Computer Aided System Design Laboratory Record

B.E (ECE) – FULL TIME

V SEMESTER

(For the Academic Year 2021-22)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

(Deemed to be University u/s 3 of UGC Act, 1956)

(Accredited with ‘A’ Grade by NAAC)

Enathur, Kanchipuram – 631561

BONAFIDE CERTIFICATE

Certified to be the Bonafide Record of the work done by ……………………………………(Name)……………….(Reg.No.) ………… (Semester) …………. (Branch) in the Computer Aided System Design Laboratory during the year 2021-2022.

Place:

Date:

……………………… ………………………

Faculty in-charge
Dr. G. Senthil Kumar

Head of the department
Prof. V. Swaminathan

Submitted for the Practical Examination held on …………...

Register No: ……………….

Internal Examiner **External examiner**

LIST OF EXPERIMENTS

PSPICE:

(Modelling, Design, Simulation and Analysis using Schematic / Circuit file / both)

1. Study of PSPICE
2. RC circuits – Transient and AC analysis
3. MOS Device Characterization and CMOS Inverter Characteristics – DC analysis
4. Diode based circuits (like, Rectifiers, Clampers, etc.,) – Transient, Worst-case, MC, Sensitivity, etc. analysis
5. Amplifiers and Current mirrors using BJT/MOSFET
6. Op-Amp based Wein Bridge Oscillator and DAC using sub-circuit and Analog behavioural modelling
7. Digital Circuits – Logic switches / Multiplexer / Counter

HDL:

(Logic Design and Simulation of Digital Circuits using VHDL / Verilog HDL / Both)

8. Study of VHDL and Verilog
9. Full Adder and Multiplexer using different Modelling / Descriptions and Concurrent and Sequential execution in VHDL
10. 8-bit Adder / Multiplier (min 4-bit) – Port Map, Generics, Technology Mapping in VHDL
11. 8-bit Counter – Bottom up approach design and Test vector generation in Verilog HDL
12. NAND / NOR / Transmission gates using Switch level modelling in Verilog HDL
13. Design of simple sequential and combinational circuits
14. Design of ALU
15. Design of FSM and Control Unit
16. FPGA real time programming and I/O Interfacing– Waveform generation / Traffic light controller

PSPICE

Exp no. 1	Study of PSPICE
Date:	

Objective:

A primary purpose of this lab is to become familiar with the use of PSpice and to learn to use it in the analysis of circuits. The software is already installed in the computer of every station.

Introduction:

SPICE is an acronym for Simulation Program with Integrated Circuit Emphasis. The original SPICE program was developed at the University of California Berkley in the 1970s. Computer aided simulation is common practice in industry and is a very useful tool. SPICE is useful way of verifying your lab test results, and experimenting with changes to your own circuit designs. It is also widely used in industry for simulating designs prior to production.Internal numerical accuracy of programs such as SPICE is very high with errors that seldom exceeding 1%.

Transistor circuit analysis is burdensome as the number of transistors increases beyond more than a few. Consequently SPICE is used to test and simulate complex transistor circuits. There are several versions of the SPICE software now available. Aim Spice and PSPICE are two versions. PSPICE is a graphical simulator, whereas Aim Spice is text based. All SPICE programs are based on the core SPICE programming.

While PSPICE makes extensive use of part libraries, Aim Spice uses text entries. Circuits may contain passive components such as resistors, capacitors, and inductors, and active devices such as transistors and diodes as well as independent voltage and current sources. To write code describing a circuit, nodes must be defined in the code. With nodes clearly defined, various elements are then connected between nodes to specified values.

SPICE allows the user to perform various analysis of the circuit such as nonlinear dc, large-signal time domain (transient), small-signal frequency domain, nonlinear transient, and linear ac analyses. The dc and transient analysis

capabilities are of greatest interest for digital circuit studies. In addition to performing the differing analysis types, SPICE also generates graphical outputs for which the various nodes and inputs can be graphed individually or together. SPICE software is based on the same logic core in which the code is either manually generated as with Aim Spice, or converted from a graphical representation by the software as PSPICE does. A netlist file is manually written when using Aim Spice, whereas PSPICE generates the netlist file containing the circuit elements and their interconnections for you based on the graphical representation.

Despite the accuracy of computer simulation, hand analysis is still necessary. SPICE simulation is a tool to enhance circuit analysis not replaces hand computations. For instance, hand calculations are the best method for developing appropriate simulation time intervals or rise times for a given circuit.

A curve tracer is a special type instrument similar to an oscilloscope designed to display voltage-current characteristics of three terminal devices such as transistors. The graphical display of an oscilloscope enables a user to easily view and identify the operating regions for a specific transistor and see how quickly the transistor saturates.

In PSpice the program we run in order to draw circuit schematics is called CAPTURE. The program that will let us run simulations and see graphic results is called PSPICE. You can run simulation from the program where your schematic is. There are a lot of things we can do with PSpice, but the most important things for you to learn are

- Design and draw circuits
- Simulate circuits
- Analyze simulation results (Probe for older versions).

The devices that we will use are resistors, inductors, capacitors and various independent/dependent sources. It is good to know that CAPTURE has extensive symbol libraries and includes a fully integrated symbol editor for creating your own symbols or modifying existing symbols.

The main tasks in CAPTURE are :

- Creating and editing designs
- Creating and editing symbols
- Creating and editing hierarchical designs
- Preparing your design for simulation

PROCEDURE:

1. Run the CAPTURE program.
2. Select File/New/Project from the File menu.
3. On the New Project window select Analog or Mixed A/D, and give a name to your project then click OK.
4. The Create PSpice Project window will pop up, select Create a blank project, and then click OK.
5. Now you will be in the schematic environment where you are to build your circuit.
6. Select Place/Part from the Place menu.
7. Click ANALOG from the box called Libraries:, then look for the part called R. You can do it either by scrolling down on the Part List: box or by typing R on the Part box. Then click OK.
8. Use the mouse to place the resistor where you want and then click to leave the resistor there. You can continue placing as many resistors as you need and once you have finished placing the resistors right-click your mouse and select end mode.
9. To rotate the components there are two options: • Rotate a component once it is placed: Select the component by clicking on it then Ctrl-R • Rotate the component before it is placed: Just Ctrl-R.
10. Select Place/Part from the Place menu.
11. Click SOURCE from the box called Libraries:, then look for the part called VDC. You can do it either by scrolling down on the Part List: box or by typing VDC on the Part box, and then click OK. Place the Source.
12. Repeat steps 10 - 12 to get and place a current source named IDC.

13. Select Place/Wire and start wiring the circuit. To start a wire click on the component terminal where you want it to begin, and then click on the component terminal where you want it to finish. You can continue placing wires until all components are wired. Then right-click and select end wire.

14. Select Place/Ground from the Place menu, click on GND/CAPSYM. Now you will see the ground symbol. EE/CE 3111 Electronic Circuits Laboratory Spring 2015 Professor Y. Chiu 3

15. Type 0 on the Name: box and then click OK. Then place the ground. Wire it if necessary.

16. Now change the component values to the required ones. To do this you just need to double-click on the parameter you want to change. A window will pop up where you will be able to set a new value for that parameter.

17. Once you have finished building your circuit, you can move on to the next step – prepare it for simulation.

18. Select PSpice/New Simulation Profile and type a name, this can be the same name as your project, and click Create.

19. The Simulations Settings window will now appear. You can set up the type of analysis you want PSpice to perform. In this case it will be Bias Point. Click Apply then OK.

20. Now you are ready to simulate the circuit. Select PSpice/Run and wait until the PSpice finishes. Go back to Capture and see the voltages and currents on all the nodes.

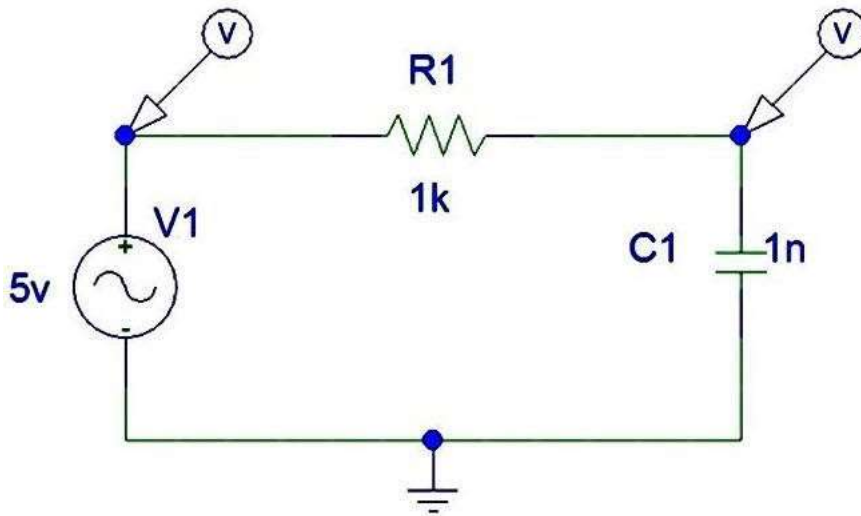
21. If you are not seeing any readout of the voltages and currents then select PSpice/Bias Point/Enable Bias Voltage Display and PSpice/Bias Point/Enable Bias Current Display. Make sure that PSpice/ Bias Point/Enable is checked

Conclusion:

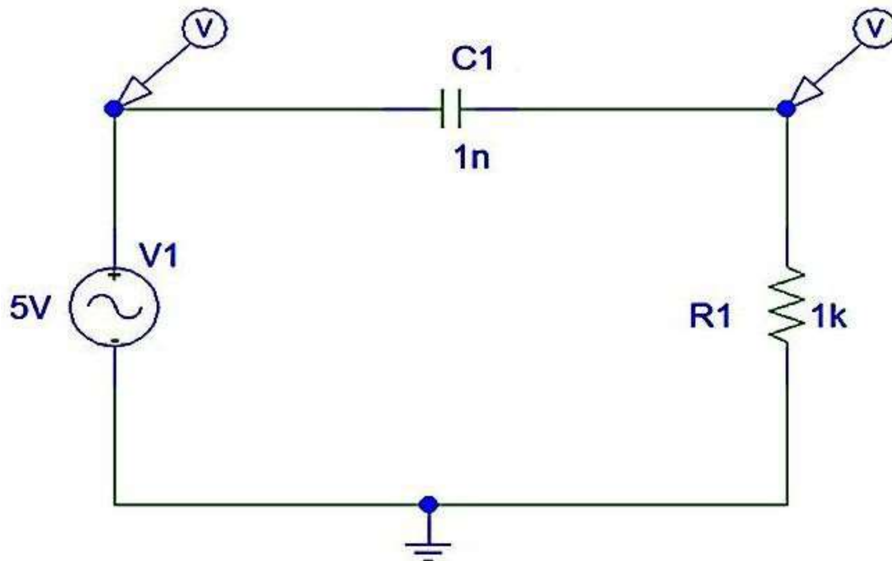
Thus introduction is given to PSPICE SOFTWARE and operation procedure for simulation was also done.

CIRCUIT DIAGRAM:

LOW PASS CIRCUIT:



HIGH PASS CIRCUIT:



Exp no. 2	RC circuits – Transient and AC analysis
Date:	

Aim:

To Perform AC and transient analysis of low pass and high pass filters using Pspice.

System Requirements:

PC with PSPICE software 9.1

Theory:

A resistor–capacitor circuit (RC circuit), or RC filter or RC network, is an electric circuit composed of resistors and capacitors driven by a voltage or current source. A first order RC circuit is composed of one resistor and one capacitor and is the simplest type of RC circuit.

RC circuits can be used to filter a signal by blocking certain frequencies and passing others. The two most common RC filters are the high-pass filters and low-pass filters; band-pass filters and band-stop filters usually require RLC filters, though crude ones can be made with RC filters.

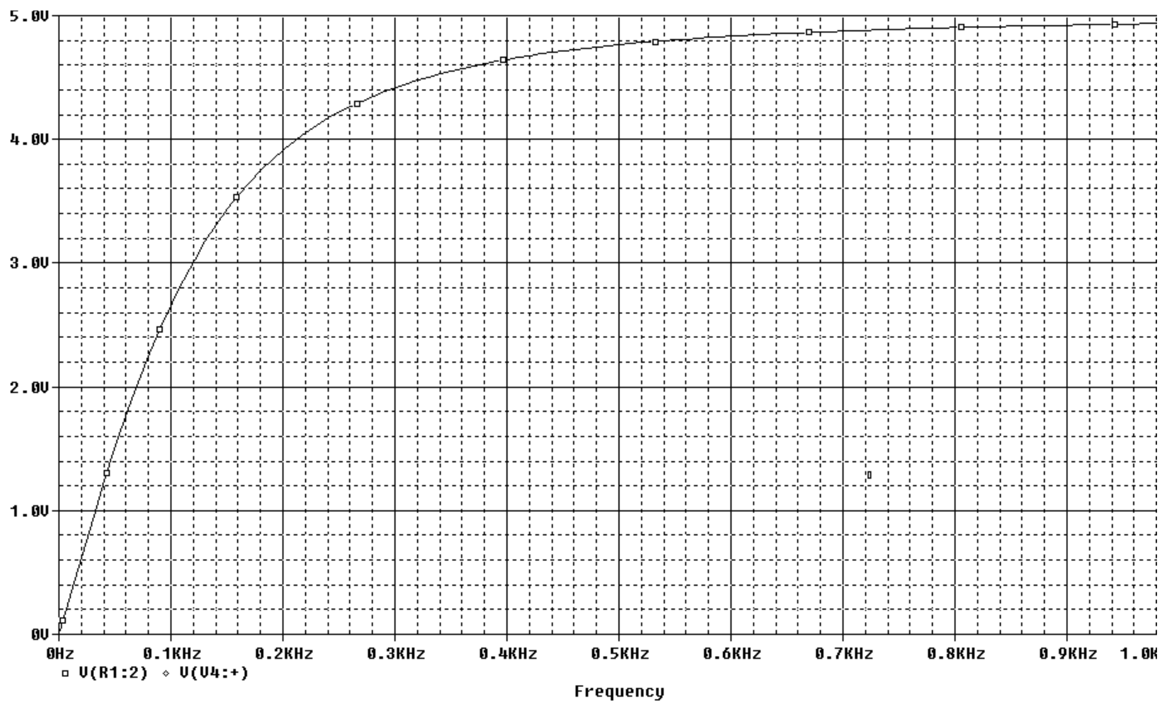
Design:

Procedure:

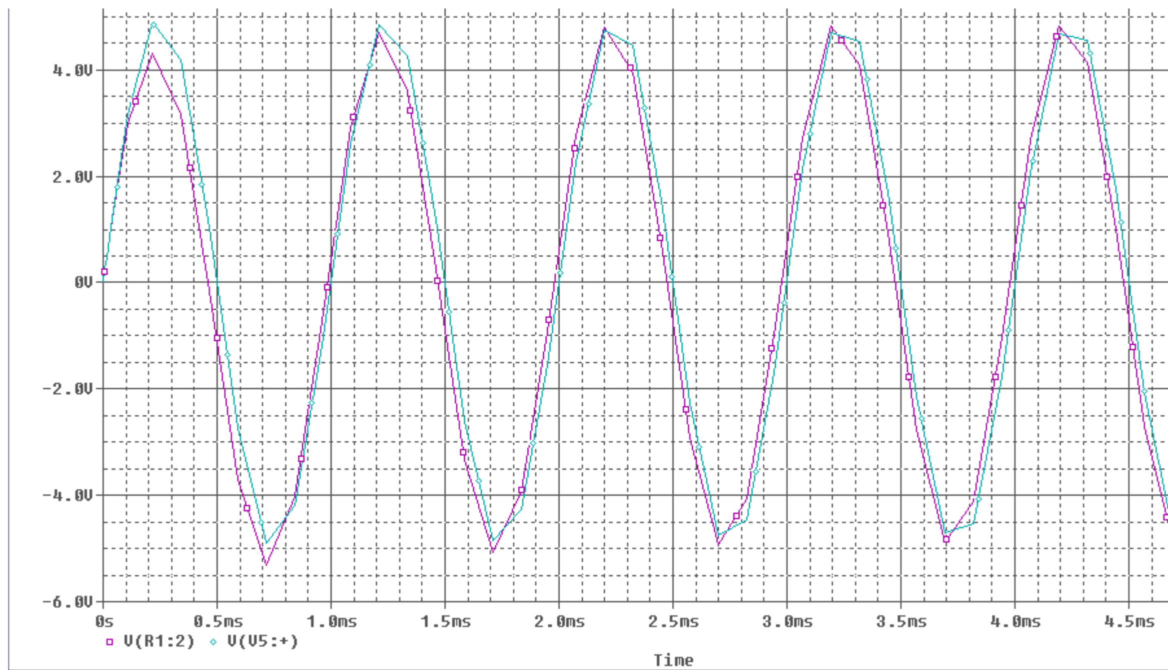
1. Rig-up the circuit as shown in figure by choosing appropriate devices from the menu titled devices
2. Choose the wire drawing tool from the tool bag and draw the lines.
3. Give the appropriate names and values for all elements present in the circuit.
4. An AC voltage source of ‘0’ phase, 1V amplitude, variable frequency is applied as input signal by editing the voltage source.
5. Then choose set up simulation from simulation menu.
6. Choose option of AC frequency analysis and give starting and ending frequency ranges. Select the option of view table and view graph.
7. Now choose run simulation.
8. Observe the output frequency response graph and take the maximum gain and 3dB frequencies.
9. Note down the tabular column

Output for RC circuit

Ac Analysis



Transient Analysis



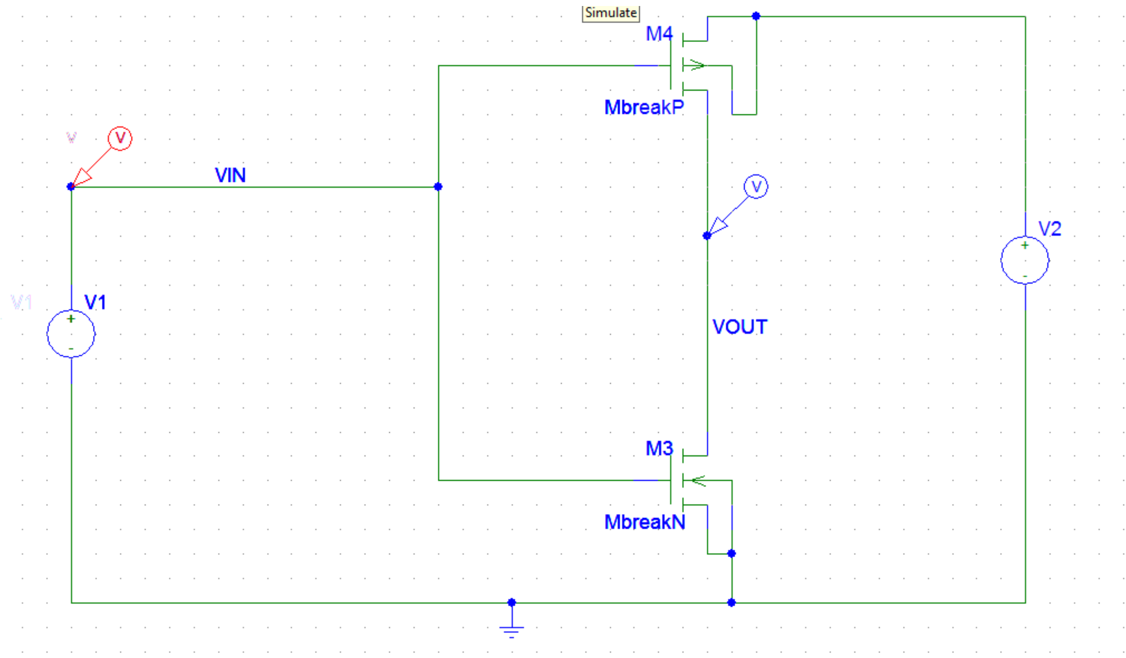
CIRCUIT FILE:

```
VIN 1 0 AC 5
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 1K
C1 2 0 0.01U
.AC LIN 1000 1K 1MEG
.TRAN 5e-005 4MS 0
.PROBE
.END
```

Conclusion:

Thus the AC and transient analysis of low pass and high pass filters using Pspice is performed.

Circuit diagram:



DC Sweep

Swept Var. Type
- (•) Voltage Source
- ( ) Temperature
- ( ) Current Source
- ( ) Model Parameter
- ( ) Global Parameter

Name: v1

Model Type:

Model Name:

Param. Name:

Sweep Type
- (•) Linear
- ( ) Octave
- ( ) Decade
- ( ) Value List

Start Value: 0

End Value: 5

Increment: 0.001

Values:

Nested Sweep... | OK | Cancel

Transfer Function

Output Variable: V(vout)

Input Source: v1

OK | Cancel

Exp no. 3	MOS Device Characterization and CMOS Inverter Characteristics – DC analysis
Date:	

Aim:

To understand the MOS devices characters and to design the CMOS inverter and to understand characteristics using DC analysis using LT spice.

Apparatus:

PSPICE 9.1

Description:

The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom.

CMOS is also sometimes referred to as **complementary-symmetry metal–oxide–semiconductor**. The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.

Design:

Procedure:

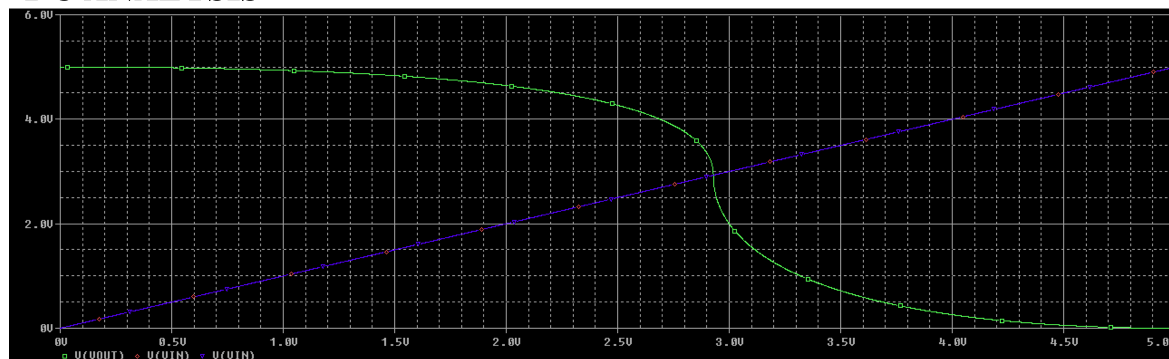
1. Rig-up the circuit as shown in figure by choosing appropriate devices from the menu titled devices
2. Choose the wire drawing tool from the tool bag and draw the lines.
3. Give appropriate names and values for all elements present in circuit.
4. Then choose set up simulation from simulation menu.
5. Choose option of DC analysis. Select the option of view table and view

graph.

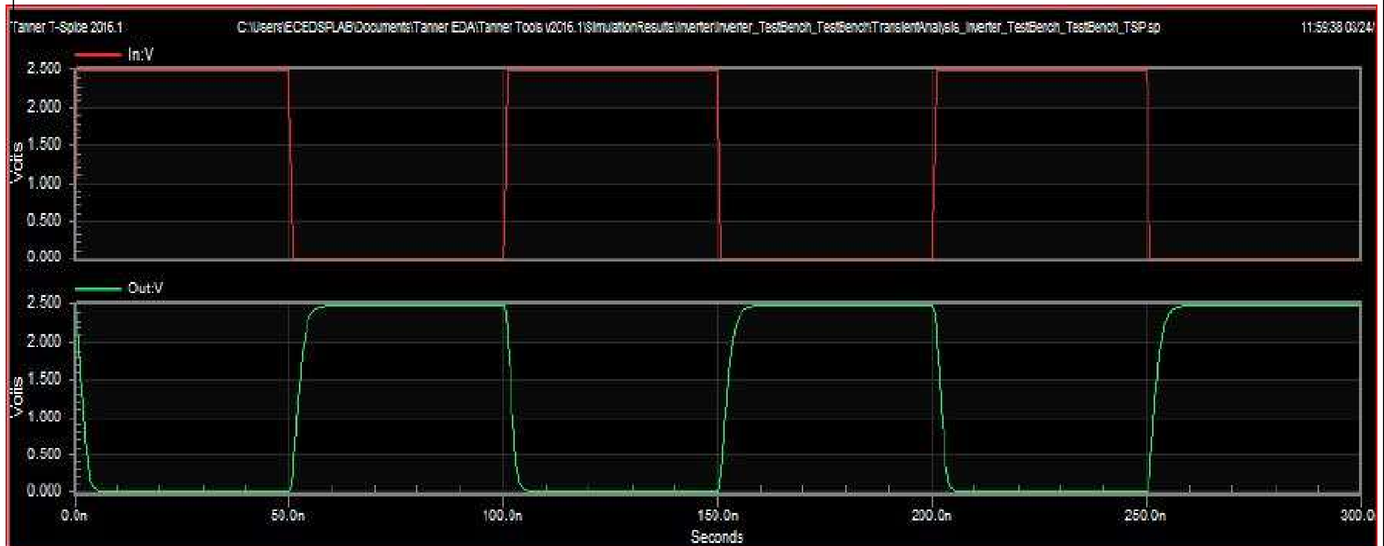
6. Now choose run simulation.
7. Observe the output wave forms and its characteristics.
8. Note down the tabular column

RESULT:

DC ANALYSIS



TRANSIENT ANALYSIS

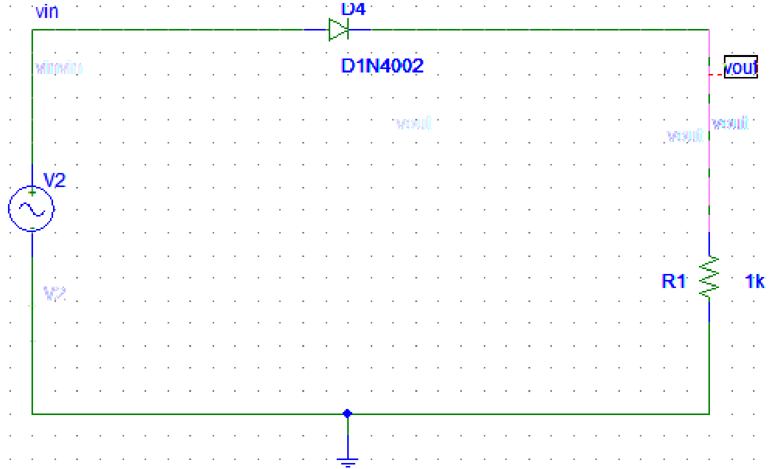


CONCLUSION:

The MOSFET characteristics are analysed using transient analysis and CMOS inverter is designed and its characteristics are derived using DC analysis in PSPICE.

CIRCUIT DIAGRAM:

Half Rectifiers:



V2 PartName: VSIN

Name | Value
REFDES = V2

- REFDES=V2
- TEMPLATE=V^@REFDES %+ %- ?DC|DC @DC| ?AC|AC @
- DC=0
- AC=0
- VOFF=0
- VAMPL=5v
- FREQ=1khz

Save Attr | Change Display | Delete

☑ Include Non-changeable Attributes

☑ Include System-defined Attributes

OK | Cancel

Transient

Transient Analysis

- Print Step: 0ms
- Final Time: 10ms
- No-Print Delay:
- Step Ceiling:

☐ Detailed Bias Pt.

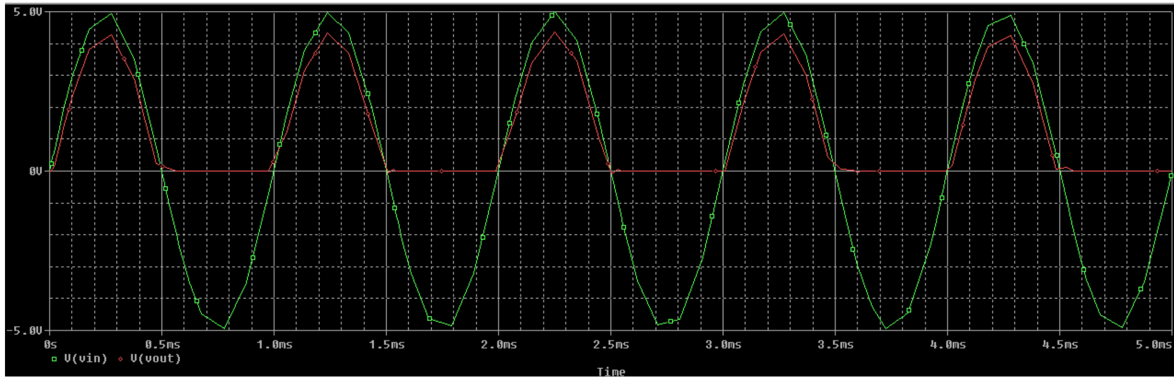
☐ Skip initial transient solution

Fourier Analysis

☐ Enable Fourier

- Center Frequency:
- Number of harmonics:
- Output Vars.:

OK | Cancel



Exp no. 4	Diode based circuits – Transient, Worst-case, MC, Sensitivity, etc. analysis
Date:	

Aim:

To Perform AC and transient analysis of inverting and non-inverting amplifiers and clippers and rectifier circuits using Pspice.

Apparatus:

PC with PSPICE VERSION 9.1

Description:

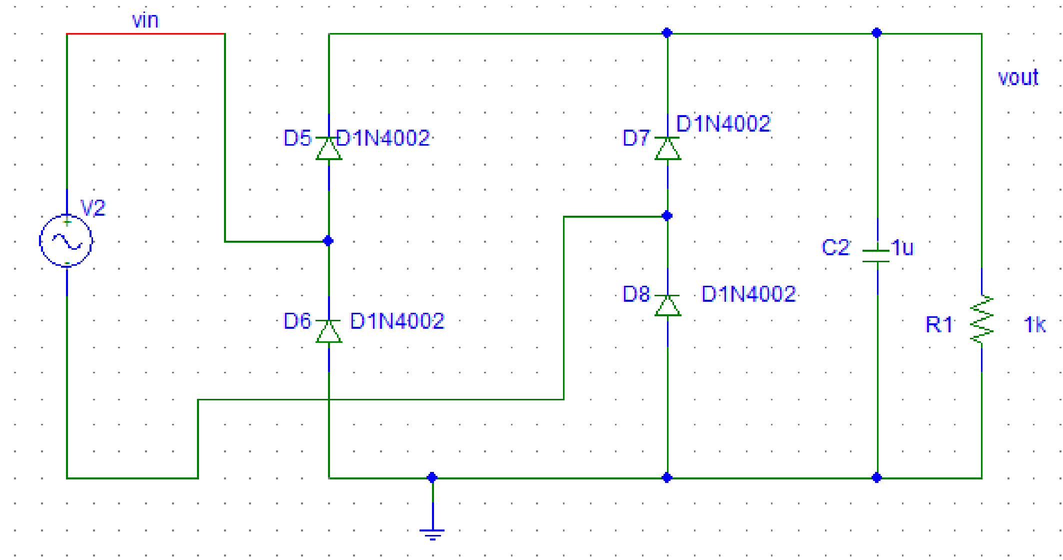
In half wave rectification, either the positive or negative half of the AC wave is passed, while the other half is blocked. Because only one half of the input waveform reaches the output, it is very inefficient if used for power transfer. Half-wave rectification can be achieved with a single diode in a one-phase supply, or with three diodes in a three-phase supply. Half wave rectifiers yield a unidirectional but pulsating direct current.

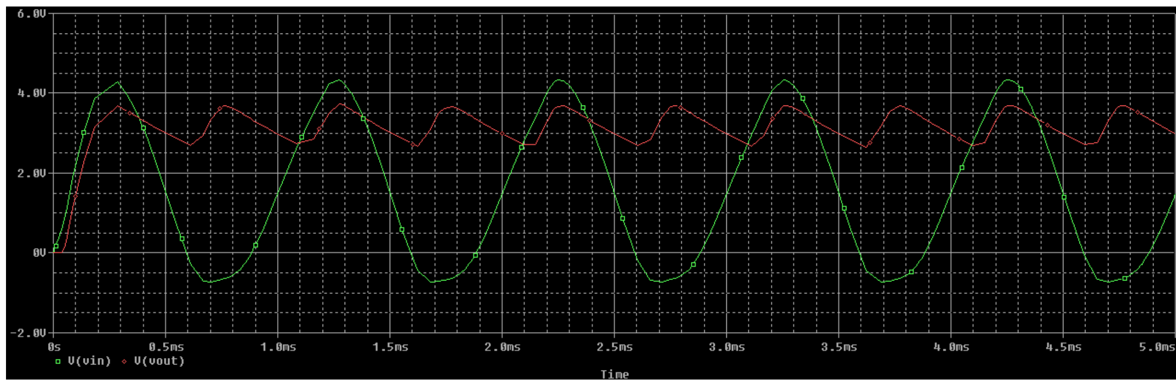
A full-wave rectifier converts the whole of the input waveform to one of constant polarity (positive or negative) at its output. Full-wave rectification converts both polarities of the input waveform to DC (direct current), and is more efficient.

A circuit which removes the peak of a waveform is known as a clipper. A negative clipper is shown in Figure above. During the positive half cycle of the 5 V peak input, the diode is reversed biased. The diode does not conduct. It is as if the diode were not there. The positive half cycle is unchanged at the output V(2) in Figure below. Since the output positive peaks actually overlays the input sinewave V(1), the input has been shifted upward in the plot for clarity.

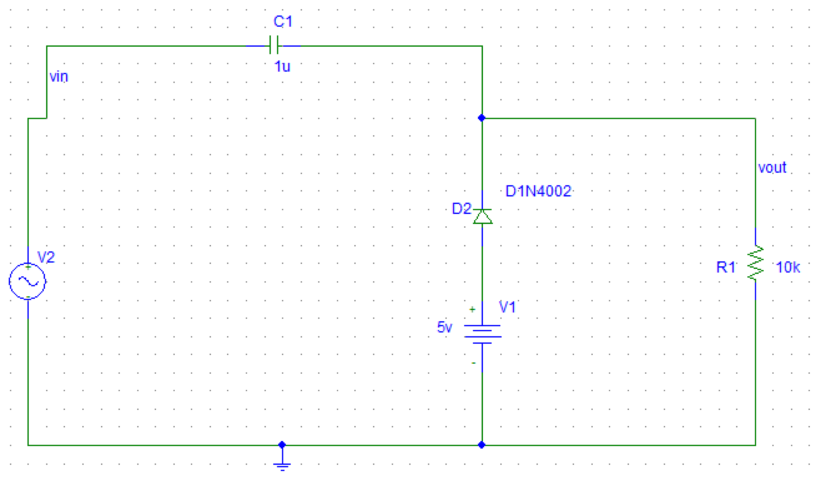
The circuits in Figure above are known as clampers or DC restorers. These circuits clamp a peak of a waveform to a specific DC level compared with a capacitively coupled signal which swings about its average DC level (usually 0V). If the diode is removed from the clamper, it defaults to a simple coupling capacitor– no clamping.

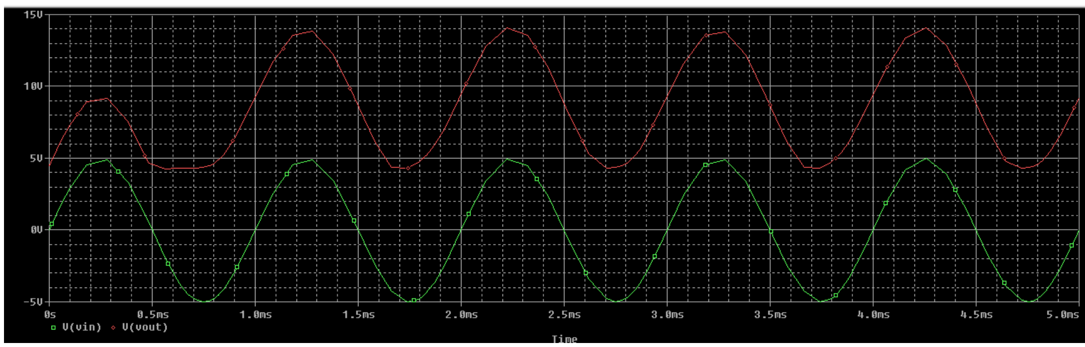
Full rectifier :



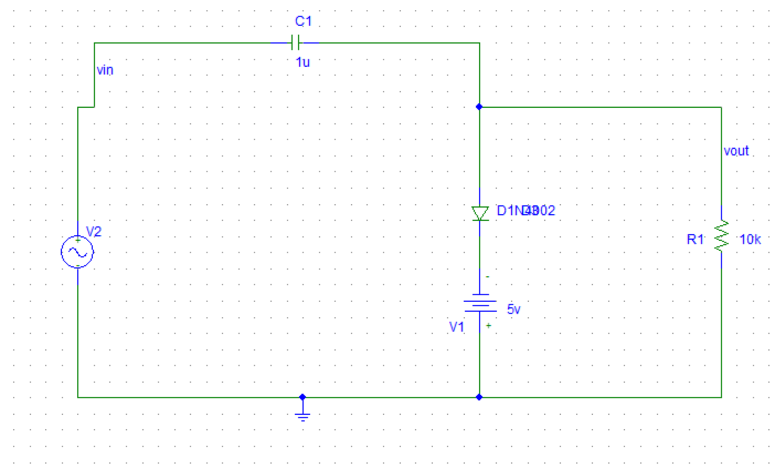


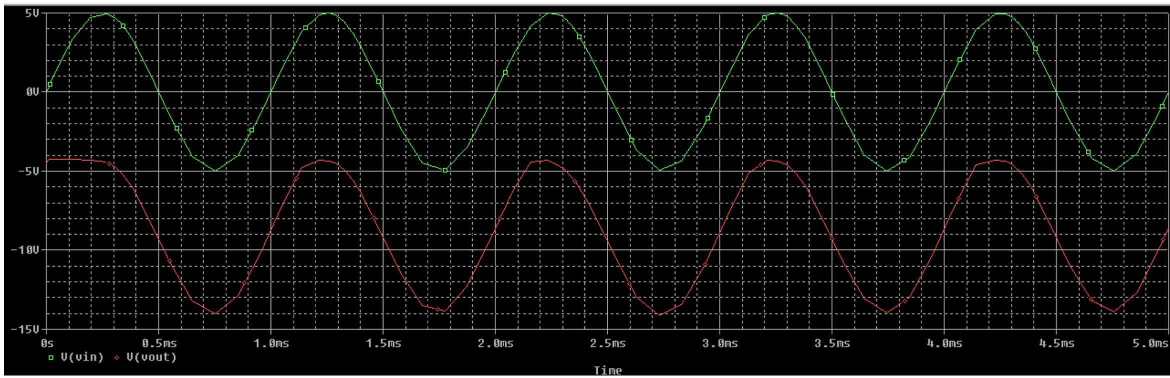
Positive clampper





Negative clampper





Design:

Procedure:

1. Draw the circuit as shown in figure by choosing appropriate devices from the menu titled devices
2. Choose the wire drawing tool from the tool bag and draw the lines.
3. Give the appropriate names and values for all elements present in the circuit.
4. An AC voltage source of ‘0’ phase, 1V amplitude, variable frequency is applied as input signal by editing the voltage source.
5. Then choose set up simulation from simulation menu.
6. Choose the option of AC frequency analysis and give starting and ending

frequency ranges.

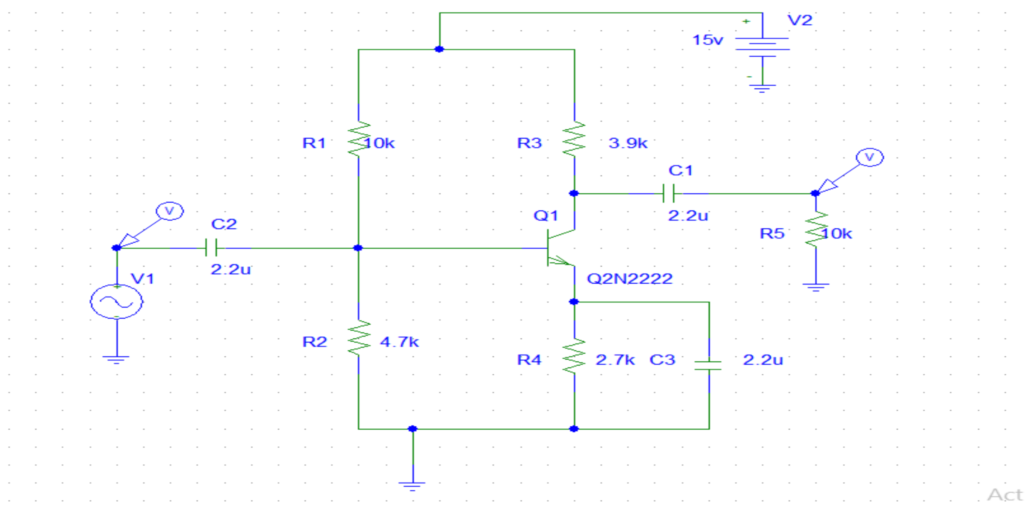
7. Select the option of view table and view graph.
8. Now choose run simulation.
9. Observe the output frequency response graph and take the maximum gain and 3dB frequencies.
10. Note down the tabular column.

Conclusion:

Thus the diode based circuits like clippers,clampers, rectifiers are designed and characteristics are verified using PSPICE.

Circuit diagram:

BJT amplifier:



V1 PartName: VSIN

Name Value

REFDES = V1

Save Attr

Change Display

Delete

REFDES=V1
TEMPLATE=V^@REFDES %+ %- ?DC|DC @DC| ?AC|AC @
DC=
AC=
VOFF=0v
VAMPL=10mv
FREQ=100hz

☑ Include Non-changeable Attributes
☑ Include System-defined Attributes

OK

Cancel

Transient

Transient Analysis

Print Step: 0ms
Final Time: 100ms
No-Print Delay:
Step Ceiling: 100us

☐ Detailed Bias Pt.
☐ Skip initial transient solution

Fourier Analysis

☐ Enable Fourier
Center Frequency:
Number of harmonics:
Output Vars.:

OK Cancel

Exp no. 5	Amplifiers and Current mirrors using BJT/MOSFET
Date:	

Aim:

To analyse the characteristics of amplifiers and current mirrors BJT/MOSFET circuits using Pspice.

Apparatus:

PC with PSPICE VERSION 9.1

Description:

So far we have looked at the bipolar type transistor amplifier and especially the common emitter amplifier, but small signal amplifiers can also be made using Field Effect Transistors or FET's for short. These devices have the advantage over bipolar transistors of having an extremely high input impedance along with a low noise output making them ideal for use in amplifier circuits that have very small input signals. The design of an amplifier circuit based around a junction field effect transistor or "JFET", (n-channel FET) or even a metal oxide silicon FET or "MOSFET" is exactly the same principle as that for the bipolar transistor circuit used for a Class A amplifier circuit we looked at in the previous experiment. Firstly, a suitable quiescent point or "Q-point" needs to be found for the correct biasing of the JFET amplifier circuit with single amplifier configurations of Common-source (CS), Common-drain (CD) or Source-follower (SF) and the Common-gate (CG) available for most FET devices. These three JFET amplifier configurations correspond to the common-emitter, emitter-follower and the common-base configurations using bipolar transistors.

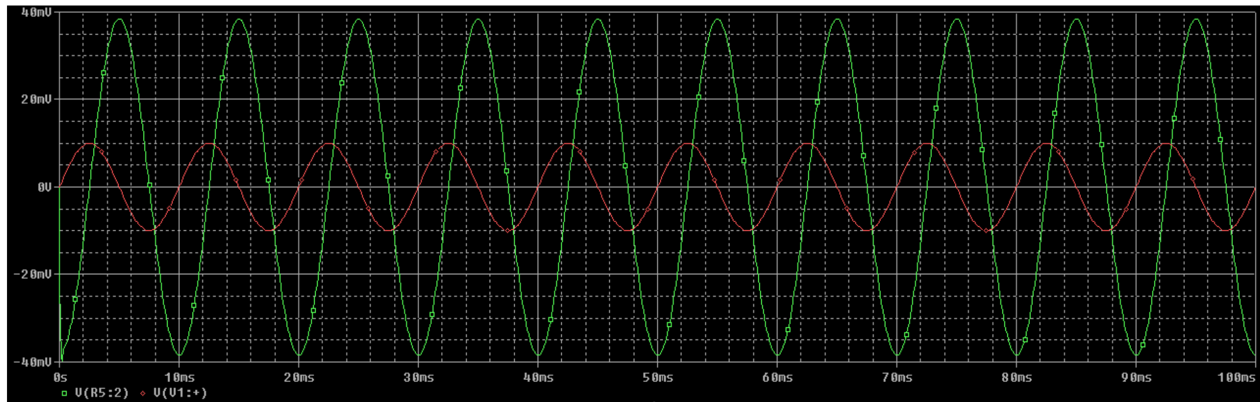
Design:

Procedure:

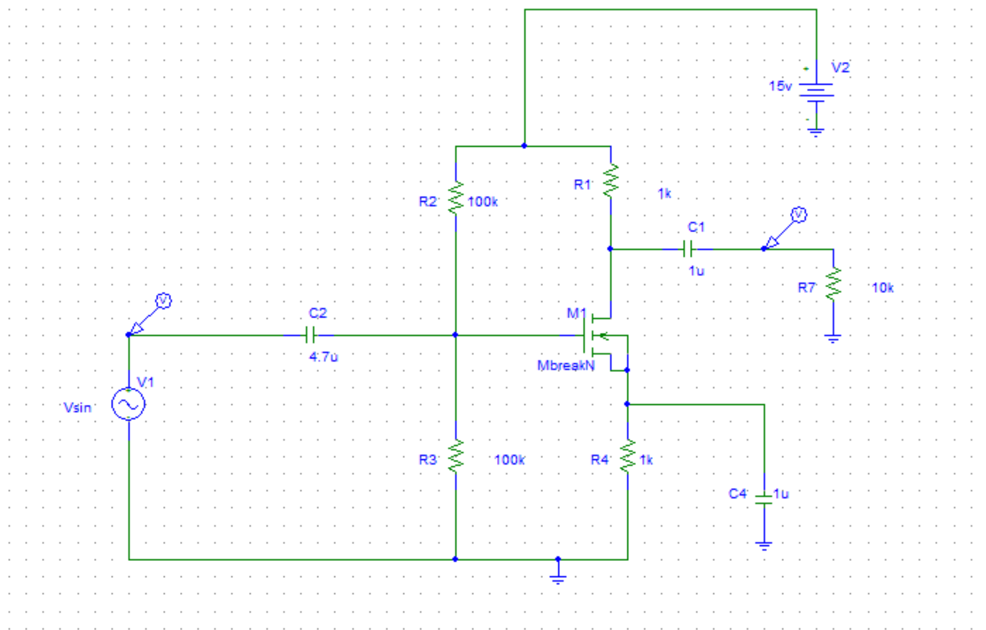
1. Draw the circuit as shown in figure by choosing appropriate devices from the menu titled devices.

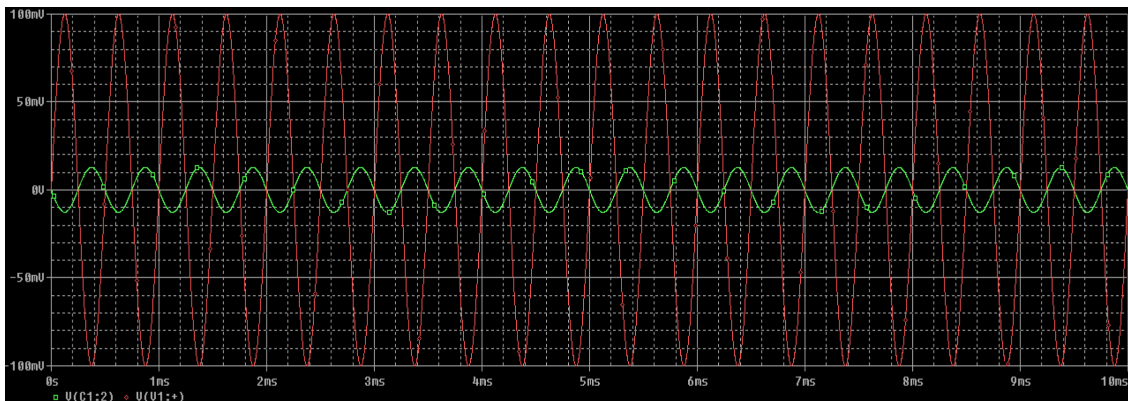
2. Choose the wire drawing tool from the tool bag and draw the lines.

3. Give the appropriate names and values for all elements present in the circuit.



Common source amplifier





4. An AC voltage source of ‘0’ phase, 1V amplitude, variable frequency is applied as input signal by editing the voltage source.

5. Then choose set up simulation from simulation menu.

6. Choose the option of AC frequency analysis and give starting and ending frequency ranges.

7. Select the option of view table and view graph.

8. Now choose run simulation.

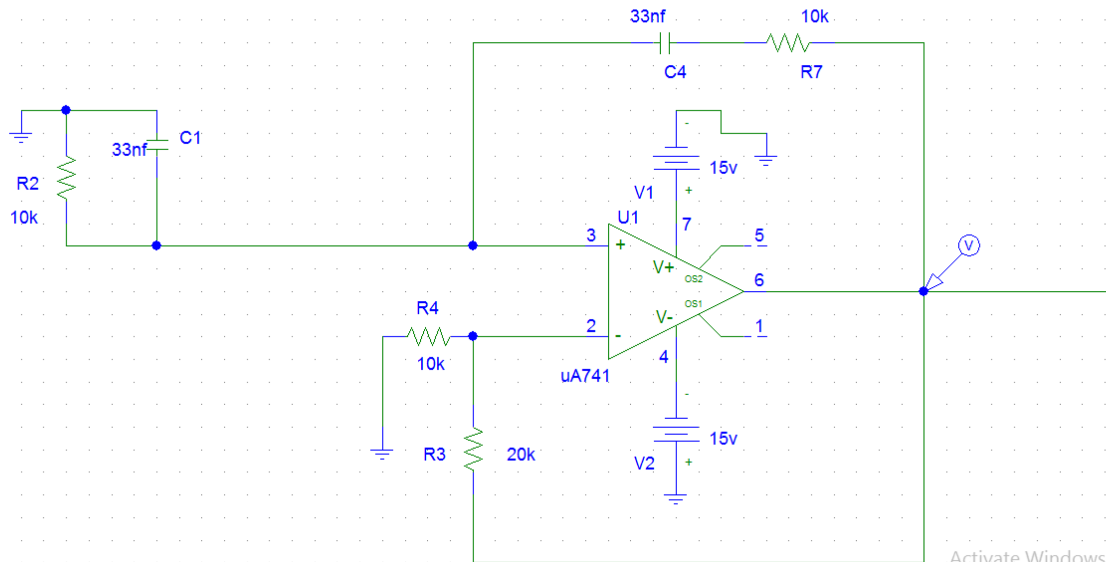
9. Observe the output frequency response graph and take the maximum gain and 3dB frequencies.

10. Note down the tabular column.

Conclusion:

The characteristics of amplifiers and current mirrors using BJT/MOSFET circuits and characteristics are verified in PSPICE.

Circuit diagram:
Wein bridge oscillator:



Transient

Transient Analysis
Print Step: 0ms
Final Time: 20ms
No-Print Delay:
Step Ceiling: 0.01ms
☐ Detailed Bias Pt.
☑ Skip initial transient solution

Fourier Analysis
☐ Enable Fourier
Center Frequency:
Number of harmonics:
Output Vars.:

OK Cancel

Analysis Setup

Enabled		Enabled	
☐	AC Sweep...		Options...
☐	Load Bias Point...	☐	Parametric...
☐	Save Bias Point...	☐	Sensitivity...
☐	DC Sweep...	☐	Temperature...
☐	Monte Carlo/Worst Case...	☐	Transfer Function...
☑	Bias Point Detail	☑	Transient...
	Digital Setup...		

Close

Exp no. 6	Op-Amp based Wein Bridge Oscillator and DAC using sub-circuit and Analog behavioural modelling
Date:	

Aim:

To analyse the characteristics of Op-Amp based wein bridge oscillator and DAC circuits using behavior modelling using Pspice.

Apparatus:

PC with PSPICE VERSION 9.1

Description:

The opamp Wien-bridge oscillator provides a nice view into classic oscillator design using feedback analysis. Feedback analysis reveals if your circuit is stable (well behaved) or unstable (may oscillate). When designing amplifiers (especially high-speed ones), the trick is to avoid the conditions that make the circuit oscillate. When designing oscillators, you strive to achieve those conditions in a predictable way.

FEEDBACK ANALYSIS

Feedback analysis simply means opening the circuit and injecting an AC signal VTEST at one end of the circuit. Then, by looking at the magnitude (gain) and phase (time-shift) of signal as it travels around the opened loop, you can tell whether you've got an amplifier or an oscillator on your hands.

Digital to Analog Converters:

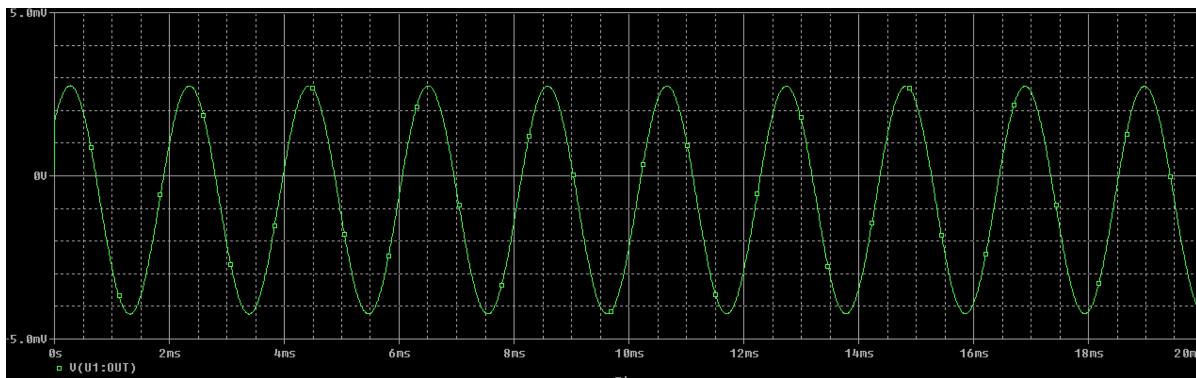
There are several ways to build digital to analog converters, but the simplest way is with a scaled resistor network.This is just an Op-Amp adder circuit, in which each bit's b_n is added in with weight 2^{b_n}. In practice, it is difficult to make a high resolution DAC with a scaled resistor network because it requires precisely scaled resistors over a very wide range. It is much easier to make precise resistors over a narrow range; a common DAC that takes advantage of this fact uses a ladder network:

Design:

Procedure:

1. Draw the circuit as shown in figure by choosing appropriate devices from the menu titled devices
2. Choose the wire drawing tool from the tool bag and draw the lines.
3. Give the appropriate names and values for all elements present in the circuit.
4. An AC voltage source of ‘0’ phase, 1V amplitude, variable frequency is applied as input signal by editing the voltage source.
5. Then choose set up simulation from simulation menu.
6. Choose the option of analysis and give starting and ending frequency ranges.
7. Select the option of view table and view graph.
8. Now choose run simulation.
9. Note down the tabular column.

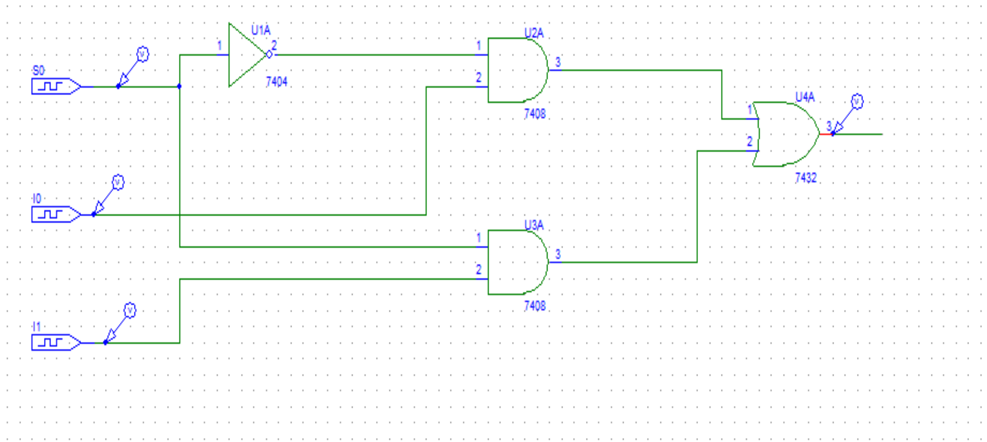
RESULT:



Conclusion:

Thus the characteristics of the opamp based wein bridge oscillator and DAC are verified by designing and simulation using Pspice.

Circuit Diagram:

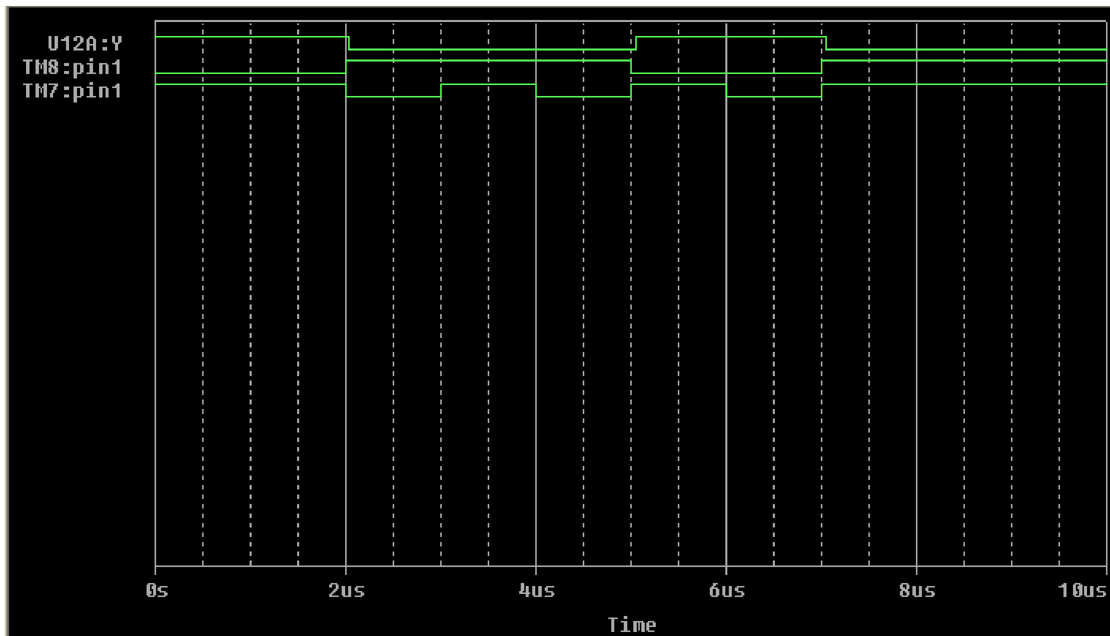


-

Truth Table

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

Output:



Exp no. 7	Digital Circuits – Logic switches / Multiplexer / Counter
Date:	

Aim :

To Implement the 4:1 multiplexer, counter, logic switches using PSPICE

System Requirements:

PC with PSPICE VERSION 9.1

DESCRIPTION:

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

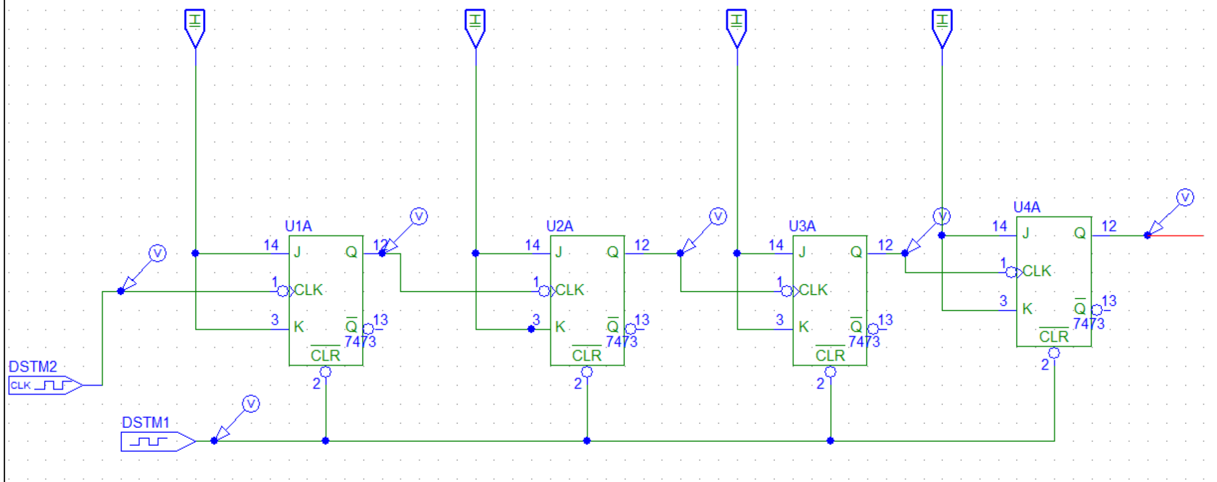
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

Procedure:

1. Regup the circuit as shown in figure by choosing appropriate devices from the menu titled devices
2. Choose the wire drawing tool from the tool bag and draw the lines.
3. Give the appropriate names and values for all elements present in the circuit.
4. An AC voltage source of ‘0’ phase, 1V amplitude, variable frequency is applied as input signal by editing the voltage source.
5. Then choose set up simulation from simulation menu.
6. Choose the analysis and give starting and ending Frequency ranges.
7. Select the option of view table and view graph.
8. Now choose run simulation.
9. Note down the tabular column.

Circuit diagram: COUNTER



DSTM1 PartName: STIM1

Name Value

TIMESTEP =

TIMESTEP=
COMMAND1=0s 0
COMMAND2=1us 1
COMMAND3=
COMMAND4=
COMMAND5=
COMMAND6=

Save Attr
Change Display
Delete
OK
Cancel

☑ Include Non-changeable Attributes
☑ Include System-defined Attributes

DSTM2 PartName: DigClock

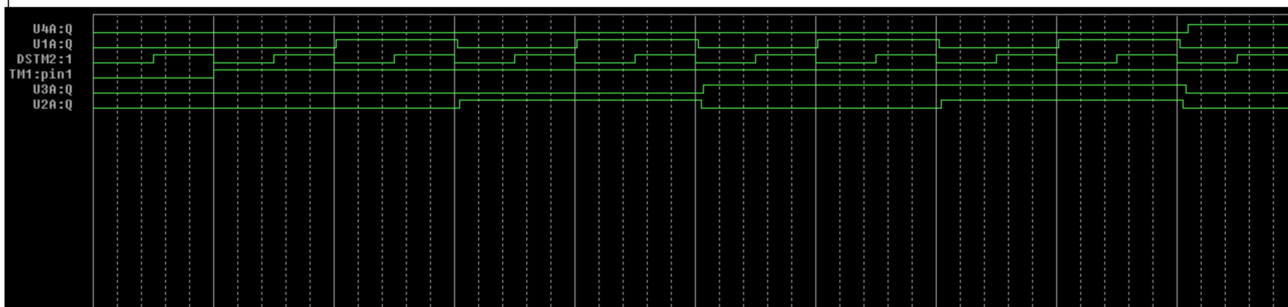
Name Value

DELAY =

DELAY=
ONTIME=.5uS
OFFTIME=.5uS
STARTVAL=0
OPPVAL=1
IO_MODEL=IO_STM
IO_LEVEL=0

Save Attr
Change Display
Delete
OK
Cancel

☑ Include Non-changeable Attributes
☑ Include System-defined Attributes



Conclusion:

Thus the 4:1 multiplexer, counter and logic switchesare designed using Pspice.

VHDL AND VERILOG

Exp no. 8	Study of VHDL and Verilog
Date:	

Aim :

To understand system Verilog and VHDL.

System Requirements:

PC with PSPICE VERSION 9

DESCRIPTION:

HDL

In electronics, a hardware description language or HDL is any language from a class of Computer languages for formal description of electronic circuits. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation HDLs are standard text-based expressions of the spatial, temporal structure and behavior of electronic systems. In contrast to a software programming language, HDL syntax, semantics include explicit notations for expressing time and concurrency, which are the attributes of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchies of blocks are properly classified as netlist languages.

HDLs are used to write executable specifications of some piece of hardware. A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this execute ability that gives HDLs the illusion of being programming languages. Simulators capable of supporting discrete-event and continuous-time (analog) modeling exist, and HDLs targeted for each are available.

It is certainly possible to represent hardware semantics using traditional programming languages such as C++, although to function such programs must be augmented with extensive and unwieldy class libraries. Primarily, however, software programming languages function as a hardware description language Using the proper subset of virtually any language, a software program called a synthesizer can infer hardware logic operations from the language statements and

produce an equivalent netlist of generic hardware primitives to implement the specified behavior. This typically requires the synthesizer to ignore the expression of any timing constructs in the text. The two most widely-used and well-supported HDL varieties used in industry are

- VHDL (VHSIC HDL)
- Verilog

VHDL

VHDL (Very High Speed Integrated Circuit Hardware Description Language) is commonly used as a design-entry language for field-programmable gate arrays and applicationspecific integrated circuits in electronic design automation of digital circuits. VHDL is a fairly general-purpose language, and it doesn’t require a simulator on which to run the code. There are a lot of VHDL compilers, which build executable binaries. It can read and write files on the host computer, so a VHDL program can be written that generates another VHDL program to be incorporated in the design being developed. Because of this generalpurpose nature, it is possible to use VHDL to write a test bench that verifies with the user, and compares results with those expected.

This is similar to the capabilities of the Verilog language VHDL is not a case sensitive language. One can design hardware in a VHDL IDE (such as Xilinx) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software (such as ModelSim) which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. To generate an appropriate test bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

The key advantage of VHDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). When a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, then it is

the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip. Both VHDL and Verilog emerged as the dominant HDLs in the electronics industry while older and less-capable HDLs gradually disappeared from use. But VHDL and Verilog share many of the same limitations: neither HDL is suitable for analog/mixed-signal circuit simulation. Neither possesses language constructs to describe recursively-generated logic structures

Verilog:

Verilog is a hardware description language (HDL) used to model electronic systems. The language supports the design, verification, and implementation of analog, digital, and mixed - signal circuits at various levels of abstraction The designers of Verilog wanted a language with syntax similar to the C programming language so that it would be familiar to engineers and readily accepted. The language is case sensitive, has a preprocessor like C, and the major control flow keywords, such as "if" and "while", are similar. The formatting mechanism in the printing routines and language operators and their precedence are also similar The language differs in some fundamental ways.

Verilog uses Begin/End instead of curly braces to define a block of code. The concept of time, so important to a HDL won't be found in C.The language differs from a conventional programming language in that the execution of statements is not strictly sequential. A Verilog design consists of a hierarchy of modules are defined with a set of input, output, and bidirectional ports. Internally, a module contains a list of wires and registers. Concurrent and sequential statements define the behavior of the module by defining the relationships between the ports, wires, and registers Sequential statements are placed inside a begin/end block and executed in sequential order within the block.

But all concurrent statements and all begin/end blocks in the design are executed in parallel, qualifying Verilog as a Dataflow language. A module can also contain one or more instances of another module to define sub-behavior.

A subset of statements in the language is synthesizable. If the modules in a design contains a netlist that describes the basic components and connections to be implemented in hardware only synthesizable statements, software can be used to transform or synthesize the design into the net list may then be transformed into, for example, a form describing the standard cells of an integrated circuit (e.g. ASIC) or a bit stream for a programmable logic device (e.g. FPGA).

Design using HDL

The vast majority of modern digital circuit design revolves around an HDL description of the desired circuit, device, or subsystem. Most designs begin as a written set of requirements or a high-level architectural diagram. The process of writing the HDL description is highly dependent on the designer's diagram. The process of writing the HDL description is highly dependent on the designer's background and the circuit's nature. The HDL is merely the 'capture language'–often begin with a high-level algorithmic description such as MATLAB or a C++ mathematical model Control and decision structures are often prototyped in flowchart applications, or entered in a state-diagram editor. Designers even use scripting languages (such as PERL) to automatically generate repetitive circuit structures in the HDL language. Advanced text editors (such as PERL) to automatically generate repetitive circuit structures in the HDL language. Advanced text editors (such as Emacs) offer editor templates for automatic indentation, syntaxdependent coloration, and macro-based expansion of entity/architecture/signal declaration. As the design's implementation is fleshed out, the HDL code invariably must undergo code review, or auditing. In preparation for synthesis, the HDL description is subject to an array of automated checkers. The checkers enforce standardized code guidelines, identifying ambiguous code construct before they can cause misinterpretation by downstream synthesis, and check for common logical coding errors, such as dangling ports or shorted outputs. In industry parlance, HDL design generally ends at the synthesis stage. Once the synthesis tool has mapped the HDL description into a gate net list, this net list is passed off to the back - end stage. Depending on the physical technology (FPGA, ASIC gate-array, ASIC standardcell), HDLs may or may not play a significant role in the back-end flow. In

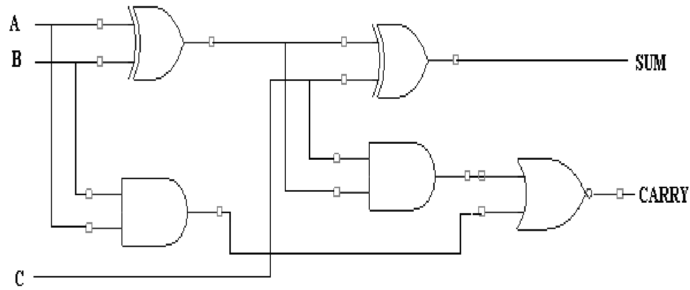
general, as the design flow progresses toward a physically realizable form, the design database becomes progressively more laden with technology-specific information, which cannot be becomes progressively more laden with technology-specific information, which cannot be stored in a generic HDL-description. Finally, a silicon chip is manufactured

HDL Programming using Xilinx ISE design suite Xilinx ISE means Xilinx® Integrated Software Environment (ISE), i.e programmable logic design tool in electronics industry. This Xilinx ® design software suite allows taking design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes design through several steps in the ISE design flow. These steps are Design Entry, Synthesis, Implementation, Simulation/Verification, and Device Configuration. Xilinx is one of most popular software tool used to synthesize VHDL/Verilog code.

CONCLUSION:

Thus the introduction to Verilog and VHDL is written and process of designing is written.

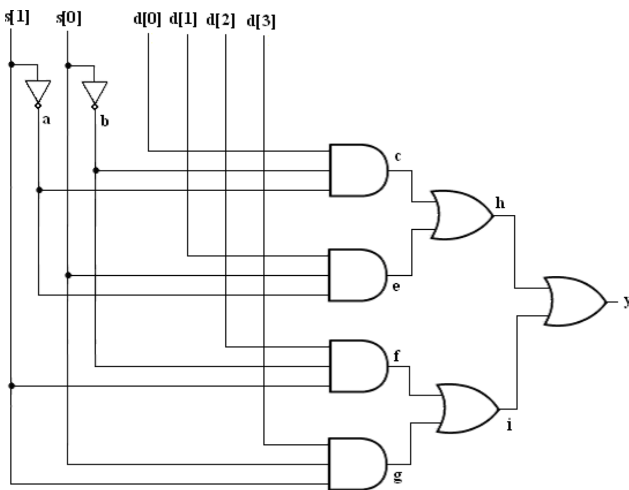
LOGIC DIAGRAM:



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

MULTIPLEXER:



TRUTH TABLE

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

Exp no. 9	Full Adder and Multiplexer using different Modelling / Descriptions and Concurrent and Sequential execution in VHDL
Date:	

Aim :

To design the full adder and multiplexer using different model styles in VHDL.

System Requirements:

a) Xilinx (ISE) simulator 9.1
b) FPGA KIT

DESCRIPTION:

The full adder is a digital circuit that performs the addition of three numbers. It is implemented using logic gates. A one-bit full adder adds three one-bit binary numbers (two input bits, mostly A and B, and one carry bit Cin being carried forward from previous addition) and outputs a sum and a carry bit.

A multiplexer is a data selector device that selects one input from several input lines, depending upon the enabled, select lines, and yields one single output.

A multiplexer of 2^n inputs has n select lines, are used to select which input line to send to the output. There is only one output in the multiplexer, no matter what's its configuration. These devices are used extensively in the areas where the multiple data can be transferred over a single line like in the communication systems and bus architecture hardware. Visit this post for a crystal clear explanation to multiplexers.

DESIGN:

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.

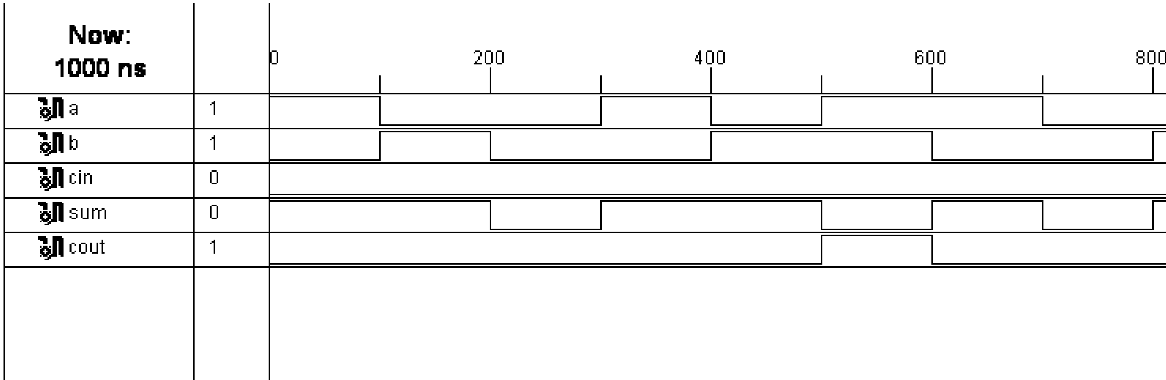
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE simulator.
8	Verify the output.
9	Follow the instructions given to implement it with the available FPGA kit.

VHDL CODE:

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;
```

RESULT:



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----

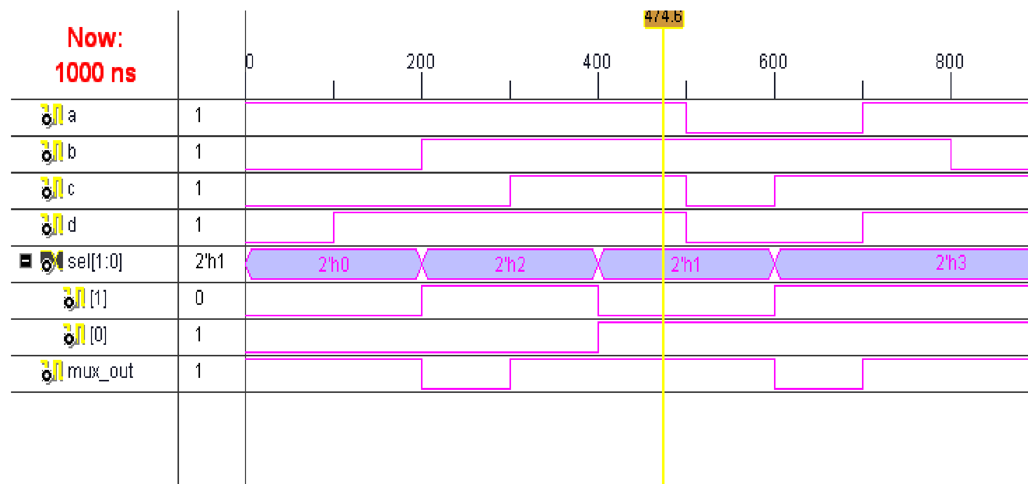
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fulladder_structural is
```

```
 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;
```

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;
```

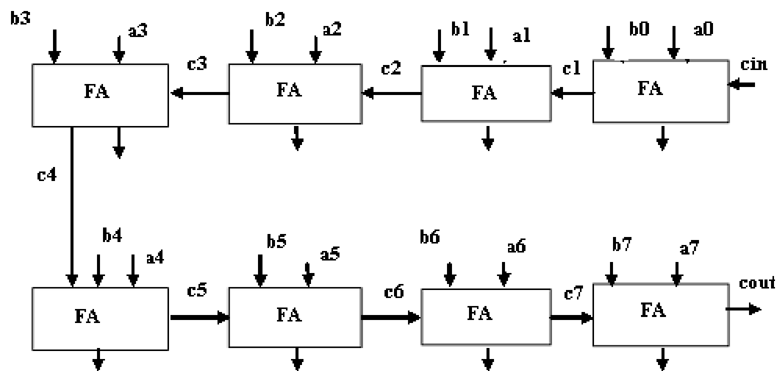
OUTPUT:



CONCLUSION:

Thus the Full adder and 4:1 multiplexer Circuits has been simulated with the truth table by using Xilinx ISE Simulator.

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
```

Exp no. 10	8-bit Adder / Multiplier (min 4-bit) – in VHDL
Date:	

Aim :

To design the 8- bit adder and multiplier using Port Map, Generics, Technology MappingLin VHDL.

System Requirements:

1. Xilinx (ISE) simulator 9.1
2. FPGA KIT

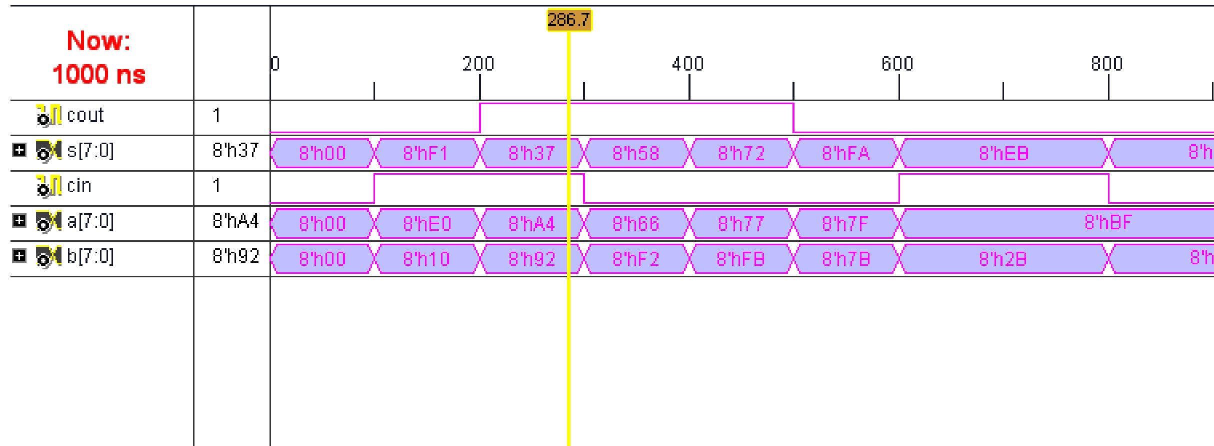
Description :

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. The most common type is a sequential digital logic circuit with an input line called the *clock* and multiple output lines. The values on the output lines represent a number in the binary or BCD number system. Each pulse applied to the clock input increments or decrements the number in the counter.

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.

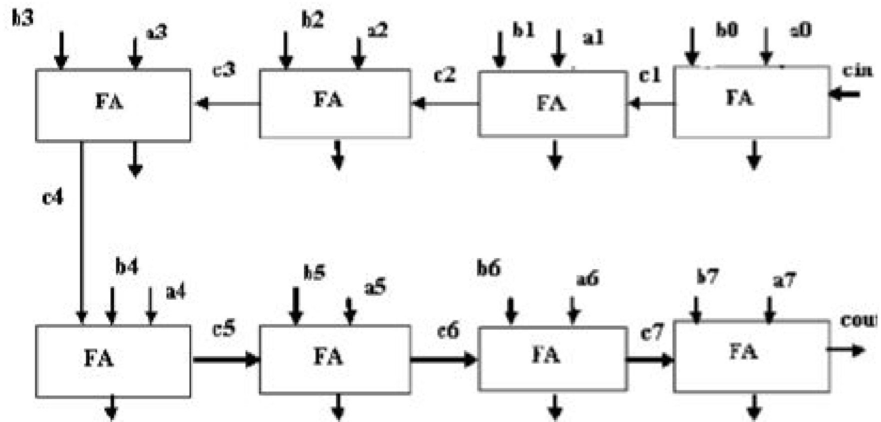
OUTPUT:



CONCLUSION:

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

LOGIC DIAGRAM:



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

Exp no. 11	8-bit Counter – Bottom up approach design and Test vector generation in Verilog HDL
Date:	

Aim:

1. To Simulate the ripple counter with the tools available in Xilinx Project Navigator using Verilog.
2. To Implement the above with the available FPGA kit.

Facilities Required:

Xilinx (ISE) simulator 9.1
FPGA KIT

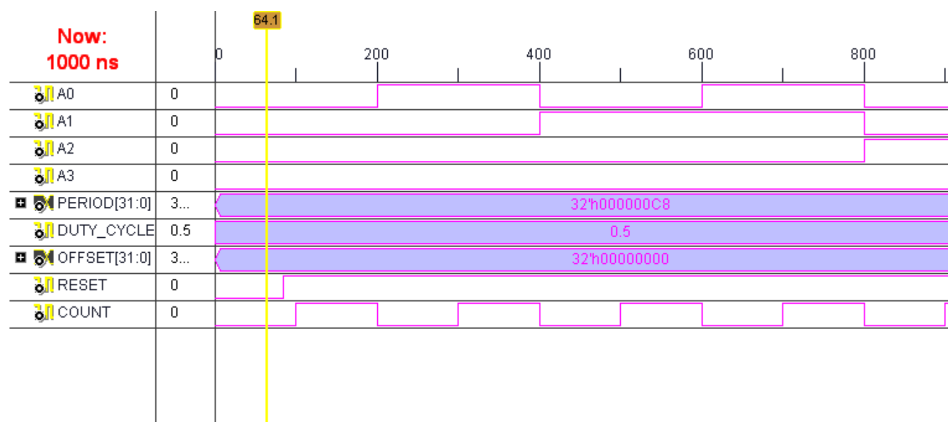
Procedure for doing the experiment:

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.

VERILOG SOURSE CODE: //Structural description of Ripple Counter

```
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 Ripple Counter Circuit has been simulated with the truth table by using Xilinx ISE Simulator

NAND GATE

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	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;
```

Exp no. 12	NAND / NOR / Transmission gates using Switch level modelling in Verilog HDL
Date:	

AIM:

To design the NAND/NOR gates using switch level modelling in Verilog HDL.

Apparatus:

Xilinx (ISE) simulator 9.1

FPGA KIT

DESCRIPTION:

Digital logic circuits are non linear networks that use transistors as electronic switches to divert one of the supply voltages VDD or 0V to the output. This corresponds to a logic result of f-1 or f=0. An important characteristic of a CMOS circuit is the duality that exists between its PMOS transistors and NMOS transistors.

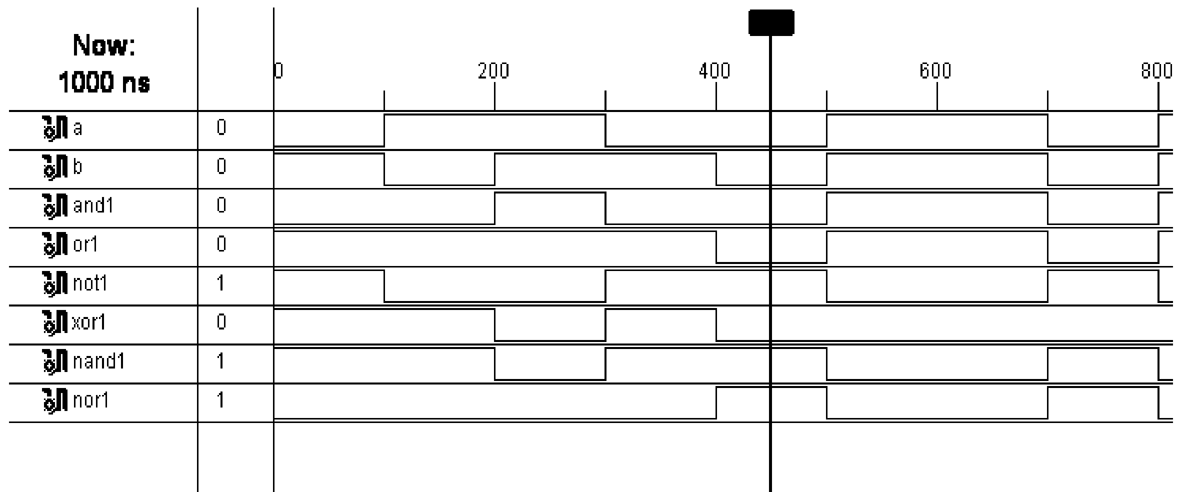
According to the De Morgan's laws based logic, the PMOS transistors in parallel have corresponding NMOS transistors in series while the PMOS transistors in series have corresponding NMOS transistors in parallel. The NOT or the INVERT function is the simplest Boolean operation. It has an input _a' and produces output f(a) i.e. it implements logical negation. An important feature of CMOS is the manner in which complementary FET pair ensures that always a path from the output to either the power source VDD or ground. To accomplish this, the set of all paths to the voltage source must be the complement of the set of all paths to ground. This can be easily accomplished by defining one in terms of the NOT of the other.

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.

RESULT:

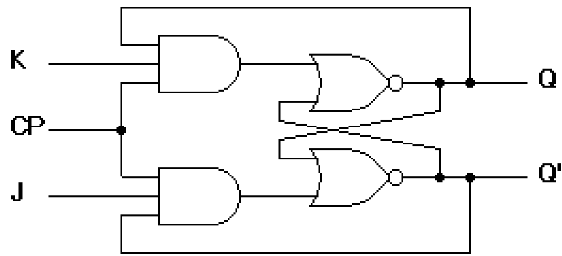


RESULT:

Thus the design of the NAND/NOR gates using switch level modelling in Verilog HDL is done and truth table is verified.

LOGIC DIAGRAM:

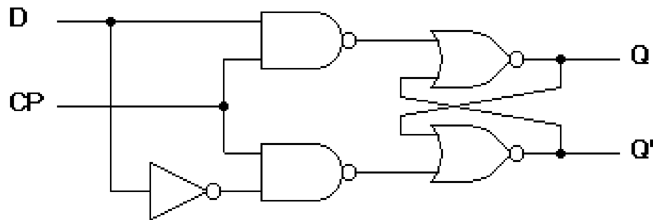
JK FLIP FLOP:



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

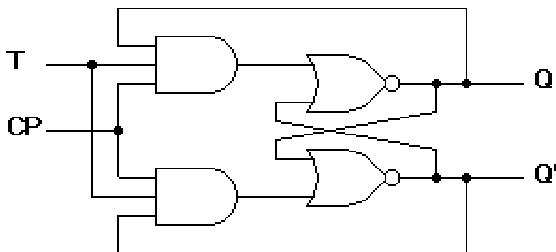
D Flip Flop:



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

Exp no. 13	design of simple sequential and combinational circuits
Date:	

Aim:

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:

Xilinx (ISE) simulator 9.1 **&** 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.

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=~te
mp;
 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
     end
endmodule
```

Result:

Thus the flipflop circuit has been simulated with the truth table by using Xilinx ise simulator.

Exp no. 14	Design of ALU
Date:	

Aim:

a) To design the arithemetical and logical unit(ALU) with the tools available in Xilinx Project Navigator using Verilog.

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

Facilities Required:

Xilinx (ISE) simulator 9.1 **&** FPGA KIT

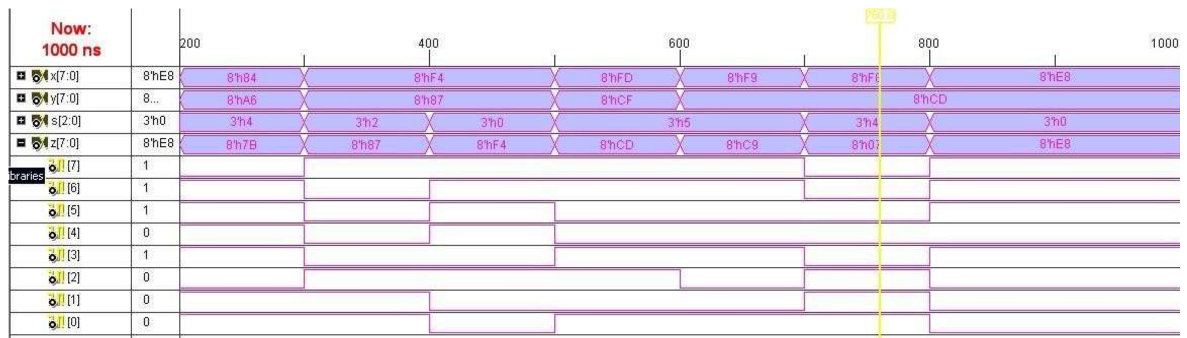
Procedure for doing the experiment:

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;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity ALU is
    Port ( x,y : in STD_LOGIC_VECTOR (7 downto 0);
           s : in STD_LOGIC_VECTOR (2 downto 0);
           z : out STD_LOGIC_VECTOR (7 downto 0));
end ALU;
architecture dataflow of ALU is
signal arith,logic:std_logic_vector(7 downto 0);
begin
With s(2 downto 0)select
arith <=x when "000",
      x+1 when "001",
      y when "010",
      x+y when others;
With s(2 downto 0)select
        logic <=not x when "100",
         x and y when "101",
         x or y when "110",
         x xor y when others;
with s(2)
select z <= arith when '0',
         logic when others;
end dataflow;
```

RESULT



CONCLUSION

Thus the implement and stimulate of ALU using VHDL is done.

Exp no. 15	Design of FSM and Control Unit
Date:	

Aim:

a) To design the FSM and Control Unit with the tools available in Xilinx Project Navigator using Verilog

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

Facilities Required:

Xilinx (ISE) simulator 9.1 **&** FPGA KIT

Procedure for doing the experiment:

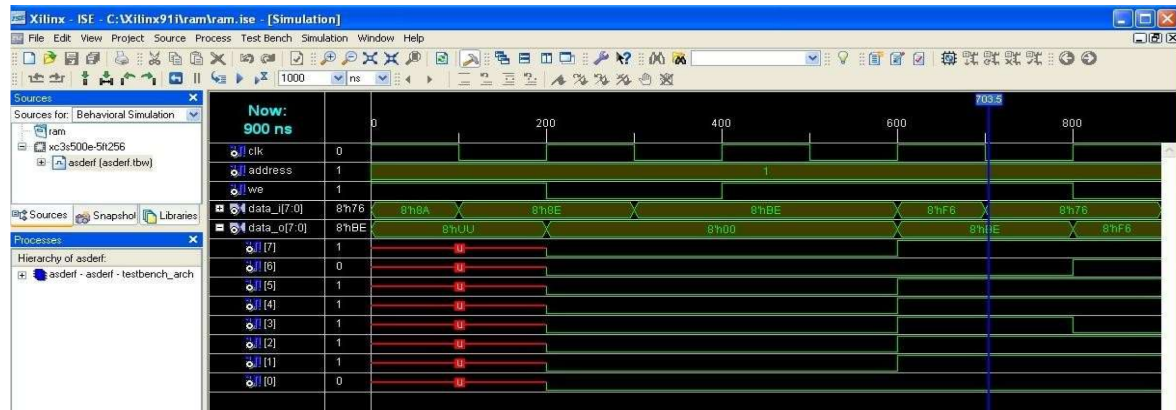
S.No	Details of the step
1	Double click the project navigator and select the option File-New project.
2	Give the project name.
3	Select VHDL module.
4	Type your VHDL coding.
5	Check for syntax.
6	Select the new source of test bench waveform
7	Choose behavioural simulation and simulate it by Xilinx ISE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

VHDL code:

1) To implement and simulate RAM using VHDL

```
entity ram_example is
port (Clk : in std_logic;
        address : in integer;
        we : in std_logic;
        data_i : in std_logic_vector(7 downto 0);
        data_o : out std_logic_vector(7 downto 0)     );
end ram_example;
architecture Behavioral of ram_example is
--Declaration of type and signal of a 256 element RAM
--with each element being 8 bit wide.
type ram_t is array (0 to 255) of std_logic_vector(7 downto 0);
signal ram : ram_t := (others => (others => '0'));
begin
--process for read and write operation.
PROCESS(Clk)
BEGIN
    if(rising_edge(Clk)) then
        if(we='1') then
            ram(address) <= data_i;
        end if;
        data_o <= ram(address);
    end if;
END PROCESS;
end Behavioral;
```

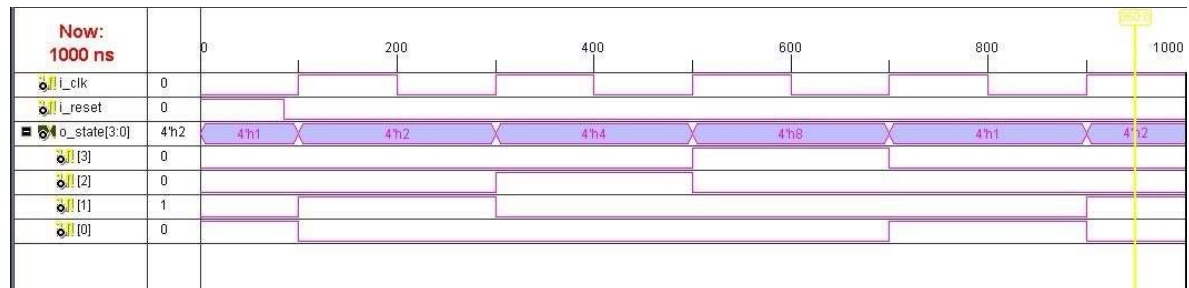
OUTPUT:



To implement and simulate Control Unit VHDL

```
entity control is
   Port ( I_clk : in STD_LOGIC;
          I_reset : in  STD_LOGIC;
          O_state : out STD_LOGIC_VECTOR (3 downto 0));
end control;
architecture Behavioral of control is
  signal s_state: STD_LOGIC_VECTOR(3 downto 0) := "0001";
begin
  process(I_clk)
  begin
    if rising_edge(I_clk) then
      if I_reset = '1' then
      s_state <= "0001";
      else
        case s_state is
          when "0001" =>
            s_state <= "0010";
          when "0010" =>
            s_state <= "0100";
          when "0100" =>
            s_state <= "1000";
          when "1000" =>
            s_state <= "0001";
        when others =>
        s_state <= "0001";
        end case;
      end if;
    end if;
  end process;
O_state <= s_state;
end Behavioral;
```

OUTPUT:



<u>**Result:**</u>

Thus the implementation and stimulation of Control unit and ram using VHDL is done.