श्रीचन्द्रशेखरेन्द्रसरस्वतीविश्वमहाविद्यालयः

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

Deemed to be University u/s 3 of UGC Act 1956 | Accredited with "A" grade by NAAC

Enathur, Kanchipuram - 631 561. Tamilnadu, India | www.kanchiuniv.ac.in

Sponsored and run by Sri Kanchi Kamakoti Peetam Charitable Trust

LABORATORY MANUAL
of
COMPUTER AIDED SYSTEM DESIGN LABORATORY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Prepared by: Dr.M.Vinoth
Assistant Professor

Computer Aided System Design Laboratory Manual

B.E (ECE) – FULL TIME

V SEMESTER

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA
(A University U/S 3 of UGC Act 1956)
Enathur, Kanchipuram 631561

LIST OF EXPERIMENTS

1. Verify the characteristics (AC and Transient Analysis) of RLC Circuits by using the PSPICE Software.

2. Verify the characteristics Positive and Negative clamper by using the PSPICE Software.

3. Verify the characteristics Half Wave Rectifier and Full Wave Rectifier (Bridge rectifier) by using the PSPICE Software.

4. Verify the Input and Output characteristics of BJT by using the PSPICE Software.

5. Verify the characteristics Transient analysis of Common Emitter Amplifier by using the PSPICE Software.

6. Verify the Input and Output characteristics of CMOS by using the PSPICE Software

7. Verify the Truth Table for the concept of 4:1 Multiplexer using the PSPICE software.

8. Verify the Truth Table for the concept of Counter Circuit using the PSPICE software.

9. Verify the Truth Table for the concept of Flip-flop Circuit using the PSPICE software.

10. Write the VHDL code for the concept of Basic Logic Gates, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

11. Write the VHDL code for the concept of Full Adder, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

12. Write the VHDL code for the concept of Full Subtractor, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

13. Write the VHDL code for the concept of Multiplexer and Demultiplexer, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

14. Write the Verilog code for the concept of 8 bit Adder (Ripple Carry Adder), simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

15. Write the Verilog code for the concept of Flip-Flops, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

16. Write the Verilog code for the concept of Ripple Counter, Simulate it by using the Xilinx ISE Simulator and implement it with the available FPGA device.

Exp:-1

AC AND TRANSIENT ANALYSIS OF RC CIRCUITS

Aim :

To verify the AC AND TRANSIENT ANALYSIS of a Low Pass and High Pass filter circuits by using the PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

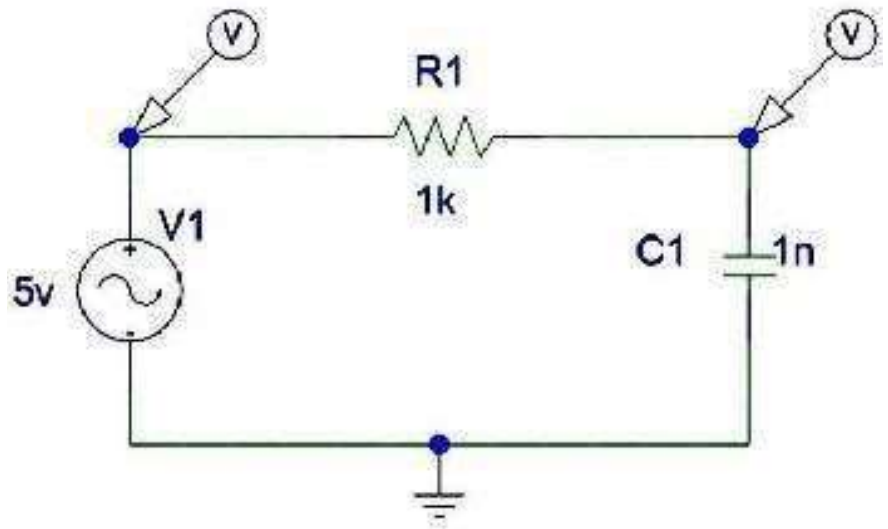
COMPONENTS:

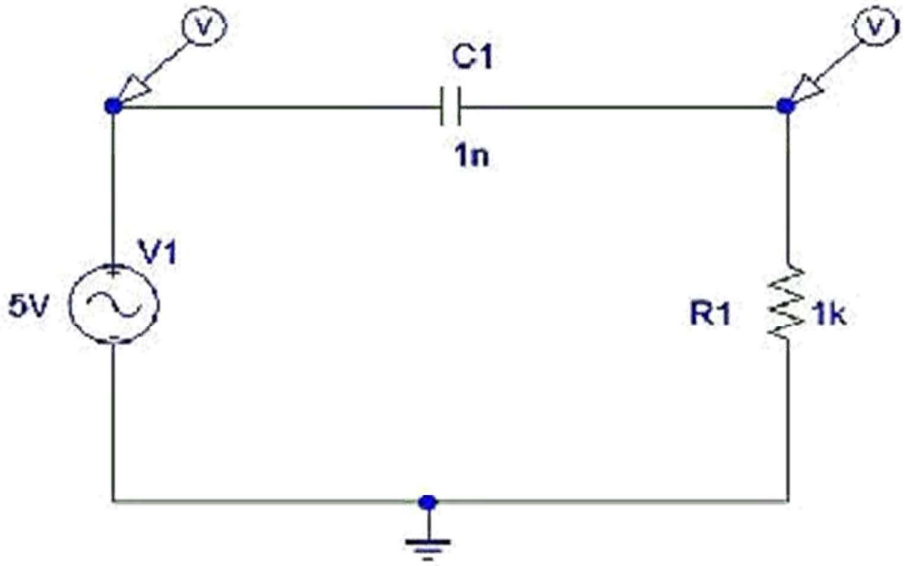
sno	component	Range	quantity
1	AC source(vac)	5v	1
2	resistor	1k	1
3	capacitor	1n	1
4	ground	-	1
5	Voltage probe	-	1
6	vsin	0,5v,1khz	1

PROCEDURE:

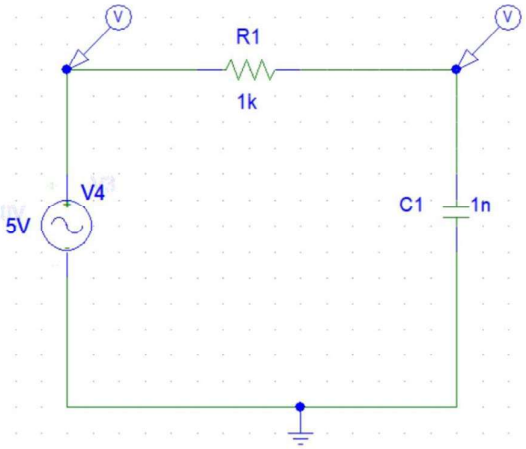
- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For ac analysis choose vsin 0,5v,1khz.
- For transient analysis choose vdc and 5v.
- Give the ac analysis the default values
- For transient analysis give the 0-10ms time.
- Simulate the circuit.

CIRCUIT DIAGRAM:

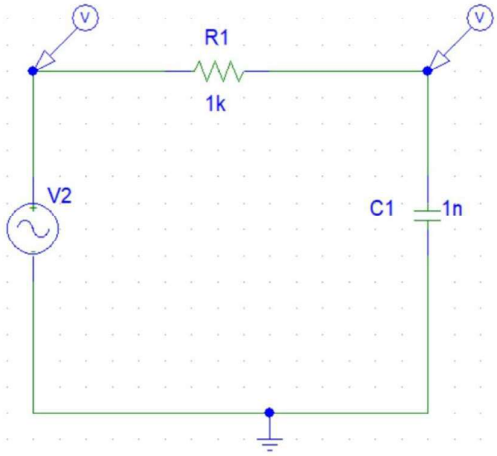




LOW PASS CIRCUIT:

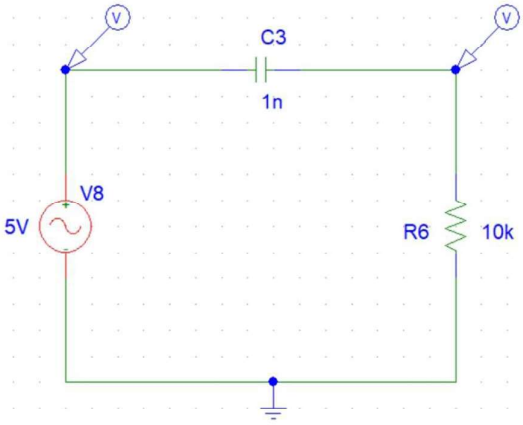


Ac analysis

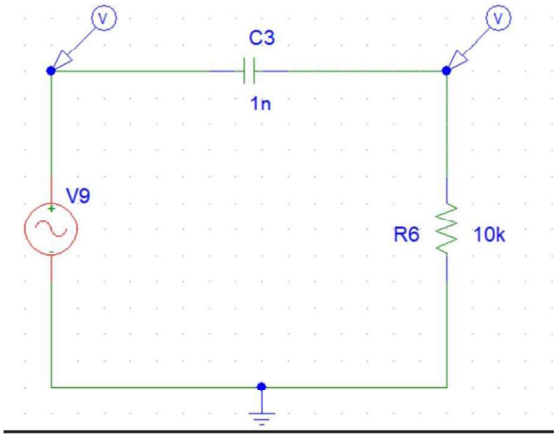


transient analysis

HIGH PASS CIRCUIT:

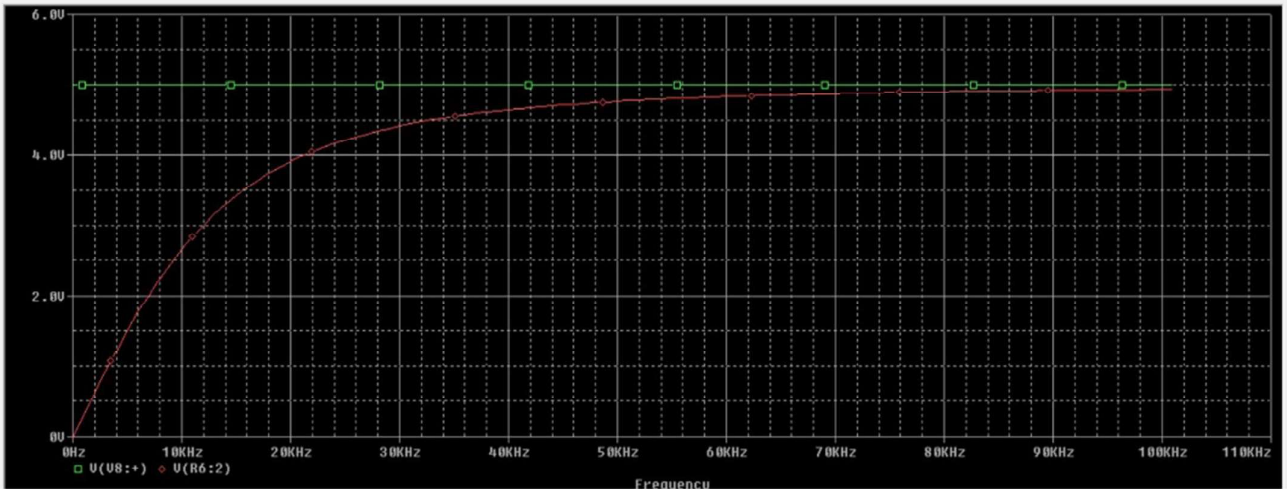


Ac analysis

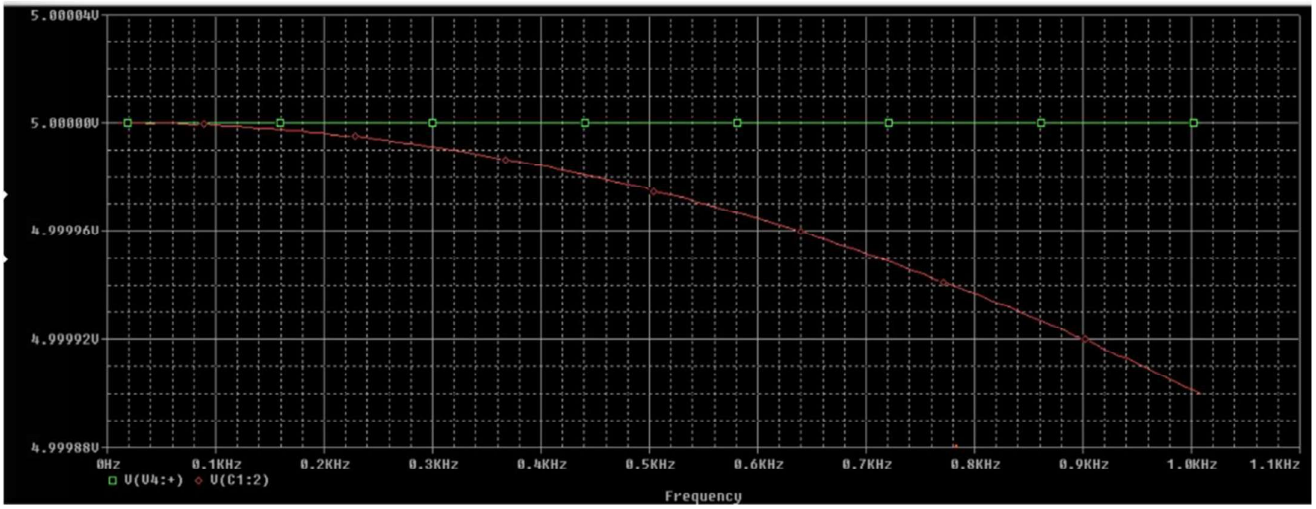


transient analysis

Ac Analysis

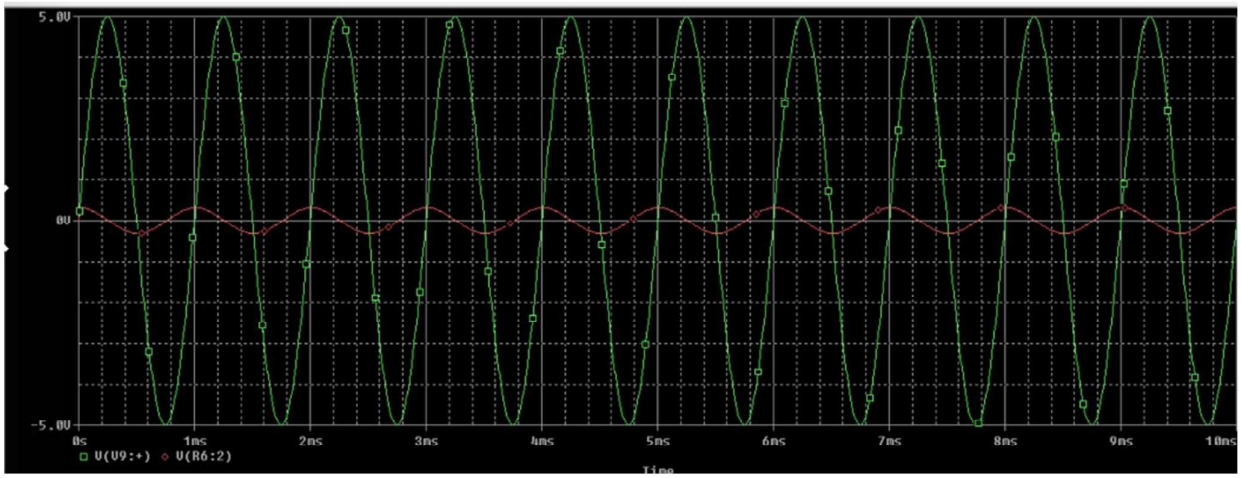


High pass filter

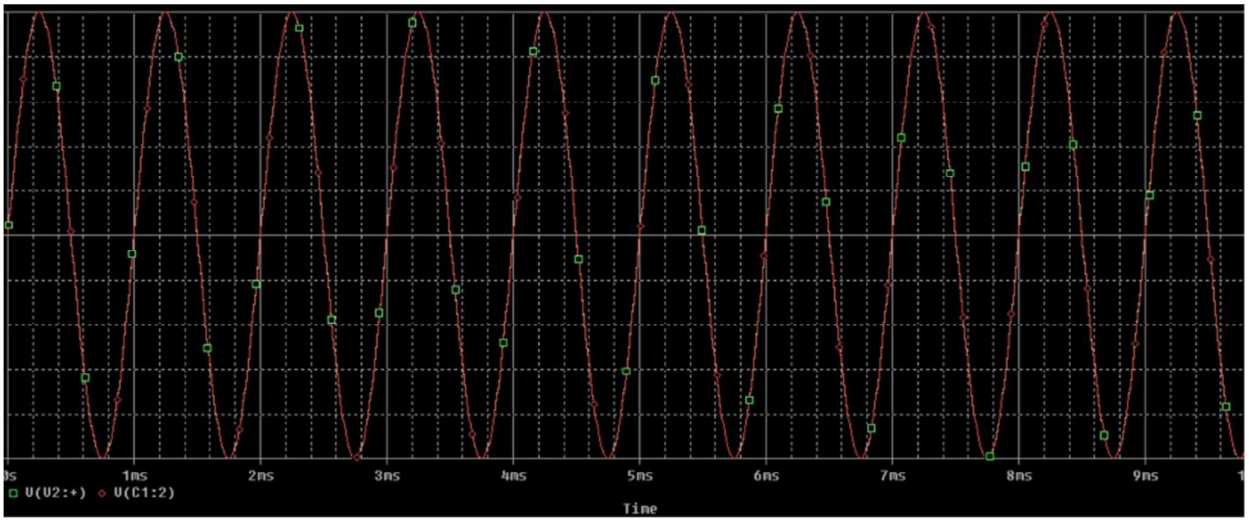


Low pass filter

Transient Analysis



HIGH PASS FILTER



Low pass filter

Low pass filter code:-

**** Analysis setup ****
.ac LIN 101 10 1.00K
.tran 0ns 10m 0 0.01ms

.OP

*** From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:**
.lib "nom.lib"

.INC "low pass filter.net"

****** INCLUDING "low pass filter.net" ******
*** Schematics Netlist ***

R_R1 $N_0002 $N_0001 1k
C_C1 0 $N_0001 1n
V_V4 $N_0002 0 DC 0V AC 5V 0

****** RESUMING "low pass filter.cir" ******
.PROBE V(*) I(*) W(*) D(*) NOISE(*)

.END
High pass filter code:-
**** Analysis setup ****
.tran 0ns 10m 0 0.01ms
.ac LIN 101 10 1.00K

.OP
.STMLIB "low pass filter.stl"

*** From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:**
.lib "nom.lib"

.INC "high pass filter.net"

****** INCLUDING "high pass filter.net" ******
*** Schematics Netlist ***

C_C3 $N_0001 $N_0002 1n
R_R6 0 $N_0002 10k

```
V_V9         $N_0001 0
+SIN 0 5 1khz 0 0 0

**** RESUMING "high pass filter.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

RESULT:

Thus the characteristics of Low Pass and High Pass filter circuit have been verified .

Exp : 2

POSITIVE AND NEGATIVE CLAMPER

Aim:

To verify the characteristics of a POSITIVE AND NEGATIVE CLAMPER circuits by using PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

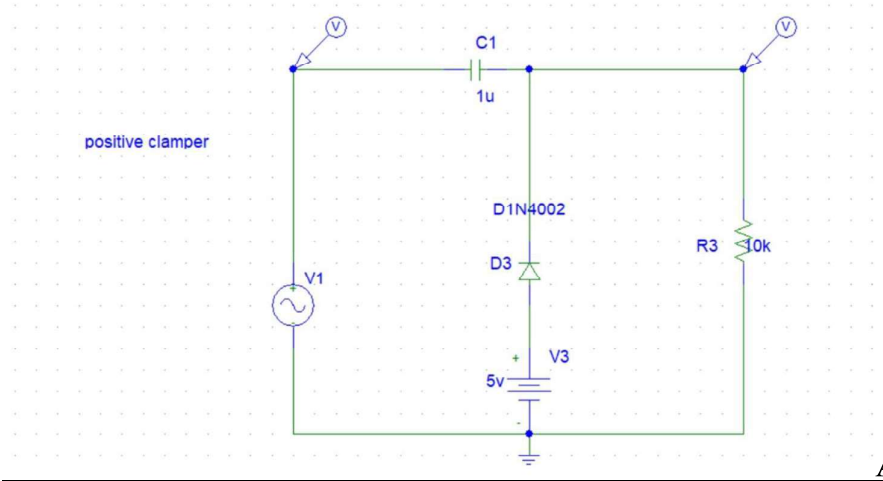
COMPONENTS:-

sno	component	Range	quantity
1	vsin	0,5v,1khz	1
2	resistor	10k	1
3	capacitor	1u	1
4	ground	-	1
5	Voltage probe	-	1
6	diode	D1n4002	1
7	vdc	5v	1

PROCEDURE:

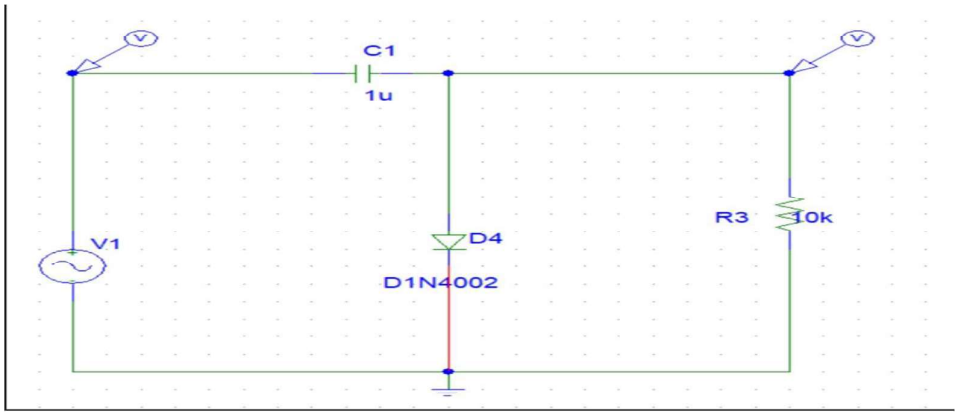
- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For transient analysis choose vac and 5v.
- For transient analysis give the 0-10ms time.
- Simulate the circuit.

CIRCUIT DIAGRAMS:



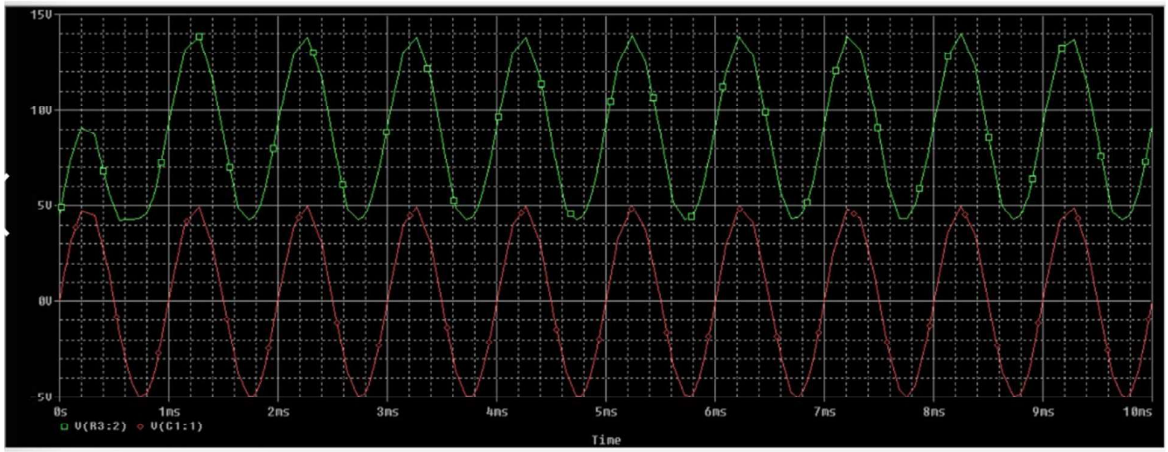
AM:

POSITIVE CLAMPER

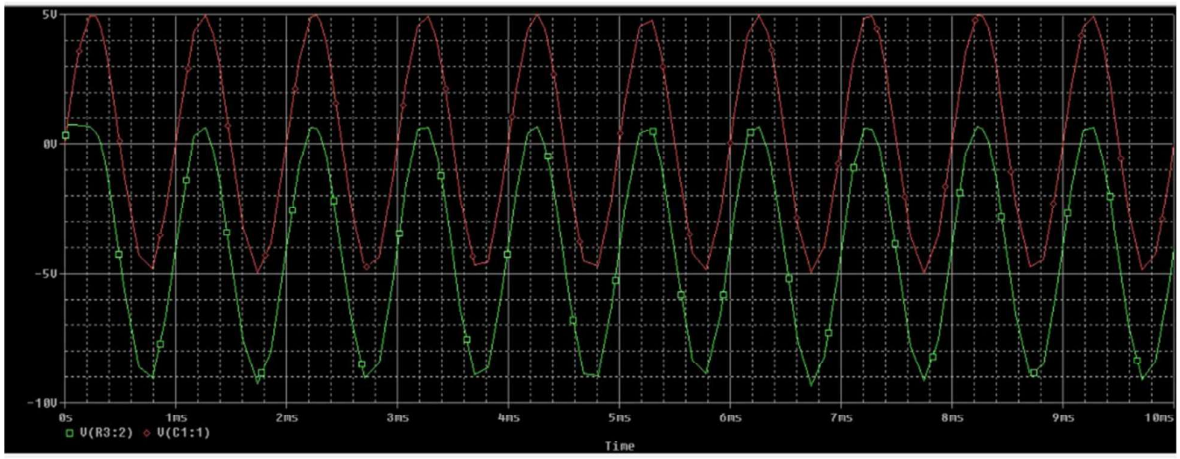


NEGATIVE CLAMPER

WAVEFORMS:



POSITIVE CLAMPRER



NEGATIVE CLAMPER

Positive clamper code:-

```
** Analysis setup **
.tran 0ns 10ms
.OP


* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:
.lib "nom.lib"

.INC "1628574735364_positive clamper.1.net"



**** INCLUDING "1628574735364_positive clamper.1.net" ****
* Schematics Netlist *


D_D3          $N_0001 $N_0002 D1N4002
C_C1          $N_0003 $N_0002  1u
V_V3          $N_0001 0 5v
V_V1          $N_0003 0
+SIN 0v 5v 1khz 0 0 0
R_R3          0 $N_0002  10k

**** RESUMING "1628574735364_positive clamper.1.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)
.END
```

Negative clamper code:-

```
** Analysis setup **
.tran 0ns 10ms
.OP


* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:
.lib "nom.lib"

.INC "1628574735364_negative clamper.1.net"



**** INCLUDING "1628574735364_negative clamper.1.net" ****
* Schematics Netlist *



C_C1         $N_0001 $N_0002  1u
V_V1         $N_0001 0
+SIN 0v 5v 1khz 0 0 0
R_R3         0 $N_0002  10k
D_D4         $N_0002 0 D1N4002

**** RESUMING "1628574735364_positive clamper.1.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

RESULT:

Thus the characteristics of positive and negative clamper have been verified .

Exp : 3

HALF WAVE ANF FULL WAVE RECTIFIER

Aim:

To verify the characteristics of a HALF WAVE ANF FULL WAVE RECTIFIER (BRIDGE RECTIFIER) circuits by using PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

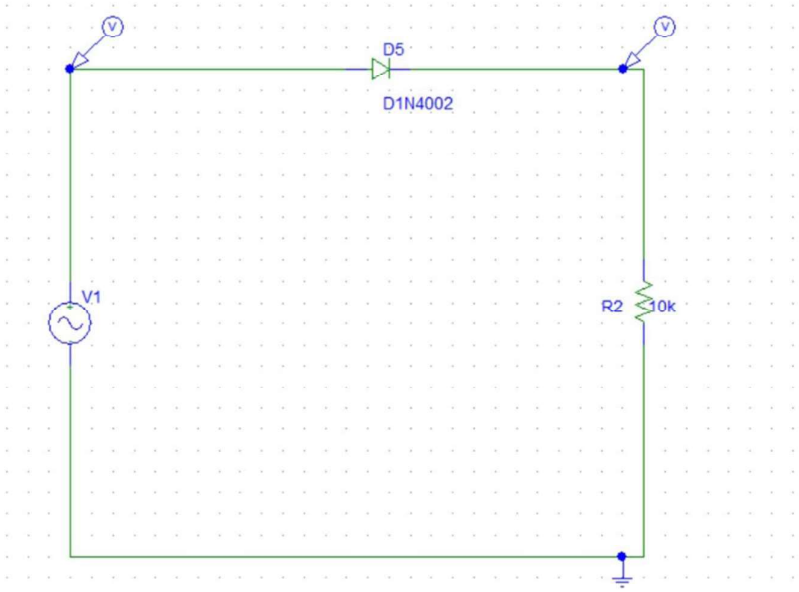
COMPONENTS:-

sno	component	Range	quantity
1	vsin	0,5v,1khz	1
2	resistor	10k,1k	1 each
3	capacitor	1u	1
4	ground	-	1
5	Voltage probe	-	1
6	diode	D1n4002	4

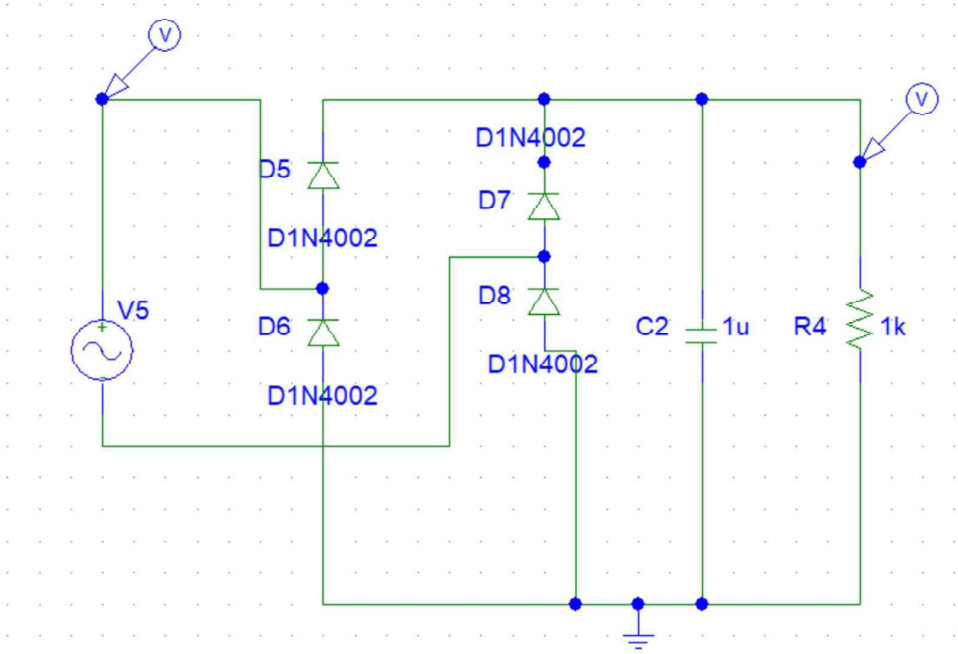
PROCEDURE:

- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For transient analysis choose vdc and 5v.
- For transient analysis give the 0-10ms time.
- Simulate the circuit.

CIRCUIT DIAGRAMS:

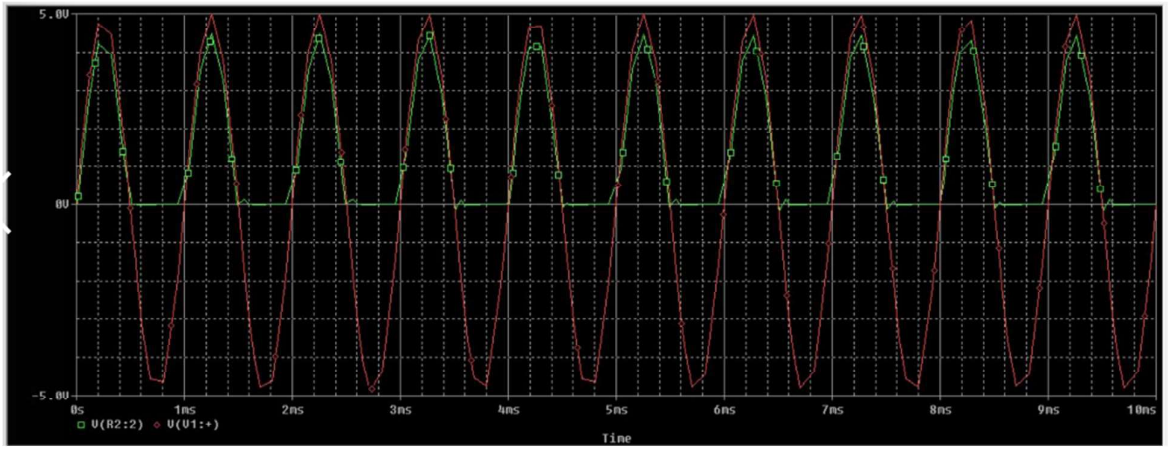


HALF WAVE RECTIFIER

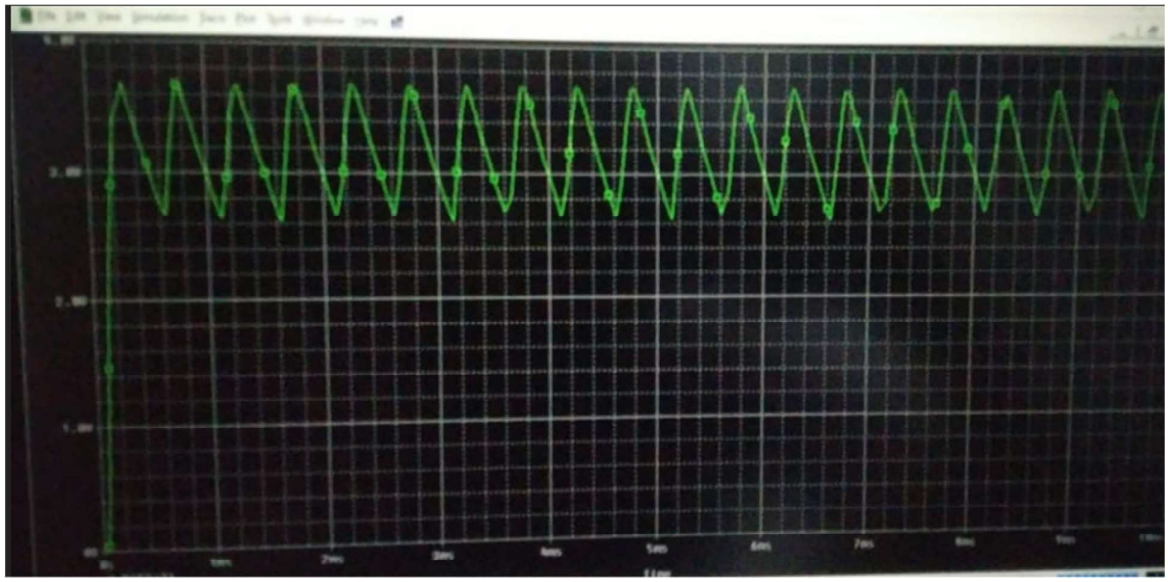


FULL WAVE RECTIFIER

WAVEFORMS:



HALF WAVE RECTIFIER



FULL WAVE BRIDGE RECTIFIER

Halfwave rectifier code:-

** Analysis setup **

.tran 0ns 10ms

.OP

* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:

.lib "nom.lib"

.INC "half wave rectifier (1).net"

```
**** INCLUDING "half wave rectifier (1).net" ****
* Schematics Netlist *



D_D5         $N_0001 $N_0002 D1N4002
V_V1         $N_0001 0
+SIN 0v 5v 1khz 0 0 0
R_R2         0 $N_0002  10k

**** RESUMING "half wave rectifier (1).cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)



.END
```

Full wave rectifier code:-

```
** Analysis setup **
.tran 0ms 10ms
.OP


* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:
.lib "nom.lib"

.INC "bridge rectifier.net"



**** INCLUDING "bridge rectifier.net" ****
* Schematics Netlist *



V_V5         $N_0001 $N_0002
+SIN 0V 5V 1KHZ 0 0 0
D_D6         0 $N_0001 D1N4002
D_D5         $N_0001 $N_0003 D1N4002
R_R4         0 $N_0003  1k
```

```
C_C2         0 $N_0003  1u
D_D7         $N_0002 $N_0003 D1N4002
D_D8         0 $N_0002 D1N4002

**** RESUMING "positive clamper.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

RESULT:

Thus the characteristics of half wave and fullwave rectifier have been verified .

Exp : 4

WEIN BRIDGE OSCILLATOR

Aim:

To verify the characteristics of a WEIN BRIDGE OSCILLATOR circuit by using PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

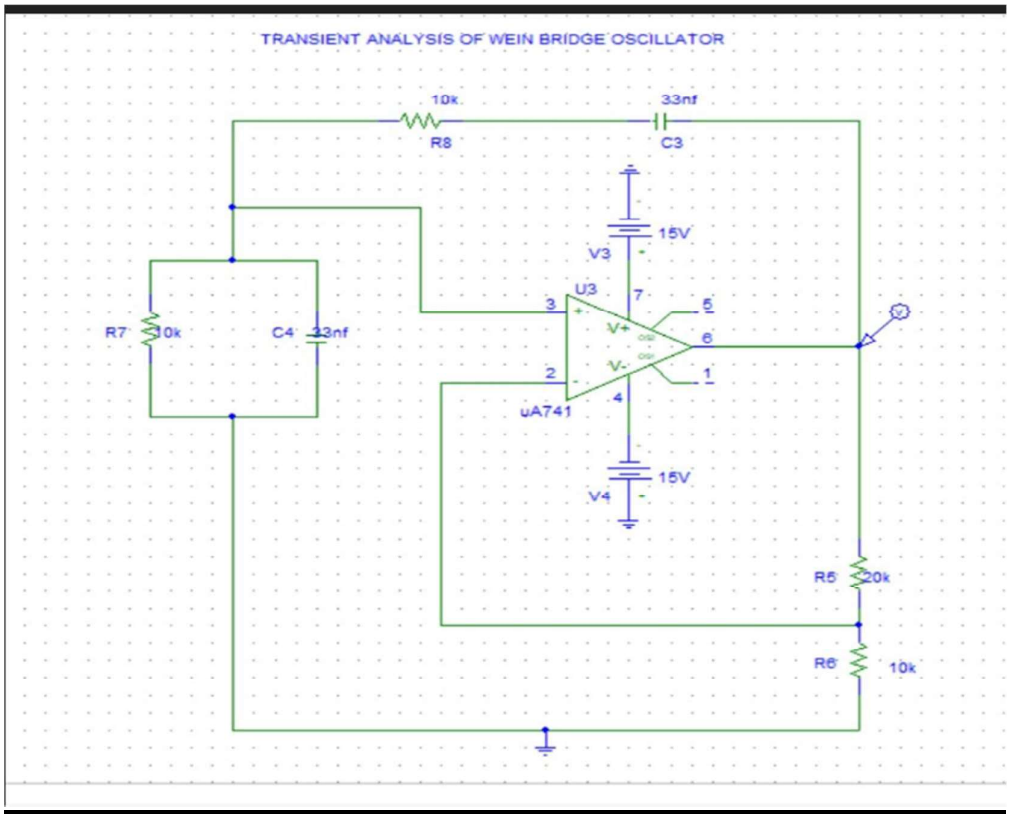
COMPONENTS:-

sno	component	Range	quantity
1	resistor	10k,20k	3,1
2	capacitor	33nf	2
3	ground	-	1
4	Voltage probe	-	1
5	amplifier	Ua741	1
6	vdc	15v	2

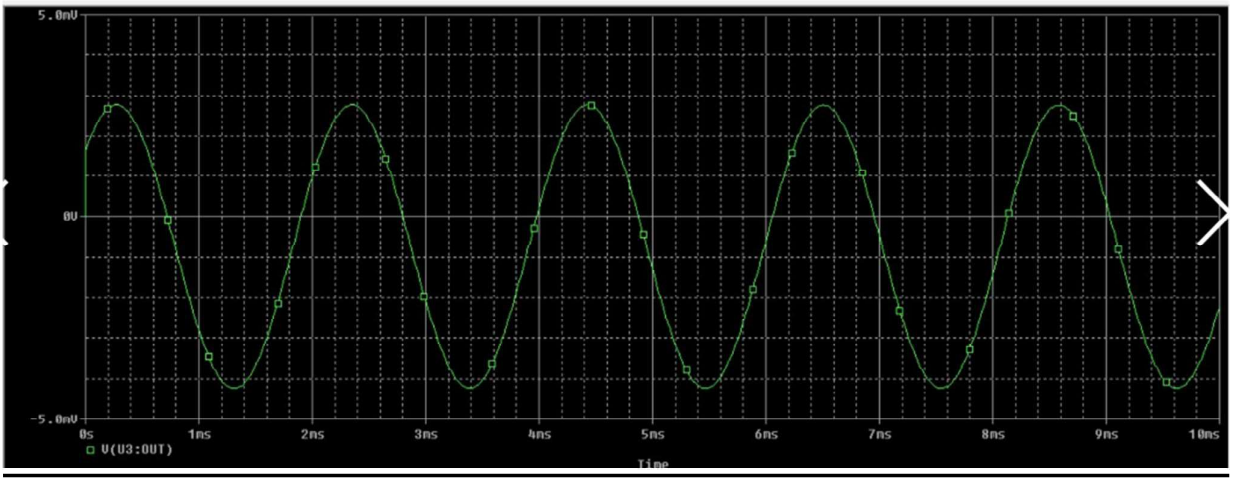
PROCEDURE:

- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For transient analysis choose vdc and 5v.
- For transient analysis give the 0-10ms time.
- Set to skip the initial conditions in transient setup.
- Simulate the circuit.

CIRCUIT DIAGRAMS:



Wave forms:-



Code:-

```
** Analysis setup **
.tran 0ns 10ms 0 0.01ms SKIPBP
.OP
```

```
* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:
.lib "nom.lib"

.INC "wein bridge oscillator.net"



**** INCLUDING "wein bridge oscillator.net" ****
* Schematics Netlist *



C_C2        $N_0001 $N_0002  33nf
R_R3        $N_0003 $N_0001  20k
R_R2        0 $N_0003  10k
C_C1        0 $N_0004  33nf
R_R1        0 $N_0004  10k
X_U2        $N_0004 $N_0003 $N_0005 $N_0006 $N_0001 uA741
R_R4        $N_0002 $N_0004  10k
V_V1        $N_0005 0 15V
V_V2        0 $N_0006 15V

**** RESUMING "wein bridge oscillator.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

RESULT:

Thus the characteristics of wein bridge oscillator have been verified

Exp : 5

INVERTING & NON-INVERTING AMPLIFIER

Aim:

To verify the characteristics of a Inverting & Non Inverting Amplifier circuits by using PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

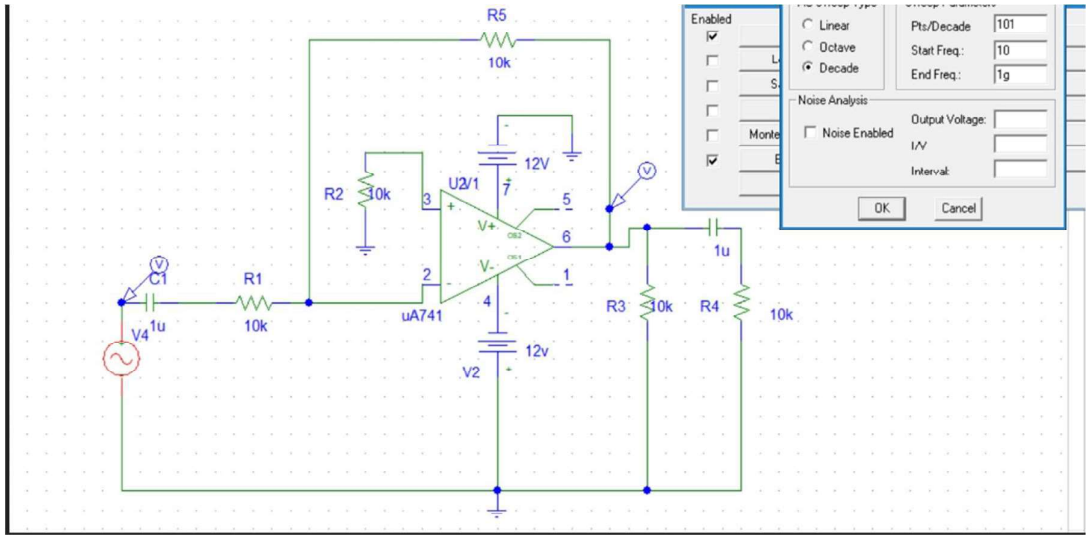
COMPONENTS:-

sno	component	Range	quantity
1	AC source(vac)	5v	1
2	resistor	10k	5
3	capacitor	1u	2
4	ground	-	1
5	Voltage probe	-	1
6	amplifier	Ua741	1
7	vdc	12v	2
6	vsin	0,5v,1khz	1

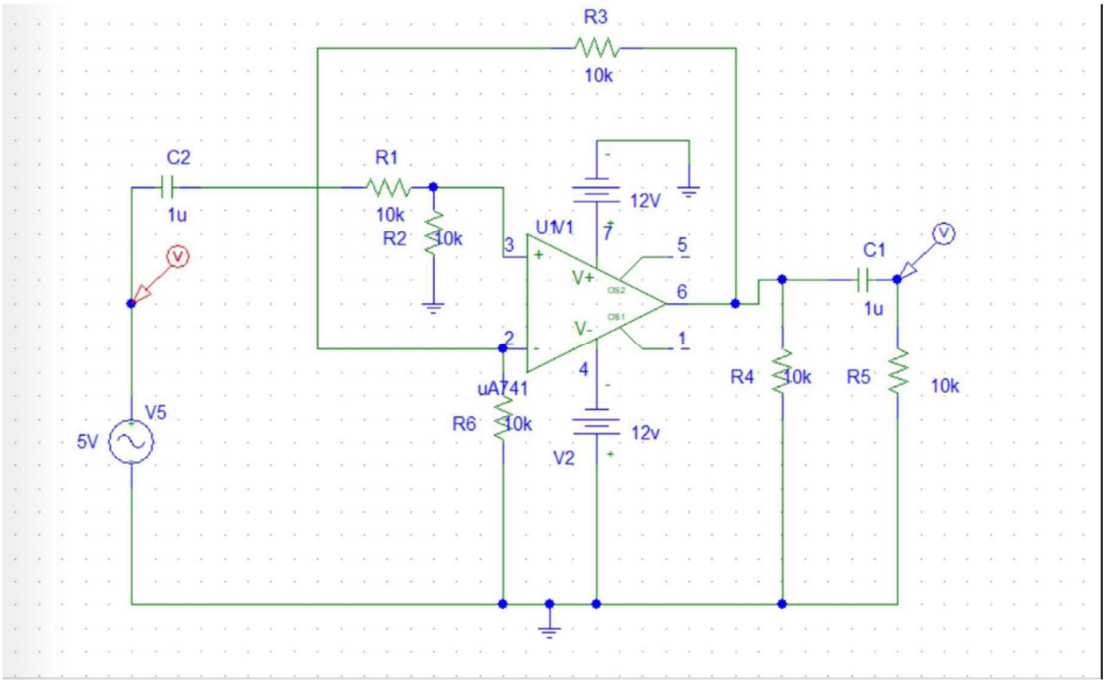
PROCEDURE:

- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For transient analysis choose vdc and 5v.
- For ac analysis choose vsin 0,5v,1khz.
- He ac analysis is for voltage and in decade mode and settings shown in figure.
- For transient analysis give the 0-10ms time.
- Simulate the circuit.

Circuit diagrams:-



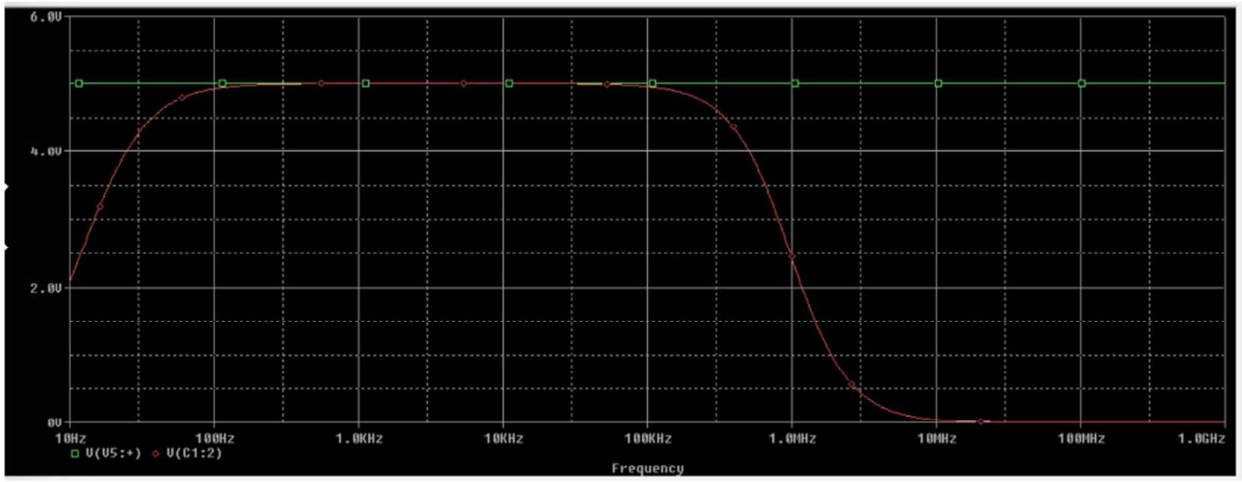
INVERTING AMPLIFIER CIRCUIT



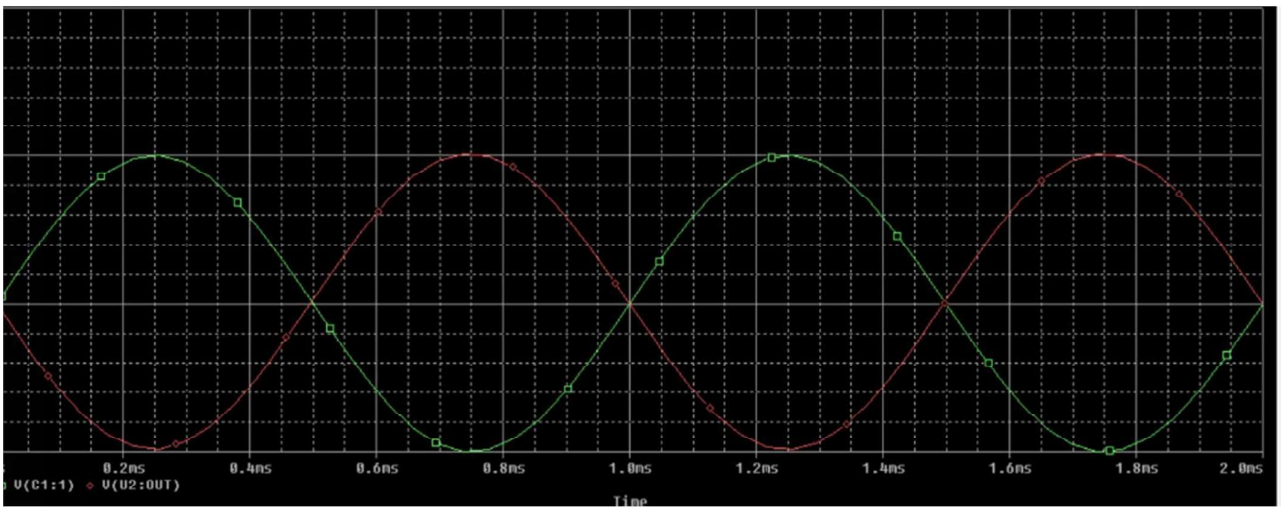
NON-INVERTING AMPLIFIER

OUTPUT OF INVERTING AMPLIFIER:-

Ac analysis

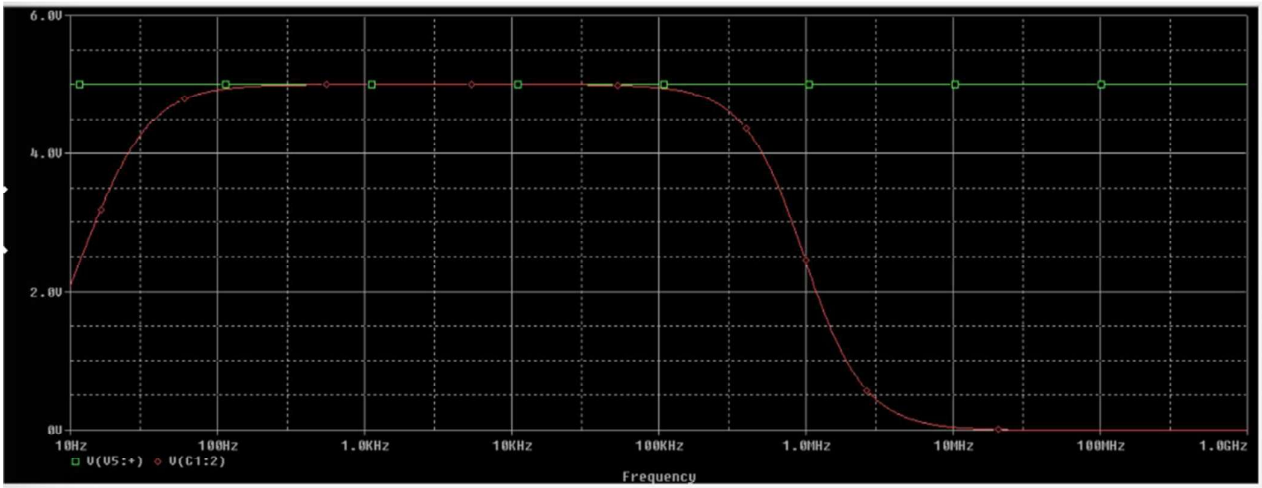


Transient analysis for inverting amplifier :-

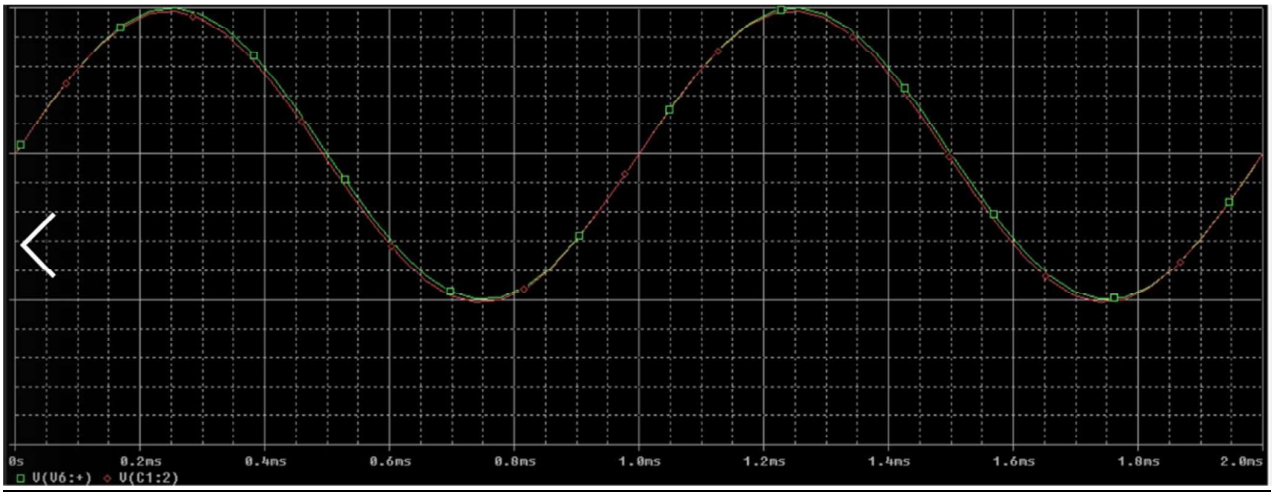


NON INVERTING AMPLIFIER:-

Ac analysis



Transient analysis:-



CIRCUIT FILE:

INV OP (AC & TRANSIENT)

VIN 1 0 AC 1.5V PWL(0 0 1NS -1V 1MS -1V 1.0001MS 1V 2MS 1V 2.0001MS -1V 3MS -1V 3.0001MS 1V 4MS 1V)

R1 1 2 2.5K

RF2310K

RL 3 0 100K

XA1 2 0 3 0 OPAMP

.SUBCKT OPAMP 1 2 7 4

RI 1 2 2.0E6

GB43120.1M

R13410K

C1 3 4 1.5619UF

EA45342E+5

RO5775

.ENDS

.PROBE

.AC DEC 50 100 1E9

.PLOT AC VP(3) VM(3) 0,250

.TRAN 5e-005 4MS 0

.TEMP 27

.PLOT

TRAN V(3) V(1) -10,15

.END

RESULT: Thus the characteristics of Inverting and Non - inverting circuit have been verified.

Exp:-6

INPUT AND OUTPUT CHARACTERISTICS OF BJT

Aim :

To verify the input and output characteristics of BJT by using the PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

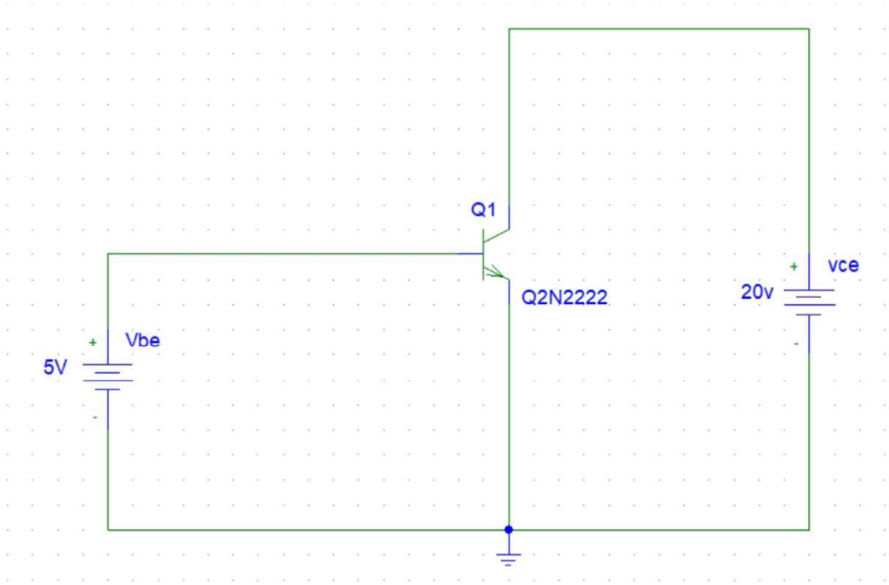
COMPONENTS:-

sno	component	Range	quantity
1	ground	-	1
2	bjt	Q2n222	1
3	vdc	5v,20v	1 each

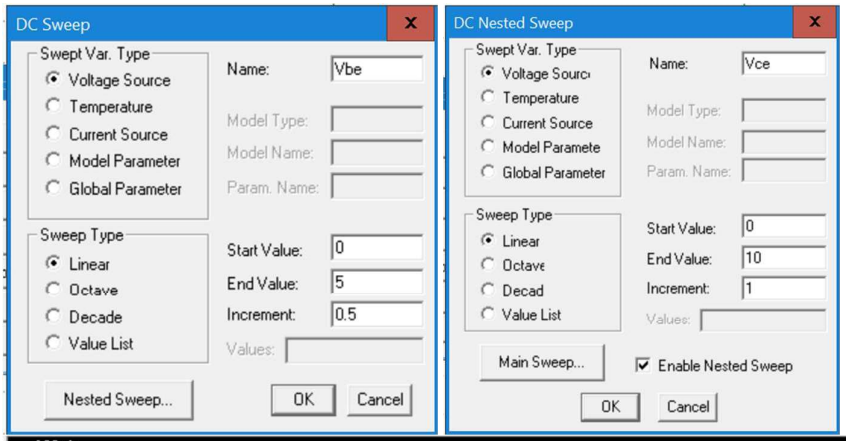
PROCEDURE:

- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- Perform dc sweep in the bjt and nested sweep for input and output values.
- For input characteristics set the trace to IB(q1).
- For output characteristics set the trace to IC(q1).
- The dc sweep values are in figure.
- Simulate the circuit.

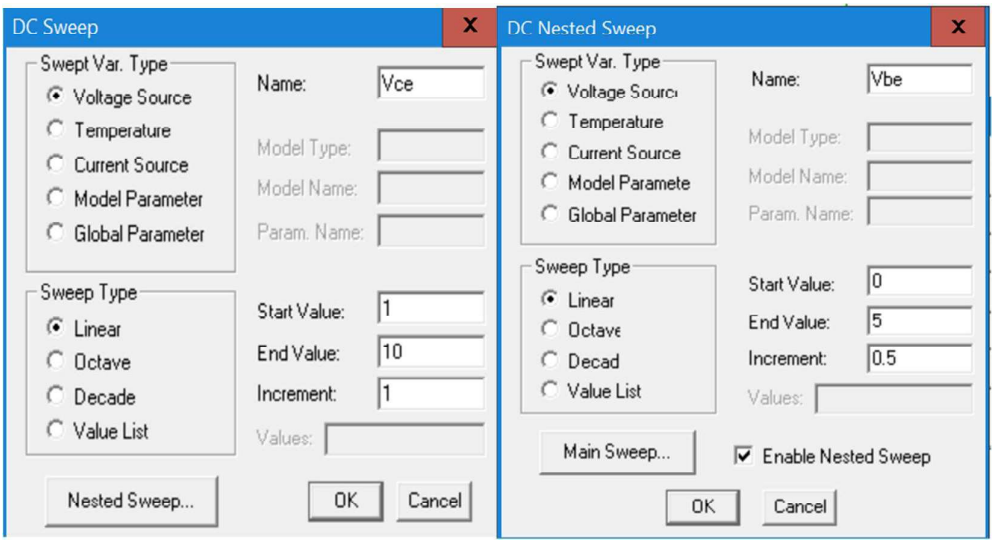
CIRCUIT DIAGRAMS:

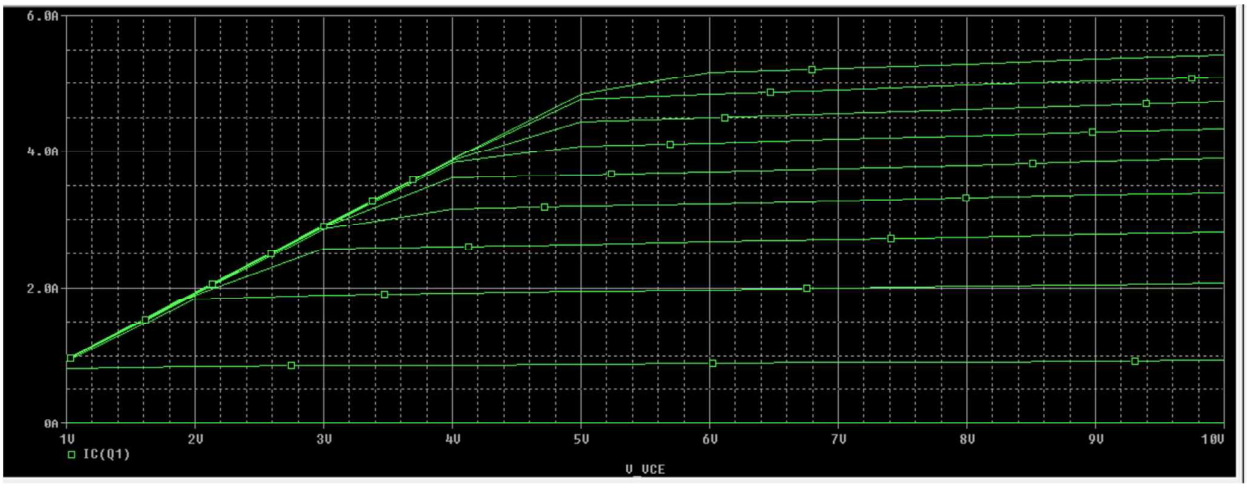


BJT INPUT DC SWEEP VALUES



Output characteristics:-





Code:-

**** Analysis setup ****

.tran 0ns 2ms

.OP

*** From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:**

.lib "nom.lib"

.INC "bjt input.net"

.PROBE V(*) I(*) W(*) D(*) NOISE(*)

```
.END
Output code:-
** Analysis setup **
.DC LIN V_VCE 1 10 1
+ LIN V_VBE 0 5 0.5
.DC LIN V_VBE 0 5 0.5
+ LIN V_VCE 1 10 1

.OP


* From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:
.lib "nom.lib"

.INC "BJT output characteristics.net"


.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

Result:- Thus the input and output characteristics of BJT have been verified.

Exp:-7

TRANSIENT ANALYSIS OF COMMON EMITTER PNP AMPLIFIER

Aim :

To verify the transient analysis of common emitter pnp amplifier by using the PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

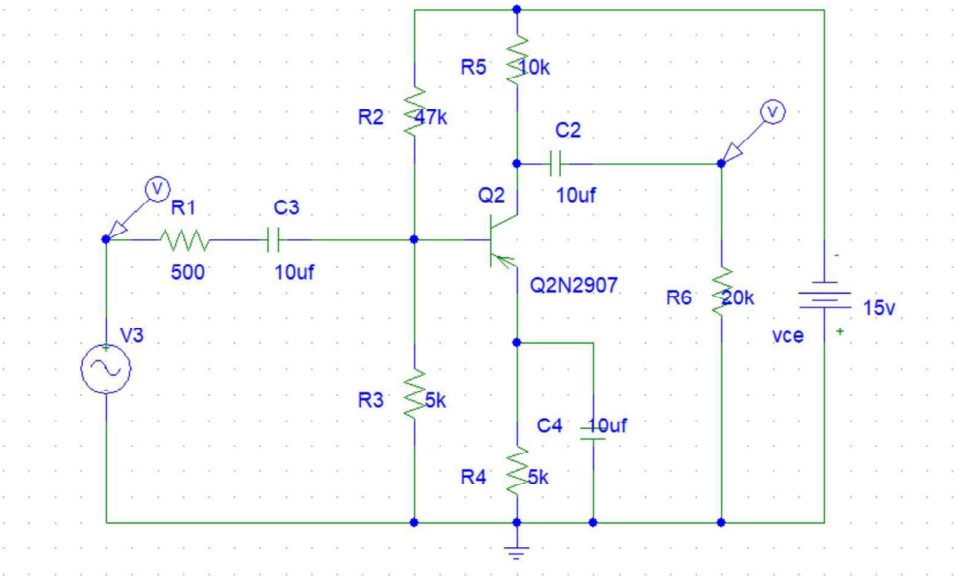
COMPONENTS:-

sno	component	Range	quantity
1	vsin	0,5v,1khz	1
2	resistor	500,47k,20k,10k,5k	1,1,1,1,2
3	capacitor	10uf	3
4	ground	-	1
6	bjt	Q2N2907	1
7	vdc	15v	1

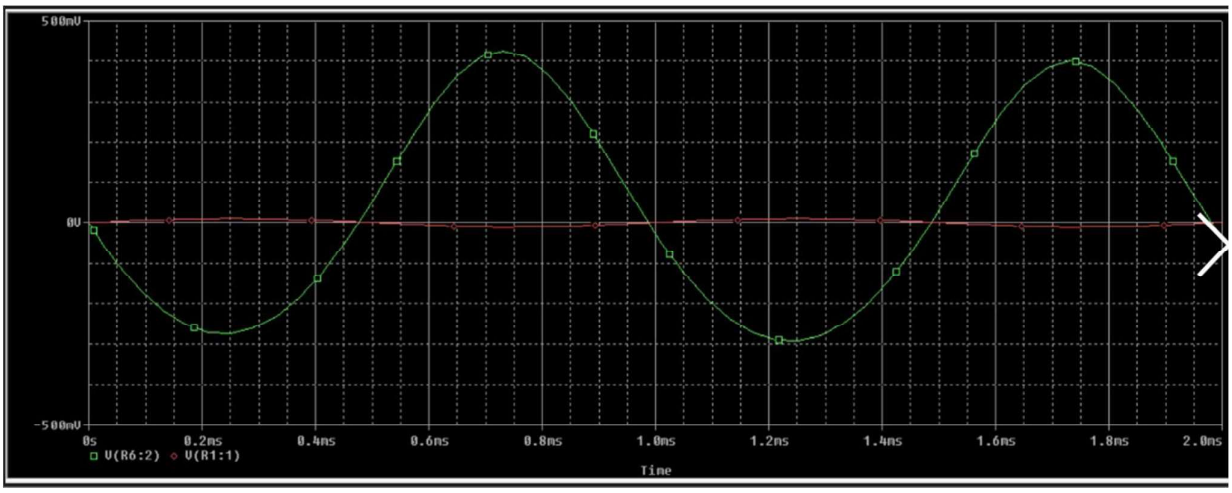
PROCEDURE:

- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- Perform the transient analysis on common emitter pnp transistor with 0-2ms.
- Simulate the circuit.

Circuit diagram:-



Waveform :-



Code:-

.AC DEC 10 1hz 10khz;

VIN 1 0 AC 10mv sin(0 10mv 1khz);

VCC 0 7 DC 15v;

Rs 1 2 500;

R1 7 3 47k;

R2 3 0 5k;

RC 7 4 10k;

RE 5 0 2k;

Rl 6 0 20k;

```
C1 2 3 10uF;
C2 4 6 10uF;
CE 5 0 10uF;
Q1 4 3 5 0 Q2n2907;
.model Q2N2907 PNP(Is = 650.6E-18 Xti=3 Eg=1.11 Vaf=115.7 Bf=231.7 Ne=1.829
+Ise =54.81f Ikf=1.079 Xtb=1.5 Br=3.563 Nc=2 Isc=0 Ikr=0 Rc=.715
+Cjc=14.76p Mjc=.5383 Vjc=.75 Fc=.5 Cje=19.82p Mje=.3357 Vje=.75
+Tr=111.3m Tf=603.7p Itf=.65 Vtf=5 Xtf=1.7 Rb=10)
* National pid=63 case=TO18
* 88-09-09 bam creation
.tran 50us 2ms;
.plot tran V(4) V(6) V(1);
.plot AC VM(6) VP(6);
.probe;
.end;
```

Result:-

Thus the transient analysis of common emitter pnp amplifier have been verified.

Exp:-8

INPUT AND OUTPUT CHARACTERISTICS OF MOSFET

Aim :

To verify the input and output characteristics of MOSFET by using the PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

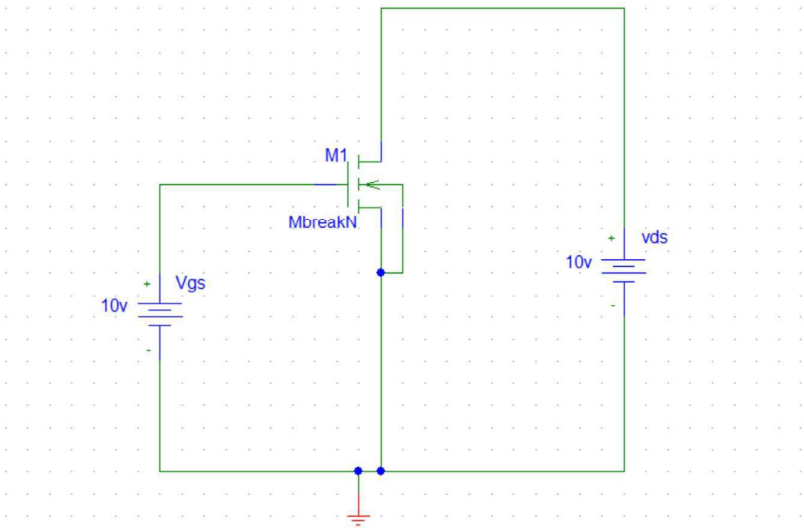
COMPONENTS:-

sno	component	Range	quantity
1	ground	-	1
2	bjt	MbreakN	1
3	vdc	10v	2

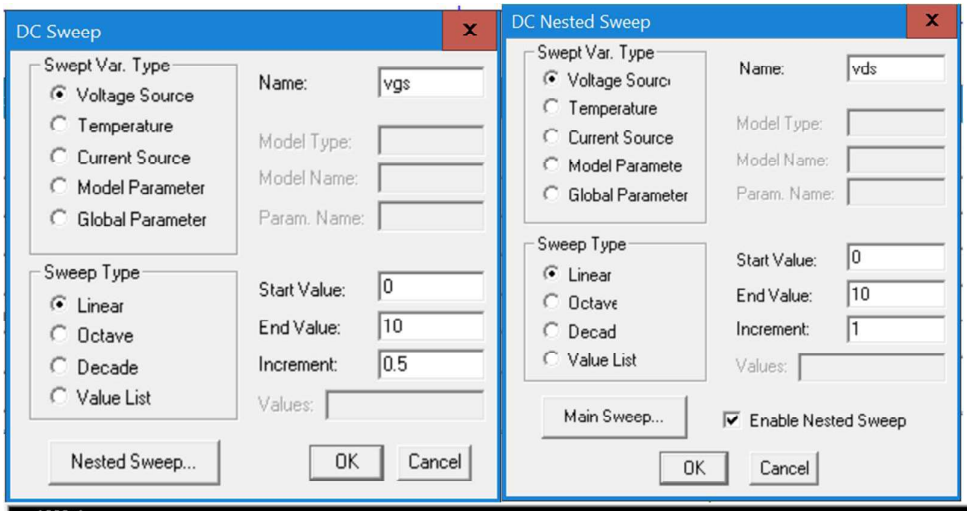
PROCEDURE:

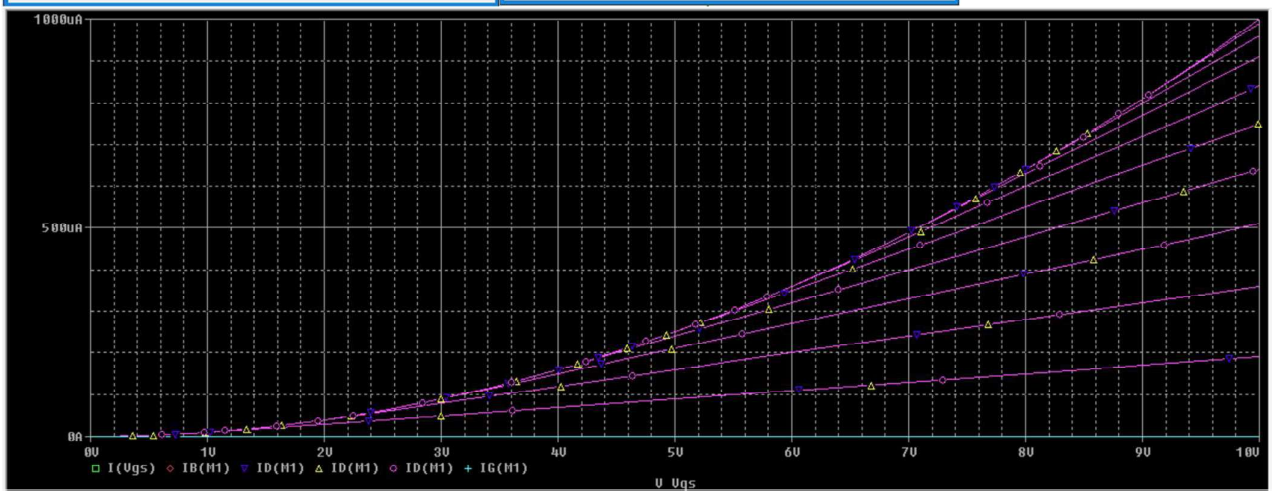
- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- Perform dc sweep in the mosfet and nested sweep for input and output values.
- For input characteristics set the trace to IG(M1).
- For output characteristics set the trace to ID(M1).
- The dc sweep values are in figure.
- Simulate the circuit.

CIRCUIT DIAGRAMS:

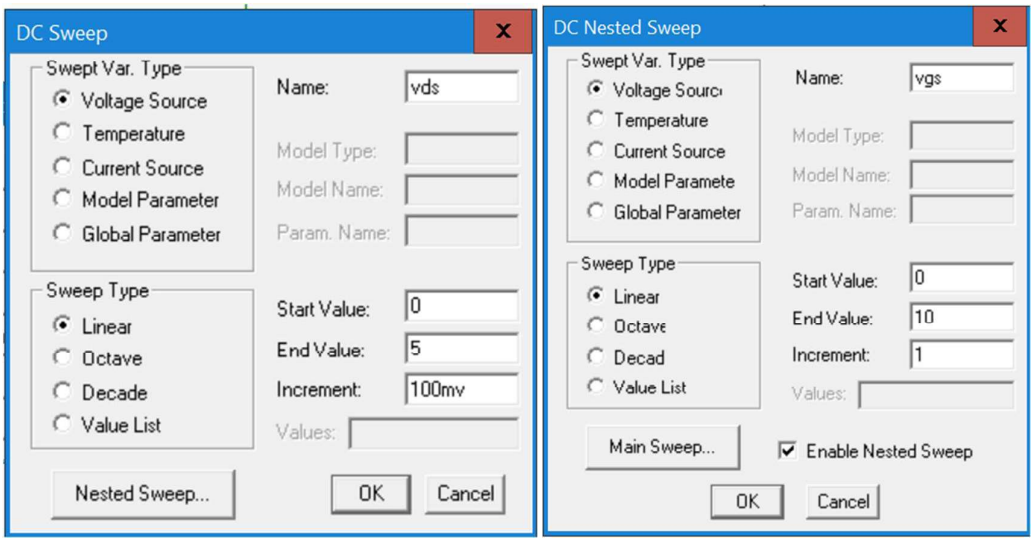


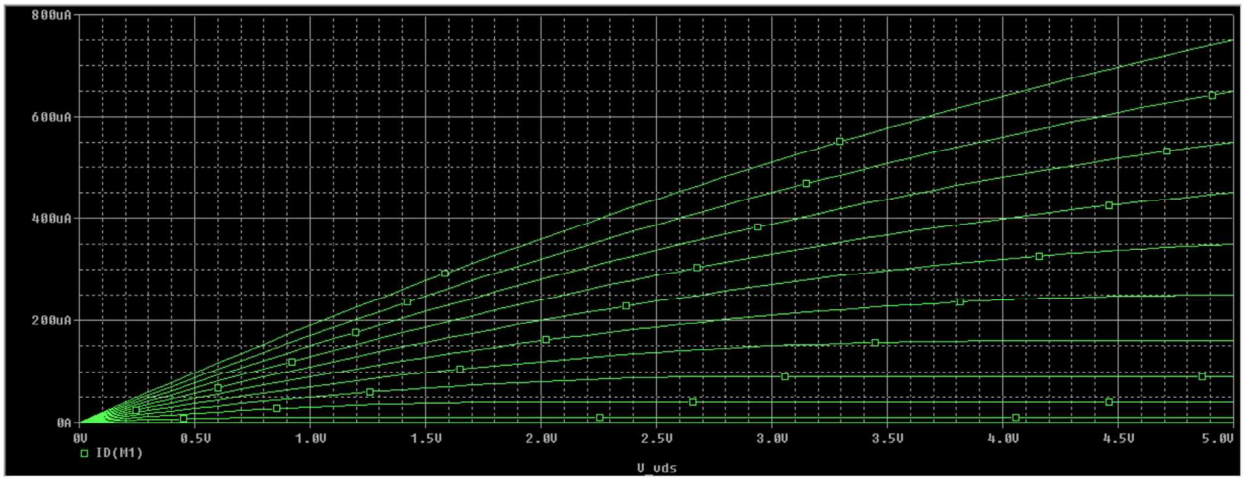
Mosfet INPUT DC SWEEP VALUES





Output characteristics:-





code:-

**** Analysis setup ****

.DC LIN V_Vgs 0 10 0.5

+ LIN V_vds 0 10 1

.DC LIN V_vds 0 10 1

+ LIN V_Vgs 0 10 0.5

.OP

*** From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:**

```
.lib "nom.lib"

.INC "mosfet char.net"



**** INCLUDING "mosfet char.net" ****
* Schematics Netlist *



V_Vgs        $N_0001 0 10v
M_M1         $N_0002 $N_0001 0 0 MbreakN
V_vds        $N_0002 0 10v

**** RESUMING "mosfet char.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

Result:-

Thus the input and output characteristics of BJT have been verified.

Exp:-9

DC AND TRANSIENT ANALYSIS OF CMOS INVERTER

Aim :

To verify the DC AND TRANSIENT ANALYSIS OF CMOS INVERTERcircuits by using the PSPICE software.

Apparatus:

PC with PSPICE VERSION 9.1 installed

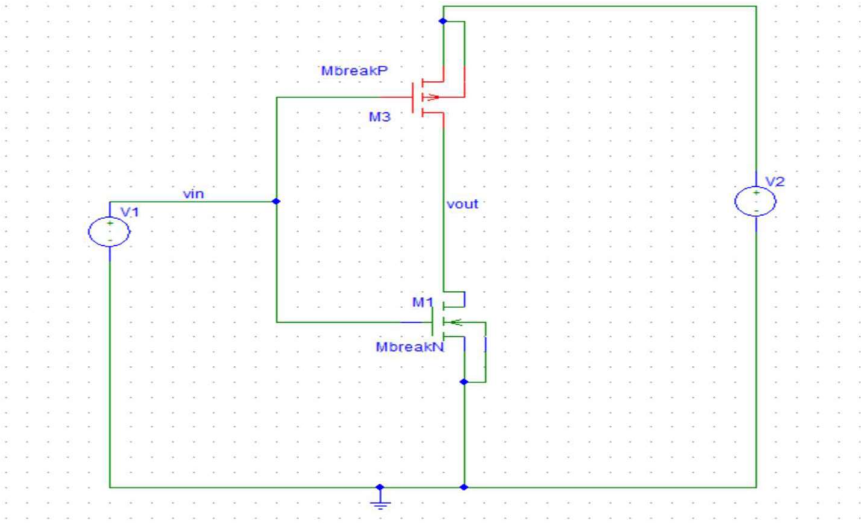
COMPONENTS:-

sno	component	Range	quantity
1	ground	-	1
2	mosfet	MbreakN,MbreakP	1 each
3	vsrc	5v	2
4	vpulse	-	1

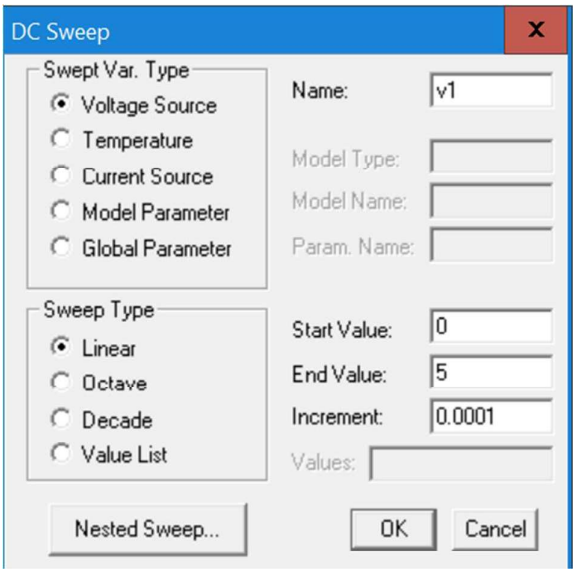
PROCEDURE:

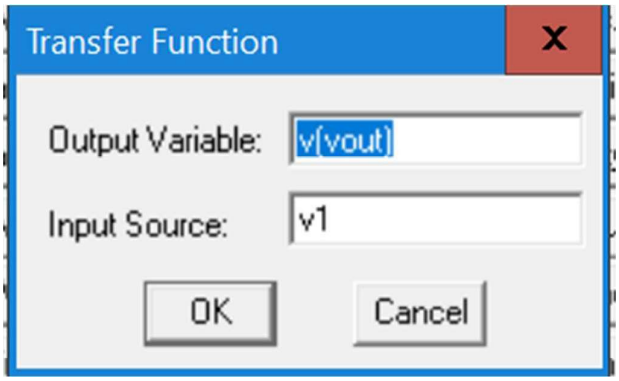
- Get the components from the library
- Choose wire to join them
- Give appropriate names and values to elements
- For mbreakp L = 2u,W = 4u;
- For mbreakn L = 2u,W = 2u;
- For DC analysis choose vsrc and set to 5v.
- Add trace to v_v1 and v(m1:d).
- For transient analysis choose vpulse and 5v.
- Add trace to v(m1:g) and v(m1:d).
- Give the ac analysis the default values
- For transient analysis give the 0-10ms time.
- Simulate the circuit.

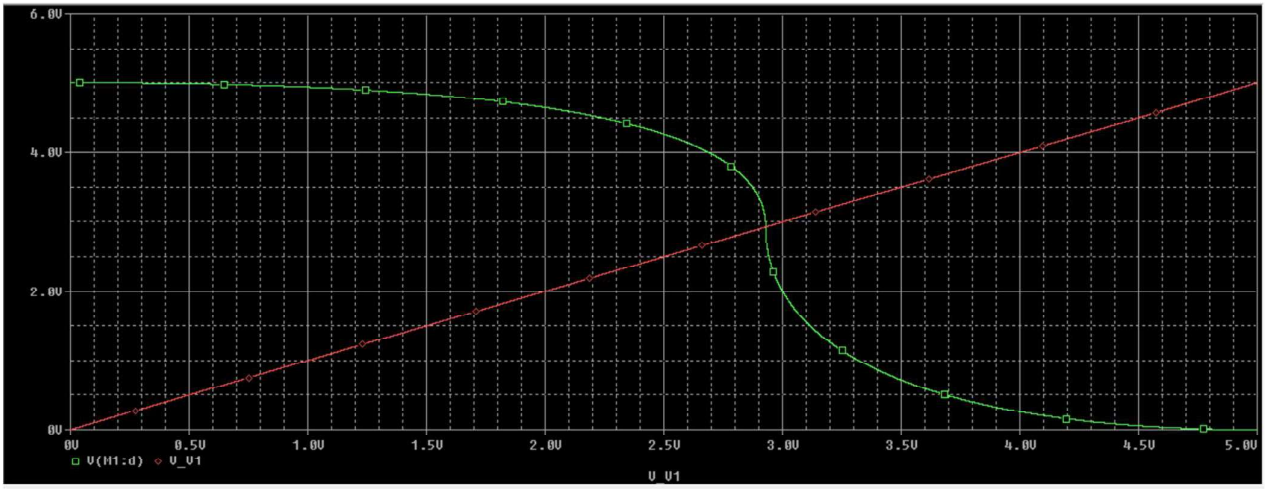
Circuit diagram:-



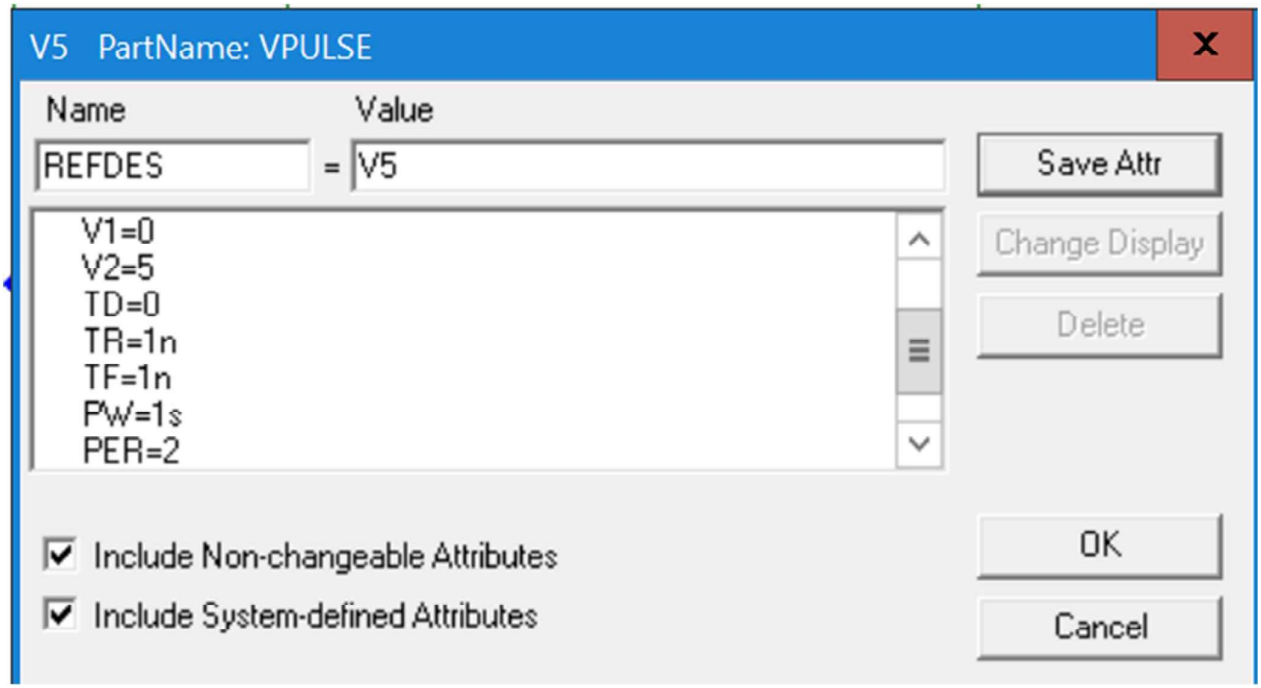
Output for dc sweep:-

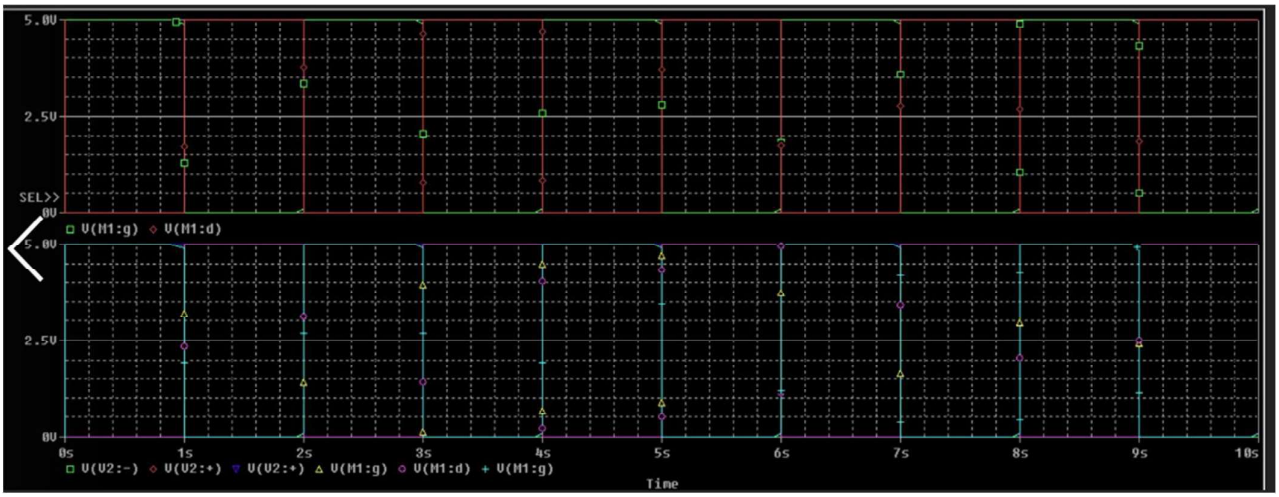






Output of transient analysis:-





Code:-

**** Analysis setup ****

.tran 0ns 10s

.OP

*** From [PSPICE NETLIST] section of d:\pspice\PSpice\PSpice.ini:**

.lib "nom.lib"

.INC "cmos inverter.net"

****** INCLUDING "cmos inverter.net" ******

*** Schematics Netlist ***

V_V2 $N_0001 0 DC 5v

M_M1 vout vin 0 0 MbreakN

+ L=2u

+ W=2u

M_M3 vout vin $N_0001 $N_0001 MbreakP

```
+ L=2u
+ W=4u
V_V5        vin 0 DC 5v
+PULSE 0 5 0 1n 1n 1s 2

**** RESUMING "cmos inverter.cir" ****
.PROBE V(*) I(*) W(*) D(*) NOISE(*)


.END
```

Result:-

Thus the DC and transient analysis of cmos inverterhave been verified.

Experiment Number:

Title of the Experiment: **BASIC LOGIC GATES**

Date of the Experiment:

Objective :

a) To Simulate the Basic Logic Gates with the tools available in Xilinx Project Navigator using VHDL.

b) To Implement the above with the available FPGA kit.

Facilities Required:

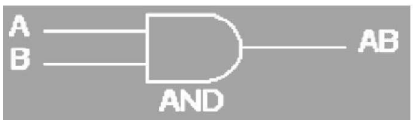
c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

CIRCUIT DIAGRAM:

AND GATE:

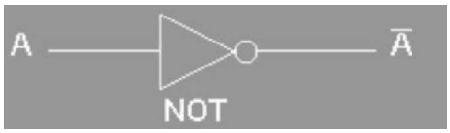


2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

2 Input OR gate		
A	B	A+B
0	1	1
1	1	1

NOT GATE:



NOT gate	
A	$\overline{A}$
0	1
1	0

XOR GATE:

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

NAND GATE:

2 Input NAND gate		
A	B	A B
0	1	1
0	1	1
1	0	1
1	1	0

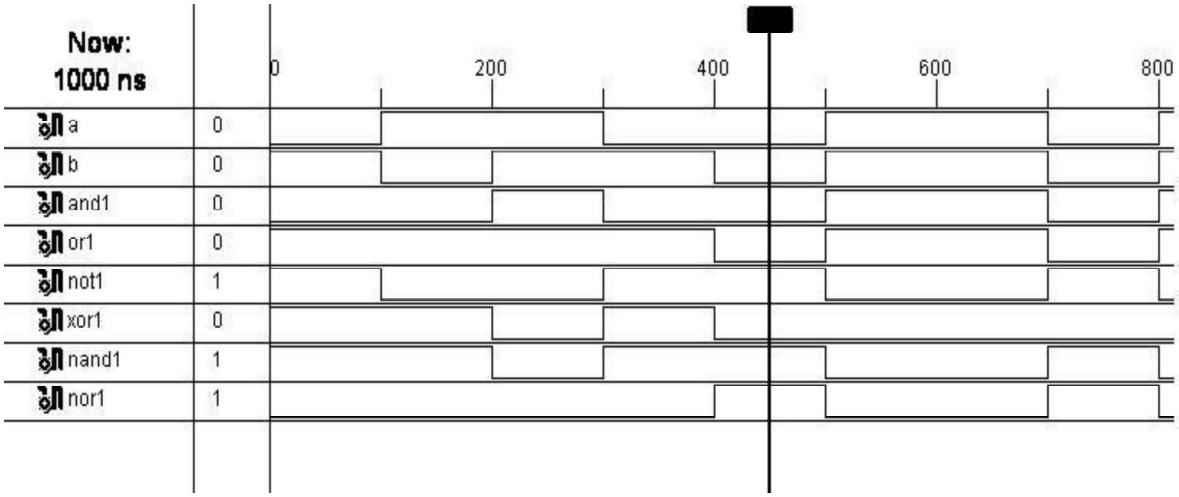
NOR GATE:

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity logic_gates is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           AND1 : out STD_LOGIC;
           OR1 : out STD_LOGIC;
           NOT1 : out  STD_LOGIC;
           XOR1 : out  STD_LOGIC;
           NAND1 : out  STD_LOGIC;
           NOR1 : out  STD_LOGIC);
end logic_gates;
architecture Behavioral of logic_gates is
begin
AND1<=A AND B;
OR1<=A OR B;
NOT1<=NOT A;
XOR1<=A XOR B;
NAND1<= A NAND B;
NOR1<=A NOR B;
end Behavioral;
```

OUTPUT:



RESULT:

Thus the Basic Logic Gates have been simulated with the Truth Table by using Xilinx ISE Simulator.

Experiment Number:

Title of the Experiment: FULL ADDER

Date of the Experiment:

Objective :

a) To Simulate the Full Adder with the tools available in Xilinx Project Navigator using VHDL.

b) To Implement the above with the available FPGA kit.

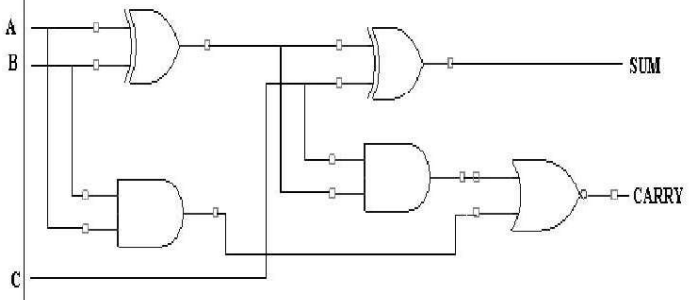
Facilities Required:

c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	SUM	CARRY
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 USING STRUCTURAL MODELLING:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity or_gate is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : out  STD_LOGIC);
end or_gate;
architecture Behavioral of or_gate is
   begin
    c<=a or b;
end Behavioral;
```

-------VHDL Code for and Gate----

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_g is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : out  STD_LOGIC);
end and_g;
architecture Behavioral of and_g is
begin
c<=a and b;
end Behavioral;
```

-------VHDL Code for XOr Gate----

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xor_g is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : out  STD_LOGIC);
end xor_g;
architecture Behavioral of xor_g is
begin
c<=a xor b;
```

```
end Behavioral;
```

-------VHDL Code for Full Adder----

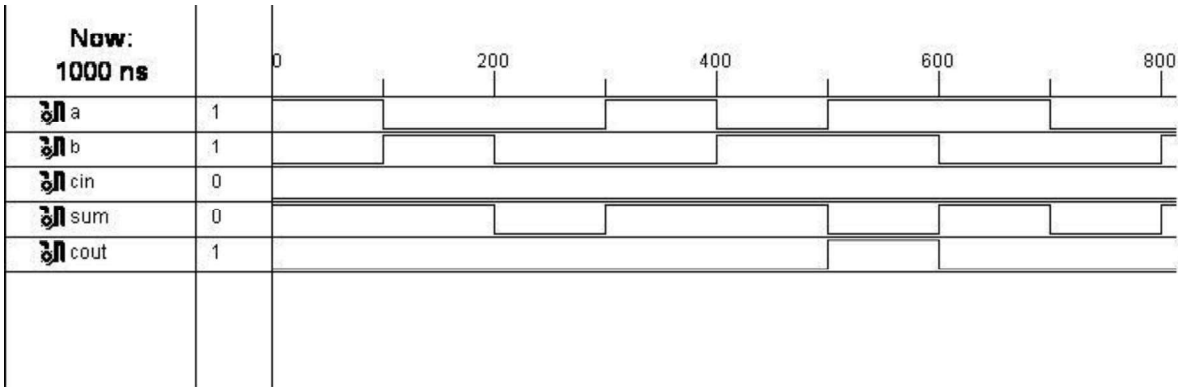
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fulladder_structural is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           SUM : out  STD_LOGIC;
           Cout : out STD_LOGIC);
end fulladder_structural;
architecture Behavioral of fulladder_structural is
 component or_gate is
         port(a,b:in std_logic;
                 c:out std_logic);
    end component;
  component and_g is
          port(a,b:in std_logic;
                    c:out std_logic);
end component;
component xor_g is
         port(a,b:in std_logic;
                    c:out std_logic);
end component;
              signal y1,y2,y3:std_logic;
begin
x1:xor_g port map(A,B,y1);
a1:and_g port map(A,B,y2);
x2:xor_g port map(y1,Cin,sum);
a2:and_g port map(y1,Cin,y3);
r1:or_gate port map(y2,y3,Cout);
end Behavioral;
```

VHDL CODE:

FULL ADDER USING DATAFLOW MODELLING:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Full_Adder_Dataflow is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Sum : out STD_LOGIC;
           Cout : out STD_LOGIC);
end Full_Adder_Dataflow;
architecture Dataflow of Full_Adder_Dataflow is
signal X: STD_LOGIC;
begin
X<= (A xor B) and Cin;
Sum<= A xor B xor Cin;
Cout<=X or (A and B);
end Dataflow;
```

OUTPUT:



RESULT:

Thus the Full adder Circuit has been simulated with the truth table by using Xilinx ISE Simulator.

Experiment Number:

Title of the Experiment: **FULL SUBTRACTOR**

Date of the Experiment:

Objective :

a) To Simulate the Full Subtractor with the tools available in Xilinx Project Navigator using VHDL.

b) To Implement the above with the available FPGA kit.

Facilities Required:

c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

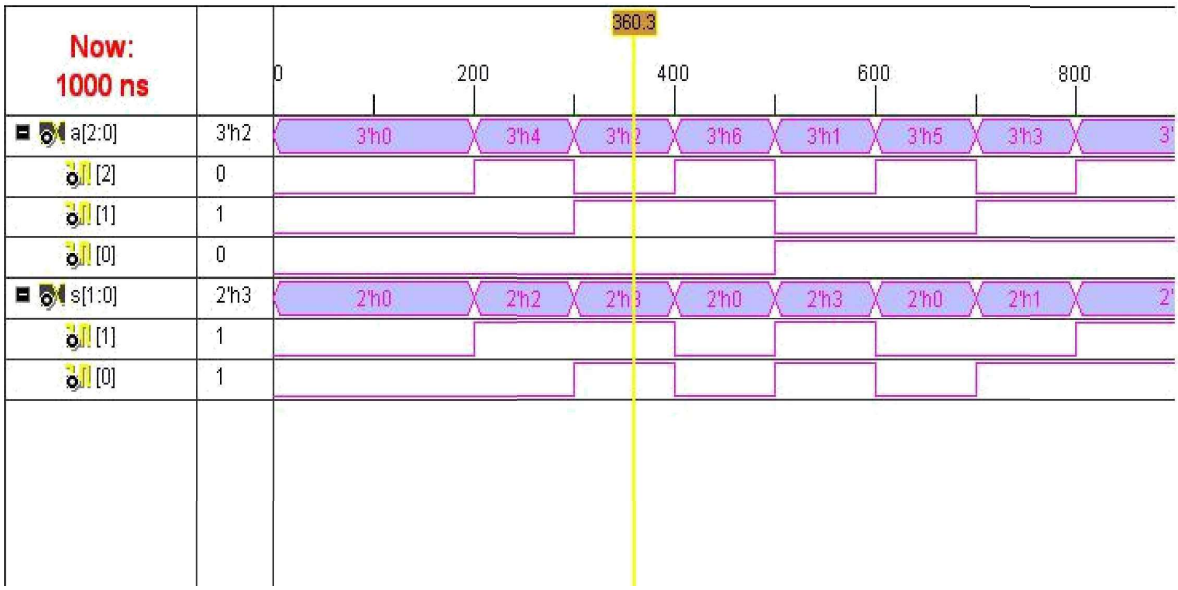
VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
      entity full_subs is
       port (a : in STD_LOGIC_VECTOR (0 to 2);
            s : out STD_LOGIC_VECTOR (0 to
      1)); end full_subs;
      architecture behavioral of full_subs
      is begin
      process
      (a) begin
      case a is
      when “000”=> s<=“00”;
      when “001”=> s<=“11”;
      when “010”=> s<=“11”;
      when “011”=> s<=“01”;
      when “100”=> s<=“10”;
      when “101”=> s<=“00”;
      when “110”=> s<=“00”;
      when                  others
      =>s<=“11”; end case;
      end process;
      end behavioral;
```

TRUTH TABLE:

A	B	C	Diff	Borrow
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

OUTPUT:



RESULT:

Thus the Full Subtractor Circuit has been simulated with the truth table by using Xilinx ISE Simulator

Experiment Number:

Title of the Experiment: MULTIPLEXER & DE-MULTIPLEXER

Date of the Experiment:

Objective :

a) To Simulate the Multiplexer & De-Multiplexer with the tools available in Xilinx Project Navigator using VHDL.

b) To Implement the above with the available FPGA kit.

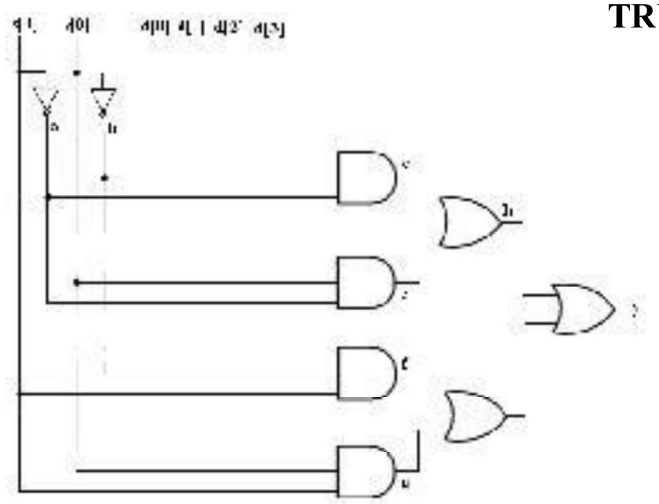
Facilities Required:

c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

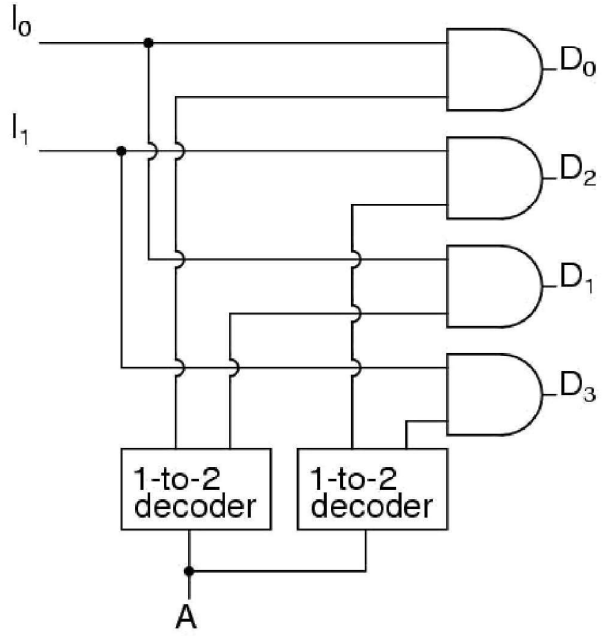
MULTIPLEXER:



TRUTH TABLE:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

DEMULTIPLEXER:



TRUTH TABLE:

INPUT			OUTPUT			
A	I0	I1	D0	D1	D2	D3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

VHDL CODE:

MULTIPLEXER:

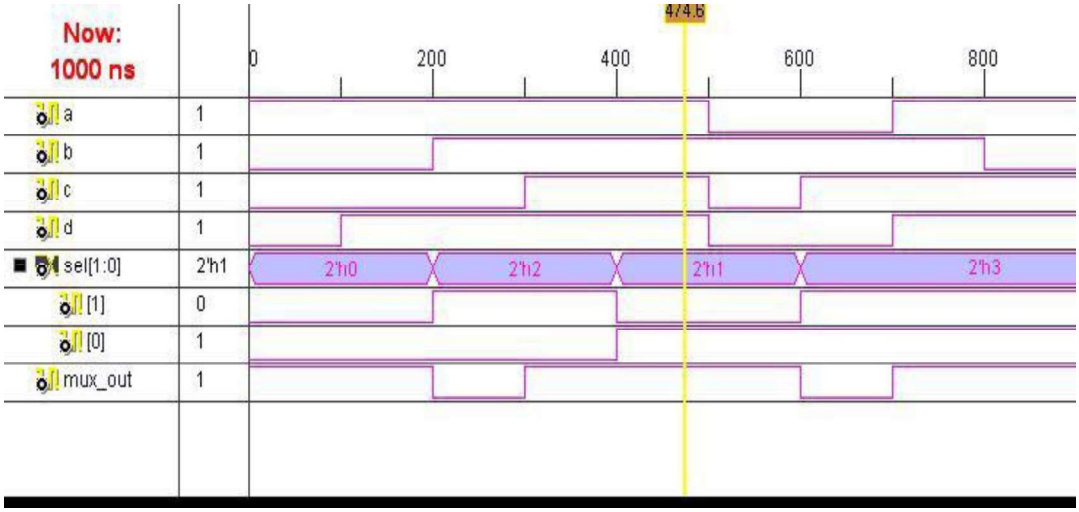
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mux4_1 is
Port ( a : in STD_LOGIC;
b : in STD_LOGIC;
c : in STD_LOGIC;
d : in STD_LOGIC;
sel : in STD_LOGIC_VECTOR(1 downto 0);
mux_out : out STD_LOGIC);
end mux4_1;
architecture behavioral of mux4_1 is
begin
process(sel,a,b,c,d)
begin
casesel is
when "00"=>mux_out<=a;
when "01"=>mux_out<=b;
when "10"=>mux_out<=c;
when "11"=>mux_out<=d;
when others=>null;
end case;
end process;
end behavioral
```

DEMULTIPLEXER:

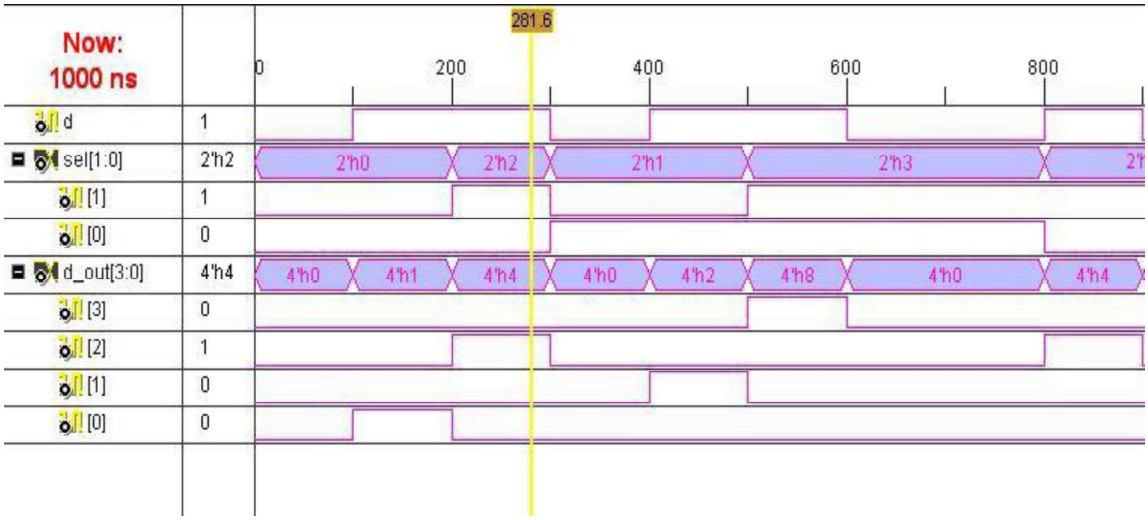
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity demux is
        Port ( d : in STD_LOGIC;
        sel : in STD_LOGIC_VECTOR(1 downto 0);
        d_out : out STD_LOGIC_VECTOR(3 downto
        0)); end demux;
        architecture demux of sai is
        begin
        process(d,sel)
        begin
        d_out<="0000";
        casesel is
        when"00"=>d_out(0)<=d;
        when"01"=>d_out(1)<=d;
        when"10"=>d_out(2)<=d;
        when others=>d_out(3)<=d;
        end case;
        end process;
        end sai;
```

OUTPUT:

MULTIPLEXER:



DEMULTIPLEXER:



RESULT:

Thus the Multiplexer and Demultiplexer Circuit has been simulated with the truth table by using Xilinx ISE Simulator

Experiment Number:

Title of the Experiment: **8 BIT ADDER(RIPPLE CARRY ADDER)**

Date of the Experiment:

Objective :

a) To Simulate the 8 bit Adder (RIPPLE CARRY ADDER) with the tools available in Xilinx Project Navigator using Verilog.

b) To Implement the above with the available FPGA kit.

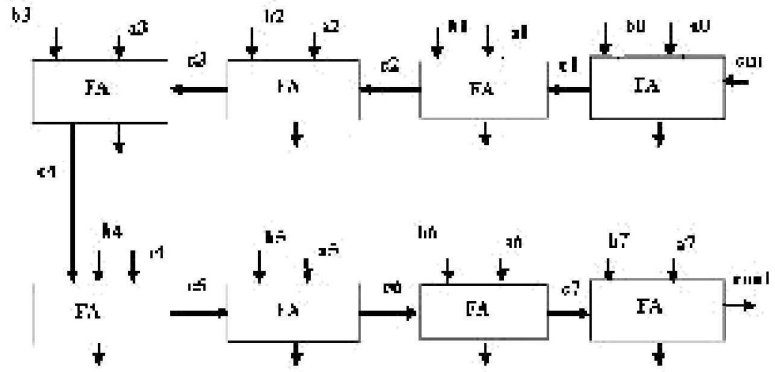
Facilities Required:

c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

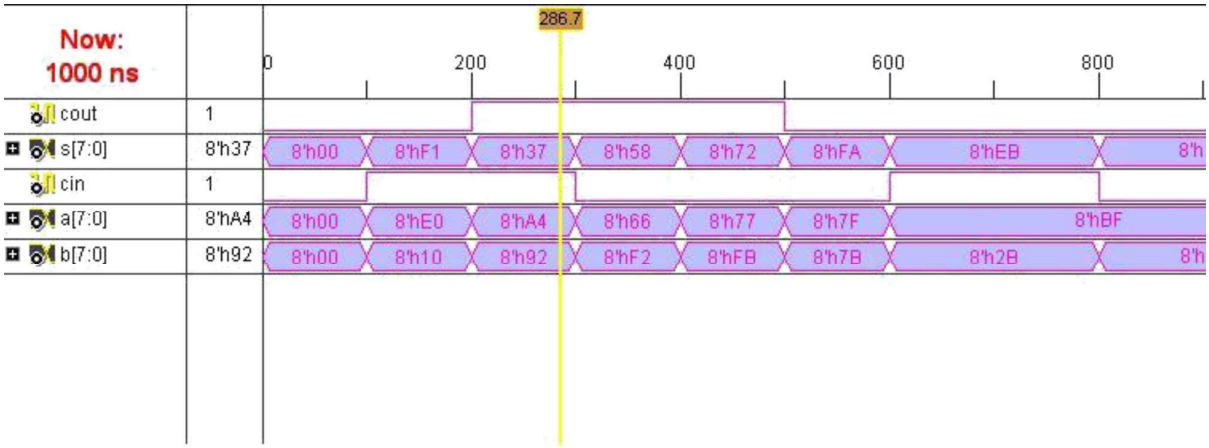
LOGIC DIAGRAM:



Verilog Code:

```
module ripplecarryadder(s,cout,a,b,cin);
output[7:0]s;
output cout;
input[7:0]a,b;
input cin;
wire c1,c2,c3,c4,c5,c6,c7;
fulladd fa0(s[0],c1,a[0],b[0],cin);
fulladd fa1(s[1],c2,a[1],b[1],c1);
fulladd fa2(s[2],c3,a[2],b[2],c2);
fulladd fa3(s[3],c4,a[3],b[3],c3);
fulladd fa4(s[4],c5,a[4],b[4],c4);
fulladd fa5(s[5],c6,a[5],b[5],c5);
fulladd fa6(s[6],c7,a[6],b[6],c6);
fulladd fa7(s[7],cout,a[7],b[7],c7);
endmodule
module fulladd(s,cout,a,b,cin);
output s,cout;
input a,b,cin;
wire s1,c1,c2;
xor(s1,a,b);
xor(s,s1,cin);
and(c1,a,b);
and(c2,s1,cin);
xor(cout,c2,c1);
endmodule
```

OUTPUT:



RESULT:

Thus the 8 Bit Adder Circuit has been simulated with the truth table by using Xilinx ISE Simulator

Experiment Number:

Title of the Experiment: FLIP-FLOPS

Date of the Experiment:

Objective :

a) To Simulate the Flip-Flops with the tools available in Xilinx Project Navigator using Verilog.

b) To Implement the above with the available FPGA kit.

Facilities Required:

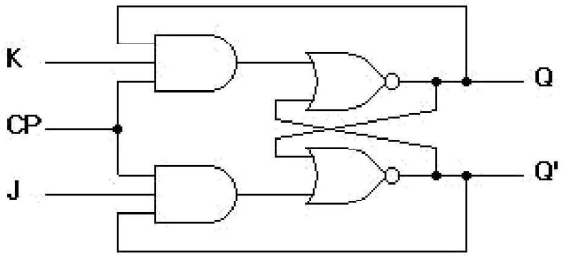
c) Xilinx (ISE) simulator 9.1
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

LOGIC DIAGRAM:

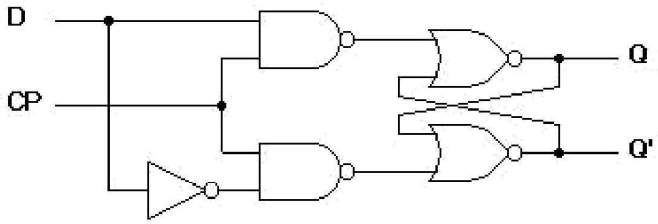
JK FLIP FLOP:



TRUTH TABLE:

Q(t)	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

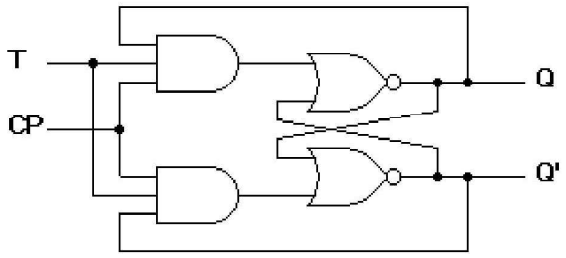
D Flip Flop:



TRUTH TABLE:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

T Flip Flop:



TRUTH TABLE:

Q T	Q(t+1)
0 0	0
0 1	1
1 0	1
1 1	0

VERILOG SOURCE CODE: T FlipFlop

Behavioral Modeling:

```
+module tff(t,clk,rst, q,qb);
  input t,clk,rst;
 output q,qb;
  reg q,qb;
  reg temp=0;
always@(posedge clk,posedge rst)
  begin
  if (rst==0) begin
if(t==1) begin
 temp=~ temp;
  end
 temp=temp;
q=temp;qb=~temp;
  end
  else
end module
```

VERILOG SOURCE CODE: JK Flip Flop

Behavioral Modeling:

```
  module jk(q,q1,j,k,c);
output q,q1;
input j,k,c;
reg q,q1;
initial begin q=1'b0; q1=1'b1; end
always @ (posedge c)
          begin
          case({j,k})
          {1'b0,1'b0}:begin q=q; q1=q1; end
          {1'b0,1'b1}: begin q=1'b0; q1=1'b1; end
          {1'b1,1'b0}:begin q=1'b1; q1=1'b0; end
          {1'b1,1'b1}: begin q=~q; q1=~q1; end
          endcase
 end
endmodule
```

VERILOG SOURCE CODE: D Flip Flop

```
module d(q,q1,d,c);
output q,q1;
 input d,c;
 reg q,q1;
     initial
       begin
                q=1'b0; q1=1'b1;
      end
     always @ (posedge c)
       begin
              q=d;
              q1= ~d;
       end
endmodule
```

RESULT:

Thus the Flip Flop Circuit has been simulated with the truth table by using Xilinx ISE Simulator

Experiment Number:

Title of the Experiment: **RIPPLE COUNTER**

Date of the Experiment:

Objective :

a) To Simulate the ripple counter with the tools available in Xilinx Project Navigator using Verilog.

b) To Implement the above with the available FPGA kit.

Facilities Required:

c) Xilinx (ISE) simulator 9.1

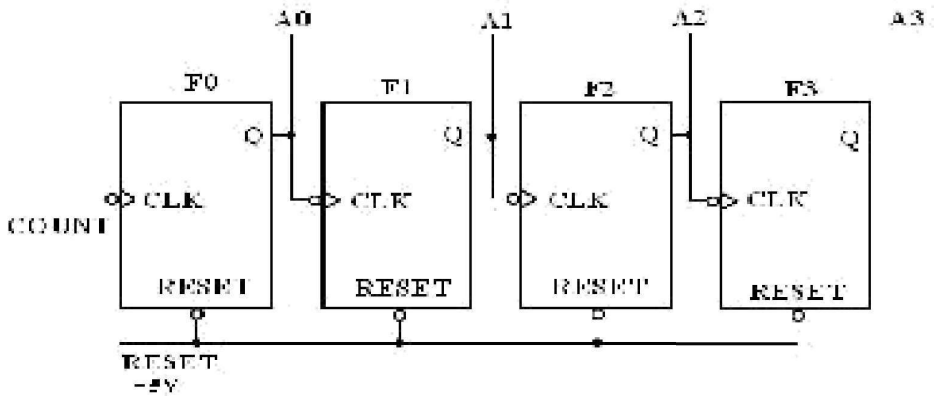
d) FPGA KIT

Procedure for doing the experiment:

S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

LOGIC DIAGRAM:

4-Bit Ripple Counter:



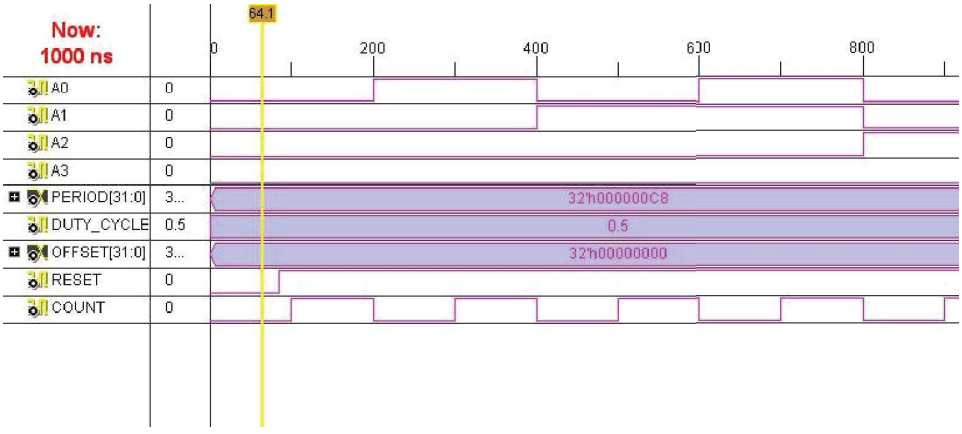
TRUTH TABLE:

COUNT	A0	A1	A2	A3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	0	1
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

VERILOG SOURSE CODE: //Structural description of Ripple Counter

```
module ripplecounter(A0,A1,A2,A3,COUNT,RESET);
  output A0,A1,A2,A3;
  input COUNT,RESET;
  //Instantiate Flip-Flop
  FF F0(A0,COUNT,RESET);
  FF F1(A1,A0,RESET);
  FF F2(A2,A1,RESET);
  FF F3(A3,A2,RESET)
  endmodule
  //Description of Flip-Flop
  module FF(Q,CLK,RESET);
  output Q;
  input CLK,RESET;
  reg Q;
  always @(negedge CLK or negedge RESET)
  if(~RESET)
  Q=1'b0;
  else
  Q=(~Q);
  endmodule
```

OUTPUT:



RESULT:

Thus the 4 bit Ripple Counter Circuit has been simulated with the truth table by using Xilinx ISE Simulator