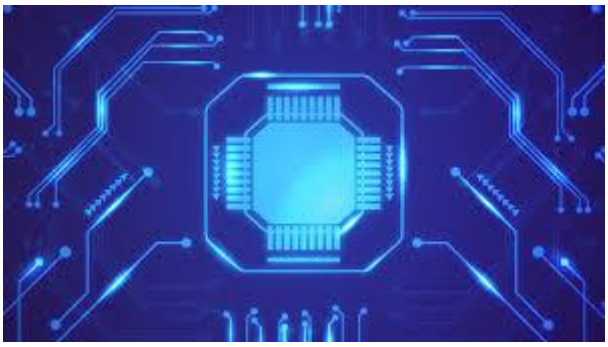
Digital Circuit Design Using Verilog

1BET302

III Semester



Dayananda Sagar College of Engineering

(An Autonomous Institute Affiliated to VTU, Belagavi, Accredited by NAAC with 'A' Grade)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560111

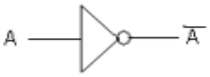
Department Of Electronics and Telecommunication Engineering

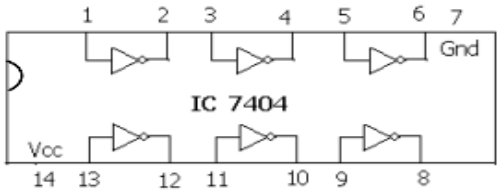
NOT GATE:

Truth Table

I/P (A)	O/P $\overline{(A)}$
0	1
1	0

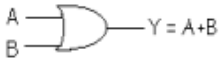
Symbol

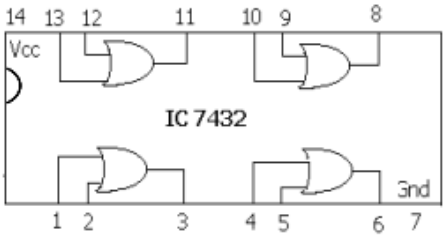




OR GATE

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

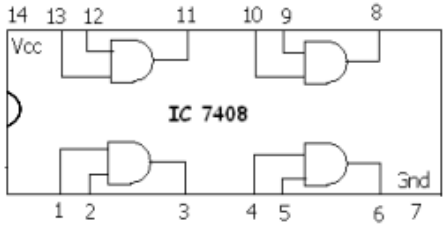




AND GATE

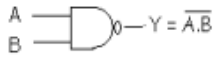
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

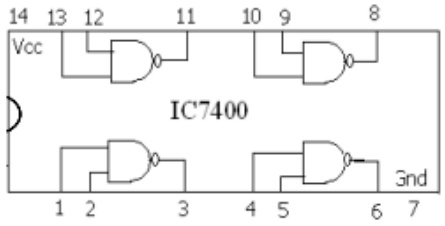




NAND GATE

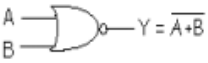
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

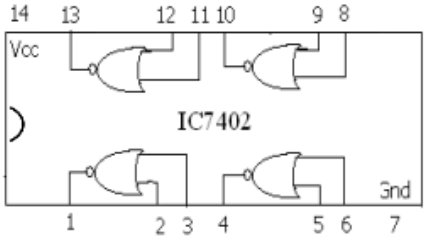




NOR GATE

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0





EX-OR GATE

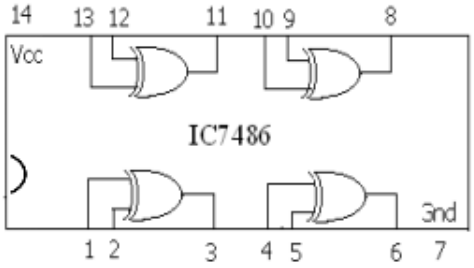
Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Symbol

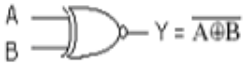


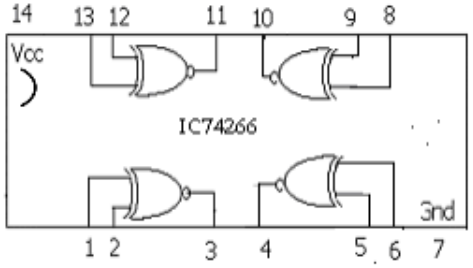
IC7486



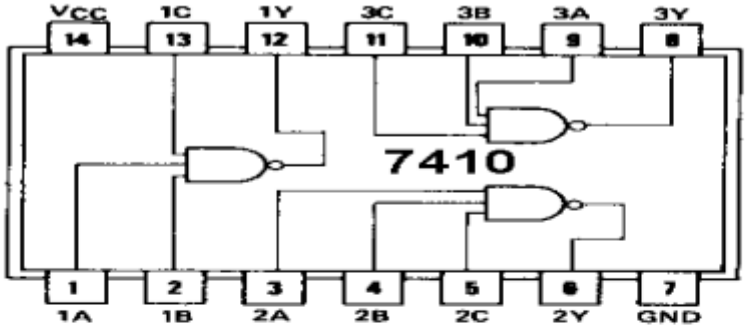
EX-NOR GATE (IC74266)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

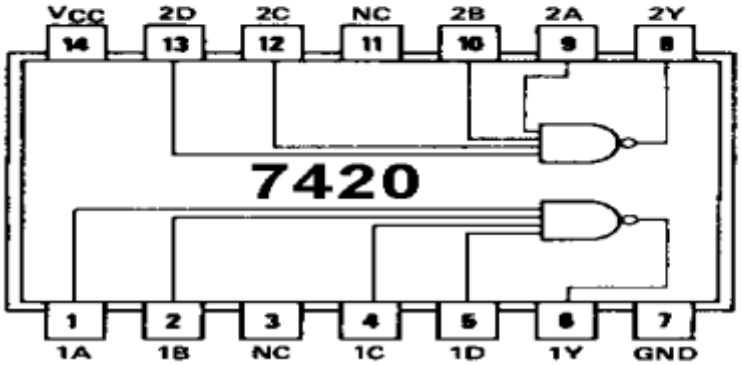




THREE INPUT NAND GATE



FOUR INPUT NAND GATE



Experiment 1: 1. Realization of full adder and full subtractor

AIM: To realize Full Adder and Full Subtractor by using Basic gates

COMPONENTS REQUIRED: IC 7408, IC 7486, IC 7432, IC 7404, Patch Cords & IC Trainer Kit.

THEORY: Combinational Circuit is one in which value of the output at any instant of time depend only on present state of the input ie it does not have memory. Adder and Subtractor is a combinational circuit. Adders and Subtractor find application in Arithmetic and Logic Unit, Array Multipliers, tri State buffers etc.

Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that Adds two data bits, A and B, and a carry-in bit, Cin is called a full-adder. The Boolean Functions describing the full-adder are:

$$Sum = A \oplus B \oplus Cin$$

$$Carry = AB + ACin + BCin$$

Full Subtractor: Subtracting two single-bit binary values, B, Bin from a single-bit value A Produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-Subtractor are:

$$Diff = A \oplus B \oplus Bin$$

$$Bout = \overline{A}B + \overline{A}Bin + BBin$$

PROCEDURE:

1. Place the IC in the socket of the trainer kit.
2. Make the connections for the gate as shown in the circuit diagram.
3. Verify the Truth Table.
4. Repeat the above steps for other gates in the different IC chips.

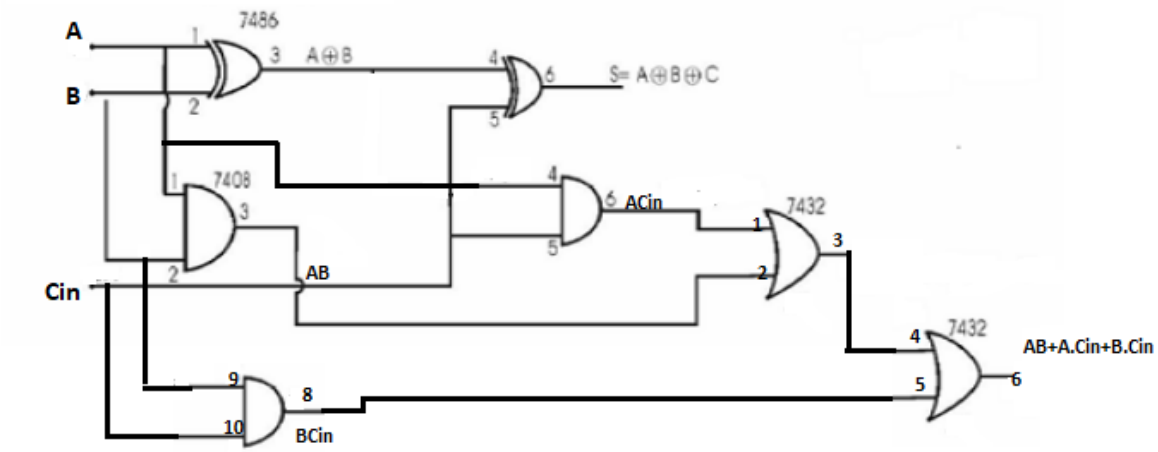
1. **Full Adder:**

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$Sum = A \oplus B \oplus Cin$$

$$Carry = AB + ACin + BCin$$

Full adder using Basic Gates:



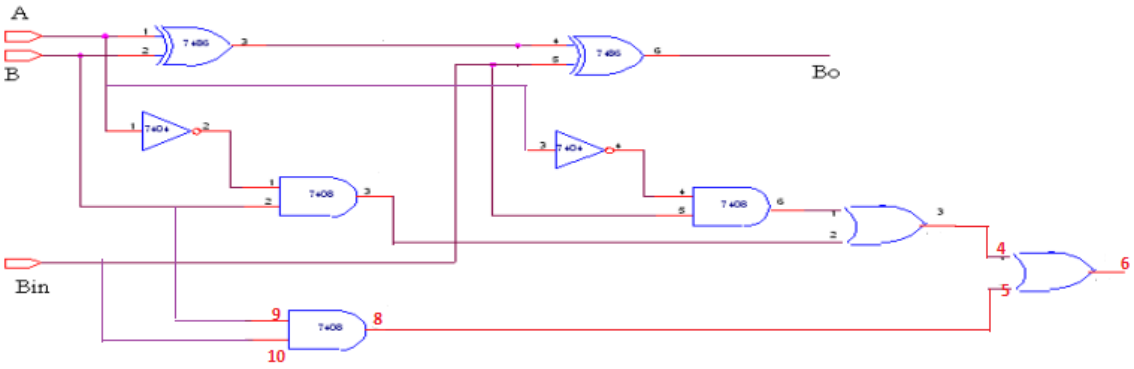
2. FULL SUBTRACTOR:

TRUTH TABLE

INPUTS			OUTPUTS	
A	B	Bin	D	Bo
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$Diff = A \oplus B \oplus Bin$$

$$Bout = \overline{A}B + \overline{A}Bin + BBin$$



EXPERIMENT-2
BINARY TO GRAY CODE CONVERSION VICE &VERSA

AIM:

1) To Design and set up 4-bit Binary to Gray code converter circuit

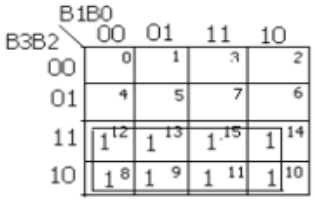
2) To Design and set up 4-bit Gray to binary code circuit using gates.

THEORY: This is another binary code which finds application in input/output devices, shaft position encoders and several analog to digital converters. This is an unweighted code. The gray code exhibits only a single bit change from one code number to the next. Code converter is a combinational circuit that translates the input code word into a new corresponding word.

BINARY TO GRAY CONVERSION:

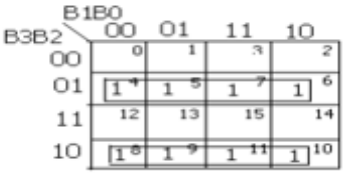
BINARY				GRAY CODE			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

$G3=\sum(8,9,10,11,12,13,14,15)$



G3= B3

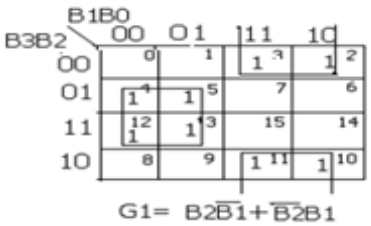
$G2= \sum(4,5,6,7,8,9,10,11)$



$\overline{B3}B2 + B3\overline{B2}$

$G2= B3 \oplus B2$

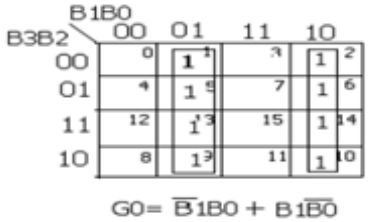
$G1= \sum(2,3,4,5,10.,11,12,13)$



$G1= B2\overline{B1} + \overline{B2}B1$

$G1= B1 \oplus B2$

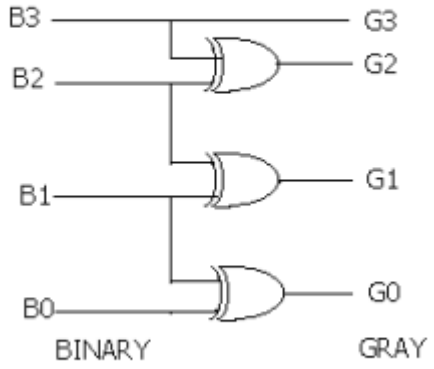
$G0 = \sum(1,2,5,6,9,10,13,14)$



$G0= \overline{B1}B0 + B1\overline{B0}$

$G0= B1 \oplus B0$

Binary to Gray Conversion USING EX-OR GATES



$G3 = B3$

$G2 = \overline{B3}B2 + B3\overline{B2}$

$G2 = B3 \oplus B2$

$G1 = B2\overline{B1} + \overline{B2}B1$

$G1 = B1 \oplus B2$

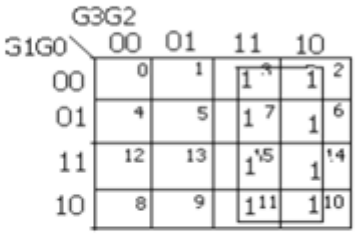
$G0 = \overline{B1}B0 + B1\overline{B0}$

$G0 = B1 \oplus B0$

Gray to Binary code converter

GRAY CODE				BINARY CODE			
G3	G2	G1	G0	B3	B2	B1	B0
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	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

K MAP FOR B3



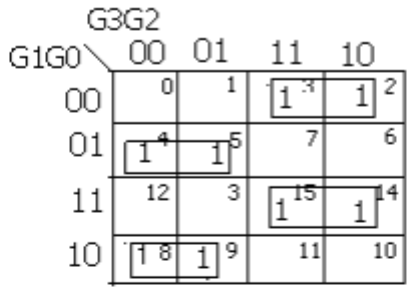
$B3 = G3$

K MAP FOR B2



$B2 = \overline{G3}G2 + G3\overline{G2}$

K MAP FOR B1



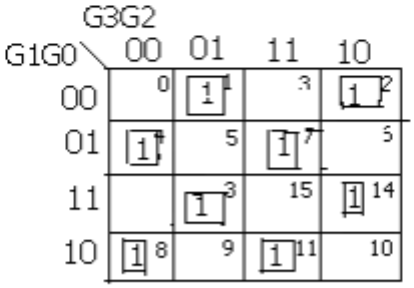
$B1=\overline{G1}G0\overline{G3}+\overline{G1}\overline{G0}G3+G1G0G3+G1\overline{G0}\overline{G3}$

$=\overline{G1}(G0\overline{G3}+\overline{G0}G3)+G1(G0G3+\overline{G0}\overline{G3})$

$=\overline{G1}(G0 \oplus G3)+G1(\overline{G0 \oplus G3})$

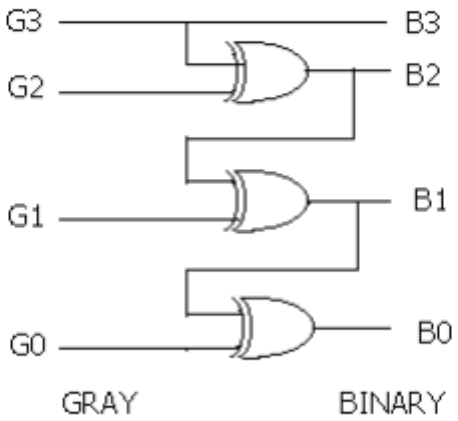
$B1=G3 \oplus G2 \oplus G1$

K MAP FOR B0



$BO=G0 \oplus G1 \oplus G2 \oplus G3$

Gray to Binary Conversion USING EX-OR GATES



Result: The digital circuit for Binary to Gray & Gray to binary is built and verified the Conversion Table

Experiment 3:
Parallel Adder/Subtractor

AIM: (a) Realization of Parallel adder/Subtractor using 7483chip
(b) BCD to XS3 code conversion and vice versa

COMPONENTS REQUIRED: IC Trainer kit, IC 7483 Parallel Adder and IC7486 NAND gate.

THEORY: The Full adder can add single-digit binary numbers and carries. The largest sum that can be obtained using a full adder is $(11)_2$. Parallel adders can add multiple-digit numbers. If full adders are placed in parallel, we can add two- or four-digit numbers or any other size desired.

i) 4 BIT BINARY PARALLER ADDER:

1. Make Cin = 0. The circuit works as an Adder.
2. Set A3A2A1A0 = Binary equivalent of the Augend using toggle switches. B3B2B1B0 = Binary equivalent of the Addend using toggle switches.
3. Verify the Result on Cout S3S2S1S0 using LEDs.

Example: 7+2=11 (1001)
* 7 is realized at A3 A2 A1 A0 = 0111
* 2 is realized at B3 B2 B1 B0 = 0010

Sum = 1001

ii) 4 BIT BINARY SUBTRACTOR:

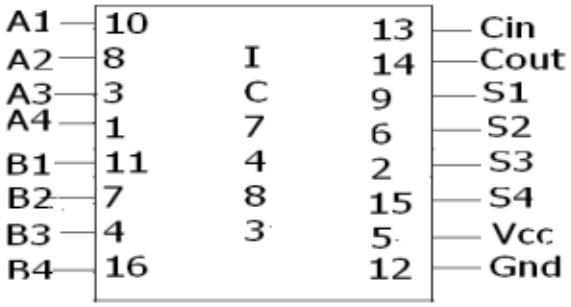
1. Make Cin = 1. The circuit works as a Subtractor.
2. Set A3A2A1A0 = Binary equivalent of the Minuend using toggle switches. B3B2B1B0 = Binary equivalent of the Subtrahend using toggle switches.
3. Verify the Difference on S3S2S1S0 and Carry generated on Cout using LEDs
4. Subtraction is carried out by adding 2's complement of the subtrahend.

Example: 8 – 3 = 5 (0101)
- 8 is realized at A3 A2 A1 A0 = 1000
- 3 is realized at B3 B2 B1 B0 through X-OR gates = 0011
- Output of X-OR gate is 1's complement of 3 = 1100
- 2's Complement can be obtained by adding Cin (M) = 1

A3 A2 A1 A0 = 1 0 0 0
B3 B2 B1 B0 = 0 0 1 1
1's Complement of 0011 is 1100 + Cin=1
2's Complement of 0011 is 1101
S3 S2 S1 S0 = 0 1 0 1
Cout = 1 (Ignored)

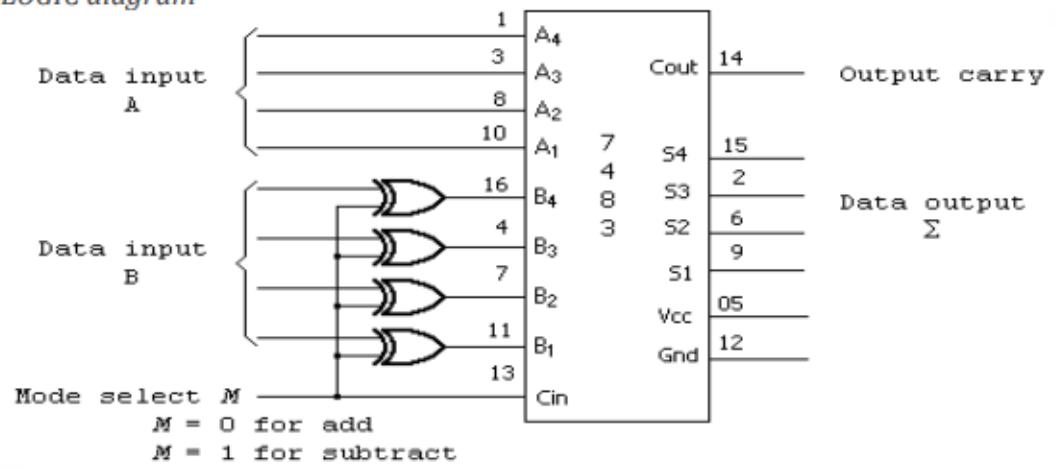
IC 7483 PIN DIAGRAM

Pin Diagram



a) Realization of Parallel adder/Subtractor using 7483chip

LOGIC diagram



Result: Addition and subtraction for the given numbers is performed using 7483.

(ii) BCD to Excess-3 code conversion and Excess-3 to BCD code conversion:

Theory: Combinations of binary digits that represent numbers, letters, or symbols are digital codes. The 8421 is a type of Binary Coded Decimal (BCD) and is composed of four bits representing the decimal digits 0 through 9. The designation 8421 indicates the binary weights of the four bits (2^3 2^2 2^1 2^0). Another form of BCD code is the excess-3 Code. This is unweighted code.ie it does not have definite weights. Excess-3 code can be obtained from BCD code by adding 0011(= 3_{10}) to each digit of a decimal number expressed in BCD code. Since three is added this code gets the name excess-3 code. BCD Code can be obtained from excess-3 code by Subtracting 0011(= 3_{10}) to each digit. Code conversion and vice-versa can be implemented using IC 7483 along with 7486 Xor gates. The four i/p bits of 'B' ie B4, B3, B2, B1, are fixed as 0011. cin =0, performs addition and Cin =1 performs subtraction.

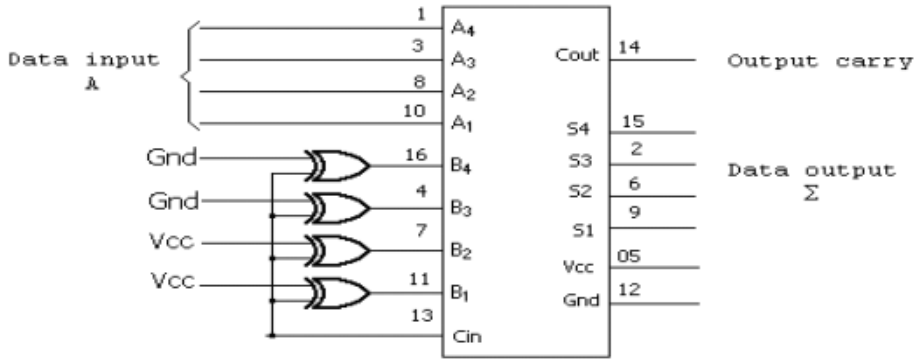
Procedure: - (BCD Excess 3 and Excess-3 to BCD)

1. Make the connections as shown in the fig.
2. Make Cin = 0
3. The inputs are applied at A4, A3, A2 and A1

4. B4, B3, B2 & B1 are fixed as 0011
5. The corresponding outputs at S4, S3, S2, and S1
6. Make Cin = 1 for (Excess-3 to BCD conversion)

Circuit Diagram

BCD to XS-3 / XS-3 to BCD



BCD TO EXCESS-3 CODE (Cin=0)

BCD				XS3			
A4	A3	A2	A1	S4	S3	S2	S1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

EXCESS-3 TO BCD CODE (Cin=1)

XS3			BCD				
A4	A3	A2	A1	S4	S3	S2	S1
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

RESULT: Verified the truth Table of BCD TO EXCESS-3 CODE & VICE VERSA

Experiment 4:
Realization of Boolean Function using MUX/DEMUX (IC-74153, IC-74139)

AIM: To design and set up the following circuit

1) To verify the various functions of IC 74153(MUX) and IC 74139(DEMUX).
2) To set up a Full Adder and Full Subtractor using IC 74153 and IC 74139

LEARNING OBJECTIVE:

- To learn about various applications of multiplexer and de-multiplexer
- To learn and understand the working of IC 74153 and IC 74139
- To learn to realize any function using Multiplexer

THEORY:

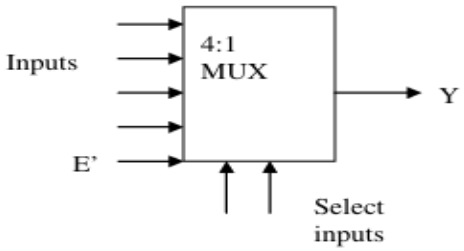
Multiplexers are very useful components in digital systems. They transfer a large number of information units over a smaller number of channels, (usually one channel) under the control of selection signals. Multiplexer means many to one. A multiplexer is a circuit with many inputs but only one output. By using control signals (select lines) we can select any input to the output. Multiplexer is also called as data selector because the output bit depends on the input data bit that is selected. The general multiplexer circuit has 2^n input signals, n control/select signals and 1 output signal.

De-multiplexers perform the opposite function of multiplexers. They transfer a small number of information units (usually one unit) over a larger number of channels under the control of selection signals. The general de-multiplexer circuit has 1 input signal, n control/select signals and 2^n output signals. De-multiplexer circuit can also be realized using a decoder circuit with enable.

COMPONENTS REQUIRED:

IC 7400, IC 7410, IC 7420, IC 7404, IC 74153, IC 74139, Patch Cords & IC Trainer Kit.

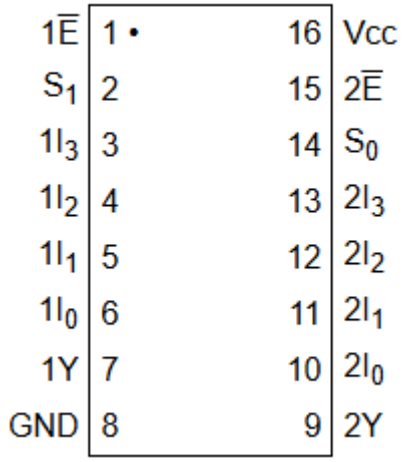
i) 4:1 MULTIPLEXER



Output Y= E'S1'S0'I0 + E'S1'S0I1 + E'S1S0'I2 + E'S1S0I3

TRUTH TABLE

Select Inputs		Enable Input	Inputs				Outputs
S_1	S_0	E	I_0	I_1	I_2	I_3	Y
X	X	1	X	X	X	X	0
0	0	0	0	X	X	X	0
0	0	0	1	X	X	X	1
0	1	0	X	0	X	X	0
0	1	0	X	1	X	X	1
1	0	0	X	X	0	X	0
1	0	0	X	X	1	X	1
1	1	0	X	X	X	0	0
1	1	0	X	X	X	1	1



FULL ADDER USING MUX:

DESIGN:

SUM

I0	I1	I3	I3
0	1	2	3
4	5	6	7
A	**A'**	**A'**	**A**

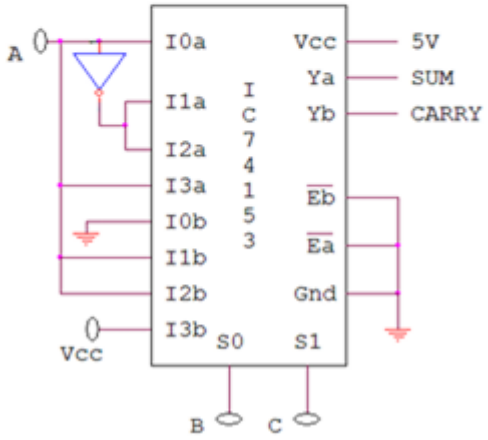
CARRY

I0	I1	I3	I3
0	1	2	3
4	5	6	7
0	**A**	**A**	**1**

TRUTH TABLE

Inputs			Outputs	
A	**B**	**C**	**S**	**C**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

FULL ADDER CIRCUIT



FULL SUBTRACTOR USING MUX:

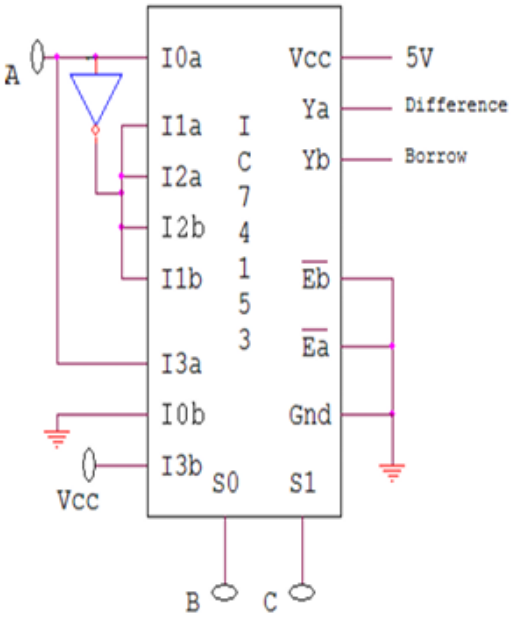
DESIGN:

DIFFERENCE

I0	I1	I2	I3
0	1	2	3
4	5	6	7
A	**A'**	**A'**	**A**

BORROW

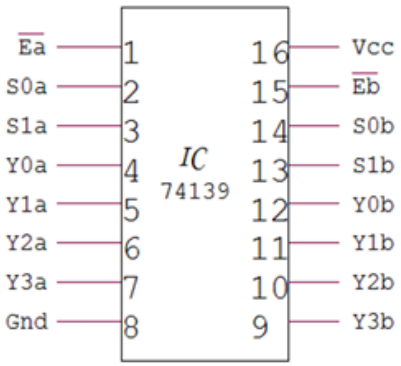
I0	I1	I2	I3
0	1	2	3
4	5	6	7
0	**A'**	**A'**	**1**



TRUTH TABLE

Inputs			Outputs	
A	**B**	**C**	**D**	**Br**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

VERIFICATION OF IC 74139 (DEMUX)



TRUTH TABLE

Inputs			Outputs			
Ea	**S_1**	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Experiment 5:
Design of synchronous 3-bit up/down counter using IC-7476 on IC trainer kit

AIM: To design and test 3-bit binary synchronous counter using flip-flop IC 7476 for the given sequence.

LEARNING OBJECTIVE:

- To learn about synchronous Counter and its application
- To learn the design of synchronous counter counter
-

COMPONENTS REQUIRED:

IC 7476, Patch Cords & IC Trainer Kit

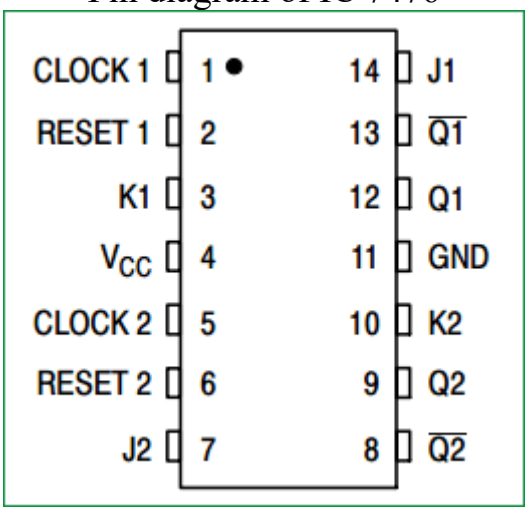
THEORY:

A counter in which each flip-flop is triggered by the output goes to previous flip-flop. As all the flip-flops do not change states simultaneously in asynchronous counter, spike occur at the output. To avoid this, strobe pulse is required. Because of the propagation delay the operating speed of asynchronous counter is low. This problem can be solved by triggering all the flip-flops in synchronous with the clock signal and such counters are called synchronous counters.

PROCEDURE:

- Check all the components for their working.
- Insert the appropriate IC into the IC base.
- Make connections as shown in the circuit diagram.
- Verify the Truth Table and observe the outputs.

Pin diagram of IC 7476



MOD 8 COUNTER:

TRUTH TABLE:

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

Present count / next count

Q_C	Q_B	Q_A	Q_C	Q_B	Q_A
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

JK FF excitation table:

Q	**Q+**	**J**	**K**
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

DESIGN:

1	X	X	1
1	X	X	1

$J_A = 1$

X	1	1	X
X	1	1	X

$K_A = 1$

0	1	X	X
X	1	X	X

$J_B = Q_A$

X	X	1	0
X	X	1	0

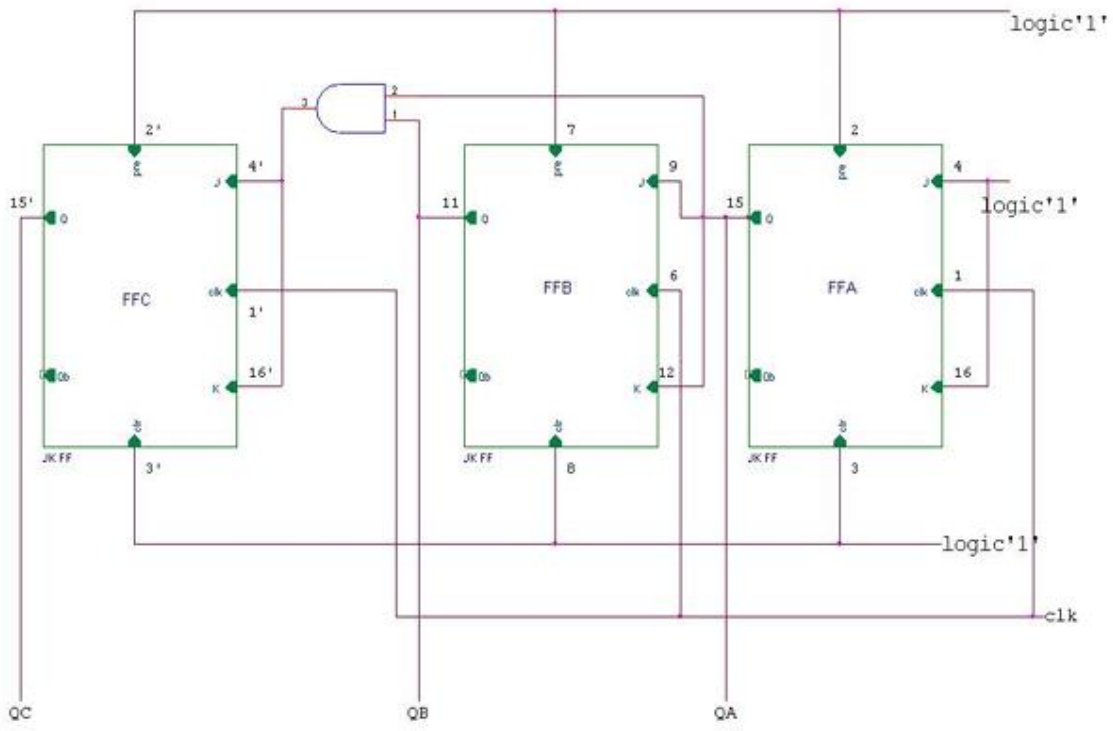
$K_B = Q_A$

0	0	1	0
X	X	X	X

$J_C = Q_BQ_A$

X	X	X	X
0	0	1	0

$K_C = Q_BQ_A$


logic'1'
FFC
FFB
FFA
JK FF
JK FF
JK FF
logic'1'
logic'1'
clk
QC
QB
QA

Experiment 6: Design of Shift register, ring counter, Johnson counter using IC 7495

AIM: To realize and study of Shift Register.

1) SISO (Serial in Serial out)
2) SIPO (Serial in Parallel out)
3) PIPO (Parallel in Parallel out)
4) PISO (Parallel in Serial out)

COMPONENTS REQUIRED: IC 7495, Patch Cords & IC Trainer Kit.

PROCEDURE:

- Check all the components for their working.
- Insert the appropriate IC into the IC base.
- Make connections as shown in the circuit diagram.
- Verify the Truth Table and observe the outputs.

1) SERIAL IN SERIAL OUT (SISO) (Right Shift)

Serial i/p data	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
0	t1	0	X	X	X
1	t2	1	0	X	X
0	t3	0	1	0	X
1	t4	1	0	1	0
X	t5	X	1	0	1
X	t6	X	X	1	0
X	t7	X	X	X	1
X	t8	X	X	X	X

2) SERIAL IN PARALLEL OUT (SIPO)

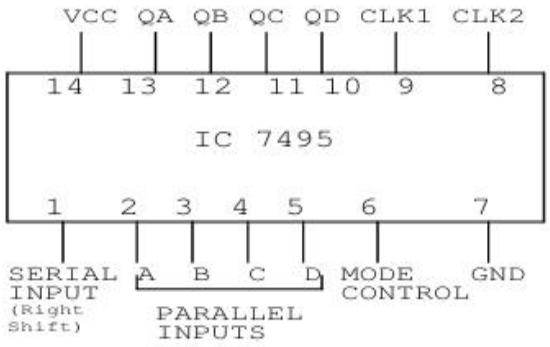
Serial i/p data	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
0	t1	0	X	X	X
1	t2	1	0	X	X
0	t3	0	1	0	X
1	t4	**1**	**0**	**1**	**0**

3) PARALLEL IN PARALLEL OUT (PIPO)

Clock Input Terminal	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
CLK_2	t1	1	0	1	0

4) PARALLEL IN SERIAL OUT (PISO)

Clock Input Terminal	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
CLK_2	t1	1	0	1	0
CLK_2	t2	X	1	0	1
0	t3	X	X	1	0
1	t4	X	X	X	1
X	t5	X	X	X	X



RESULT: The various operations of a shift register is verified

b) Ring counter and Johnson counter

THEORY:

Ring counter is a basic register with direct feedback such that the contents of the register simply circulate around the register when the clock is running. Here the last output that is Q_D in a shift register is connected back to the serial input.

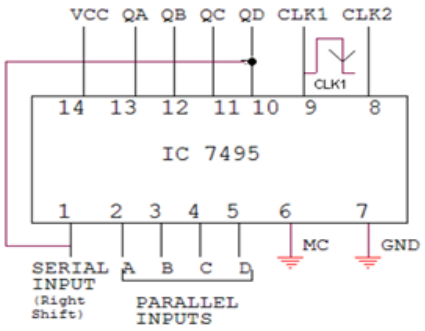
A basic ring counter can be slightly modified to produce another type of shift register counter called Johnson counter. Here complement of last output is connected back to the not gate input and not gate output is connected back to serial input. A four bit Johnson counter gives 8 state output.

PROCEDURE:

- Check all the components for their working.
- Insert the appropriate IC into the IC base.
- Make connections as shown in the circuit diagram.
- Apply clock to pin number 9 and observe the output

CIRCUIT DIAGRAM:

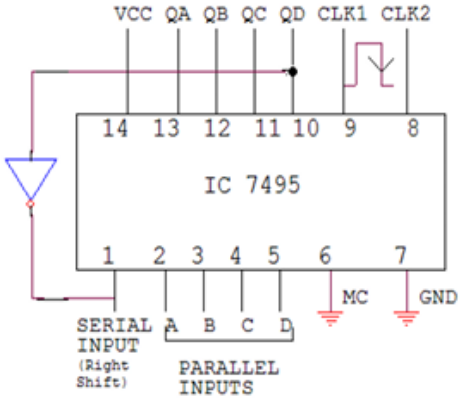
1) RING COUNTER



TRUTH TABLE

Clock pulses	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1
8	1	0	0	0

2) JOHNSON COUNTER



TRUTH TABLE

Clock pulses	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

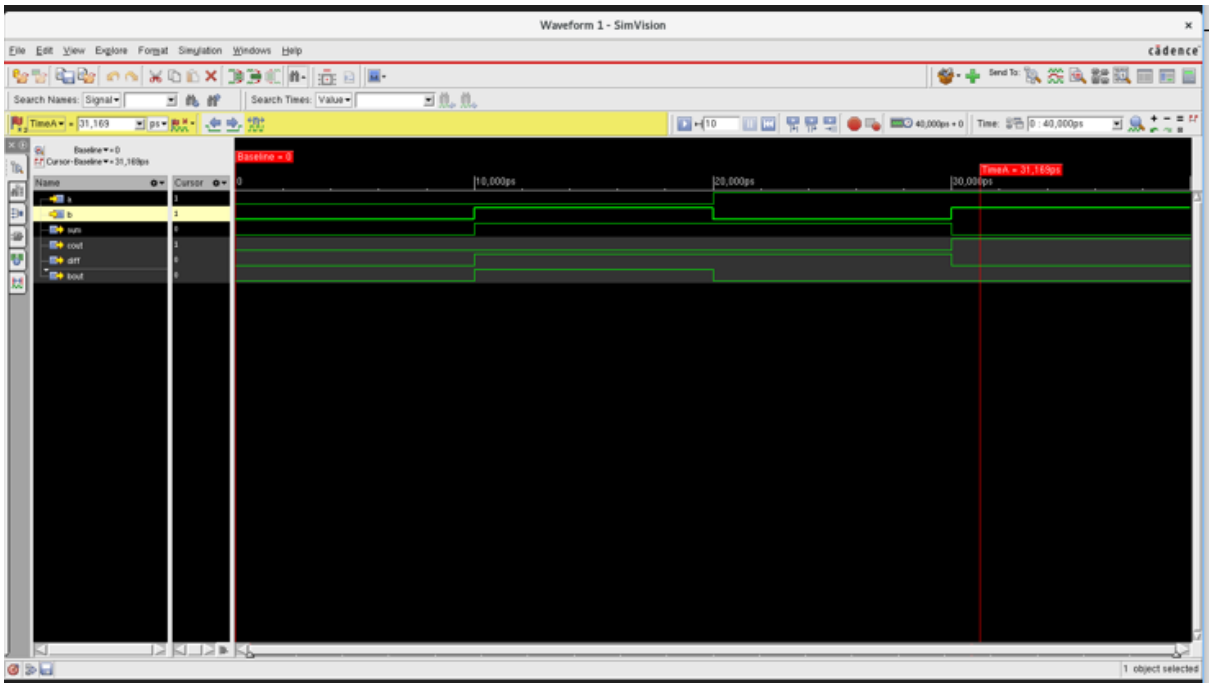
RESULT: The truth table & working of Ring and Johnson counters is verified

Experiments to be realized using Verilog on Cadence Suite

7. To realize Adder/Subtractor (Full/half) circuits using Verilog data flow description

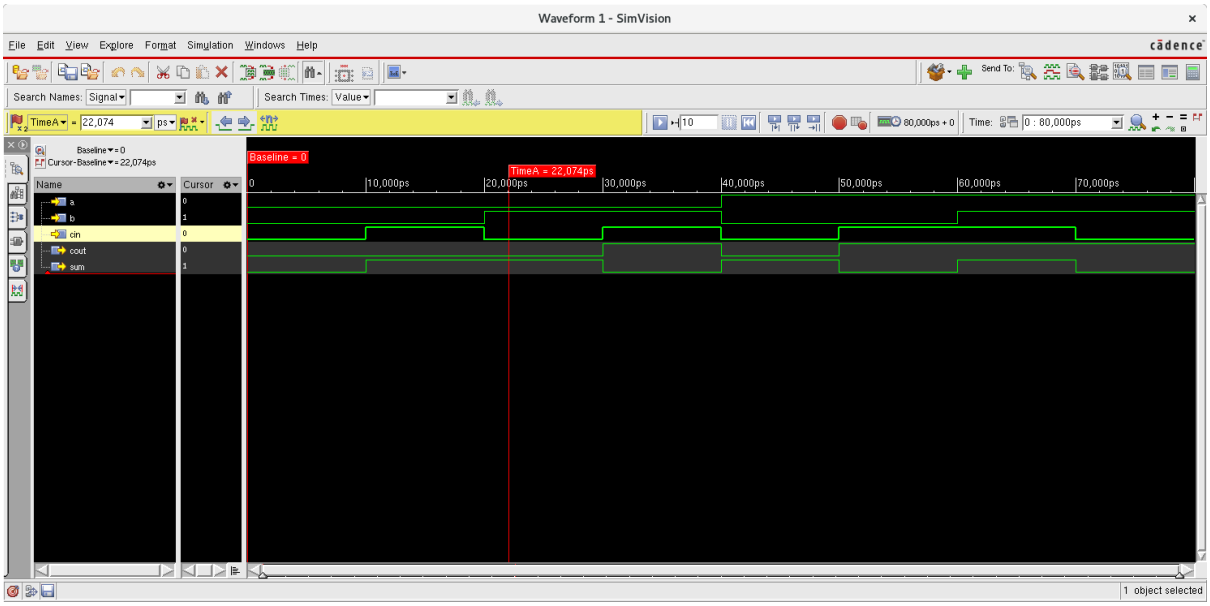
a) Half Adder/ half Subtractor

```
`timescale 1ns/1ps
module ha_hs(a,b,sum,cout,diff,bout);
input a,b;
output sum,cout,diff,bout;
assign sum=a^b;
assign cout=a&b;
assign diff=a^b;
assign bout=(~a)&b;
endmodule
```



b) Full adder

```
`timescale 1ns/1ps
module full_adder(a,b,cin,sum,cout);
input a,b,cin;
output sum,cout;
assign sum=(a^b)^cin;
assign cout=(a&b)|(a&cin)|(b&cin);
endmodule
```



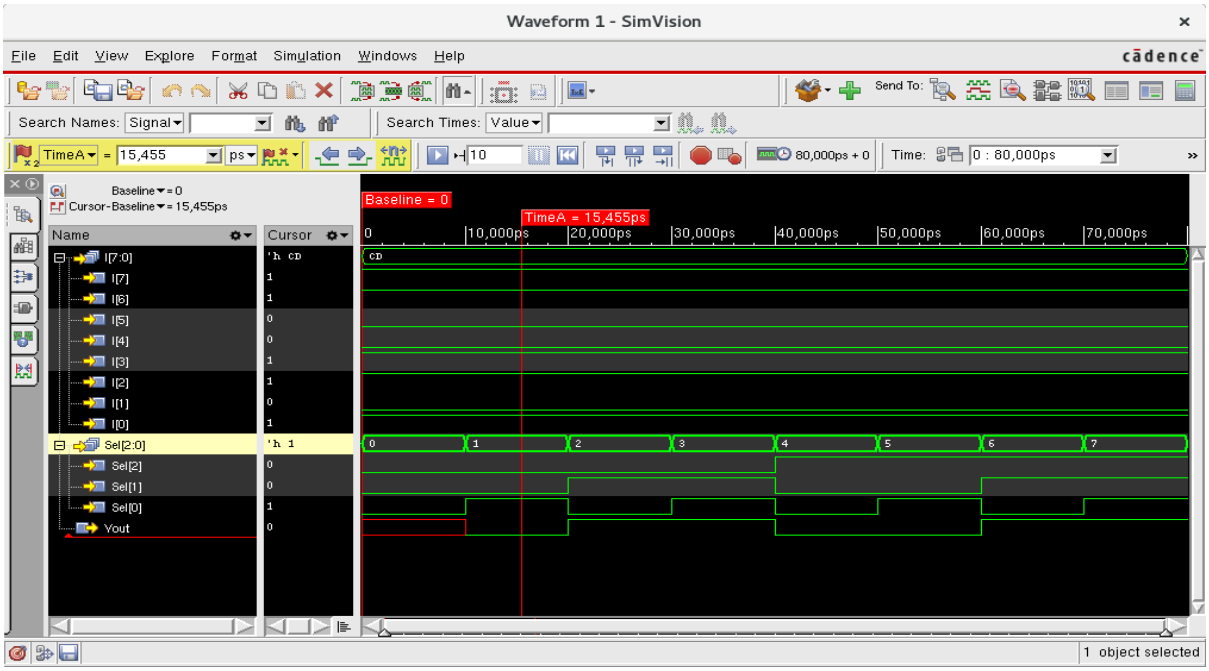
c)Full subtractor

```
`timescale 1ns/1ps
module full_subtractor(a,b,bin,diff,bout);
input a,b,bin;
output diff,bout;
assign diff=(a^b)^bin;
assign bout=(~a&b)|(~a&cin)|(b&cin);
endmodule
```

8. To realize using Verilog Behavioral description: 8:1 mux, 8:3 encoder, Priority Encoder

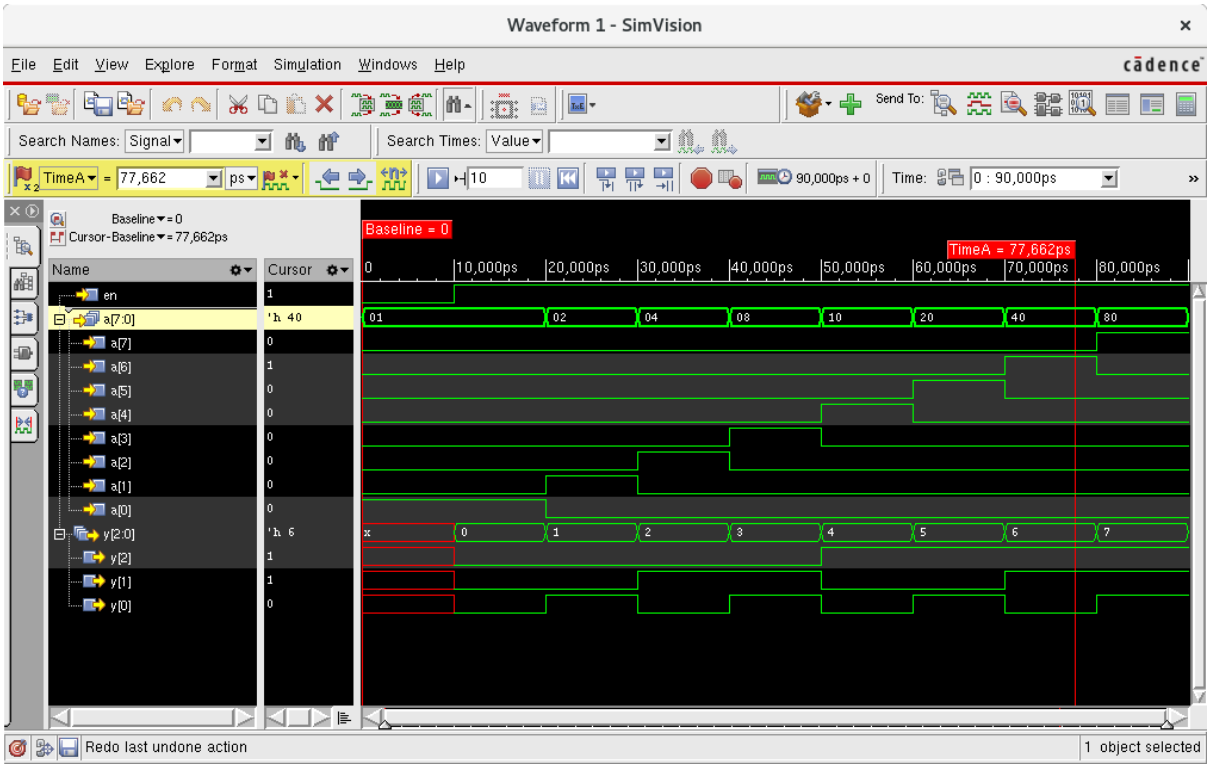
a)8:1 Mux:

```
`timescale 1ns/1ps
module mux(I,Sel,Yout);
input [7:0]I;
input [2:0]Sel;
output Yout;
wire [7:0]I;
wire [2:0]Sel;
reg Yout;
always@(I,Sel)
begin
case(Sel)
3'b000: Yout=I[0];
3'b001: Yout=I[1];
3'b010: Yout=I[2];
3'b011: Yout=I[3];
3'b100: Yout=I[4];
3'b101: Yout=I[5];
3'b110: Yout=I[6];
3'b111: Yout=I[7];
default: Yout=1'b0;
endcase
end
endmodule
```



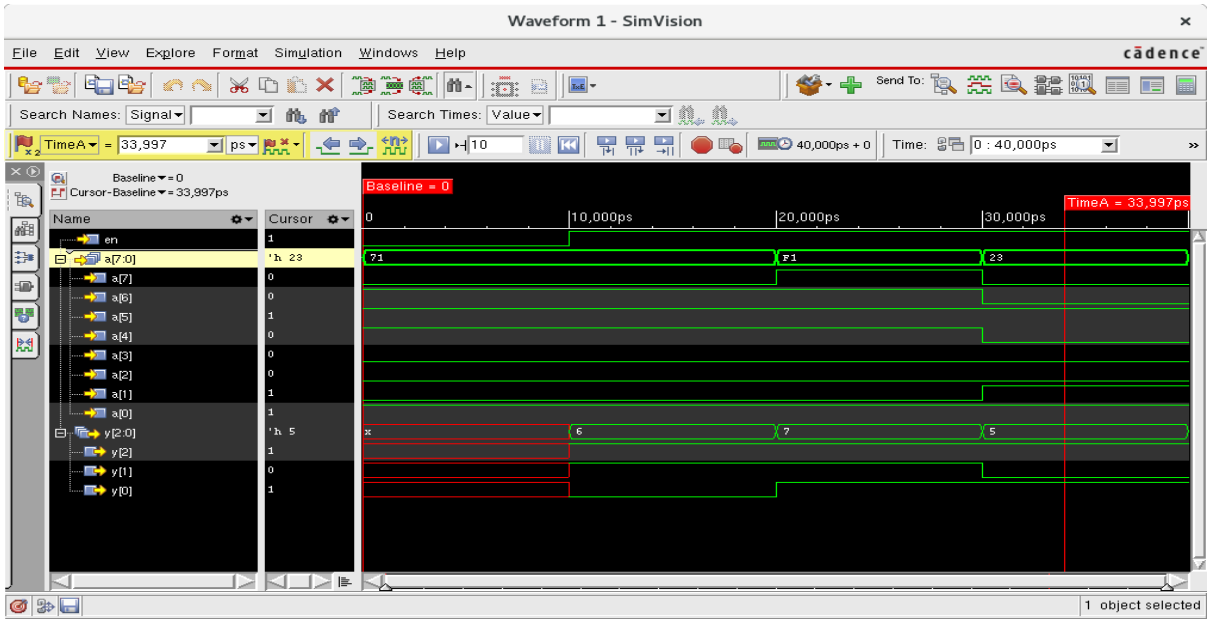
b) 8:3 Encoder

```
`timescale 1ns/1ps
module encoder(en,a,y);
input en;
input [7:0]a;
output [2:0]y;
wire en;
wire [7:0]a;
reg [2:0]y;
always@(en,a)
begin
if(en==1)
case(a)
8'b00000001: y=3'b000;
8'b00000010: y=3'b001;
8'b00000100: y=3'b010;
8'b00001000: y=3'b011;
8'b00010000: y=3'b100;
8'b00100000: y=3'b101;
8'b01000000: y=3'b110;
8'b10000000: y=3'b111;
default: y=3'bz;
endcase
else
y=3'bz;
end
endmodule
```



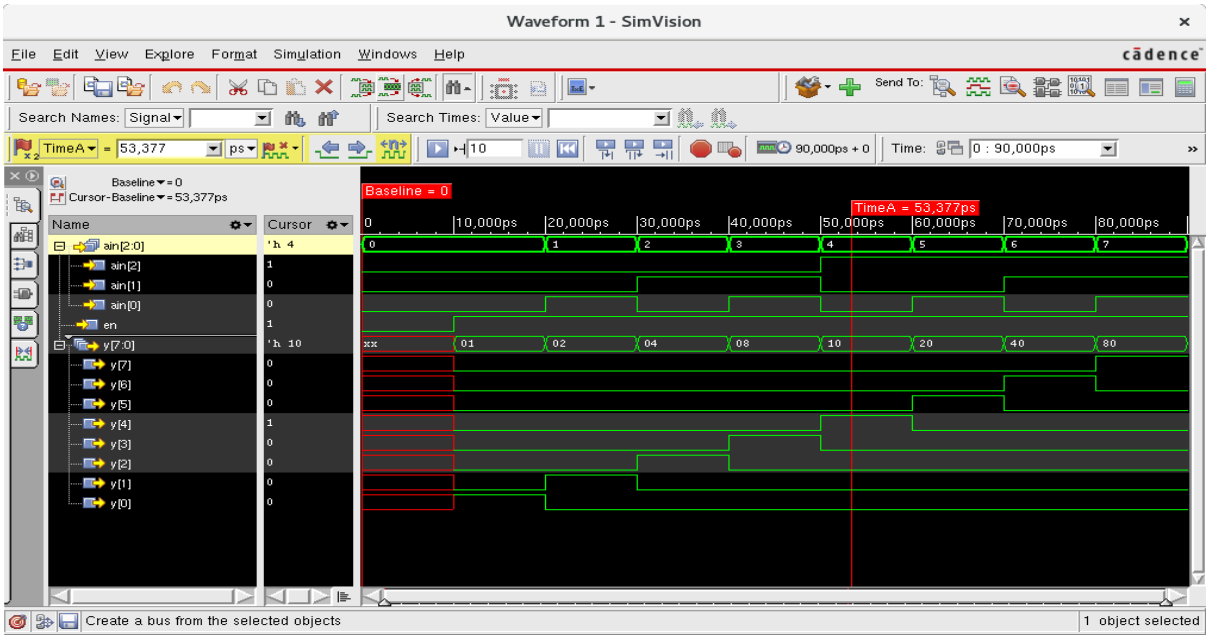
c)8:3 Priority Encoder

```
`timescale 1ns/1ps
module priority_encoder(en,a,y);
input en;
input [7:0]a;
output [2:0]y;
wire en;
wire [7:0]a;
reg [2:0]y;
always@(en,a)
begin
if(en==0)
y=3'bz;
else
begin
if(a[7]==1'b1) y=3'b111;
  else if(a[6]==1'b1) y=3'b110;
  else if(a[5]==1'b1) y=3'b101;
  else if(a[4]==1'b1) y=3'b100;
  else if(a[3]==1'b1) y=3'b011;
  else if(a[2]==1'b1) y=3'b010;
  else if(a[1]==1'b1) y=3'b001;
  else if(a[0]==1'b1) y=3'b000;
  else y=3'bz;
end
end
endmodule
```



d) Decoder

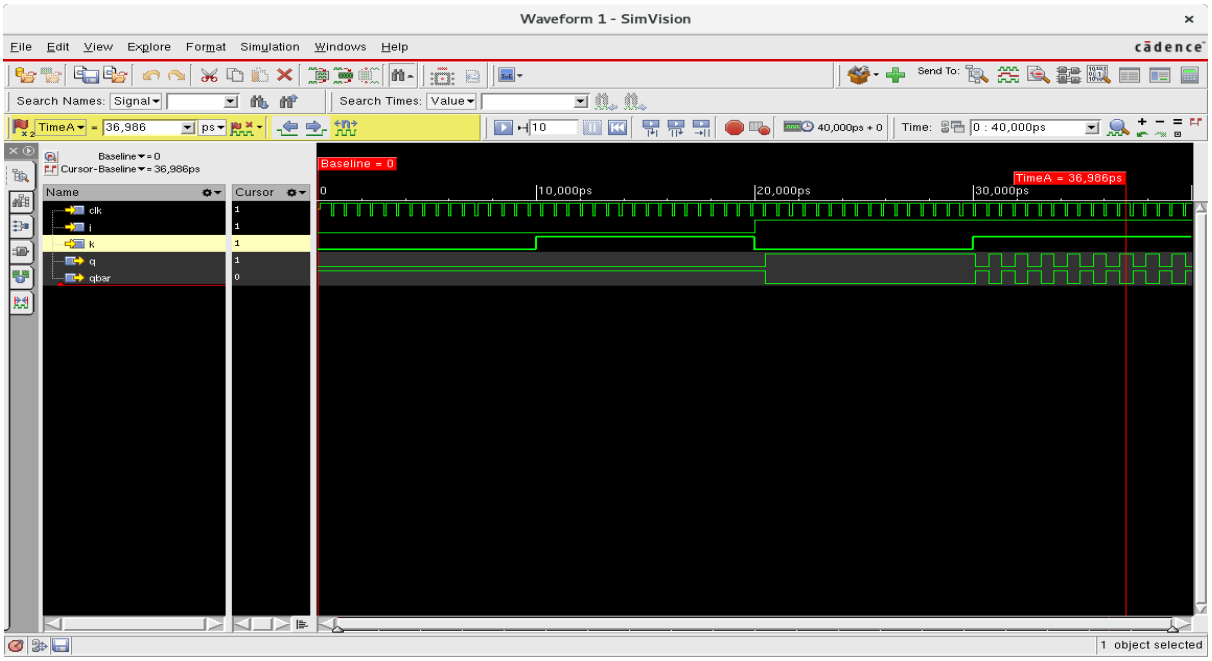
```
`timescale 1ns/1ps
module decoder(en,ain,y);
input en;
input [2:0]ain;
output [7:0]y;
wire en;
wire [2:0]ain;
reg [7:0]y;
always@(en,ain)
begin
if(en==1)
case(ain)
3'b000: y=8'b00000001;
3'b001: y=8'b00000010;
3'b010: y=8'b00000100;
3'b011: y=8'b00001000;
3'b100: y=8'b00010000;
3'b101: y=8'b00100000;
3'b110: y=8'b01000000;
3'b111: y=8'b10000000;
default: y=8'bZ;
endcase
else
y=8'bZ;
end
endmodule
```



9. To realize using Verilog Behavioral description:
Flip-flops: a) JK type b) SR type c) T type and d) D type

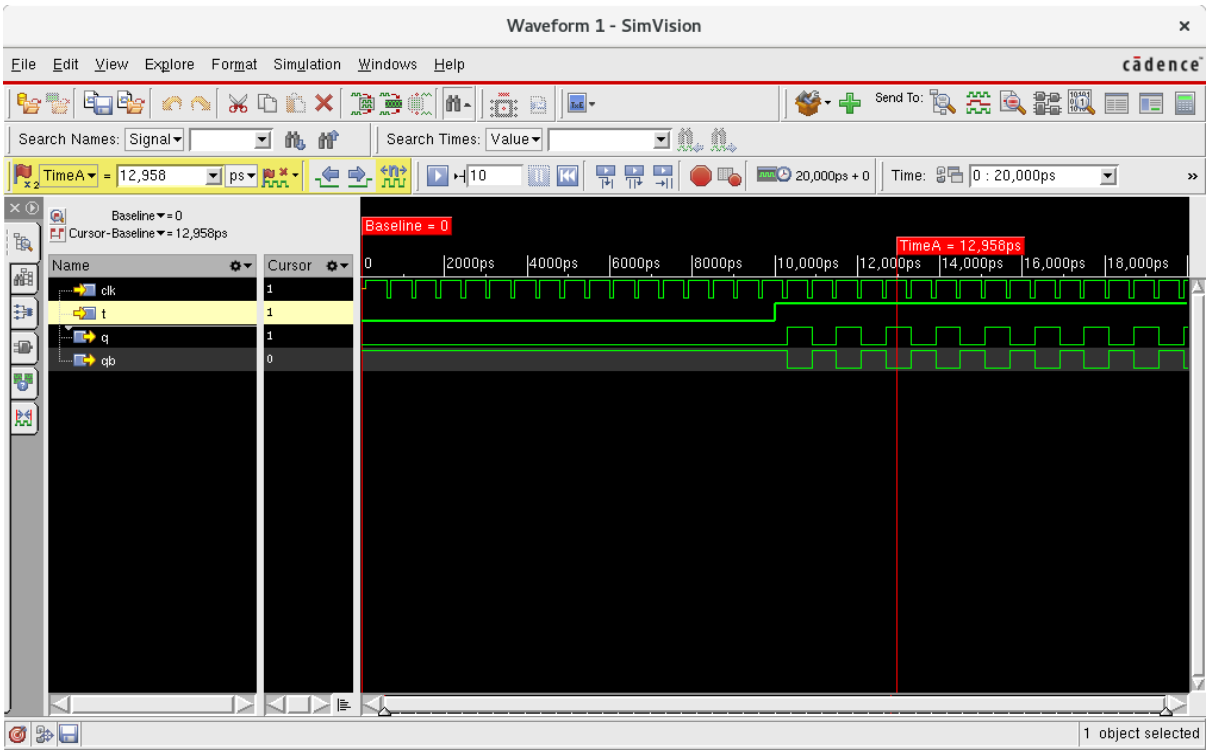
a) JK Flip flop

```
`timescale 1ns/1ps
module jkff(j,k,clk,q,qbar);
input j,k;
input clk;
output q,qbar;
wire j,k;
wire clk;
reg q,qbar;
initial begin
q=1'b0;
qbar=1'b1;
end
always@(posedge clk)
begin
case({j,k})
2'b00: begin q=q; qbar=qbar;end
2'b01: begin q=1'b0; qbar=1'b1; end
2'b10: begin q=1'b1; qbar=1'b0; end
2'b11: begin q=~q; qbar=~qbar; end
default: begin q=1'b0; qbar=1'b1; end
endcase
end
endmodule
```



b)T flip flop

```
`timescale 1ns/1ps
module tff(t,clk,q,qb);
input t,clk;
output q,qb;
wire t,clk;
reg q,qb;
initial
begin
q=1'b0;
qb=1'b1;
end
always@(posedge clk)
begin
case (t)
1'b0: q=q;
1'b1: q=~q;
endcase
qb=~q;
end
endmodule
```



c) `timescale 1ns/1ps
module dff(d,clk,q,qb);
input d,clk;
output q,qb;
wire d,clk;
reg q,qb;

initial
begin
q=1'b0;
qb=1'b1;
end

always@(posedge clk)
begin
q=d;
qb=~q;
end
endmodule

