .

CONCLUSION:

By observing the truth table of binary code we found that the successive number is change by only one bit. They can be used to design different circuit.

EXPERIMENT NO: 06

Verilog module for Boolean Expression

AIM: Simulation of verilog module for Boolean Expression.

OBJECTIVE : To verify verilog module by Test Bench using XILINK.

PROCEDURE:

Verilog Module:-

```
module fourvariable (A,B,C,D,X,Y)
input A;
input B;
input C;
input D;
output X;
output Y;

wire and _op1,or_op2;
and g1(and _op1,A,B);
or g2(or_op2,B,D);
nor g3(X,C,and_op1);
nand g4(Y,C,or_op2);
endmodule
```

Test Bench:-

```
Module fourvariable_test_fourvariable_v_tf();
//DATE:10:32:04 06/22/2018
//MODULE:  fourvariable
//DESIGN: fourvariable
//FILENAME:test_fourvariable.v
//PROJECT: booleanexpression
//VERSION:
//Inputs
reg A;
reg B;
reg C;
reg D;

//OUTPUTS
wire x;
wire Y;

//Bidirs


//Instantiate the UUT
```

```
fourvariable uut(
.A(A),
.B(B),
.C(C),
.D(D),
.X(X),
Y(Y)
);
//Initialize Inputs
ifdef auto_init

initial begin

A=0;  B=0;  C=0;  D=0;
ABCD=1'b0000;#15; ABCD=1'b0001;#15; ABCD=1'b0010;#15;

ABCD=1'b0011;#15; ABCD=1'b0100;#15; ABCD=1'b0101;#15;

ABCD=1'b0110;#15; ABCD=1'b0111;#15; ABCD=1'b1000;#15;

ABCD=1'b1001;#15; ABCD=1'b1010;#15; ABCD=1'b1011;#15;

ABCD=1'b1100;#15; ABCD=1'b1101;#15; ABCD=1'b1110;#15;

ABCD=1'b1111;#15;
end

endif

Endmodule
```

THEORY:-

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL.
HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits.

Verilog can be used to describe designs at four levels of abstraction:
(i) Algorithmic level (much like c code with if, case and loop statements).
(ii) Register transfer level (RTL uses registers connected by Boolean equations).
(iii) Gate level (interconnected AND, NOR etc.).
(iv) Switch level (the switches are MOS transistors inside gates).
The language also defines constructs that can be used to control the input and output of simulation.

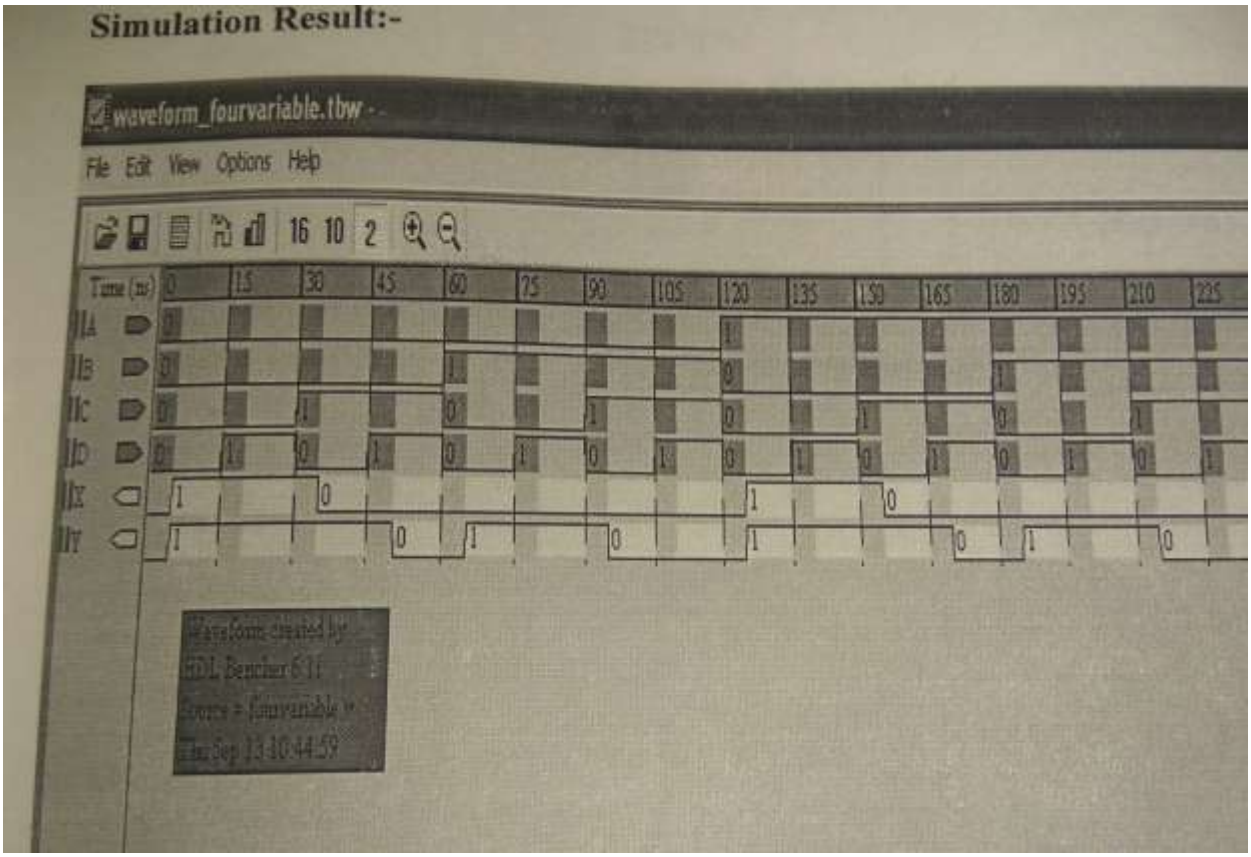
More recently Verilog is used as an input for synthesis programs which will generate a gate-level description (a netlist) for the circuit. Some Verilog constructs are not synthesizable. Also the way the code is written will greatly effect the size and speed of the synthesized circuit. Most readers will want to synthesize their circuits, so nonsynthe- sizable constructs should be used only for test benches. These are program modules used to generate I/O needed to simulate the rest of the design. The words “not synthesizable” will be used for examples and constructs as needed that do not synthesize.

There are two types of code in most HDLs:
Structural, which is a verbal wiring diagram without storage. assign a=b & c | d; /* “|” is a OR */
assign d = e & (~c);
Here the order of the statements does not matter. Changing e will change a.
Procedural which is used for circuits with storage, or as a convenient way to write conditional logic. always @(posedge clk) // Execute the next statement on every rising clock edge.
count <= count+1;
Procedural code is written like c code and assumes every assignment is stored in memory until over written. For syn- thesis, with flip-flop storage, this type of thinking generates too much storage. However people prefer procedural code because it is usually much easier to write, for example, if and case statements are only allowed in procedural code. As a result, the synthesizers have been constructed which can recognize certain styles of procedural code as actually combinational. They generate a flip-flop only for left-hand variables which truly need to be stored. However if you stray from this style, beware. Your synthesis will start to fill with superfluous latches.

This manual introduces the basic and most common Verilog behavioral and gate-level modelling constructs, as well as Verilog compiler directives and system functions. Full description of the language can be found in Cadence Verilog-XL Reference Manual and Synopsys HDL Compiler for Verilog Reference Manual. The latter emphasizes only those Verilog constructs that are supported for synthesis by the Synopsys Design Compiler synthesis tool.

In all examples, Verilog keyword are shown in boldface. Comments are shown in italics.

OBSERVATION:



CONCLUSION:- Thus by observing test bench waveform for four variable we have studied working of verilog module for Boolean Expression.

EXPERIMENT NO: 07
Verilog module for half adder

AIM: Simulation of verilog module for half adder.

OBJECTIVE : To verify verilog module by Test Bench using XILINK.

PROCEDURE:

Verilog Module:-

```
module halfaddr(A,B,sum,carry);
input A;
input B;
output sum;
output carry;
 assign sum=A^B;
assign carry=A&B;

endmodule
```

Test bench:-

```
module halfaddr_test_halfaddr_v_tf();
//DATE:11:05:31 06/22/2018
//MODULE:  halfaddr
//DESIGN: halfaddr
//FILENAME: test_halfaddr.v
//PROJECT: halfaddr
//VERSION:
//Inputs
reg A;
reg B;

//OUTPUTS
wire sum;
wire carry;

//Bidirs

//Instantiate the UUT
halfaddr uut(
.A(A),
.B(B),
.sum(sum),
.carry(carry),
);

//Initialize Inputs
```

ifdef auto_init

initial begin

A=0; B=0;

AB=2'b00;#15; ABCD=2'b01;#15;

AB=2'b10;#15; ABCD=2'b11;#15;

end

endif

endmodule

THEORY:- Verilog source text files consists of the following lexical tokens:

White Space

White spaces separate words and can contain spaces, tabs, new-lines and form feeds. Thus a statement can extend over multiple lines without special continuation characters.

Comments

Comments can be specified in two ways (exactly the same way as in C/C++):

- Begin the comment with double slashes (**//**). All text between these characters and the end of the line will be ignored by the Verilog compiler.
- Enclose comments between the characters **/*** and ***/**. Using this method allows you to continue comments on more than one line. This is good for "commenting out" many lines code, or for very brief in-line comments.

Example 2 .1

```
a = c + d;              // this is a simple
/* however, this comment continues on
   more than one line */
assign y = temp_reg;
assign x=ABC /* plus its compliment*/ + ABC
```

Numbers

Number storage is defined as a number of bits, but values can be specified in binary, octal, decimal or hexadecimal (See Sect. 6.1. for details on number notation). Examples are 3'**b**001, *a 3-bit numbe*r, 5'd30, (*=5'b11110*), and 16'**h**5ED4, (*=16'd24276*)

Identifiers

Identifiers are user-defined words for variables, function names, module names, block

names and instance names. Identifiers begin with a letter or underscore (Not with a number or $) and can include any number of letters, digits and underscores. Identifiers in Verilog are case-sensitive.

Syntax

allowed symbols

ABCDE . . . abcdef. . . 1234567890 _$

not allowed: anything else especially

- & # @

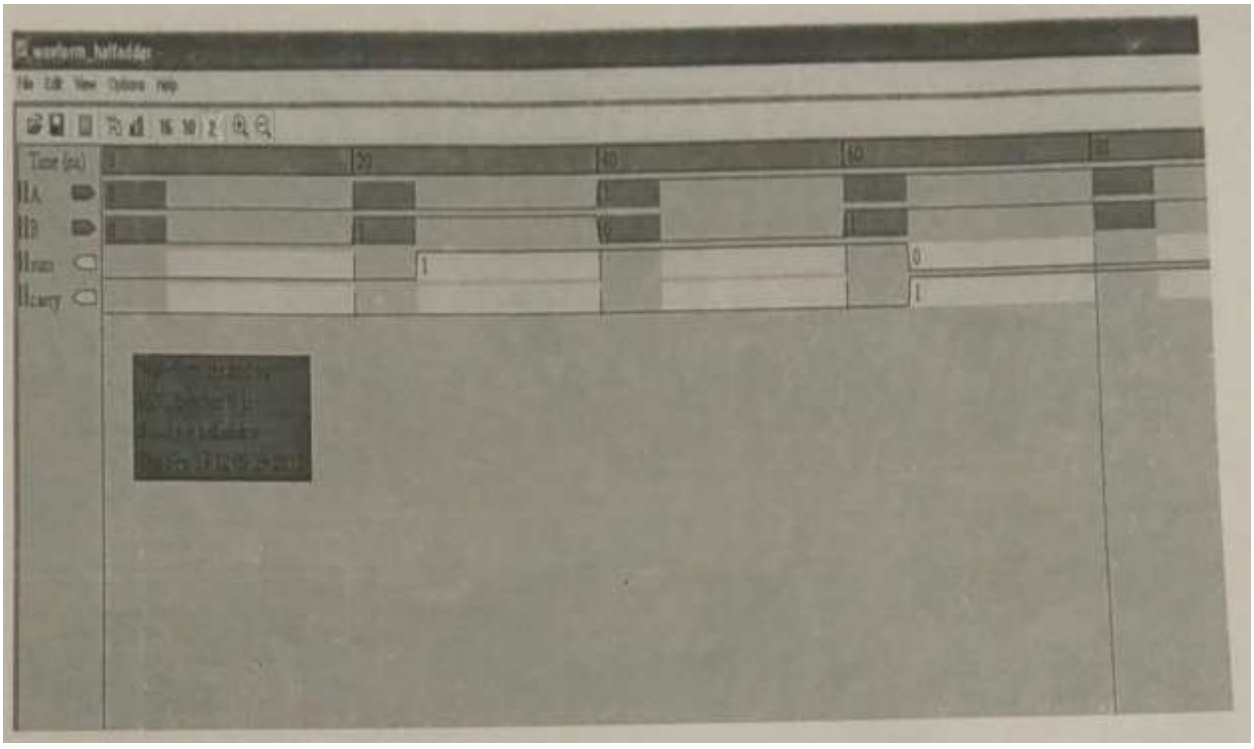
Operators

Operators are one, two and sometimes three characters used to perform operations on variables. Examples include >, **+**, **~, &**, **!=.** Operators are described in detail in “Operators” on p. 6.

Verilog Keywords

These are words that have special meaning in Verilog. Some examples are **assign, case, while, wire, reg, and, or, nand,** and **module**. They should not be used as identifiers. Refer to *Cadence Verilog-XL Reference Manual* for a complete listing of Verilog keywords. A number of them will be introduced in this manual. Verilog keywords also includes Compiler Directives (Sect. 15.) and System Tasks and Functions

OBSERVATION:



CONCLUSION:- Thus by observing test bench waveform for half adder we have studied working of verilog module for half adder.

EXPERIMENT NO: 08
Verilog module for full adder

AIM: Simulation of verilog module for full adder.

OBJECTIVE : To verify verilog module by Test Bench using XILINK.

PROCEDURE:

Verilog Module:-

```
module fulladdr(A,B,Cin,sum,carry);
input A;
input B;
input Cin;
output sum;
output carry;
 assign sum=A^B^Cin;
assign carry=(A&B)|(B&C)|(A&C);
endmodule
```

Test bench:-

```
module fulladdr_test_fulladdr_v_tf();
//DATE:11:05:31 06/22/2018
//MODULE:  fulladdr
//DESIGN: fulladdr
//FILENAME: test_fulladdr.v
//PROJECT: fulladdr
//VERSION:

//Inputs
reg A;
reg B;
reg Cin;

//OUTPUTS
wire sum;
wire carry;

//Bidirs

//Instantiate the UUT
halfaddr uut(
.A(A),
.B(B),
.Cin(Cin),
.sum(sum),
.carry(carry),
);
//Initialize Inputs
```

```
ifdef auto_init

initial begin

A=0;  B=0; Cin=0;

ABC=3'b000;#10; ABC=3'b001;#10;

ABC=3'b010;#10; ABC=3'b011;#10;

ABC=3'b100;#10; ABC=3'b101;#10;

ABC=3'b110;#10; ABC=3'b111;#10;

end

endif

endmodule
```

THEORY:-

Value Set

Verilog consists of only four basic values. Almost all Verilog data types store all these values:

0 (logic zero, or false condition)

1 (logic one, or true condition)

x (unknown logic value) x and z have limited use for synthesis.

z (high impedance state)

Wire

A **wire** represents a physical wire in a circuit and is used to connect gates or modules. The value of a **wire** can be read, but not assigned to, in a function or block. See "Functions" on p. 19, and "Procedures: Always and Initial Blocks" on p. 18. A **wire** does not store its value but must be driven by a continuous assignment statement or by con- necting it to the output of a gate or module. Other specific types of wires include:

wand (wired-AND);:the value of a wand depend on logical AND of all the drivers connected to it.

wor (wired-OR);: the value of a wor depend on logical OR of all the drivers connected to it.

tri (three-state;): all drivers connected to a tri must be z, except one (which determines the value of the tri).

Syntax

wire [msb:lsb] wire_variable_list; **wand** [msb:lsb] wand_variable_list; **wor**

Example 4 .1

```
wire  c            // simple
wand d;
assign d = a;      // value of d is the logical AND of
assign d = b;      // a and b
wire [9:0] A:      // a cable (vector) of 10 wires.
```

[msb:lsb] wor_variable_list;
tri [msb:lsb]
tri_variable_list;

Reg

Declare type **reg** for all data objects on the left hand side of expressions in **inital** and **always** procedures, or **func- tions**. See "Procedural Assignments" on page 12. A **reg** is the data type that must be used for latches, flip-flops and memorys. However it often synthesizes into leads rather than storage. In multi-bit registers, data is stored as *unsigned* numbers and no sign extension is done for what the user might have thought were two's complement numbers.

Syntax

reg [msb:lsb] reg_variable_list;

Example 4 .2

```
reg a;          // single 1-bit register variable
reg [7:0] tom;  // an 8-bit vector; a bank of 8 registers.
reg [5:0] b, c; // two 6-bit variables
```

Input, Output, Inout

These keywords declare input, output and bidirectional ports of a **module** or **task**. Input and inout ports are of type
wire. An output port can be configured to be of type **wire**, **reg**, **wand**, **wor** or **tri**. The default is **wire**.

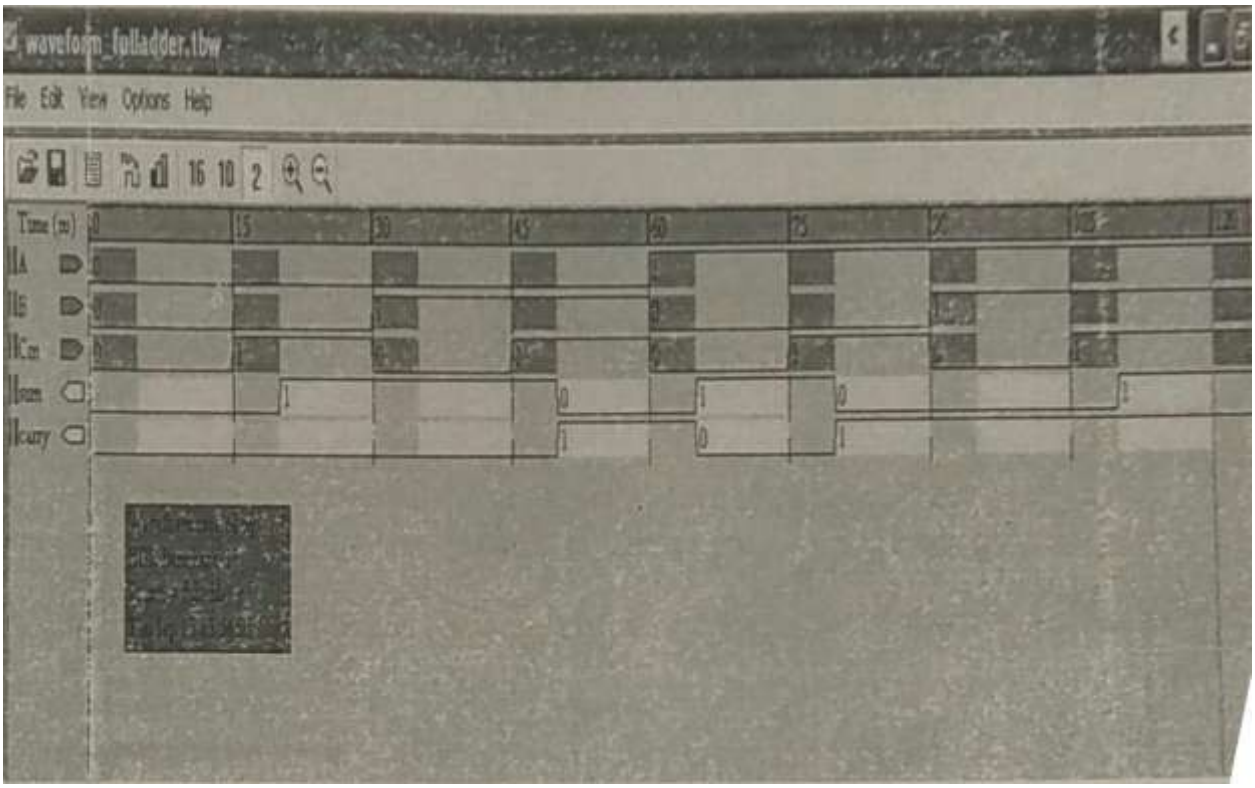
Syntax

input [msb:lsb] input_port_list; **output** [msb:lsb] output_port_list;
inout [msb:lsb] inout_port_list;

Example 4 .3

```
module sample(b, e, c, a); //See "Module Instantiations" on p. 10
  input a;          // An input which defaults to wire.
  output b, e;      // Two outputs which default to wire
  output [1:0] c;   /* A two-it output. One must declare
  its
          type in a separate statement. */
```

OBSERVATION:



CONCLUSION:- Thus by observing test bench waveform for full adder we have studied working of verilog module for full adder.

EXPERIMENT NO: 09
Verilog module for 4:1 multiplexer

AIM: Simulation of verilog module for 4:1 multiplexer.

OBJECTIVE : To verify verilog module by Test Bench using XILINK.

PROCEDURE:

Verilog Module:-

```
module mux4(I0,I1,I2,I3,S0,S1,Y);
input I0;
input I1;
input I2;
input I3;
input S0;
input S1;
output Y;reg Y;
assign Y=S[1]?(S[0]?I3:I2): (S[0]?I1:I0);
endmodule
```

Test bench:-

```
module mux4_test_mux4_v_tf();
//DATE:11:05:31 06/22/2018
//MODULE:mux4
//DESIGN: mux4
//FILENAME: test_mux4.v
//PROJECT: multiplexer
//VERSION:

//Inputs
reg I0;
reg I1;
reg I2;
reg I3;
reg S0;
reg S1;


//OUTPUTS
wire Y;
```

```
//Bidirs

//Instantiate the UUT
mux4 uut(
.I0(I0),
.I1(I1),
.I2(I2),
.I3(I3),
.S0(S0),
.S1(S1),
.Y(Y)
);

//Initialize Inputs
ifdef auto_init

initial begin

I0-0;
I1=1;
I2=1;
I3=0;
S0S1=2'b00;#10; S0S1=2'b00;#10;

S0S1=2'b10;#10; S0S1=2'b11;#10;
end
endif
endmodule
```

THEORY:-

Integer

Integers are general-purpose variables. For synthesois they are used mainly loops-indicies, parameters, and con- stants. See"Parameter" on p. 5. They are of implicitly of type **reg**. However they store data as signed numbers whereas explicitly declared **reg** types store them as unsigned. If they hold numbers which are not defined at compile time, their size will default to 32-bits. If they hold constants, the synthesizer adjusts them to the minimum width needed at compilation.

Example 4 .4

```
integer a;          // single 32-bit integer
assign b=63;        // 63 defaults to a 7-bit
```

Syntax

integer integer_variable_list;
... integer_constant ... ;

Supply0, Supply1

Supply0 and **supply1** define wires tied to logic 0 (ground) and logic 1 (power), respectively.

Syntax

supply0 logic_0_wires;
supply1 logic_1_wires;

Example 4 .5

```
supply0  my_gnd;      // equivalent to a wire assigned 0
supply1 a, b;
```

Time

Time is a 64-bit quantity that can be used in conjunction with the **$time** system task to hold simulation time. Time is not supported for synthesis and hence is used only for simulation purposes.

Syntax

time time_variable_list;

Example 4 .6

```
time c;
c = $time;          // c = current simulation time
```

Parameter

Parameters allows constants like word length to be defined symbolically in one place. This makes it easy to change the word length later, by change only the parameter. See also “Parameterized Modules” on page 11 . An alternative way to do the same thing is to use macro substitution, see “Macro Definitions” on page 26.

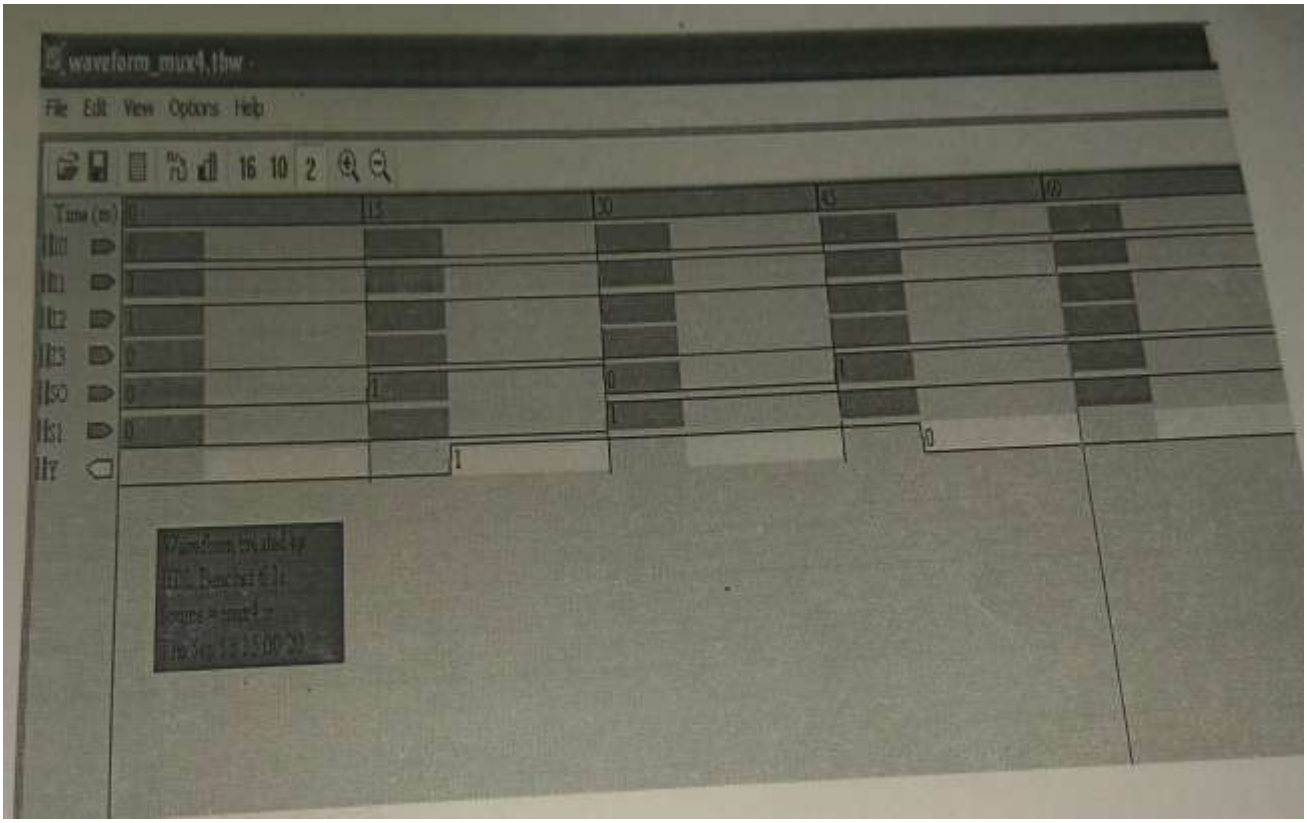
Syntax

parameter par_1 = *value*,
par_2 = *value*, ...;
parameter [range] parm_3 = valu

Example 4 .7

```
parameter add = 2'b00, sub = 3'b111;
parameter n = 4;
parameter [3:0] st4 = 4'b1010;
. . .
reg [n-1:0] harry; /* A 4-bit register whose length is
                      set by parameter n above. */
always @(x)
   y = {{(add - sub){x}}; // The replication operator Sect. 5.8.
   if (x) begin
          state = st4[1]; else state = st4[2];
        end
```

OBSERVATION:



CONCLUSION:- Thus by observing test bench waveform for mux4 we have studied working of verilog module for 4:1 multiplexer.

EXPERIMENT NO: 10
Verilog module for 8:1 multiplexer

AIM: Simulation of verilog module for 8:1 multiplexer.

OBJECTIVE : To verify verilog module by Test Bench using XILINK.

PROCEDURE:

Verilog Module:-

```
module mux8(I0,I1,I2,I3,I4,I5,I6,I7,S0,S1,S2,Y);
input (I0,I1,I2,I3,I4,I5,I6,I7);
input S0;
input S1;
input S2;
output Y;reg Y;
always@(I0 or I1 or I2 or I3 or I4 or I5 or I6 or I7)
if(s2s1s0==3'b000)
Y=I0;
else if(s2s1s0==3'b000)
Y=I0;
else if(s2s1s0==3'b000)
Y=I1;
else if(s2s1s0==3'b000)
Y=I2;
else if(s2s1s0==3'b000)
Y=I3;
else if(s2s1s0==3'b000)
Y=I4;
else if(s2s1s0==3'b000)
Y=I5;
else if(s2s1s0==3'b000)
Y=I6;
else if(s2s1s0==3'b000)
Y=I7;
endmodule
```

Test bench:-

```
module mux8_test_mux8_v_tf();
//DATE:11:05:31 06/22/2018
//MODULE:mux8
//DESIGN: mux8
//FILENAME: test_mux8.v
//PROJECT: multiplexer
//VERSION:

//Inputs
reg I0;
```

```
reg I1;
reg I2;
reg I3;
reg I4;
reg I5;
reg I6;
reg I7;
reg S0;
reg S1;
reg S2;

//OUTPUTS
wire Y;
//Bidirs
//Instantiate the UUT
mux4 uut(
.I0(I0),
.I1(I1),
.I2(I2),
.I3(I3),
.I4(I4),
.I5(I5),
.I6(I6),
.I7(I7),
.S0(S0),
.S1(S1),
.S2(S2),
.Y(Y)
);

//Initialize Inputs
ifdef auto_init
initial begin

I0-0;
I1=1;
I2=1;
I3=0;
I4=0;
I5=0;
I6=0;
I7=0;
S0=0;
S1=0;
S2=0;

S2S1S0=3'b000;#5; S2S1S0=3'b001;#5; S2S1S0=3'b010;#5;
S2S1S0=3'b011;#5; S2S1S0=3'b100;#5; S2S1S0=3'b101;#5;
```

S2S1S0=3'b110;#5; S2S1S0=3'b111;#5;
end
endif
endmodule

THEORY:-

Always Block

The always block is the primary construct in RTL modeling. Like the continuous assignment, it is a concurrent statement that is continuously executed during simulation. This also means that all always blocks in a module execute simultaneously. This is very unlike conventional programming languages, in which all statements execute sequen- tially. The always block can be used to imply latches, flip-flops or combinational logic. If the statements in the always block are enclosed within **begin ... end**, the statements are executed sequentially. If enclosed within the **fork ... join**, they are executed concurrently (simulation only).

The always block is triggered to execute by the level, positive edge or negative edge of one or more signals (sepa- rate signals by the keyword **or**). A double-edge trigger is implied if you include a signal in the event list of the always statement. The single edge-triggers are specified by **posedge** and **negedge** keywords.

Procedures can be named. In simulation one can **disable** named blocks. For synthesis it is mainly used as a com- ment.

Syntax 1

```
always @(event_1 or event_2 or ...)
  begin
    ... statements ...
end
```

Example 10 .1

```
always @(a or b) // level-triggered; if a or b changes levels
always @(posedge clk); // edge-triggered: on +ve edge of clk
```

see previous sections for complete examples

Syntax 2

```
always @(event_1 or event_2 or ...)
  begin: name_for_block
    ... statements ...
end
```

Initial Block

The initial block is like the always block except that it is executed only once at the beginning of the simulation. It is typically used to initialize variables and specify signal waveforms during simulation. Initial blocks are not supported for synthesis.

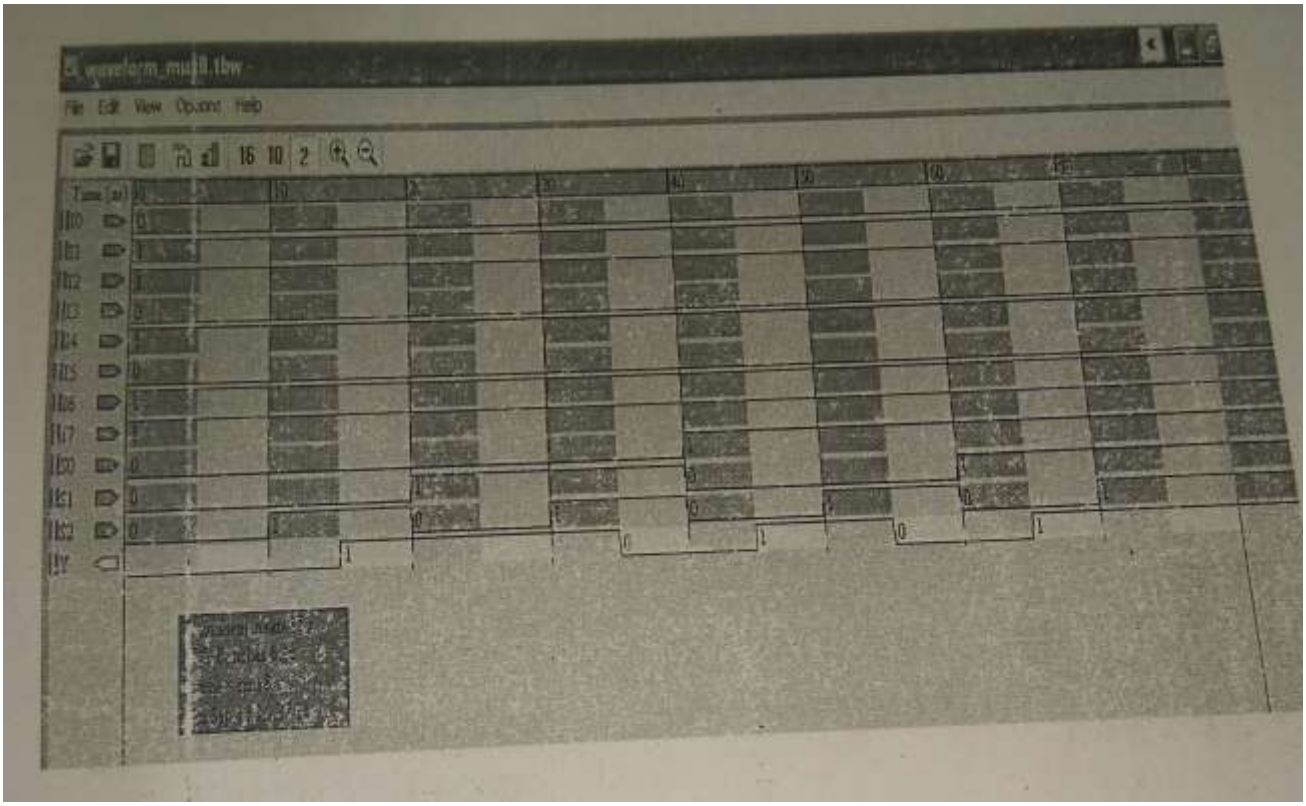
Syntax

```
initial
    begin
        ... statements ...

end
```

Example 10 .2

```
inital
 begin
   clr = 0;          // variables initialized at
   clk = 1;          // beginning of the
   simulation

inital               // specify simulation
 begin               waveforms
   a = 2'b00;
   #50 a = 2'b01; // at time = 50, a = 01
   #50 a = 2'b10; // at time = 100, a = 10
 end
```

OBSERVATION:



CONCLUSION:- Thus by observing test bench waveform for mux4 we have studied working of verilog module for 8:1 multiplexer.

3.Quiz on the subject:-

Model Viva Questions for "Digital Electronics"

Title of the Practical: Verify the truth table of logic gates AND, OR, NOT, NAND and NOR gates/ Design Basic Gates Using NAND/NOR gates.

What is DIGITAL GATE?

A1:-Digital gates are basically electronic components which are used for switching and manipulating binary data

What do u mean by universal gate?

A2:-The universal gates are those gate from which we can make any gate by using them. The universal gates are- NAND & NOR

What is truth table?

A3:-Truth table is a table from which we can get o/p of different

gates Q4. Make truth table of NAND gate?

A4:-

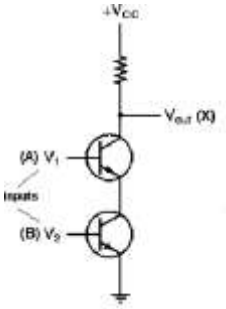
A	B	o/p
0	0	1
0	1	1
1	0	1
1	1	0

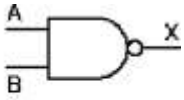
What is different between Ex-or & Ex-nor gate?

A5:-the basic difference between this two gate is that Ex-or gate gives o/p when both the i/p is different & Ex-nor gate give o/p when both i/p same.

Draw the NAND gate?

A6:-





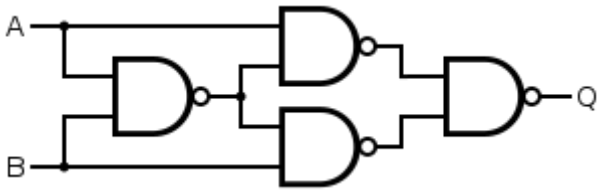
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Draw the Ex-or gate?

A7:-

Draw the EX-OR gate by using the NAND gate?

A8:-

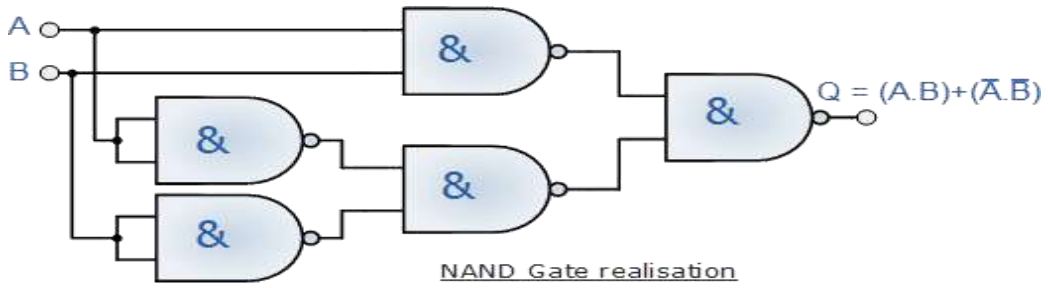


Draw the EX-NOR gate?

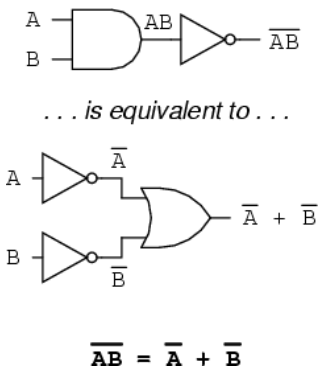
A9:-

Draw the EX-NOR by using the NAND gate?

A10:-



NAND Gate realisation



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

Title of the Practical: - Verify D'morgans theorem.

what is D'morgans theorem.

A1: - D'morgans theorem is theorems through which we can easily manipulate and reduce the given equation.

writes D'morgans theorem equations.

A2: - 1. (A+B)'=A'B'

2. A'B'= (A+B)'

solve following example by using D'morgans theorem.

A3: - (ABC)'=A'+B'+C'

solve following example by using D'morgans theorem.

(A+B+C)' A4: - (A+B+C)'=A'B'C'

solve following example by using D'morgans theorem.

(ABC)' (AB)' A5: - (ABC)' (AB)'= (A'+B'+C') (A'+B')

who invent the D'morgans theorem?

A6: - The law is named after Augustus De Morgan (1806–1871)

De Morgan theorem is used for what?

A7: - this theorem is useful for solving the different bullion expressions.

Draw the example of De Morgan theorem?

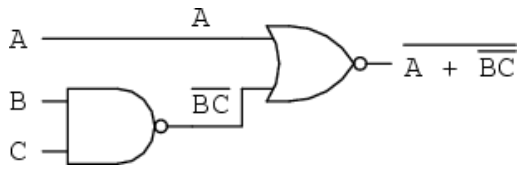
A7:-

For what De Morgan theorem is used?

A9: - De Morgan theorem may be thought of in terms of breaking a long bar symbol.

what is the output of this expression?

A10:-



Title of the Practical: - Design of half adder.

what is use of half adder?

A1:-It is used for adding 2 bit data

in Half adder how many inputs are used?

A2: - Two

In o/p of Half adder what we gate?

A3: - SUM, CARRY

in Half adder SUM=?

A4: - SUM= A (+) B.

In half adder Carry=?

A5: - AB

how many AND gate required to make a Half adder?

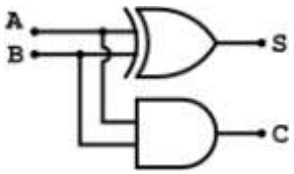
A6: - 1 one

in Half adder how many types of gates are required?

A7: - two types NAND & Ex-or

Draw the half adder diagram?

A8:-



what is difference between the half and full adder?

A9: - in half adder only 2bits can be use but in full adder we can use 3 bit data.

what is difference between half adder and half subs tractor?

A10:- The only difference is in carry & borrows expression.

Title of the Practical: - Design of full adder.

what is use of Full adder?

A1:-Full adder is used for adding three bit data.

In full adder how many inputs are used?

A2:-three

In o/p of full adder what we gate?

A3:-SUM & Carry

In full adder SUM=?

A4: - SUM= A (+) B (+) C.

in full adder Carry=?

A5:-Carry=AB+BC+AC

how many half adders required to make a full adder?

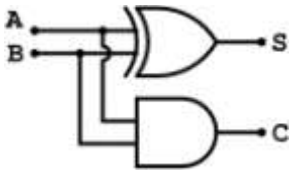
A6: - 2 Half Adder

In full adder how many types of gates are required?

A7:-three types 1. And, 2.Ex-or, 3.or

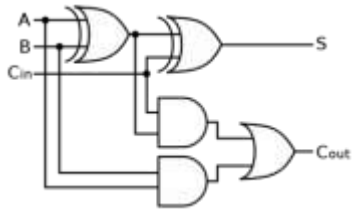
Draw the half adder diagram?

A8:-



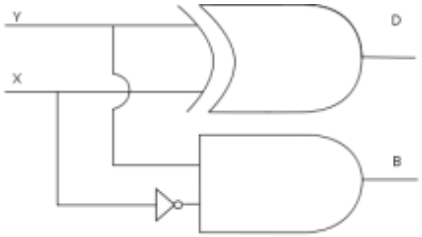
Draw the full adder diagram?

A9:-



Draw the half subtractor diagram?

A10:-



Title of the Practical: - Design half subs tractor.

what is use of half Subs tractor?

A1: - It is used for subs tract 2 bit data.

in Half Subs tractor how many inputs are used?

A2: - two

In o/p of Half Subs tractor what we gate?

A3:-Difference & Borrow

in Half Subs tractor Difference=?

A4: - Difference=A (+) B. Borrow=A'B?

how many AND gate required to make a Half Subs tractor?

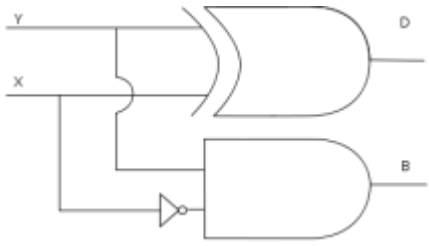
A5:- one

in Half Subs tractor how many types of gates are required?

A6:- two ANAD & Ex-or

Draw the half subtractor diagram?

A7:-



what is difference between full & half substractor?

A8: - in half substractor we can subtract only 2 bit data but in full substractor we can subtract 3 bit data.

what is difference between half adder and half substractor?

A9: - The only difference is in carry & borrows

expression. Q10 If input of half substractor is 11 then

output is?

A10: - Difference= 0, borrow=0.

Title of the Practical:- Design of full subs tractor.

what is use of Full Subs tractor?

A1: - Full substractor is used for differentiate three bit data.

in Full Subs tractor how many inputs are used?

A2: - three

In output of Full Subs tractor what we gate?

A3: - Difference & borrow

In Full Subs tractor Difference=?

A.4 Difference=A (+) B (+) C. Borrow=A'B+A'C+BC?

how many NAND gate required to make a Full Subs tractor?

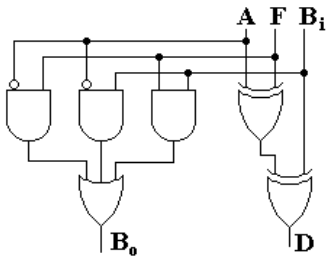
A5: - nine

in Full Subs tractor how many Half Subs tractor are required?

A6:-two

Draw the full substractor diagram?

A7:-

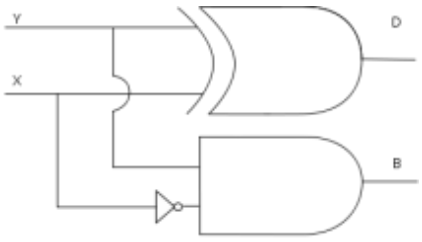


how many half substractor are required to construct a full adder?

A8: - Two half substractor are required to construct a full adder.

Draw the half substractor diagram?

A9:-



Q10what is difference between full & half substractor?

A10: - in half substractor we can subtract only 2 bit data but in full substractor we can subtract 3 bit data.

Title of the Practical: - Verify the operation of magnitude comparator (7485 IC)

Q1what is magnitude comparator?

A1 a digital comparator or magnitude comparator is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number. Comparators are used in a central processing units (CPU) and microcontrollers. Examples of digital comparator include the CMOS 4063 and 4585 and the TTL 7485 and 74682-'89.The analog equivalent of digital comparator is the voltage comparator. Many microcontrollers have analog comparators on some of their inputs that can be read or trigger an interrupt.

Q2what is most significant bit?

A2 In computing, the most significant bit (msb, also called the high-order bit) is the bit position in a binary number having the greatest value. The msb is sometimes referred to as the left-most bit on big-endian architectures, due to the convention in positional notation of writing more significant digits further to the left.

Q3explain operation of AND gate?

A3 the AND gate is a digital logic gate that implements logical conjunction - it behaves according to the truth table to the right. A HIGH output (1) results only if both the inputs to the AND gate are HIGH (1). If neither or only one input to the AND gate is HIGH, a LOW output results. In another sense, the function of AND effectively finds the minimum between two binary digits, just as the OR function finds the maximum.

Q4 explain truth table of a comparator?

A4 the operation of a single bit digital comparator can be expressed as a truth table:

Inputs		Outputs		
A	B	A < B	A = B	A > B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Q5what is equality?

A5 The binary numbers A and B will be equal if all the pairs of significant digits of both numbers are equal, i.e., $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$

Q6what is inequality?

A6 In order to manually determine the greater of two binary numbers, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant bit, gradually proceeding towards lower significant bits until an inequality is found. When an inequality is found, if the corresponding bit of A is 1 and that of B is 0 then we conclude that A>B.

Q7explain magnitude comparator7485 IC

A7 the 74F85 is a 4-bit magnitude comparator that can be expanded t almost any length. It compares two 4-bit binary, BCD, or other monotonic codes and presents the three possible magnitude results at the outputs. The 4-bit inputs are weighted (A0–A3) and (B0–B3) where A3 and B3 are the most significant bits. The operation of the 74F85 is described in the Function Table, showing all possible logic conditions. The upper part of the table describes the normal Operation under all conditions that will occur in a single device or in a series expansion scheme. In the upper part of the table the three outputs are mutually exclusive. In the lower part of the table, the outputs reflect the feed-forward conditions that exist in the parallel expansion scheme. The expansion inputs IA>B, and IA=B and IA<B are the least significant bit positions. When used for series expansion, the A>B, A=B and A<B outputs of the lease significant word are connected to the corresponding IA>B, IA=B and IA<B inputs of the next higher stage. Stages can be added in this manner to any length, but a propagation delay penalty of about 15ns is added with each additional stage. For proper operation, the expansion inputs of the least significant word should be tied as follows: IA>B = Low, IA=B = High and IA<B = Low.

Q8what is 8-input Magnitude Comparator?

A8 Magnitude Comparator. This Magnitude Comparator can be used perform comparisons of two 8-bit binary or BCD words. The output provides both a P equals Q function or P greater than Q function.
A Magnitude Comparator would be considered standard logic or glue logic when a discrete IC is used. However, because of the internal complexity, a Magnitude Comparator would also be considered an MSI Function [Medium Scale Integration].

Q9what is IC?

A9 In electronics, an integrated circuit (also known as IC, chip, or microchip) is a miniaturized electronic circuit (consisting mainly of semiconductor devices, as well as passive components) that has been manufactured in the surface of a thin substrate of semiconductor material. Integrated circuits are used in almost all electronic equipment in use today and have revolutionized the world of electronics. Computers, cellular phones, and other digital appliances are now inextricable parts of the structure of modern societies, made possible by the low cost of production of integrated circuits.

Q10 tell about advancement in integrated circuits?

A10 Among the most advanced integrated circuits are the microprocessors or "cores", which control everything from computers and cellular phones to digital microwave ovens. Digital memory chips and ASICs are examples of other families of integrated circuits that are important to the modern information society. While the cost of designing and developing a complex integrated circuit is quite high, when spread across typically millions of production units the individual IC cost is minimized. The performance of ICs is high because the small size allows short traces which in turn allows low power logic (such as CMOS) to be used at fast switching speeds.

Title of the Practical: - Study of flip-flop.

what is flip-flop?

A1: - Flip-flop is a 1 bit storing element.

how many types of flip-flop are used?

A2: - 4 types of flip –flop, S-R, J-K, D, T

what is disadvantage of SR flip-flop?

A3:- When both the input is one then it gives invalid output.

what is disadvantage of JK flip-flop?

A4: - race around condition.

to remove race around condition what we use?

A5: - master slave Flip-flop.

what is race around condition?

A6: - when pulse width is more then signal width then for signal change of pulse width many no of times signal changes its state that is called race around condition.

Q7 what are the characteristic equation for T flip-

flop? A7 Q = TQ'+ QT'

Q8 which Gates are used in SR flip flops to a JK flip-

flop? A8 Nand Gates

D flip-flop is used for?

A9: - providing delay.

what is full form of T flip-flop?

A10: - toggle flip-flop

Title of the Practical: - Design 3/4 bit Counter & Ripple 3/4 bit Counter & verify truth table.

Q1what is counter?

A1 In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

Q2give types of counter?

A2 there are two types of counters(1) Up counters, which increase (increment) in value (2) Down counters, which decrease (decrement) in value .

Q3what are the implements of counter?

A3 In electronics, counters can be implemented quite easily using register-type circuits such as the flip- flop, and a wide variety of designs exist, e.g.:(1)Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops (2)Synchronous counter – all state bits change under control of a single clock (3)Decade counter – counts through ten states per stage (4)Up–down counter – counts both up and down, under command of a control input (5)Ring counter – formed by a shift register with feedback connection in a ring (6)Johnson counter – a twisted ring counter (7)Cascaded counter

Q4 explain Asynchronous (ripple) counter?

A4 an asynchronous (ripple) counter is a single K-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0.
Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), you will get another 1 bit counter that counts half as fast. Putting them together yields a two bit counter.

Q5 explain Johnson counter?

A5 A Johnson counter (or switch tail ring counter, twisted-ring counter, walking-ring counter, or Moebius counter) is a modified ring counter, where the output from the last stage is inverted and fed back as input to the first stage.[2][3][4] A pattern of bits equal in length to twice the length of the shift register thus circulates indefinitely. These counters find specialist applications, including those similar to the decade counter, digital to analog conversion, etc. it

can be established by D flip flop and JK flip flop.

Q6 explain Decade counter?

A6 A decade counter is one that counts in decimal digits, rather than binary. A decade counter may have each digit binary encoded (that is, it may count in binary-coded decimal, as the 7490 integrated circuit did) or other binary encodings (such as the bi-quinary encoding of the 7490 integrated circuit).

Alternatively, it may have a "fully decoded" or one-hot output code in which each output goes high in turn; the 4017 is such a circuit. The latter type of circuit finds applications in multiplexers and demultiplexer, or wherever a scanning type of behavior is useful. Similar counters with different numbers of outputs are also common.

Q7what is synchronous counters?

A7 In synchronous counters, the clock inputs of all the flip-flops are connected together and aretriggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel). The circuit below is a 3-bit synchronous counter. The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.

Q8 give truth table of two bit counter?

A8 Putting they together yield a two bit counter:

Cycle	Q1	Q0	(Q1:Q0)decimal
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3
4	0	0	0

Q9 give types of ring counters?

A9 There are two types of ring counters: (1)A straight ring counter or Over beck counter connects the output of the last shift register to the first shift register input and circulates a single one (or zero) bit around the ring. For example, in a 4-register one-hot counter, with initial register values of 1000, the repeating pattern is: 1000, 0100, 0010, 0001, 1000... . Note that one of the registers must be pre-loaded with a 1 (or 0) in order to operate properly. (2)A twisted ring counter (also called Johnson counter or Moebius counter) connects the complement of the output of the last shift register to its input and circulates a stream of ones followed by zeros around the ring. For example, in a 4-register counter, with initial register values of 0000, the repeating pattern is: 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, and 0000.

Q10what is one hot counter?

A10 In digital circuits, one-hot refers to a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). For example, the output of a decoder is usually a one-hot code, and sometimes the state of a state machine is represented by a one-hot code. A similar implementation in which all bits are '1' except one '0' is sometimes called one-cold

Title of the Practical: Design a counter for given event's counting

what is counter?

Ans: - a **counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

what is the types of counter?

A1: In practice, there are two types of counters:

1. Up counters, which increase (increment) in value
2. Down counters, which decrease (decrement) in value

what is the basic type of counter made by flip-flop or resistor?

A2: - Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip- flops

Synchronous counter – all state bits change under control of a single clock

Decade counter – counts through ten states per stage

Up–down counter – counts both up and down, under command of a

control input Ring counter – formed by a shift register with feedback

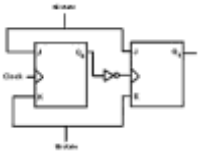
connection in a ring Johnson counter – a *twisted* ring counter

Cascaded counter

what is asynchronous counter?

A4: - An asynchronous (ripple) counter is a single K-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0.
Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), you will get another 1 bit counter that counts half as fast

make diagram of Asynchronous counter?

A5:- 

what is Synchronous counter?

A6:- A simple way of implementing the logic for each bit of an ascending counter (which is what is depicted in the image to the right) is for each bit to toggle when all of the less significant bits are at a logic high state. For example, bit 1 toggles when bit 0 is logic high; bit 2 toggles when both bit 1 and bit 0 are logic high; bit 3 toggles when bit 2, bit 1 and bit 0 are all high; and so on.

Synchronous counters can also be implemented with hardware finite state machines, which are more complex but allow for smoother, more stable transitions.

what is the ring counter?

A7: - A ring counter is a shift register (a cascade connection of flip-flops) with the output of the last one connected to the input of the first, that is, in a ring. Typically a pattern consisting of a single 1 bit is circulated, so the state repeats every N clock cycles if N flip-flops are used. It can be used as a cycle counter of N states.

what is the Johnson counter?

A8: - A Johnson counter (or switchtail ring counter, twisted-ring counter, walking-ring counter, or Moebius counter) is a modified ring counter, where the output from the last stage is inverted and fed back as input to the first stage. A pattern of bits equal in length to twice the length of the shift register thus circulates indefinitely. These counters find specialist applications, including those similar to the decade counter, digital to analog conversion, etc. it can be established by D flip flop and JK flip flop

what is the decade counter?

A9: - A decade counter is one that counts in decimal digits, rather than binary. A decade counter may have each digit binary encoded (that is, it may count in binary-coded decimal, as the 7490 integrated circuit did) or other binary encodings (such as the bi-unary encoding of the 7490 integrated circuit).
Alternatively, it may have a "fully decoded" or one-hot output code in which each output goes high in turn; the 4017 is such a circuit. The latter type of circuit finds applications in multiplexers and demultiplexers, or wherever a scanning type of behavior is useful. Similar counters with different numbers of outputs are also common.

The decade counter is also known as a mod-counter.

what do you mean by up down counter?

A10:- A counter that can change state in either direction, under the control of an up–down selector input, is known as an up–down counter. When the selector is in the up state, the counter increments its value; when the selector is in the down state, the counter decrements the count

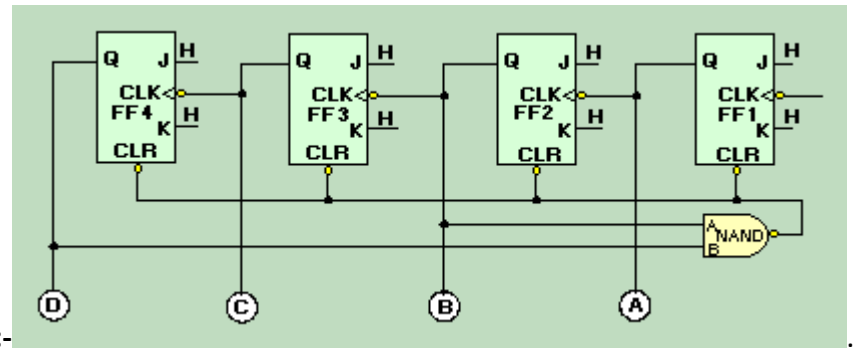
Machine cycle- the time taken by data/ opcode / operand from memory/ peripheral devices to acknowledge the external hardware. it takes 1to 6 T-state.

Title of the Practical: Design Decade Counter & verify truth table.

Q1. What is decade counter?

A1:- A decade counter is a binary counter that is designed to count to 10_{10}, or 1010_2. An ordinary four- stage counter can be easily modified to a decade counter by adding a NAND gate as shown in figure 3-
25. Notice that FF2 and FF4 provide the inputs to the NAND gate. The NAND gate outputs are connected to the CLR input of each of the FFs.

Q2. Diagram of decade counter?

A2:- 

.

Q3 How many stages are required for a decade

counter?

A3 Four.

Q4 In figure 3-25, which two FFs must be HIGH to reset the

counter?

A4. FFs 2 and 4.

Q5 Which AND gate causes FF3 to reset?
A5 Three.

Q6 what causes the specified condition to shift position?

A6 the input or clock pulse.

Q7 If the specified state is OFF, how many FFs may be off at one time?

A7 One.

Q8 how many FFs are required to count down from

15_{10}? A8 Four

Q9 what signal causes FF2 to toggle?

A9 Q output of FF 1 going LOW

Q10 How many stages are required to store a 16-bit word?

A10 16.

Title of the Practical: Design Shift Register & verify truth table

Q. what is the shift resistor?

A1 Shift resistor is a device which is used for storing and processing the bit in series or parallel.

Q.2- Types of shift resistor?

A2:-Serial-in/serial-out shift register (FIFO or

LIFO) Serial-in/parallel-out shift register

Parallel-in/parallel-out shift register

Parallel-in/serial-out shift register

. **Q3- In SISO resistor how many input cycles are required?**

A3 – it required n+1 clock pulse for input & n clock pulse for output.

Q4- In PISO resistor how many input cycles are required?

A4 - it required 1 clock pulse for input & n clock pulse for

output

In PIPO resistor how many input cycles are required?

A5 it required 1 clock pulse for input & 0 clock pulse for output

In SIPO resistor how many input cycles are required?

A6: it required n+1 clock pulse for input & 0 clock pulse for output

Q.7what do you mean by clock enable?

A7 When the Clock Enable input is High, the enabled load and shift actions take place on the next active Clock transition. When Clock Enable is Low, the register contents are unaffected by the Clock.
Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Q.8what is the clock?

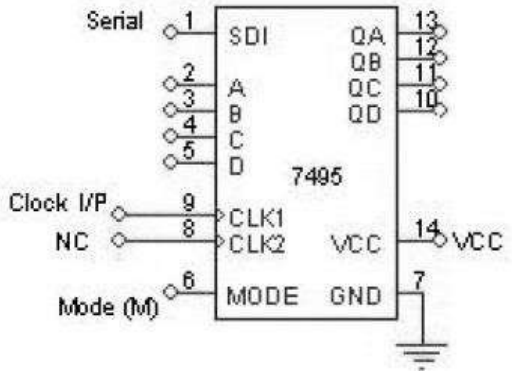
A8 clock is a trigger. By Appling the clock the information proceeds.

Pin diagram of SIPO shift resistor?

A9

Pin diagram of SISO shift resistor?

SISO:-



A10:

Title of the Practical: Design Basic Gates Using NOR gates.

Q.1what is the Universal gates?

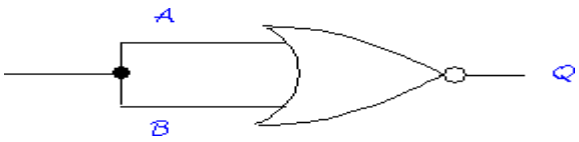
A1– the universal gates are the gate by help of which we can make other all gates.

Q. 2:- Types of universal gate?

A2: - There are two universal gates:-

1. NAND
2. NOR

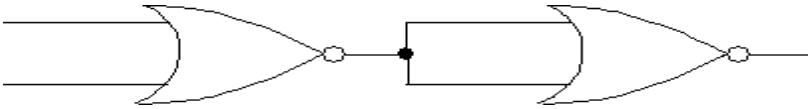
Q.3:- How would you make a NOT gate from a NOR gate?



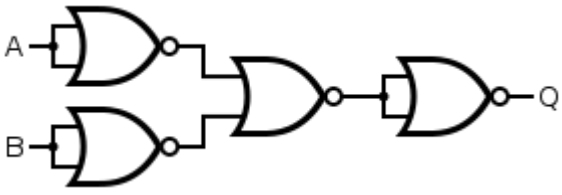
A3: -

Q.4:- How would you make an OR gate from a NOR gate?

A4: -

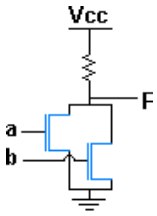


Q.5:- Draw the NAND gate by using the NOR gate?



A5: -

Q.6:- How we can make the NOR gate by using the NMOS?

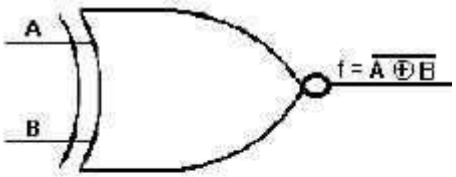


A6: -

Q.7:- make the symbol of NOR gate?

A7: -

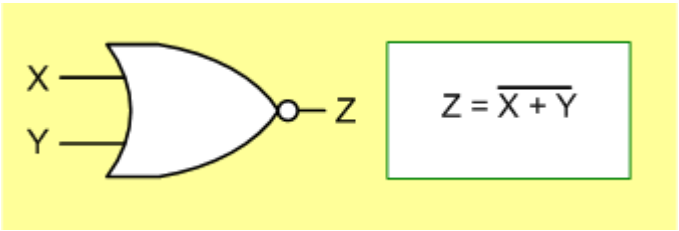
Q.8:- Draw the EX- NOR gate and also write truth table?



A	B	f
0	0	1
0	1	0
1	0	0
1	1	1

Q.9:- Draw the NOR gate?

A9-



Q.10:- what is the truth table of NOR gate?

X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

A10:

4. Conduction of Viva-Voce Examinations:

Teacher should conduct oral exams of the students with full preparation. Normally, the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested. Oral examinations are to be conducted in cordial environment amongst the teachers taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

5. Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become. It is a primary responsibility of the teacher to see that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with just circumstances.