ST.ANNE'S

COLLEGE OF ENGINEERING AND TECHNOLOGY

ANGUCHETTYPALAYAM, PANRUTI – 607 106

EC6612 VLSI DESIGN LAB

OBSERVATION NOTE

(FOR III B.E ELECTRONICS AND COMMUNICATION ENGINEERING STUDENTS)

NAME : _____

REGISTER NO : _____

SEMESTER : _____

DECEMBER 2018 - MAY 2018

AS PER ANNA UNIVERSITY (CHENNAI) SYLLABUS

2013 REGULATION

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PREPARED BY: Mr. S. BALABASKER

ABOUT OBSERVATION NOTE & PREPARATION OF RECORD

- This Observation contains the basic diagrams of the circuits enlisted in the syllabus of EC6612 VLSI DESIGN LAB course, along with the design the design of various components of the Integrated circuit
- Aim of the experiment is also given at the beginning of each experiment. Once the student is able to design the circuit as per the circuit diagram, he/she is supposed to go through the instructions carefully and do the experiments step by step.
- They should note down the readings (observations) and tabulate them as specified.
- It is also expected that the students prepare the theory relevant to the experiment referring to prescribed reference book/journals in advance, and carry out the experiment after understanding thoroughly the concept and procedure of the experiment.
- They should get their observations verified and signed by the staff within two days and prepare & submit the record of the experiment while they come for the laboratory in subsequent week.
- The record should contain experiment No., Date ,Aim, Apparatus required, Theory, Procedure and result on one side(i.e., Right hand side, where rulings are provided) and Circuit diagram, Design, Model Graphs, Tabulations and Calculations on the other side (i.e., Left hand side, where no rulings is provided)
- All the diagrams and table lines should be drawn in pencil
- The students are directed to discuss & clarify their doubts with the staff members as and when required. They are also directed to follow strictly the guidelines specified.

EC6612 VLSI DESIGN LAB

SYLLABUS

FPGA BASED EXPERIMENTS.

1. HDL based design entry and simulation of simple counters, state machines, adders and multipliers .

2. Synthesis, P&R and post P&R simulation of the components simulated in (I) above. Critical paths and static timing analysis results to be identified. Identify and verify possible conditions under which the blocks will fail to work correctly.

3. Hardware fusing and testing of each of the blocks simulated in (I). Use of either chipscope feature (Xilinx) or the signal tap feature (Altera) is a must. Invoke the PLL and demonstrate the use of the PLL module for clock generation in FPGAs.

IC DESIGN EXPERIMENTS: (BASED ON CADENCE / MENTOR GRAPHICS / EQUIVALENT)

4. Design and simulation of a simple 5 transistor differential amplifier. Measure gain, ICMR, and CMRR

5. Layout generation, parasitic extraction and resimulation of the circuit designed in (1)

6. Synthesis and Standard cell based design of an circuits simulated in 1(I) above. Identification of critical paths, power consumption.

7. For expt (c) above, P&R, power and clock routing, and post P&R simulation.

8. Analysis of results of static timing analysis.

LIST OF EXPERIMENTS

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

LIST OF EXPERIMENTS

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

XILINX

TOOLS

EXP.NO:	STUDY OF SIMULATION TOOLS
DATE:	

AIM:

To study simulation tools using Xilinx software tool.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

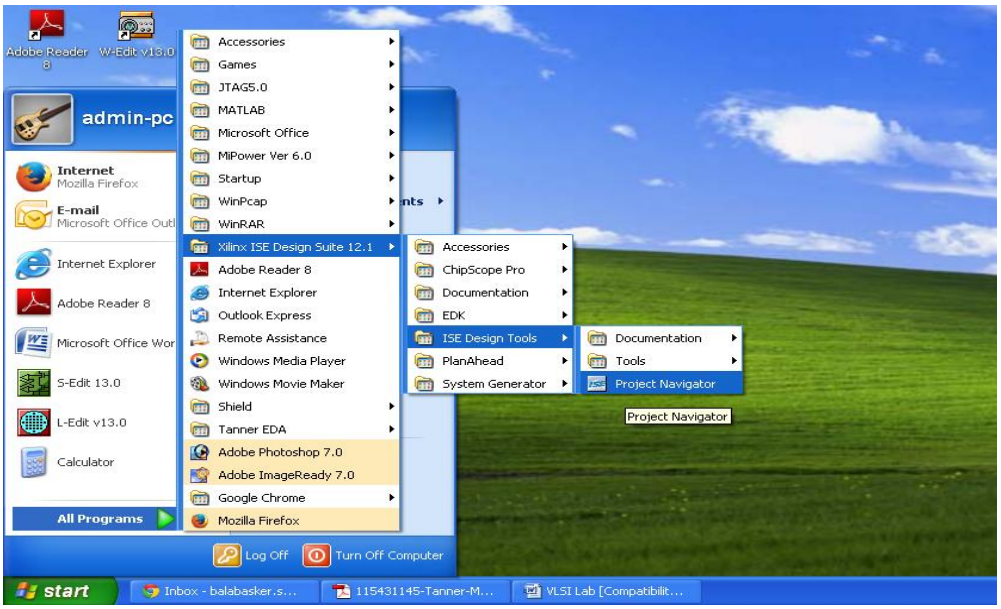
PROCEDURE:

1. Now start the Xilinx ISE Design Suite 12.1
2. Go to file and click new project
3. Enter the project name and click next
4. Select the family name is Spartan 3E, speed is -4 and simulator is verilog click next and click Finish.
5. Click new source.
6. Select verilog module and type file name and click next.
7. Assign input and output port and click next.
8. Finally the report is shown click finish.
9. Type the program save and click synthesis.
10. To see the output wave form change the source from implementation to simulation and click simulator behavior model in ISim simulator.
11. Give values to the input variables and then click run
12. In wave window, click run icon and you can see corresponding output.

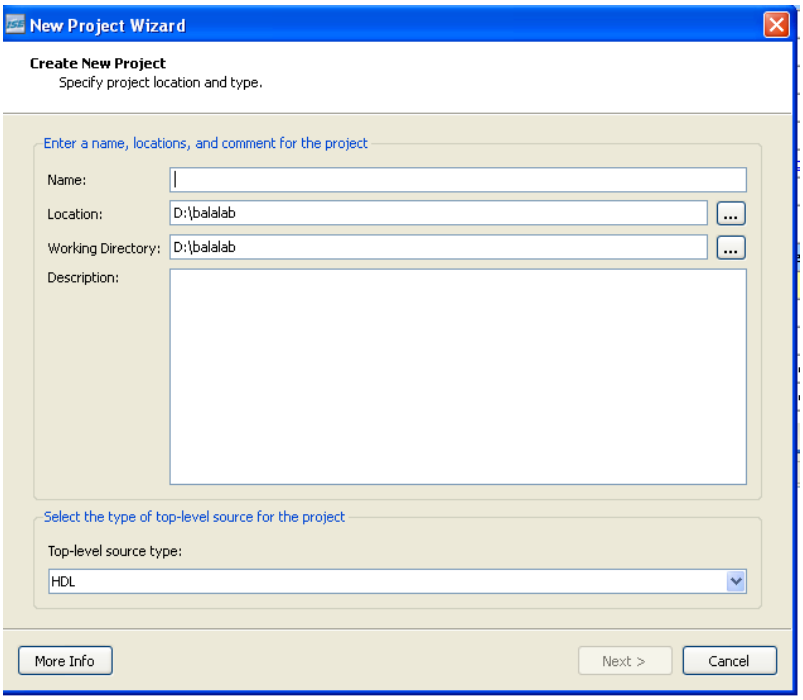
Steps to use Xilinx tool:

Start the Xilinx Project Navigator by using the desktop shortcut or by using the

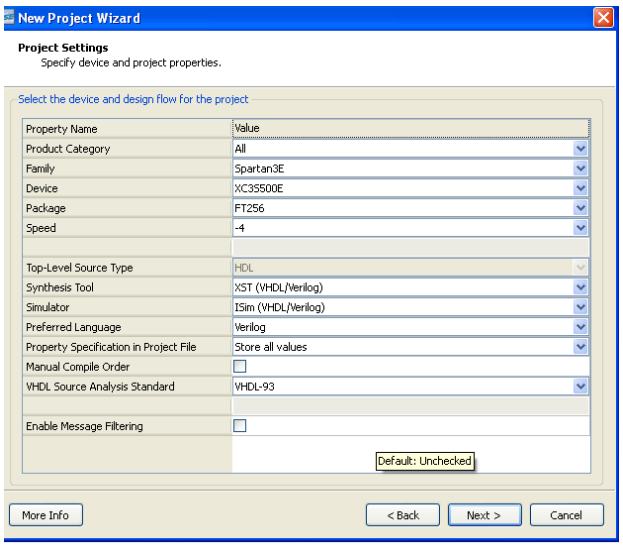
- Start → Programs → Xilinx ISE → Project Navigator.



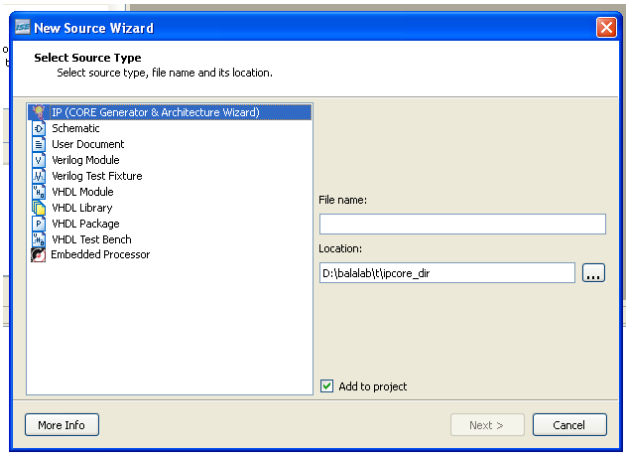
- In the Project Navigator window go to FILE →New project→ Click on new source→ verilog module and give the name inverter.v →Define ports→Finish



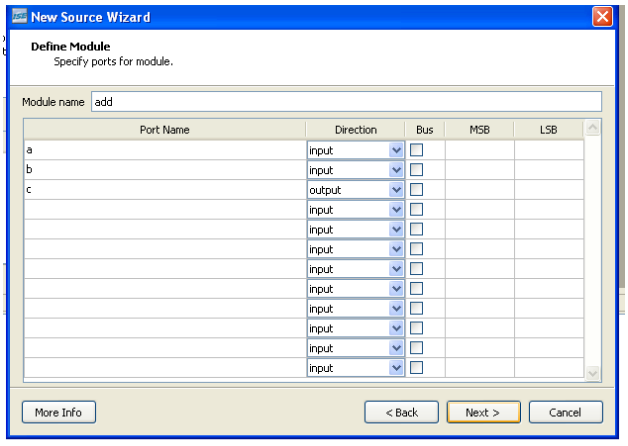
- Select devices→General purpose→ Spartan 3→ ISE simulator→verilog



- In the create new source window select source type as verilog module give file name



- assign inputs and outputs → click next →finish→yes→next →next→finish



- Double click on source file→complete the verilog code for inverter

- Check syntax, and remove errors if present
- Simulate the design using ISE Simulator Highlight **inverter.v** file in the Sources in Project window. To run the Behavioral Simulation, Click on the symbol of FPGA device and then right click→Click on new source→Click on verilog text fixture→Give file name with _tb→ finish
- Generate test bench file after initial begin assign value for inputs→ Click on simulate behavioral model →see the output.

RESULT:

EXP.NO:	STUDY OF SYNTHESIZE TOOLS
DATE:	

AIM:

To study synthesize tools using Xilinx software tool.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

Synthesis is an automatic method of converting a higher level abstraction to a lower level abstraction. The synthesis tool convert **Register Transfer Level** (RTL) description to **gate level netlists**. These gate level netlists consist of interconnected gate level macro cells. These gate level netlists currently can be optimized for **area, speed** etc., The analyzed design is synthesized to a library of components, typically gates, latches, or flipflops. Hierarchical designs are synthesized in bottom up fashion, that is lower level components are synthesized before higher level components. Once the design is synthesized we have a gate level netlist. This gate level netlist can be simulated. Delay for the individual components are available as part of the description of the component libraries. Timing accurate simulation is not possible at this point because the actual timing characteristics is determined by the physical placement of the design within the FPGA chip. However, the functional simulation that is possible at this point is quite a bit more accurate than simulation based on user specified delays. After run the synthesize in process window then full adder model is converted to **netlist file**.

To convert the RTL to gates, three steps typically occur:

* The RTL description is translated to an unoptimized boolean description usually consisting of primitive gates such as AND and OR gates, flip-flop, and latches. This is a functionally correct but completely unoptimized description.

* Boolean optimization algorithms are executed on this boolean equivalent description to produce an optimized boolean equivalent description.

* This optimized boolean equivalent description is mapped to actual logic gate by making use of a technology library of the target process.

PROCEDURE:

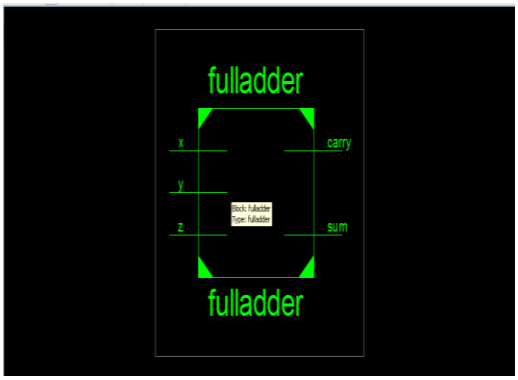
1. Now start the Xilinx ISE Design Suite 12.1
2. Go to file and click new project
3. Enter the project name and click next
4. Select the family name is Spartan 3E, speed is -4 and simulator is verilog click next and click Finish.
5. Click new source.
6. Select verilog module and type file name and click next.

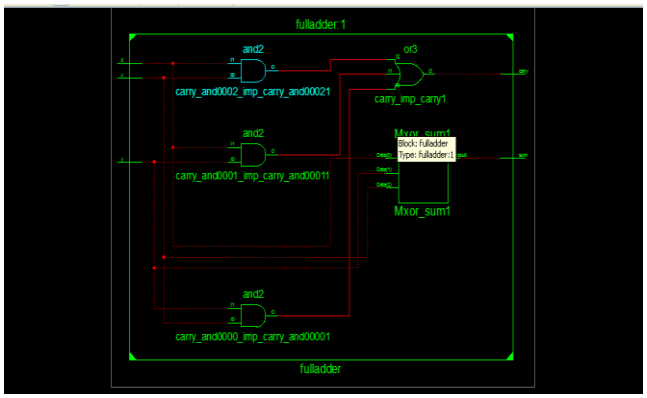
7. Assign input and output port and click next.
8. Finally the report is shown click finish.
9. Type the program save and click synthesis.
10. Go to synthesis→View RTL schematic

PROGRAM:

```
module full_adder (a,b,c,sum,carry);
output sum,carry ;
input a,b,c ;
assign sum = a ^ b ^ c;
assign carry = (a&b) | (b&c) | (c&a);
endmodule
```

RTL SCHEMATIC:





RESULT:

EXP.NO:	PLACE AND ROOT AND BACK ANNOTATION FOR FPGAS
DATE:	

AIM:

To study place and root and back annotation for FPGAs synthesize tools using Xilinx software tool.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

To map this Full adder design onto the FPGA. The primitive hardware elements that are available in Xilinx xc3s500e chip, namely lookup tables and positive-edge-triggered flip-flops are organized as a **two dimensional array of CLBs**. The netlist from synthesize is composed of **gates, latches, and flip-flops**. It is necessary to assign CLB to netlist primitives. This is the process of **mapping** a design. For example gates will be assigned to look-up tables. This process effectively translates the gate level netlist produce by the synthesize compiler into a netlist of FPGA primitive hardware components. Each elements of this new netlist corresponds to a hardware primitive in the FPGA Chip. The mapped design produces identifies the set of FPGA hardware primitives and their interconnection. The next step is to assign each of the components in the netlist to a equivalent physical primitives on the FPGA chip. Once this assignment or placement is made the interconnection between the components in the netlist must be made within the chip. This will require routing signals through the switch matrix and other inter connect resources available on FPGA Chip. This Place and route layout was generated from Xilinx ISE **Floor planner**. After place and route the design can be simulated to validate the design. At this point timing is more accurate because the propagation delays along routed signals and through CLBs can be more accurately estimated. This is particularly important for designs that are operating under tight timing tolerance.

To convert the RTL to gates, three steps typically occur:

* The RTL description is translated to an unoptimized boolean description usually consisting of primitive gates such as AND and OR gates, flip-flop, and latches. This is a functionally correct but completely unoptimized description.

* Boolean optimization algorithms are executed on this boolean equivalent description to produce an optimized boolean equivalent description.

* This optimized boolean equivalent description is mapped to actual logic gate by making use of a technology library of the target process.

PROCEDURE:

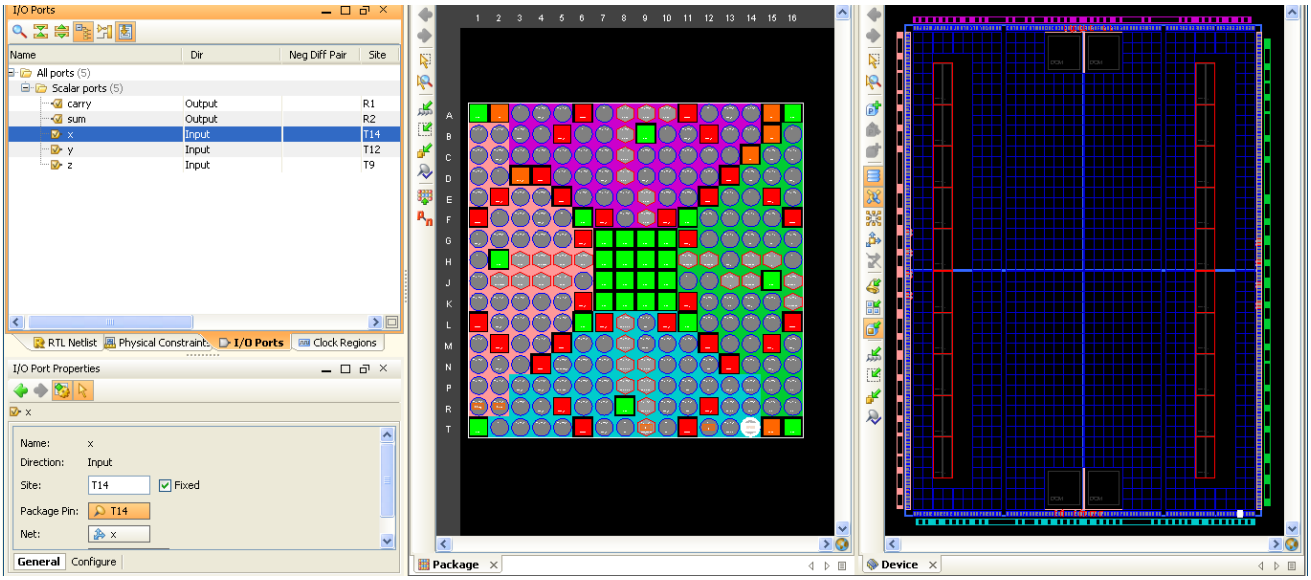
1. Now start the Xilinx ISE Design Suite 12.1
2. Go to file and click new project
3. Enter the project name and click next
4. Select the family name is Spartan 3E, speed is -4 and simulator is verilog click next and

click Finish.

5. Click new source.
6. Select verilog module and type file name and click next.
7. Assign input and output port and click next.
8. Finally the report is shown click finish.
9. Types the program saves and clicks synthesis.
10. Choose Implementation →user constraints →I/O pin planning (plan ahead) pre- synthesis, type the input /output port.

PROGRAM:

```
module full_adder (a,b,c,sum,carry);
output sum,carry ;
input a,b,c ;
assign sum = a ^ b ^ c;
assign carry = (a&b) | (b&c) | (c&a);
endmodule
```



RESULT:

EXP.NO:	STUDY OF FPGA BOARD AND ON-BOARD LED'S AND SWITCHES
DATE:	

AIM:

To Study Field Programmable Gate Array (FPGA) board and to test the on-board LEDs and Switches using Xilinx software tool.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

DIP SWITCHES:

When in the UP or ON position, a switch connects the FPGA pin to V_{cc}, a logic High. When DOWN or in the OFF position, the switch connects the FPGA pin to ground, a logic low. The switches typically exhibit about 2ms of mechanical bounce and there is no active debouncing circuitry, although such circuitry could easily be added to the FPGA design programmed on the board.

KEY SWITCHES:

The key switches can provide pulse input to the FPGA. The switches connect to an associated FPGA pin. Pressing a key generates logic High on the associated FPGA pin. There is no active debouncing circuitry on the key switches.

LEDS:

Test LEDs are provided for mapping output of FPGA or tracking particular stage in the design. A series current limiting resistor of 270 ohm is associated with every LED. To turn on an individual LED, drive the associated FPGA control signal High.

PROCEDURE:

1.Create a new project & create a new Verilog file.

2.Type the program for testing LEDs and Switches and Save it

3.Synthesize the program and view the RTL Model.

4.Create test bench waveform and simulate it.

5.Download the program using the procedure given below into the FPGA.

6.Now test the physical working of the switches and the LED's on – board.

DOWNLOADING PROCEDURE:

1. Select "Synthesis/Implementation" in the source window.
2. Select the created module (*.v file) in the source window.
3. Select "user constraint" in the process window, double click "edit constraint" to create user constraint file (*.UCF).
4. Type the net list to define the I/O pins and save it.
5. Double click "implement design" in the process window.
6. Double click "Generate programming "file and select the respective created bit file (*.bit)
7. Double click " configure device (iMPACT) ". In the impact window that appears, select 'configure device using boundary scan'. Click finish
8. Right click on the created Xilinx model & select 'program', give OK on the displayed window.

PROGRAM:

```
module leds(a, b);
   input [0:15]a;
   output [0:15]b;
   reg [0:15]b;
always@(a)begin
  b=~a;
  end
endmodule
```

RESULT:

EXP.NO:	SIMULATION OF BASIC LOGIC GATES
DATE:	

AIM:

To write a verilog program for basic logic gates to synthesize and simulate using Xilinx software tool.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

AND GATE:

An AND gate is a digital logic gate with two or more inputs and one output that performs logical conjunction. The output of an AND gate is true only when all of the inputs are true. If one or more of an AND gate's inputs are false, then the output of the AND gate is false.

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
Y = a & b		**A**	**B**	

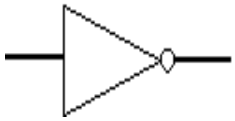
OR GATE:

An OR gate is a digital logic gate with two or more inputs and one output that performs logical disjunction. The output of an OR gate is true when one or more of its inputs are true. If all of an OR gate's inputs are false, then the output of the OR gate is false

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
Y = a | b		**A**	**B**	

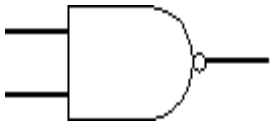
NOT GATE:

A NOT gate, often called an inverter, is a nice digital logic gate to start with because it has only a single input with simple behavior. A NOT gate performs logical negation on its input. In other words, if the input is true, then the output will be false. Similarly, a false input results in a true output.

EQUATION	LOGIC SYMBOL	TRUTH TABLE	
Y = ~a		**A**	

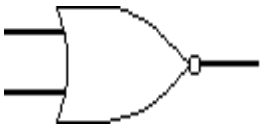
NAND GATE:

A NAND gate (sometimes referred to by its extended name, Negated AND gate) is a digital logic gate with two or more inputs and one output with behavior that is the opposite of an AND gate. The output of a NAND gate is true when one or more, but not all, of its inputs are false. If all of a NAND gate's inputs are true, then the output of the NAND gate is false.

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
Y = ~(a & b)		**A**	**B**	

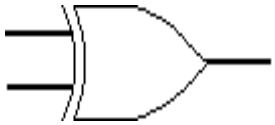
NOR GATE:

A NOR gate (sometimes referred to by its extended name, Negated OR gate) is a digital logic gate with two or more inputs and one output with behavior that is the opposite of an OR gate. The output of a NOR gate is **true** all of its inputs are**false**. If one or more of a NOR gate's inputs are **true**, then the output of the NOR gate is **false**.

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
Y = ~(a | b)		**A**	**B**	

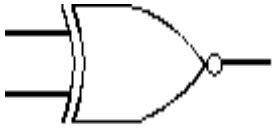
EX-OR GATE (XOR):

An XOR gate (sometimes referred to by its extended name, Exclusive OR gate) is a digital logic gate with two or more inputs and one output that performs exclusive disjunction. The output of an XOR gate is true only when exactly one of its inputs is true. If both of an XOR gate's inputs are false, or if both of its inputs are true, then the output of the XOR gate is false.

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
		A	**B**	
Y = a ^ b				

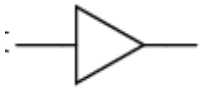
EX-NOR GATE (XNOR):

An XNOR gate (sometimes referred to by its extended name, Exclusive NOR gate) is a digital logic gate with two or more inputs and one output that performs logical equality. The output of an XNOR gate is true when all of its inputs are true or when all of its inputs are false. If some of its inputs are true and others are false, then the output of the XNOR gate is false.

EQUATION	LOGIC SYMBOL	TRUTH TABLE		
		A	**B**	
Y = ~(a ^ b)				

BUFFER GATE:

A buffer has only a single input and a single output with behavior that is the opposite of an NOT gate. It simply passes its input, unchanged, to its output. In a boolean logic simulator, a buffer is mainly used to increase propagation delay. In a real-world circuit, a buffer can be used to amplify a signal if its current is too weak.

EQUATION	LOGIC SYMBOL	TRUTH TABLE	
		A	
Y = A			

PROCEDURE:

Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator.

PROGRAM:

BASIC GATES:

DATAFLOW	BEHAVIORAL	STRUCTURAL
NOT GATE		

module notgate(a,y);
input a;
output y;
assign y= ~a;
endmodule
	module notgate (a,y);
	input a;
	output y;
	reg y;
	always @(a)
	begin
	y=~a;
	end
	endmodule
		module notgate (a,y)
		input a;
		output y;
		inv a1(y,a);
		endmodule

OR GATE		

module orgate(a,b,y);
inpput a,b;
output y;
assign y=(a|b);
endmodule
	module orgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
	begin
	y=a|b;
	end
	endmodule
		module orgate(a,b,y);
		input a,b;
		output y;
		or a1(y,a,b);
		endmodule

AND GATE		

module andgate(a,b,y);
input a,b;
output y;
assign y=(a&b);
endmodule
	module andgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
	begin
	y=a&b;
	end
	endmodule
		module andgate(a,b,y);
		input a,b;
		output y;
		and a1(y,a,b);
		endmodule

NOR GATE		

module norgate(a,b,y);
inpput a,b;
output y;
assign y=~(a|b);
endmodule
	module norgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
	begin
	y=~(a|b);
	end
	endmodule
		module norgate(a,b,y);
		input a,b;
		output y;
		nor a1(y,a,b);
		endmodule

NAND Gate		

module nandgate(a,b,y);
input a,b;
output y;
assign y=~(a&b);
endmodule
	module nandgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
	begin
	y=~(a&b);
	end
	endmodule
		module nandgate(a,b,y);
		input a,b;
		output y;
		nand a1(y,a,b);
		endmodule

XOR Gate		

module xorgate(a,b,y);
input a,b;
output y;
assign y=(a^b);
endmodule
	module xorgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
	begin
	y=a^b;
	end
	endmodule
		module xorgate(a,b,y);
		input a,b;
		output y;
		xor a1(y,a,b);
		endmodule

XNOR GATE		

module xnorgate(a,b,y);
input a,b;
output y;
assign y=(~(a^b));
endmodule
	module xnorgate(a,b,y);
	input a,b;
	output y;
	reg y;
	always @(a,b)
		module xnorgate(a,b,y);
		input a,b;
		output y;
		xnor a1(y,a,b);
		endmodule

	begin
	y=~(a^b);
	end
	endmodule	

BUFFER GATE		

module bufgate (a,y);
input a;
output y;
assign y= a;
endmodule
	module bufgate (a,y);
	input a;
	output y;
	reg y;
	always @(a)
	begin
	y=a;
	end
	endmodule
		module bufgate (a,y);
		input a;
		output y;
		buf(y,a);
		endmodule

BASIC GATES (ALL GATES IN ONE PROGRAM)		

module gates(a,b,c,d,e,f,g,h,i,j);
input a,b;
output c,d,e,f,g,h,i,j;
assign c=a&b;
assign d=a|b;
assign e=~a;
assign f=a;
assign g=~(a|b);
assign h=a^b;
assign i=~(a^b);
assign j=~(a&b);
endmodule
	module gates(a,b,c,d,e,f,g,h,i,j);
	input a,b;
	output c,d,e,f,g,h,i,j;
	reg c,d,e,f,g,h,i,j;
	always @(a,b)
	begin
	c=a&b;
	d=a|b;
	e=~a;
	f=a;
	g=~(a|b);
	h=a^b;
	i=~(a^b);
	j=~(a&b);
	end
	endmodule
		module gates(a,b,c,d,e,f,g,h,i,j);
		input a,b;
		output c,d,e,f,g,h,i,j;
		and (c,a,b);
		or(d,a,b);
		not(e,a);
		buf(f,a);
		nor(g,a,b);
		xor(h,a,b);
		xnor(i,a,b);
		nand(j,a,b);
		endmodule

SIMULATION REPORT

RESULT:

EXP.NO:	SIMULATION OF HALF ADDER AND FULL ADDER
DATE:	

AIM:

To write a verilog program for half adder and full adder to synthesize and simulate using Xilinx software tool.

TOOLS REQUIRED:

SOFTWARE:

1. Xilinx ISE Design Suite 12.1

THEORY:

HALF ADDER:

The half adder consists of two input variables designated as Augends and Addend bits. Output variables produce the Sum and Carry. The 'carry' output is 1 only when both inputs are 1 and ,sum' is 1 if any one input is 1. The Boolean expression is given by,

sum = a^ b

carry = a & b

FULL ADDER:

A Full adder is a combinational circuit that focuses the arithmetic sum of three bits. It consists of 3 inputs and 2 outputs. The third input is the carry from the previous Lower Significant Position. The two outputs are designated as Sum (S) and Carry (C). The binary variable S gives the value of the LSB of the Sum. The output S=1 only if odd number of 1's are present in the input and the output C=1 if two or three inputs are 1.

sum = a ^ b ^ c

carry= (a & b) | (b & c) | (a & c)

PROCEDURE:

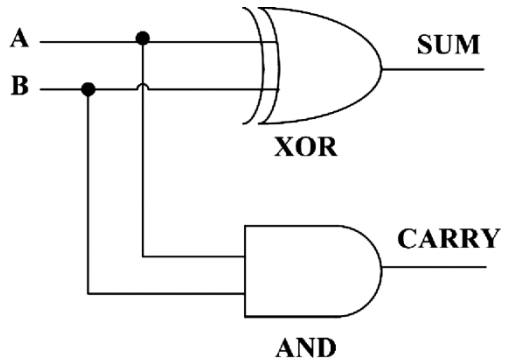
Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator.

PROGRAM:

HALF ADDER

LOGIC DIAGRAM:



TRUTH TABLE:

INPUTS		OUTPUTS	
A	**B**	**SUM**	**CARRY**
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

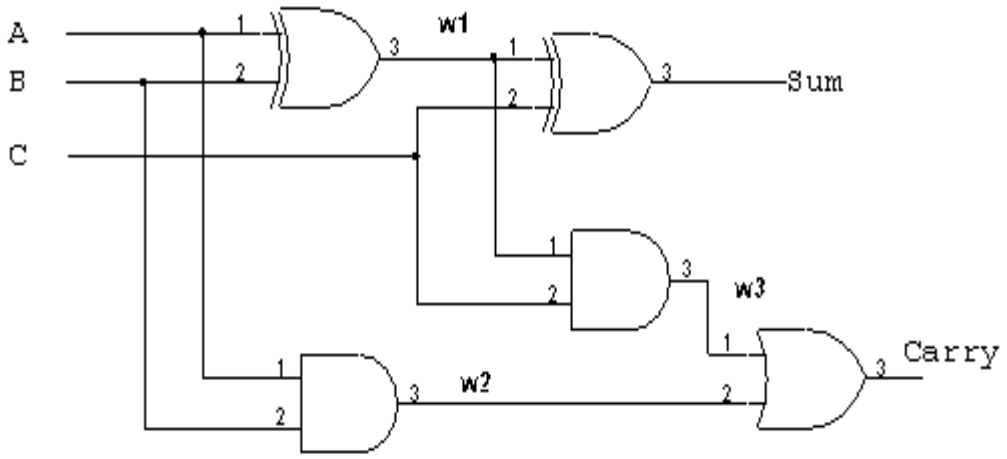
PROGRAM:

DATA FLOW	STRUCTURAL	BEHAVIOURAL

module halfadder(a,b,sum,carry);
input a,b;
output sum,carry;
assign sum = a ^ b ;
assign carry = (a&b);
endmodule
	module halfadder(a,b,sum,carry);
	input a,b;
	output sum,carry;
	reg sum,carry;
	always @(a,b)
	begin
	sum=a^b;
	carry=(a&b);
	end
	endmodule
		module halfadder(a,b,sum,carry);
		input a,b;
		output sum,carry;
		xor(sum,a,b);
		and(carry,a,b);
		endmodule

FULL ADDER

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

PROGRAM:

DATAFLOW MODELLING:

```
module full_adder (a,b,c,sum,carry);
output sum,carry ;
input a,b,c ;
assign sum = a ^ b ^ c;
assign carry = (a&b) | (b&c) | (c&a);
endmodule
```

BEHAVIORAL MODELLING:

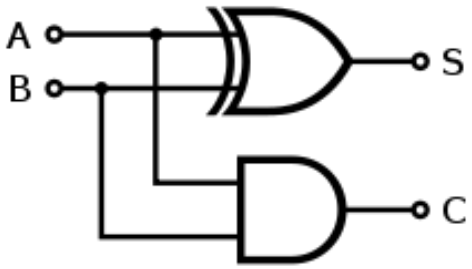
```
module full_adder (a,b,c,sum,carry);
input a,b,c;
output sum,carry;
reg sum,carry;
always @(a,b,c)
begin
sum=a^b^c;
carry=(a&b)|(b&c)|(c&a);
end
endmodule
```

STRUCURAL MODELLING:

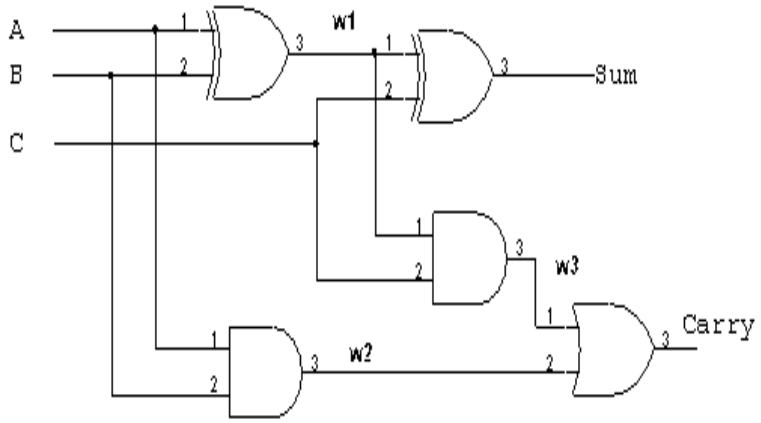
```
module full_adder(sum,carry,a,b,c);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
xor x1(w1,a,b);
xor x2(sum,w1,c);
and a1(w2,a,b);
and a2(w3,w1,c);
or(carry,w2,w3);
endmodule
```

FULL ADDER USING HALF ADDER

```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule
```




```
module full_adder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
half_adder ha1(a,b,w1,w2);
half_adder ha2(c,w1,sum,w3);
or(carry,w2,w3);
endmodule
```



SIMULATION REPORT

RESULT:

EXP.NO:	SIMULATION OF ADDER
DATE:	

AIM:

To write a Verilog program for Adder and to synthesize, simulate it using Xilinx software tool.

.

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY :

When you add large numbers carefully together the addition is done digit by digit. In the illustration, two 8 –digit binary numbers are being added. The top row contains the first number and the second row the other. Working from the right-hand side, there can be no 'carry' to add to the sum of the first two digits, so a half adder is sufficient. But for the second and subsequent pairs of digits, full adders must be use (any carry' is indicated by a f below the adder). The output will be an 8-bit and if the carry is formed that will be shown in cout output value.

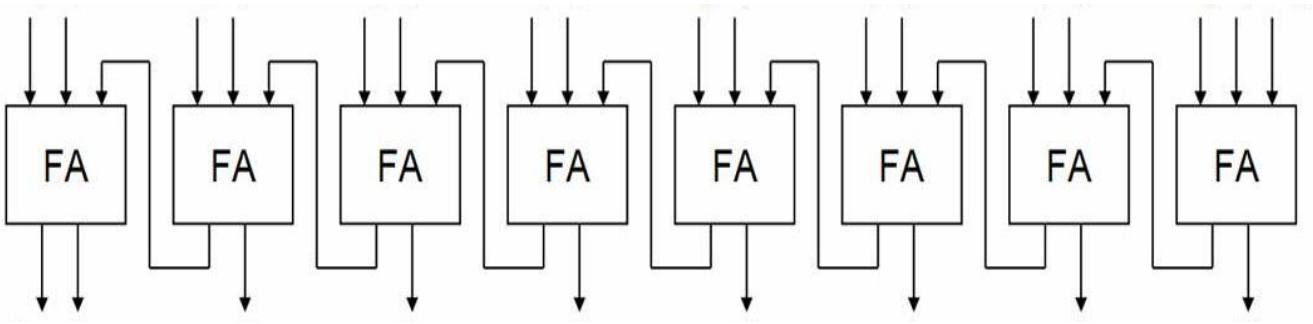
PROCEDURE:

Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. Clicking on the synthesis in the process window.
5. Perform the functional simulation using Xilinx ISE simulator.
6. The output can be observed by using ISIM Simulator.

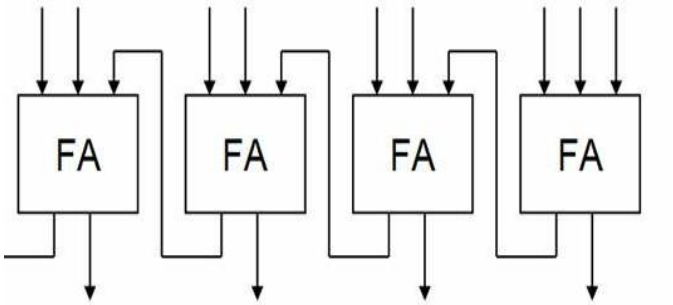
PROGRAM:

8-BIT RIPPLE CARRY ADDER:




```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule

module full_adder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
half_adder ha1(a,b,w1,w2);
half_adder ha2(c,w1,sum,w3);
or(carry,w2,w3);
endmodule

module rippleadder(a, b, cin, sum, cout);
input [7:0] a,b;
input cin;
output [7:0]sum;
output cout;
wire[6:0] c;
full_adder a1(a[0],b[0],cin,sum[0],c[0]);
full_adder a2(a[1],b[1],c[0],sum[1],c[1]);
full_adder a3(a[2],b[2],c[1],sum[2],c[2]);
full_adder a4(a[3],b[3],c[2],sum[3],c[3]);
full_adder a5(a[4],b[4],c[3],sum[4],c[4]);
full_adder a6(a[5],b[5],c[4],sum[5],c[5]);
full_adder a7(a[6],b[6],c[5],sum[6],c[6]);
full_adder a8(a[7],b[7],c[6],sum[7],cout);
endmodule
```

4-BIT RIPPLE CARRY ADDER




```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule

module full_adder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
half_adder ha1(a,b,w1,w2);
half_adder ha2(c,w1,sum,w3);
or(carry,w2,w3);
endmodule

module rippleadder (a, b, cin, sum, cout);
input [3:0]a,b;
input cin;
output [3:0]sum;
output cout;
wire[2:0] c;
full_adder a1(a[0],b[0],cin,sum[0],c[0]);
full_adder a2(a[1],b[1],c[0],sum[1],c[1]);
full_adder a3(a[2],b[2],c[1],sum[2],c[2]);
full_adder a4(a[3],b[3],c[2],sum[3],cout);
endmodule
```

SIMULATION REPORT:

RESULT:

EXP.NO:	SIMULATION OF MULTIPLIER
DATE:	

AIM:

To write a Verilog program for 4-bit multiplier and to synthesize, simulate it using Xilinx software tool.

TOOLS REQUIRED:

SOFTWARE:

1. Xilinx ISE Design Suite 12.1

THEORY:

Multiplication of two elements in the polynomial basis can be accomplished in the normal way of multiplication, but there are a number of ways to speed up multiplication, especially in hardware. In this type the multiplication can done parallel counter and it is generate carry. The multiplication is independent of the carry so we can perform N number of multiplication independent of carry.

PROCEDURE:

Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double
4. Clicking on the synthesis in the process window.
5. Perform the functional simulation using Xilinx ISE simulator.
6. The output can be observed by using ISIM Simulator.

PROGRAM:

4-BIT ARRAY MULTIPLIER:

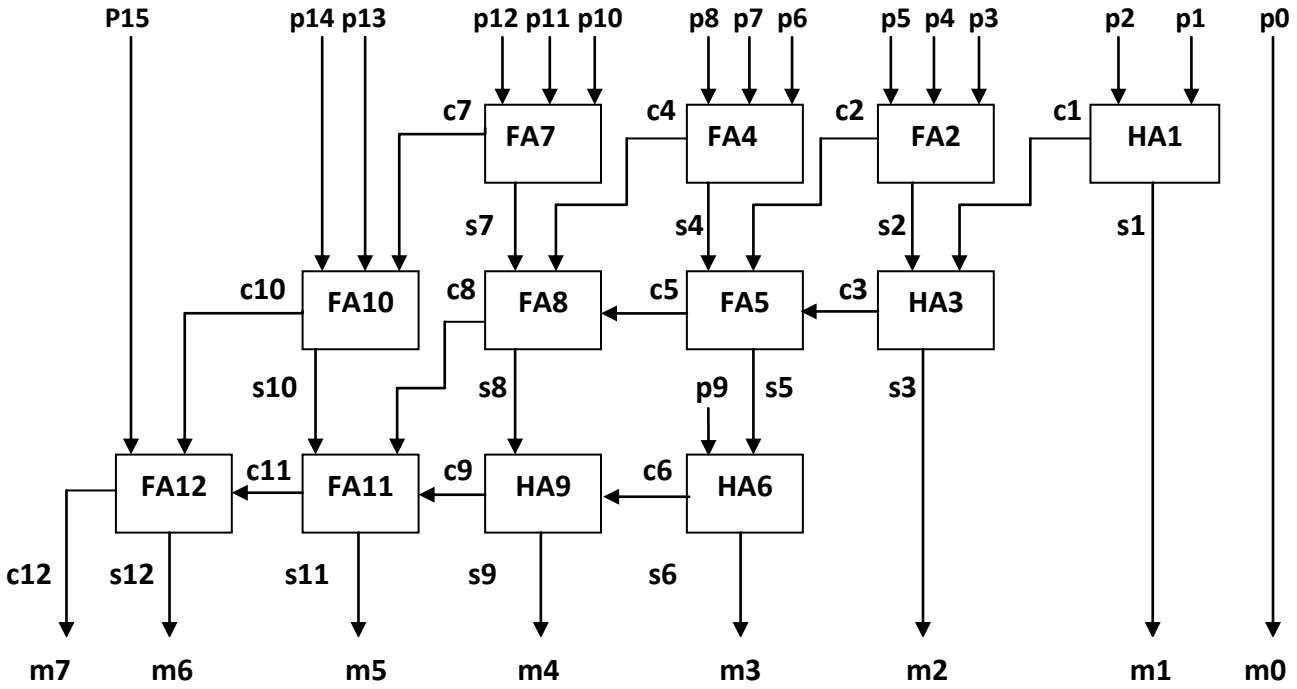
			a3	a2	a1	a0	
		X	b3	b2	b1	b0	
			a3b0	a2b0	a1b0	a0b0	
		a3b1	a2b1	a1b1	a0b1		
	a3b2	a2b2	a1b2	a0b2			
a3b3	a2b3	a1b3	a0b3				
m7	m6	m5	m4	m3	m2	m1	m0

a0b0=p0 a0b1=p2 a0b2=p5 a0b3=p9

a1b0=p1 a1b1=p4 a1b2=p8 a1b3=p12

a2b0=p3 a2b1=p7 a2b2=p11 a2b3=p14

a3b0=p6 a3b1=p10 a3b2=p13 a3b3=p15



PROGRAM:

```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule

module full_adder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire w1,w2,w3;
half_adder ha1(a,b,w1,w2);
half_adder ha2(c,w1,sum,w3);
or(carry,w2,w3);
endmodule

module arraymultiplier(m,a,b);
input [3:0]a,b;
output [7:0]m;
wire [15:0]p;
wire [12:1]s,c;
and(p[0],a[0],b[0]);
and(p[1],a[1],b[0]);
and(p[2],a[0],b[1]);
and(p[3],a[2],b[0]);
and(p[4],a[1],b[1]);
and(p[5],a[0],b[2]);
and(p[6],a[3],b[0]);
and(p[7],a[2],b[1]);
and(p[8],a[1],b[2]);
and(p[9],a[0],b[3]);
and(p[10],a[3],b[1]);
and(p[11],a[2],b[2]);
```

```
and(p[12],a[1],b[3]);
and(p[13],a[3],b[2]);
and(p[14],a[2],b[3]);
and(p[15],a[3],b[3]);
half_adder ha1(s[1],c[1],p[1],p[2]);
full_adder fa2(s[2],c[2],p[4],p[3],p[5]);
half_adder ha3(s[3],c[3],s[2],c[1]);
full_adder fa4(s[4],c[4],p[6],p[7],p[8]);
full_adder fa5(s[5],c[5],s[4],c[2],c[3]);
half_adder ha6(s[6],c[6],s[5],p[9]);
full_adder fa7(s[7],c[7],p[10],p[11],p[12]);
full_adder fa8(s[8],c[8],c[5],c[4],s[7]);
half_adder ha9(s[9],c[9],s[8],c[6]);
full_adder fa10(s[10],c[10],p[14],p[13],c[7]);
full_adder fa11(s[11],c[11],c[9],c[8],s[10]);
full_adder fa12(s[12],c[12],p[15],c[10],c[11]);
buf(m[0],p[0]);
buf(m[1],s[1]);
buf(m[2],s[3]);
buf(m[3],s[6]);
buf(m[4],s[9]);
buf(m[5],s[11]);
buf(m[6],s[12]);
buf(m[7],c[12]);
endmodule
```

4-BIT MULTIPLIER

```
module adder(a,b, out);
input [3:0]a,b;
output [7:0]out;
assign out= a*b;
endmodule
```

SIMULATION REPORT:

RESULT:

EXP.NO:	SIMULATION OF FLIP FLOPS
DATE:	

AIM:

To write a Verilog program for various flip flops and to synthesize, simulate it using Xilinx software tool.

TOOLS REQUIRED:

SOFTWARE:

1. Xilinx ISE Design Suite 12.1

THEORY:

D-FLIP FLOP:

It has only a single data input. That data input is connected to the S input of RS-flip flop, while the inverse of D is connected to the R input. This prevents that the input combination ever occurs. To allow the flip flop to be in holding state, a D-flip flop has a second input called "clock". The clock input is AND-ed with the D input, such that when clock=0, the R and S inputs of the RS-flip flop are 0 and the state is held.

D-LATCH:

It has only a single data input. That data input is connected to the S input of RS-flip flop, while the inverse of D is connected to the R input. This prevents that the input combination ever occurs. To allow the flip flop to be in holding state, a D-flip flop has a second input called "enable". The enable input is AND-ed with the D input, such that when enable=0, the R and S inputs of the RS-flip flop are 0 and the state is held.

PROCEDURE:

Software part:

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.

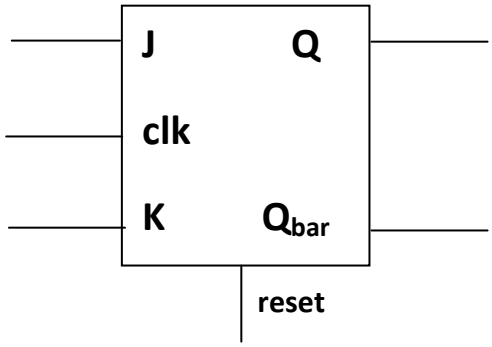
2. Write the Verilog code by choosing HDL as top level source module.

3. Check syntax, view RTL schematic and note the device utilization summary by double clicking on the synthesis in the process window.

4. Perform the functional simulation using Xilinx ISE simulator.

5. The output can be observed by using ISIM Simulator.

JK FLIP FLOP:

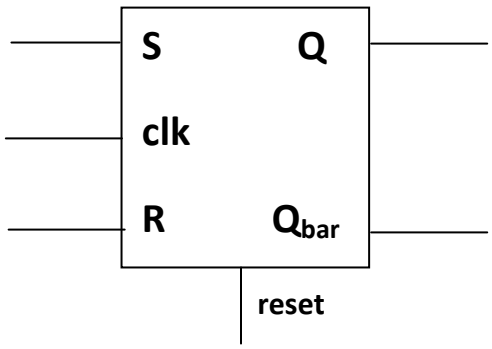


TRUTH TABLE

Q	J	K	Qbar
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

```
module JK_Flip_Flop (Q,Qbar,clk,reset,J,K);
output Q,Qbar;
input clk, reset,J,K;
reg Q, Qbar;
always @(posedge clk)
if (reset)
        begin
        Q <= 0;
        end
else if (J == 0 && K == 0)
        begin
        Qbar <= Q;
        end
else if (J == 0 && K == 1)
        begin
        Qbar <= 0;
        end
else if (J == 1 && K == 0)
        begin
        Qbar <= 1;
        end
else if (J == 1 && K == 1)
        begin
        Qbar <= Q;
        end
endmodule
```

SR FLIP FLOP:

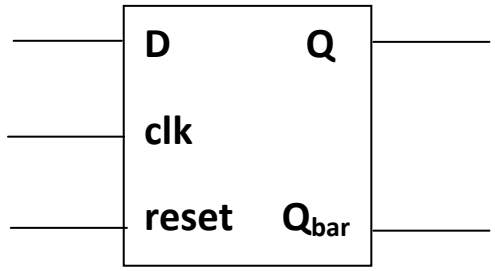


TRUTH TABLE

Q	S	R	Qbar
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

```
module SR_Flip_Flop (Q,Qbar,clk,reset,S,R);
output Q,Qbar;
input clk, reset,S,R;
reg Q,Qbar;
always @(posedge clk)
if (reset)
        begin
        Q <= 0;
        end
else if (S == 0 && R == 0)
        begin
        Qbar <= Q;
        end
else if (S == 0 && R == 1)
        begin
        Qbar <= 0;
        end
else if (S == 1 && R == 0)
        begin
        Qbar <= 1;
        end
else if (S == 1 && R == 1)
        begin
        Qbar <= 1'bx;
        end
endmodule
```

D FLIP FLOP:

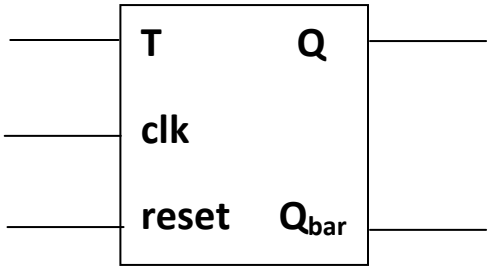


TRUTH TABLE

Q	D	Q_{bar}
0	0	0
0	1	1
1	0	0
1	1	1

```
module D_Flip_Flop (Q,Qbar,clk,reset,D);
input clk,reset,D;
output Q,Qbar;
reg Q,Qbar;
always @(posedge clk)
if (reset)
        begin
        Q <= 0;
        end
else
        begin
        Qbar <= D;
        end
endmodule
```

T FLIP FLOP:



TRUTH TABLE

Q	T	Q_{bar}
0	0	0
0	1	1
1	0	1
1	1	0

```
module T_Flip_Flop (Q,Qbar,clk,reset,t);
input clk,reset,t;
output Q, Qbar;
reg Q, Qbar;
always @(posedge clk)
if (reset)
        begin
        Q <= 0;
        end
else
 if (t == 0)
        begin
        Qbar <= Q;
        end
else
 if (t == 1)
        begin
        Qbar <= ~Q;
        end
endmodule
```

SIMULATION REPORT:

RESULT:

EXP.NO:	SIMULATION OF UP-DOWN COUNTER
DATE:	

AIM:

To write a verilog program for Up-Down Counter and to synthesize, simulate it using Xilinx software tool.

TOOLS REQUIRED:

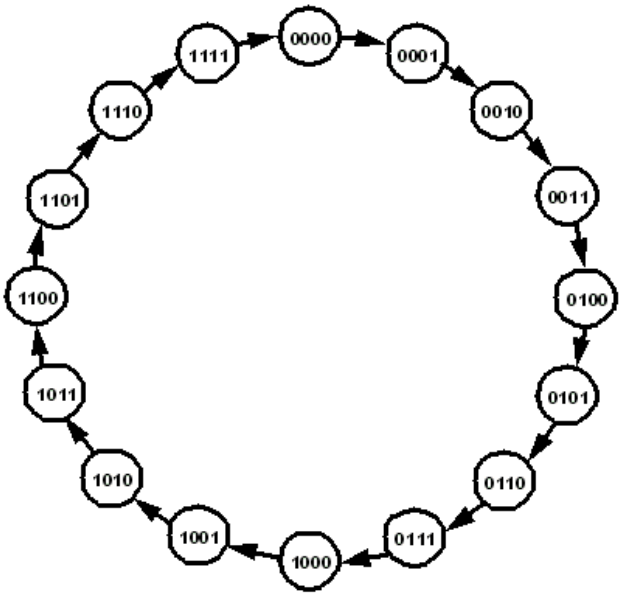
Software:

1. Xilinx ISE Design Suite 12.1

THEORY :

UP-DOWN COUNTER:

A counter that can change state in either direction, under the control of an up or down selector input, is known as an up/down counter. When the selector is in the up state, the counter increments its value. When the selector is in the down state, the counter decrements the count. Likewise the counter counts in both the directions continuously until attaining the end of the count. The count is initiated by the positive clock pulse. The counter counts from 0000 to 1111 for up count and 1111 to 0000 for down count.



PROCEDURE:

Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double Clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator.

PROGRAM:

UP COUNTER	DECIMAL VALUE	PRESENT STATE				DOWN COUNTER
		8	4	2	1	
		q[3]	q[2]	q[1]	q[0]	
↓	0	0	0	0	0	↑
	1	0	0	0	1	
	2	0	0	1	0	
	3	0	0	1	1	
	4	0	1	0	0	
	5	0	1	0	1	
	6	0	1	1	0	
	7	0	1	1	1	
	8	1	0	0	0	
	9	1	0	0	1	
	10	1	0	1	0	
	11	1	0	1	1	
	12	1	1	0	0	
	13	1	1	0	1	
	14	1	1	1	0	
	15	1	1	1	1	
	0	0	0	0	0	

4-BIT UP COUNTER

```
module counter (clk,clr,q);
      input  clk,clr;
      output [3:0]q;
      reg    [3:0]q;
      always @(posedge clk)
      begin
        if (clr)
          q<=4'b0000;
        else
          q<=q + 1'b1;
      end
 endmodule
```

4-BIT DOWN COUNTER

```
module counter (clk,clr,q);
      input  clk,clr;
      output [3:0]q;
      reg    [3:0]q;
      always @(posedge clk)
      begin
        if (clr)
          q<=4'b0000;
        else
          q<=q - 1'b1;
      end
 endmodule
```

SIMULATION REPORT:

4-BIT UP-DOWN COUNTER:

```
module updowncounterm(clk,clr,updown,q);
input clk,clr;
input updown;
output [3:0]q;
reg [3:0]q;
always@( posedge clk)
begin
if(clr)
        q <=4'b0000;
else if(updown)
        q <= q+1'b1;
else
        q <= q-1'b1;
end
endmodule
```

SIMULATION REPORT:

RESULT:

EXP.NO:	SIMULATION OF COUNTER
DATE:	

AIM:

To write a Verilog program for different Counters and to synthesize, simulate it using Xilinx software tool.

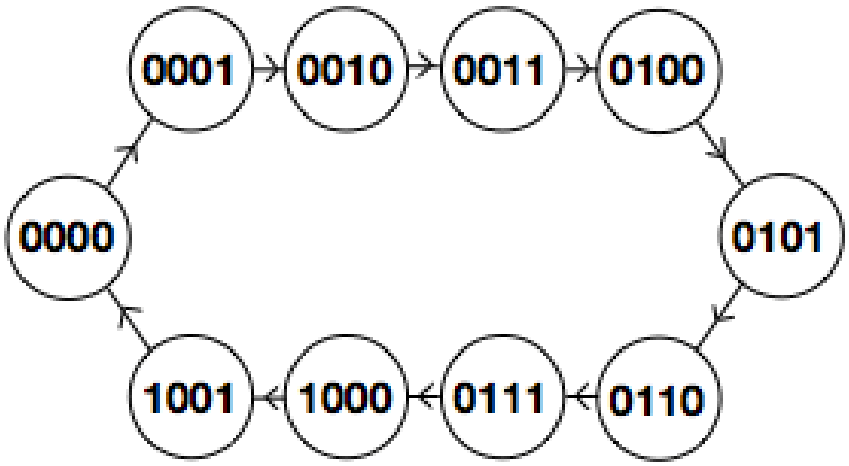
TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

MOD 10 COUNTERS:



JOHNSON COUNTER:

Johnson Counter is one kind of Ring Counter. It is also known as Twisted Ring Counter. A 4-bit Johnson Counter passes blocks of four logic "0" and then passes four logic "1". So it will produce 8-bit pattern. For example, "1000" is initial output then it will generate 1100, 1110, 1111, 0111, 0011, 0001, 0000 and this patterns will repeat so on.

RING COUNTER:

Ring Counter is composed of Shift Registers. The data pattern will recirculate as long as clock pulses are applied. For example, if we talk about 4-bit Ring Counter, then the data pattern will repeat every four clock pulses. If pattern is 1000, then it will generate 0100, 0010, 0001, 1000 and so on.

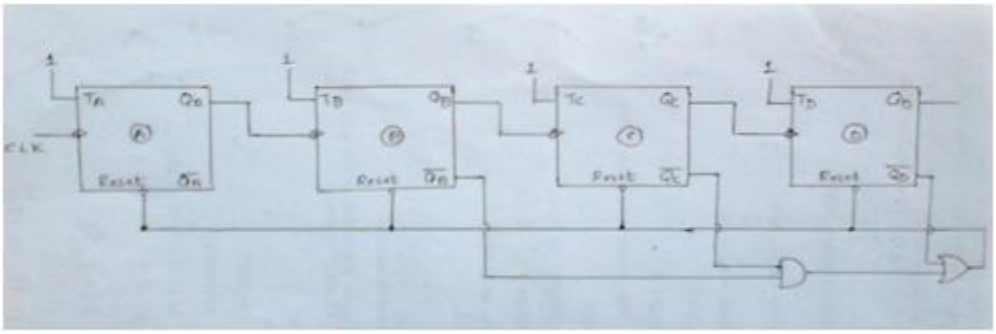
PROCEDURE:

Software part

1. Click on the Xilinx ISE Design Suite 12.1or Xilinx Project navigator icon on the desktop of PC.
2. Write the Verilog code by choosing HDL as top level source module.
3. Check syntax, view RTL schematic and note the device utilization summary by double clicking on the synthesis in the process window.
4. Perform the functional simulation using Xilinx ISE simulator.
5. The output can be observed by using ISIM Simulator.

PROGRAM:

MOD 10 COUNTERS / DECADE COUNTER:

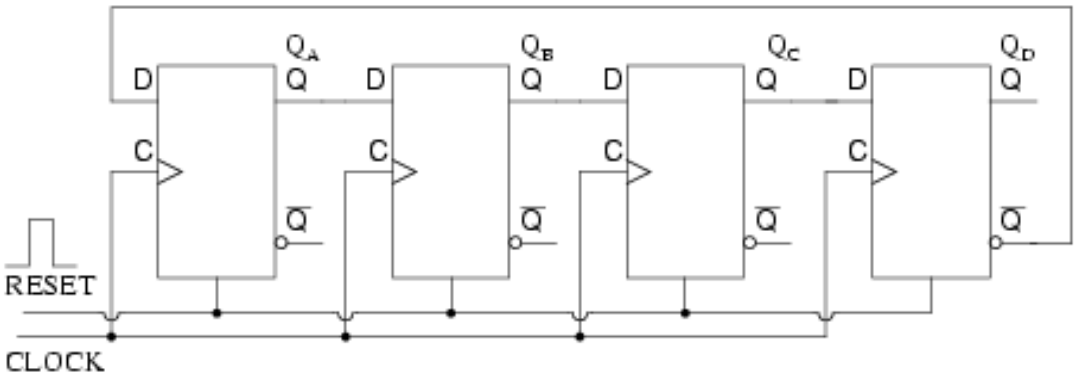



```
module mod10 (clk,clr,Q);
      input  clk,clr;
      output [3:0]Q;
      reg    [3:0]Q;
      always @(posedge clk)
      begin
        if (clr | Q==4'b1001)
          Q<=4'b0000;
        else
          Q<=Q + 1'b1;
      end
endmodule
```

TRUTH TABLE

DECIMAL VALUE	8	4	2	1
	Q[3]	Q[2]	Q[1]	Q[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
0	0	0	0	0

JOHNSON COUNTER

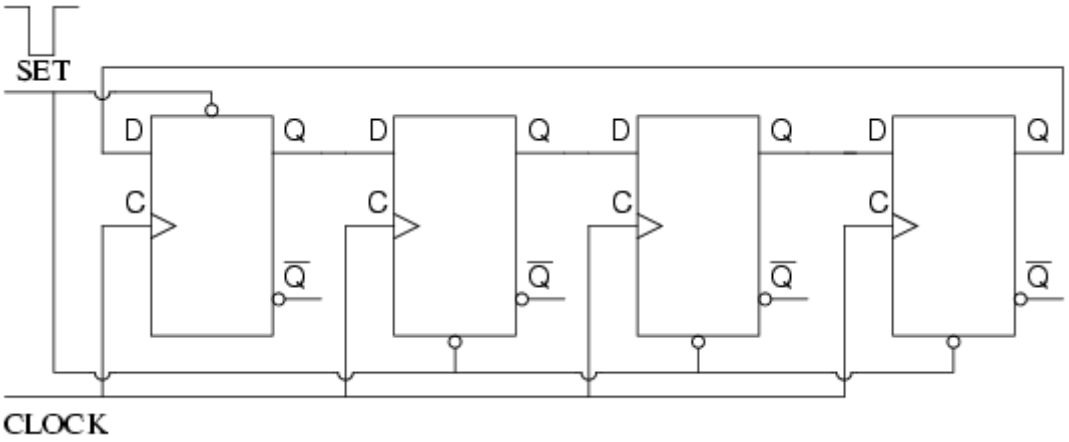



```
module johnson_count (rst, clk, Q);
input rst, clk;
output [3:0] Q;
reg [3:0] Q;
always @(posedge clk)
if (rst)
        begin
        Q[3:0] <= 0;
        end
else
        Q <= {{Q[2:0]}, {~Q[3]}};
endmodule
```

TRUTH TABLE

Q[3]	Q[2]	Q[1]	Q[0]
0	0	0	0
0	0	0	1
0	0	1	1
0	1	1	1
1	1	1	1
1	1	1	0
1	1	0	0
1	0	0	0
0	0	0	0

RING COUNTER




```
module ring_count (rst, clk, Q);
input rst, clk;
output [3:0] Q;
reg [3:0] Q;
always @(posedge clk)
if (rst)
        begin
        Q[3:1] <= 0;
        Q[0] <= 1;
        end
else
        Q <= {{Q[2:0]}, {Q[3]}};
endmodule
```

TRUTH TABLE

Q[3]	Q[2]	Q[1]	Q[0]
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

SIMULATION REPORT:

RESULT:

EXP.NO:	ANALYSIS AND SIMULATION OF STATE MACHINES
DATE:	

AIM:

To analyze, synthesize and simulate state machines using Xilinx simulation tool

TOOLS REQUIRED:

Software:

1. Xilinx ISE Design Suite 12.1

THEORY:

MEALY MODEL:

Analysis describes what a given circuit will do under certain operating conditions. The behavior of a clocked sequential circuit is determined from the inputs, the outputs, and the state of its flip-flops. The outputs and the next state are both a function of the inputs and the present state. The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs, and internal states. It is also possible to write Boolean expressions that describe the behavior of the sequential circuit. These expressions must include the necessary time sequence, either directly or indirectly. A logic diagram is recognized as a clocked sequential circuit if it includes flip-flops with clock inputs. The flip-flops may be of any type, and the logic diagram may or may not include combinational logic gates.

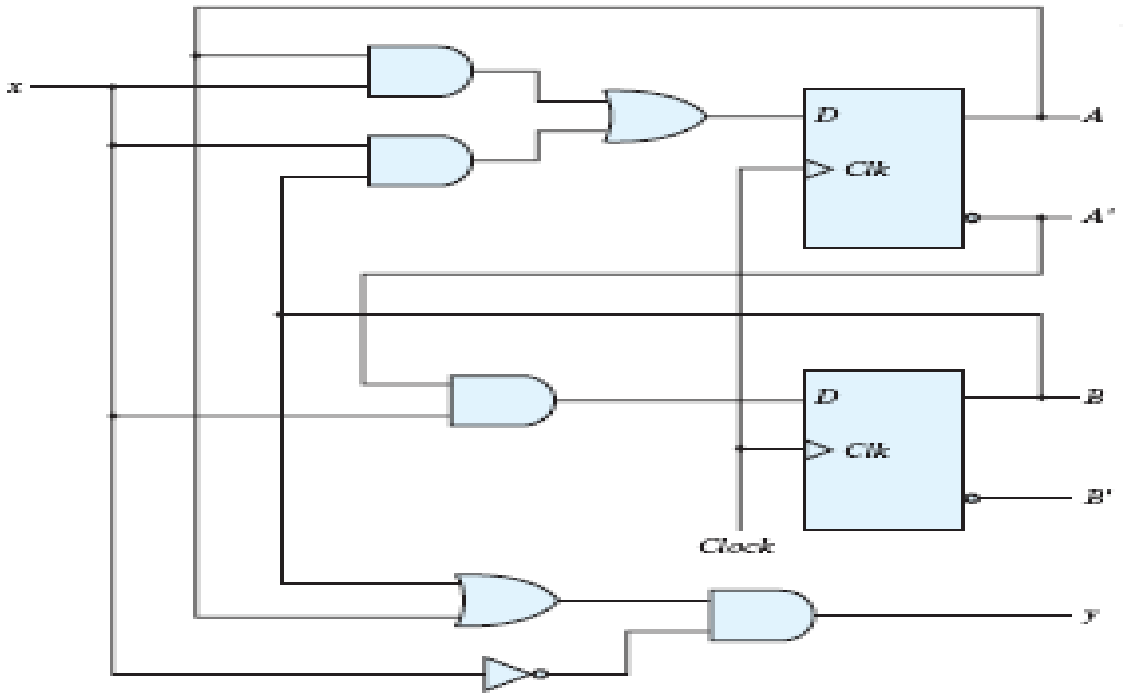
State Equations: The behavior of a clocked sequential circuit can be described algebraically by means of state equations. A *state equation* (also called a *transition equation)* specifies the next state as a function of the present state and inputs. Consider the sequential circuit shown in Fig. 1. It acts as a 0-detector by asserting its output when a 0 is detected in a stream of 1s.

$$A(t+1) = A(t)x(t) + B(t)x(t)\ ;\ B(t+1) = A'(t)x(t);\ \ y(t) = [A(t) + B(t)]x'(t)$$

A state equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation, with $(t + 1)$, denotes the next state of the flip-flop one clock edge later. The right side of the equation is a Boolean expression that specifies the present state and input conditions that make the next state equal to 1. Since all the variables in the Boolean expressions are a function of the present state, we can omit the designation (t) after each variable for convenience and can express the state equations in the more compact form

$$A(t+1) = Ax + Bx\ \ ;\ B(t+1) = A'x\ \ ;\ \ y = Ax' + Bx'$$

LOGIC DIAGRAM:



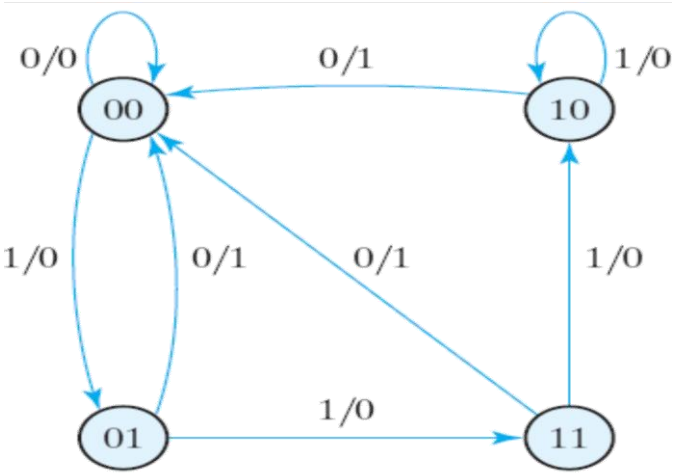
STATE TABLE:

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a *state table* (sometimes called a *transition table*). The table consists of four sections labeled *present state*, *input*, *next state*, and *output* . The present-state section shows the states of flip-flops *A* and *B* at any given time *t*. The input section gives a value of *x* for each possible present state. The next-state section shows the states of the flip-flops one clock cycle later, at time *t* + 1. The output section gives the value of *y* at time *t* for each present state and input condition. State table for the above Circuit diagram

Present State		**Input**	**Next State**		**Output**
A	**B**	**x**	**A**	**B**	**y**
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

STATE DIAGRAM:

The information available in a state table can be represented graphically in the form of a state diagram. In this type of diagram, a state is represented by a circle, and the (clock-triggered) transitions between states are indicated by directed lines connecting the circles. The state diagram provides the same information as the state table and is obtained directly from Table. The binary number inside each circle identifies the state of the flip-flops. The directed lines are labeled with two binary numbers separated by a slash. The input value during the present state is labeled first, and the number after the slash gives the output during the present state with the given input. A directed line connecting a circle with itself indicates that no change of state occurs.



PROGRAM:

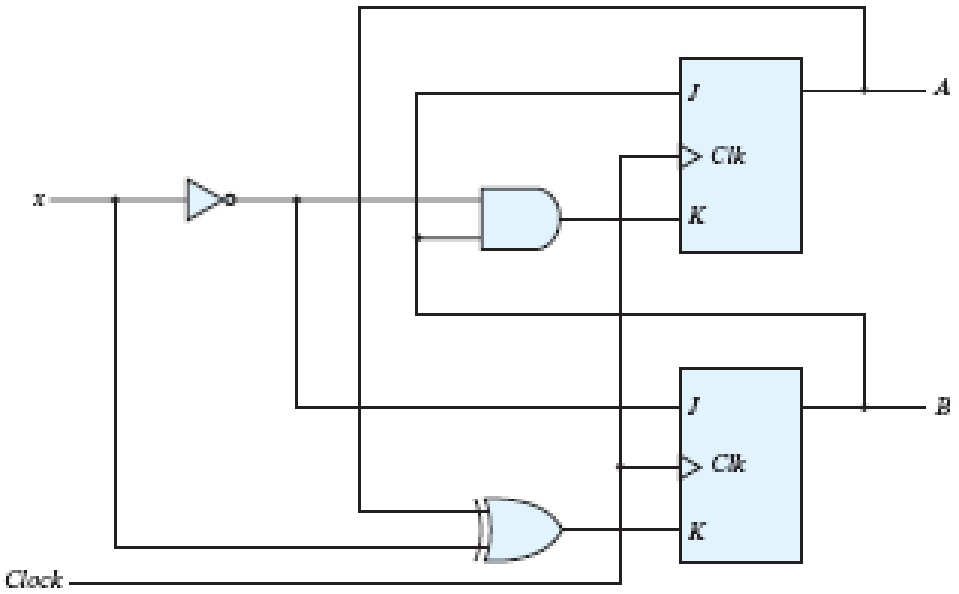
MEALY MODEL:

```
module Mealy_model(y, x, clk, reset);
input x,clk,reset;
output reg y;
reg [1:0] state, next_state;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
always @ ( posedge clk)
if (reset == 0) state <= S0;
else state <= next_state;
always @ (state, x)
case (state)
S0: if (x) next_state = S1; else next_state = S0;
S1: if (x) next_state = S3; else next_state = S0;
```

```
S2: if (x) next_state = S2; else next_state = S0;
S3: if (x) next_state = S2; else next_state = S0;
endcase
always @ (state, x)
case (state)
S0: y = 0;
S1, S2, S3: y = ~x;
endcase
endmodule
```

MOORE MODEL:

LOGIC DIAGRAM:



STATE TABLE:

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a *state table* (sometimes called a *transition table*). The table consists of four sections labeled *present state*, *input*, *next state*, and *output* . The present-state section shows the states of flip-flops *A* and *B* at any given time *t*. The input section gives a value of *x* for each possible present state. The next-state section shows the states of the flip-flops one clock cycle later, at time *t* + 1. The output section gives the value of *y* at time *t* for each present state and input condition.

Present State		**Input**	**Next State**		**Flip- Flop Inputs**			
A	**B**	**x**	**A**	**B**	$\mathbf{J_A}$	$\mathbf{K_A}$	$\mathbf{J_B}$	$\mathbf{K_B}$
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

STATE EQUATIONS:

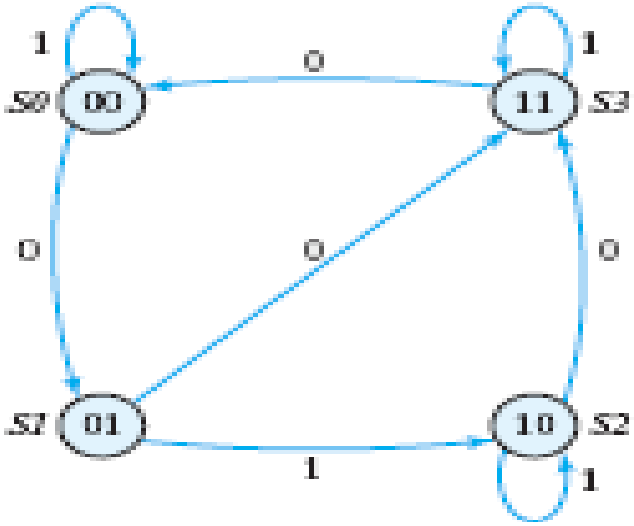
A(t+1) = JA’ + K’A

B(t+1) = JB’ + K’B

A(t+1) = BA’ + (Bx’)’A = A’B+AB’+Ax

B(t+1) = x’B’ + (A ⊕ x)’B = B’x’ + ABx + A’Bx’

STATE DIAGRAM:



MOORE MODEL:

```
module Moore_Model(y, x, clk, reset);
input x,clk,reset;
output [1:0]y;
reg [1:0] state;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
always @ (posedge clk)
if (reset == 0) state <= S0;
else case (state)
S0: if (x) state <= S0; else state <= S1;
S1: if (x) state <= S2; else state <= S3;
S2: if (x) state <= S2; else state <= S3;
S3: if (x) state <= S3; else state <= S0;
endcase
assign y=state;
endmodule
```

RESULT:

EXP.NO:	FPGA IMPLEMENTATION OF COUNTER AND TESTING USING CHIP SCOPE PRO
DATE:	

AIM:

To implement a 4 bit ripple counter using FPGA and test it using Chip Scope Pro feature tool.

TOOLS REQUIRED:

Software:

2. Xilinx ISE Design Suite 12.1

THEORY:

Follow the down loading procedure given in Expt. 3(a) to configure the FPGA device for the 4 bit ripple counter given in Expt. 1(c) and verify its working using Chip Scope Pro feature. The procedure for verification using Chip Scope Pro is outlined below:

CHIP SCOPE PRO:

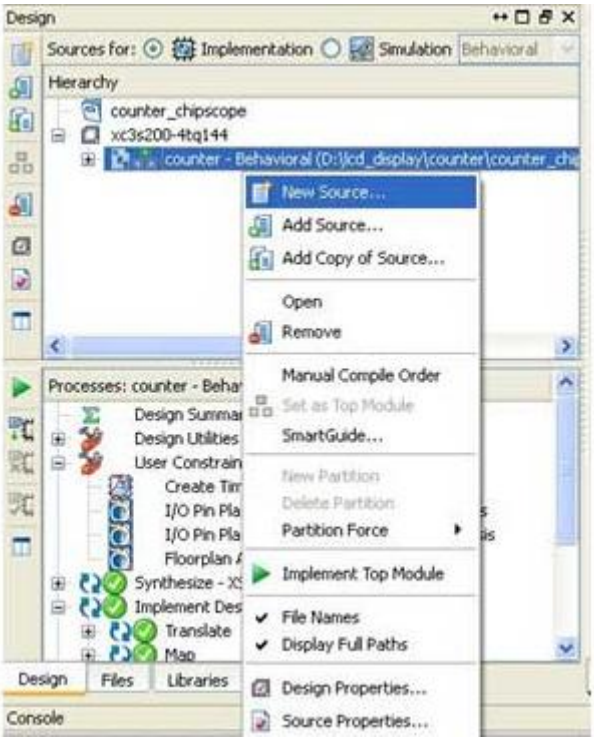
Xilinx provides Chip Scope Pro software to view hardware simulation (to view the current status of the hardware signals in a FPGA board). We can run the Chip Scope Pro for current project and also configure the FPGA through Chip Scope Pro. An FPGA may get inputs from the outside environment like switches or binary signals from external hardware and the Project output is changed depending on the input. When we change the input in hardware we get the output changes in hardware which is also updated in Chip Scope Pro simulation window. Thus we can view the real time changes of hardware signals in software. Refer Xilinx website to know more details about Chip Scope Pro.

Working procedure of Chip Scope Pro is given in the following section. Counter function is taken as an example to view the hardware simulation. Create a new project file and give the Verilog Coding, UCF file. Save the files and select New Source from Project menu. New Source Wizard window will open, here select Chip Scope Definition and Connection File. Give the file name and file location to store the file then click ‘Next’ and ‘Finish.

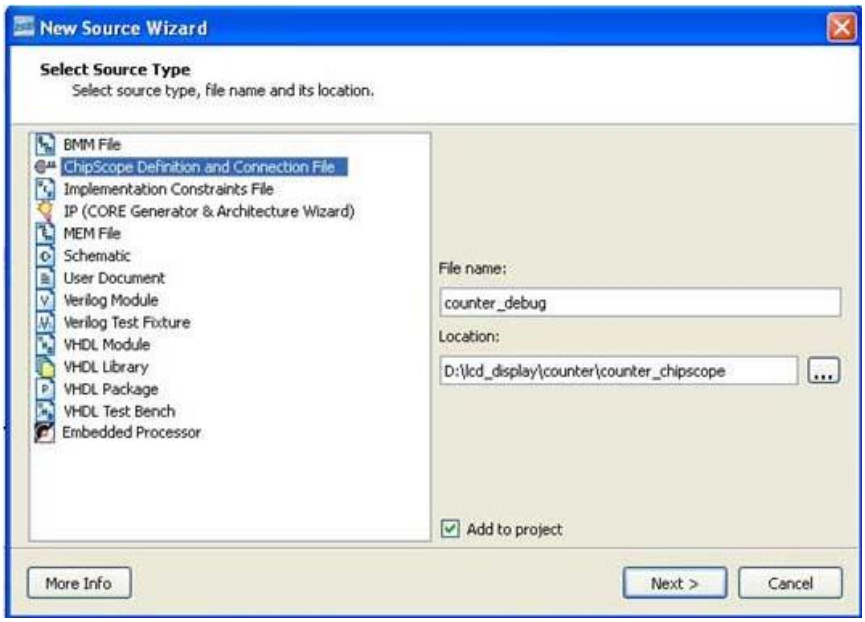
CONFIGURING THE LOGIC ANALYZER CORE:

In order to test the counter design we have to configure and insert the logic analyzer core in our design. Follow these steps:

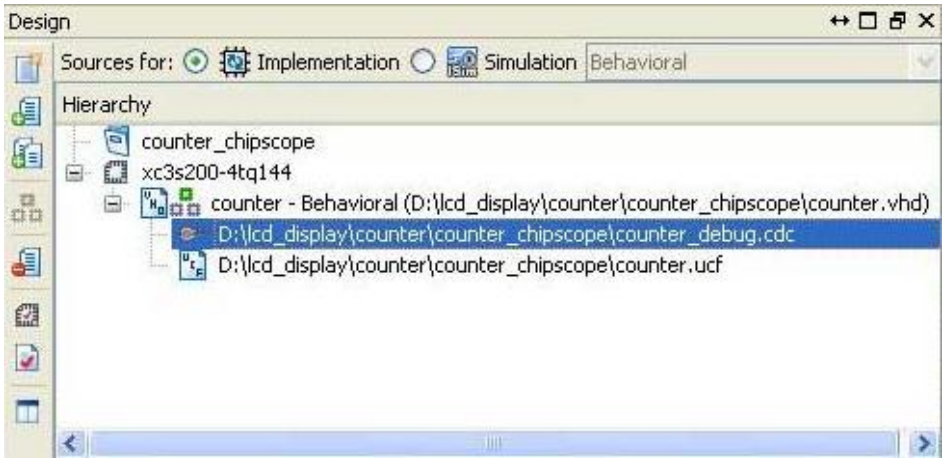
1. In the ‘Sources” view right click on the top module (counter.v) and select ‘New Source’



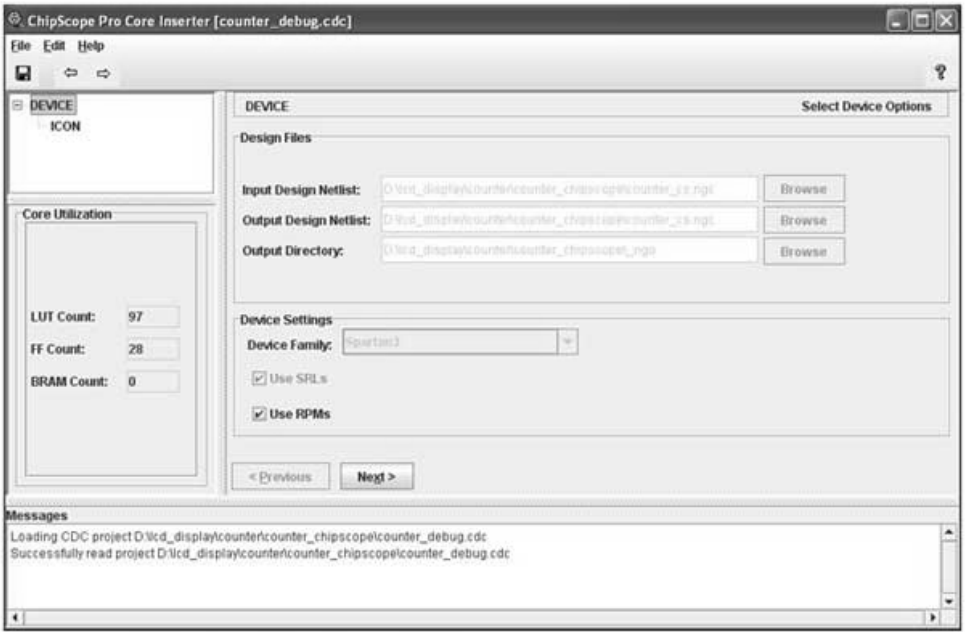
2. In the 'New Source Wizard' window, select 'Chipscope Definition and Connection File' and specify the filename as 'counter_db'. Click 'Next' and then click 'Finish'.



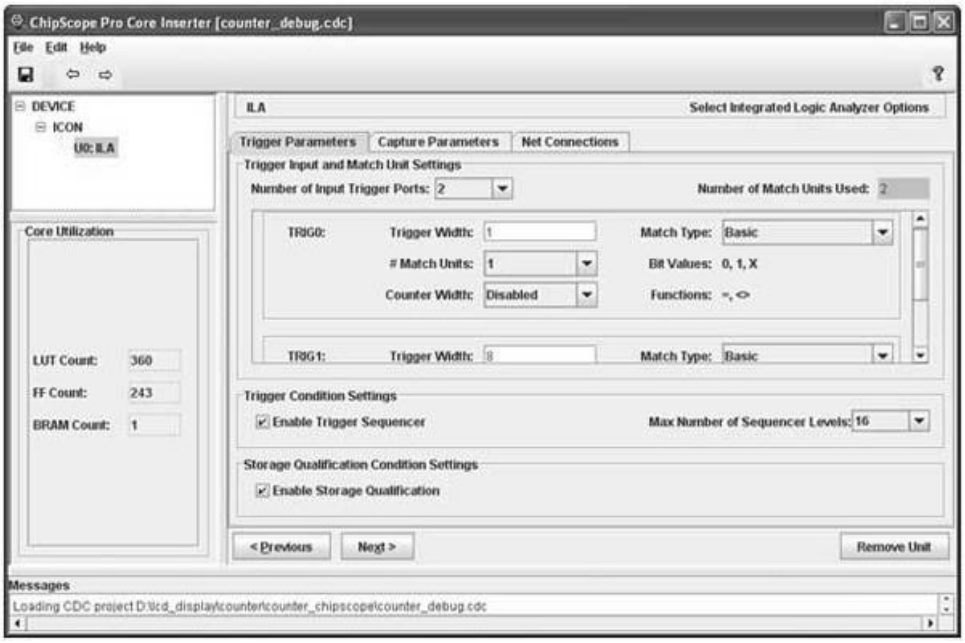
3. Note that 'counter_debug.cdc' file has been added to your 'Sources' list and is listed below the selected top module (counter).



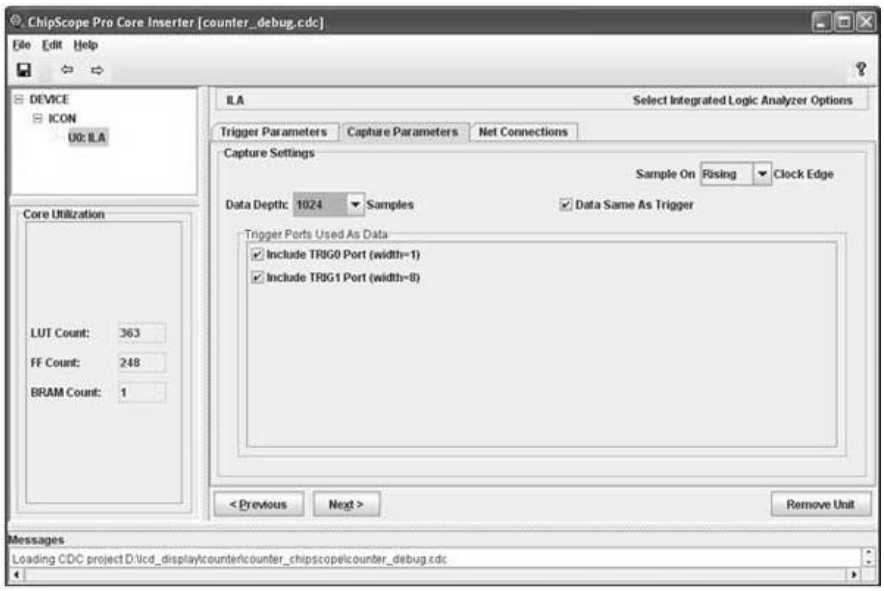
4. Double click on 'debug.cdc' to launch the ChipScope Pro Core Inserter application. This application will integrate the logic analyzer core into our counter design. Do not alter any settings on the first screen. Click 'Next'.



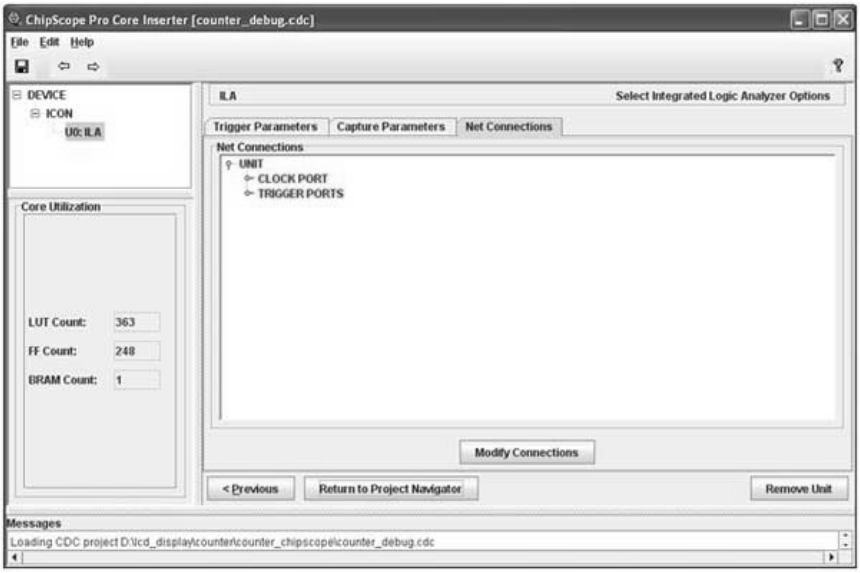
5. To observe any signal, we have to specify the trigger. Logic analyzer core will start capturing the desired signal upon activation of trigger signal. In this example we want to monitor the counter's counting action as soon as 'rst' signal is deactivated. So we will create two trigger ports. One port will be 'rst' signal and another port will be counter's eight least significant bits.
   Set 'Number of trigger ports' to 2.
   In 'TRIG0' frame set 'Trigger Width' as 1 (since 'rst' is one bit signal).
   In 'TRIG1' frame set 'Trigger Width' as 8 (as we want to observe counter's 8 least significant bits).
   Click Next.



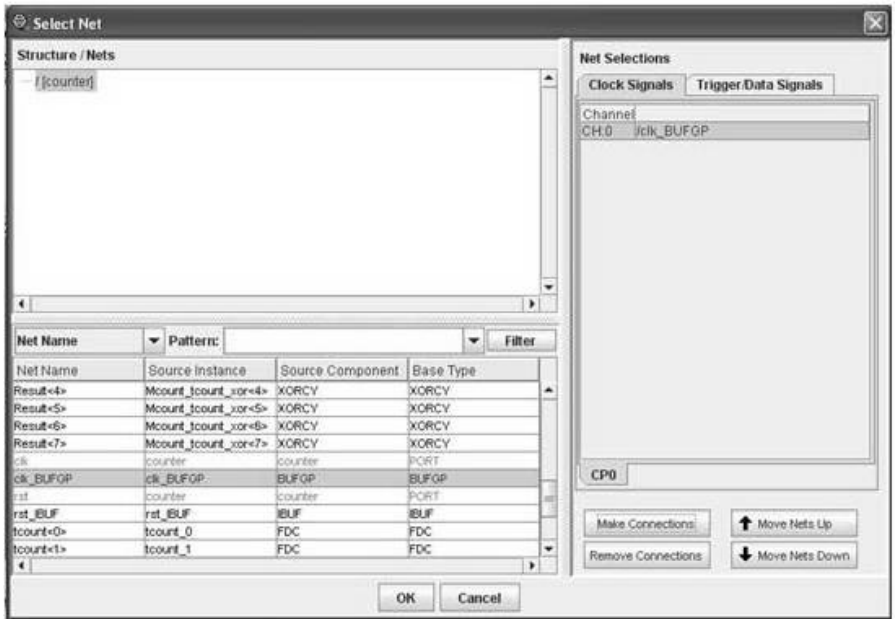
6. Now in this window we will specify capture parameters. We want to use our trigger ports as data ports which will be recorded by logic analyzer. We also want to sample data on rising clock edge.
   In 'Sample On' list select 'Rising'.
   Set Number of samples to be recorded by changing 'Data Depth' to 1024 samples. This will record 1024 samples from the trigger event. You can at the most record 16K samples.
   Select both check boxes in 'Trigger Ports Used As Data' frame.
   Click Next.



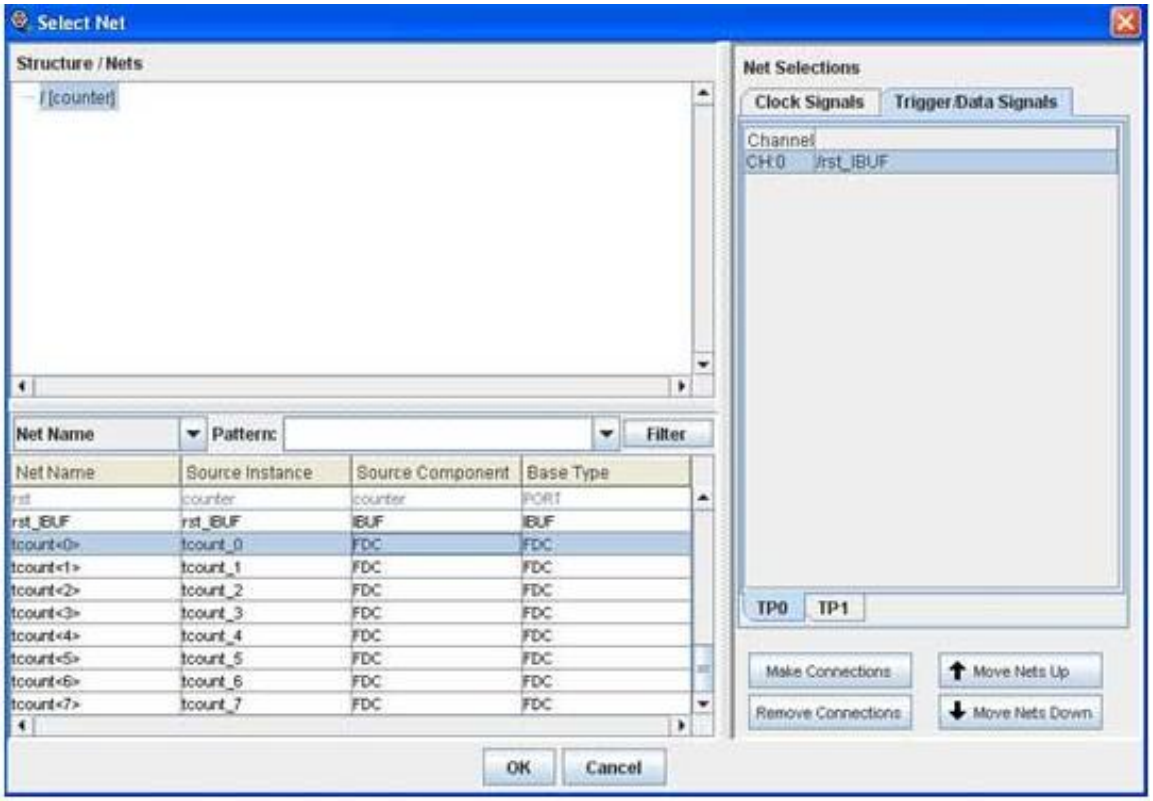
7. Now we will specify which signal(s) to be used as Clock and Trigger. Click on 'Modify Connections'.



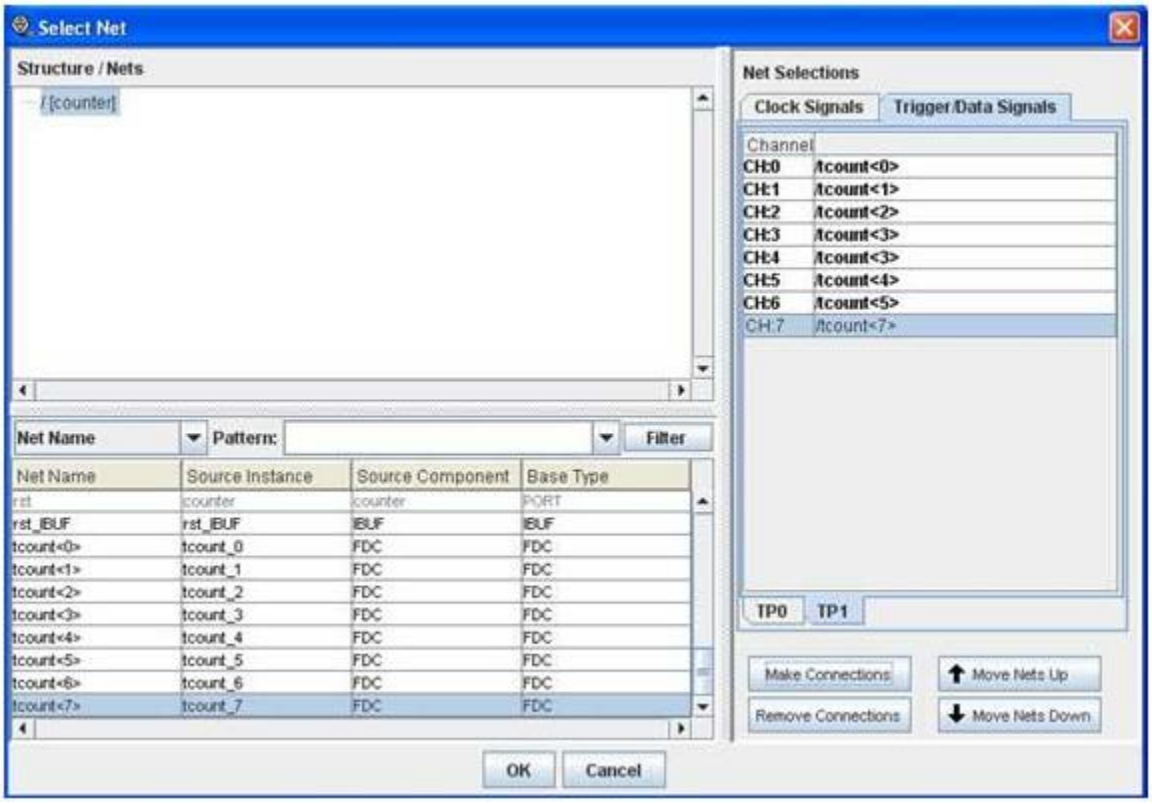
8. Select the 'Clock signals' Pane, then select 'clk_BUFG' signal from the left hand side list and then click on 'Make Connection'. This will add 'clk' signal as the clock signal for logic analyzer.



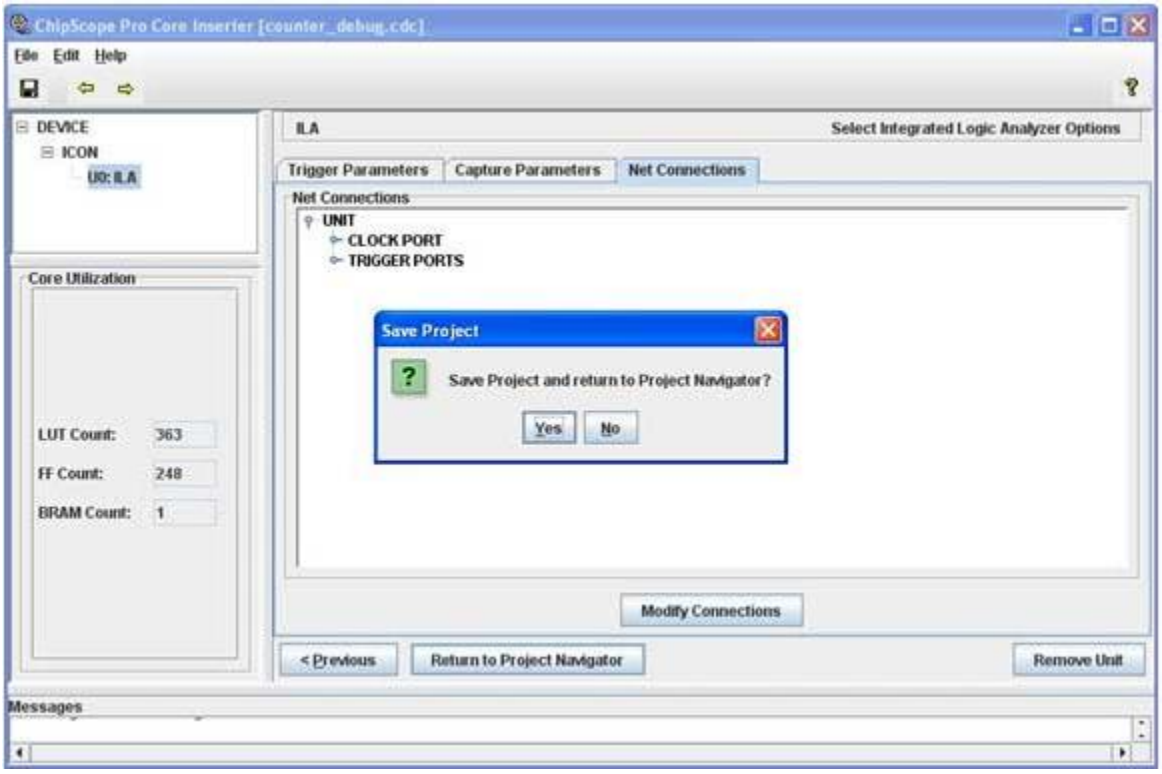
9. Now select 'Trigger/Data signals' pane. Select 'TP0' and connect 'rst_IBUF' signal to CH0 channel.



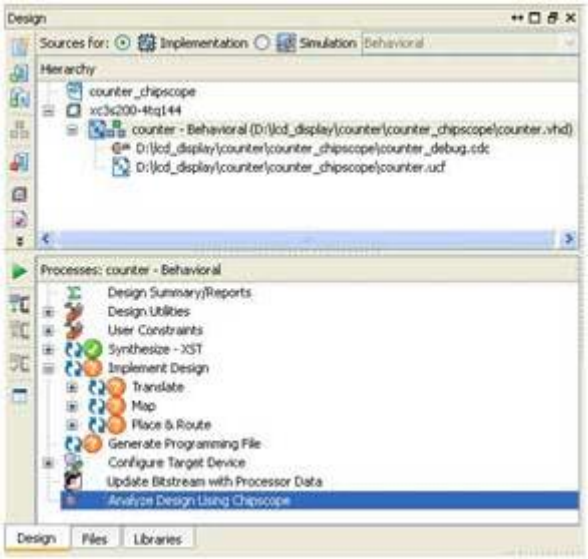
10. Similarly click on 'TP1' pane and add connect counter's lower eight bits to eight channels. Click 'OK' once you finish making connections.



11. Now in the main window click on 'Return to Project Navigator'. It will ask for saving the project, click 'Yes'. Now we are ready to compile the entire counter design along with the logic analyzer core.



12. In the ISE, select top level module ‘counter’ and in the ‘Processes’ pane double click on ‘Analyze Design Using ChipScope’. This will start the process to synthesize combined unit consisting of design under test (in this case counter) and the chipscope cores.



DEBUGGING THE DESIGN USING CHIPSCOPE ANALYZER TOOL:

Once the synthesis gets over, ISE will launch the Analyzer tool. Make sure that FPGA board is connected to PC.

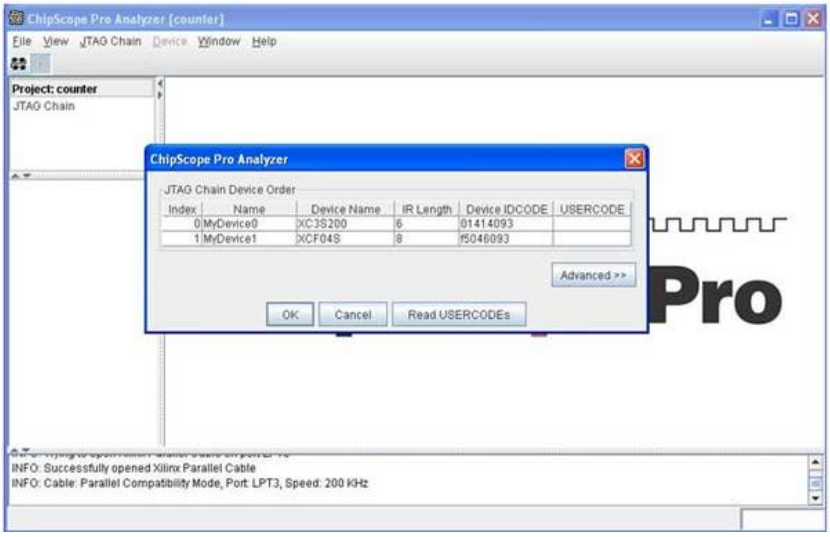
1. Once the analyzer tool is running, click on ‘Initialize JTAG Chain’ icon located at the top right corner of the window. This will initialize the JTAG chain and identify the devices found in the chain. A dialog box will appear showing the devices discovered. Click ‘OK’.

2. Now select the FPGA device from the JTAG chain, right click and then select ‘Configure’ to specify the configuration bit stream file.

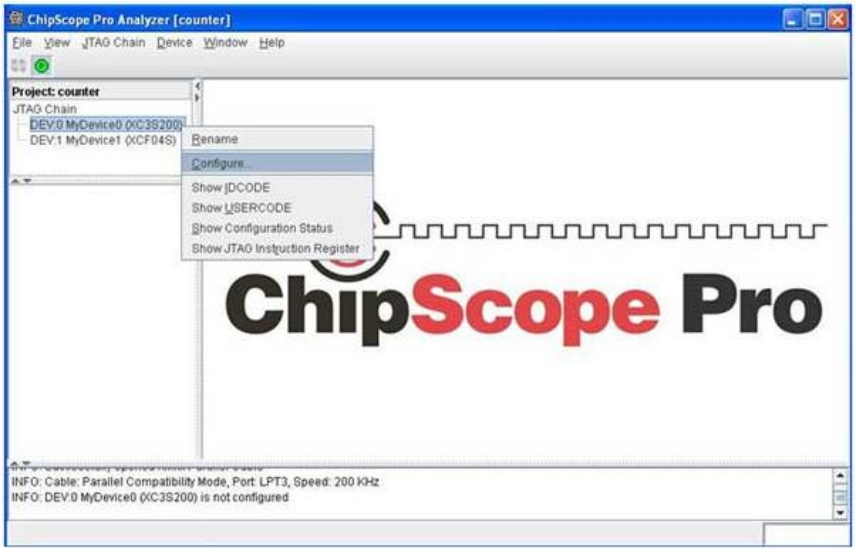
3. Select the bit stream file ‘cntr.bit’ from the bit stream folder. Then click ‘OK’.

IMPORTANT:

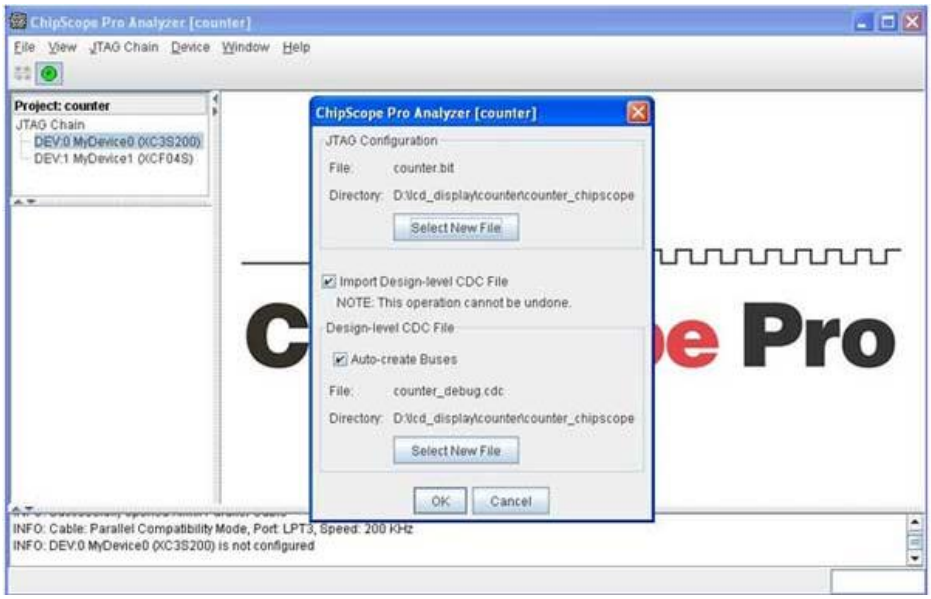
4. After clicking ‘OK’, tool will load the bit stream file into FPGA and check the availability of debugging cores. If debugging core is found tool will show ‘INFO: Found 1 Core Unit in the JTAG device Chain.’ Message in status window.

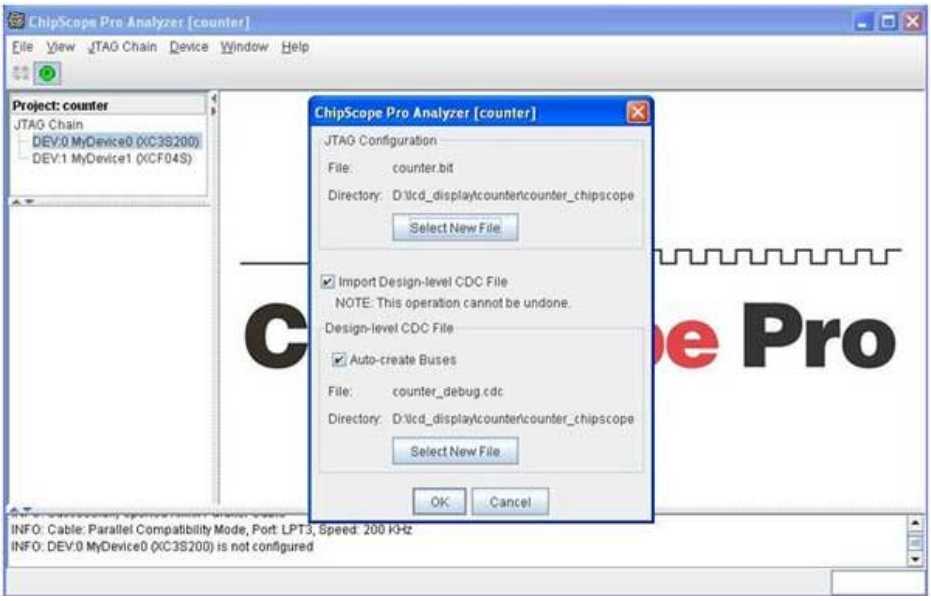


If you see ‘Found 0 Core …’ message instead, then either you have selected wrong bit stream file or something has gone wrong in one of the previous steps and debugging core has not been inserted properly into the design.



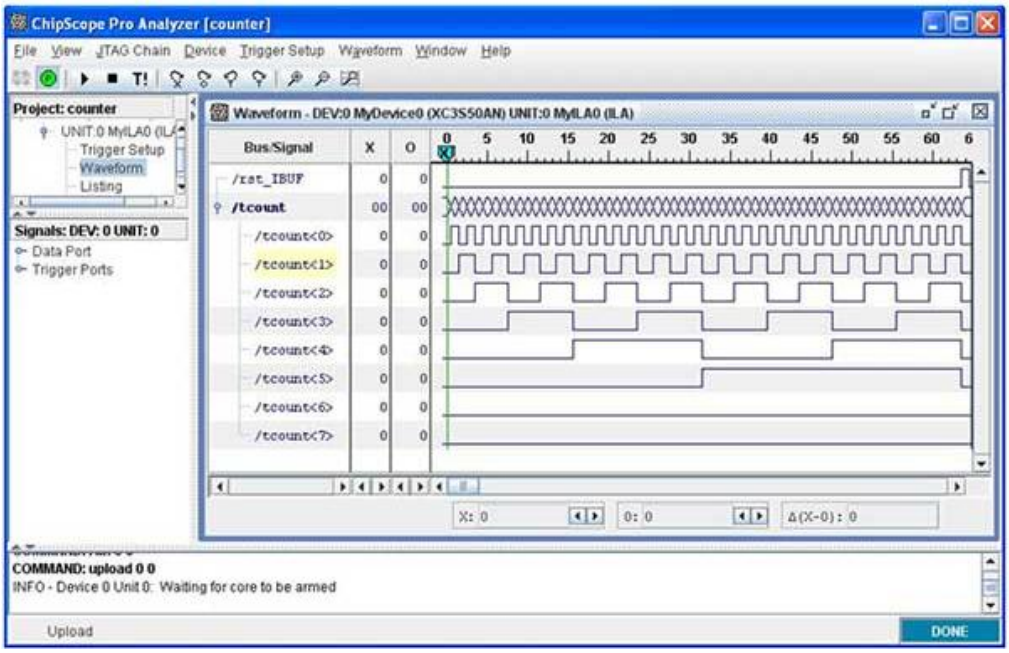
If everything is fine then you will see options for Logic Analyzer core inserted in our design. Now double click on the ‘Trigger Setup’ element to launch trigger setup window. And for trigger port 0 (i.e. ‘rst’ signal) specify the trigger Value 0.





5. This will make logic analyzer to trigger as soon as ‘rst’ become zero and record 1024 samples on successive clock edges. Note that trigger signals are sampled on rising clock edge. Double click on ‘Waveform’ element to see the waveform. 5. Now everything is ready. To apply the settings and ARM the trigger click on button. After that press the ‘Down’ button on the development board to release the ‘rst’ signal. This will trigger the logic analyzer. Once 1024 samples are recorded, this data will be

transferred to PC and will be displayed in the waveform window.



NOTE: To see the names of the trigger ports, you can import the 'debug.cdc' file in analyzer tool. Click on File>Import and then select 'counter_debug.cdc'

PROGRAM:

```
module counter (clk,clr,q);
      input  clk,clr;
      output [3:0]q;
      reg    [3:0]q;
      always @(posedge clk)
      begin
        if (clr)
          q<=4'b0000;
        else
          q<=q + 1'b1;
      end
 endmodule
```

NETLIST:

```
# PlanAhead Generated physical constraints
NET "clk" LOC = A8;
NET "clr" LOC = T14;
NET "q[0]" LOC = R1;
NET "q[1]" LOC = R2;
NET "q[2]" LOC = K3;
NET "q[3]" LOC = T4;
```

RESULT:

TANNER

TOOLS

Introduction to Tanner Tool

Tanner tool is a Spice Computer Analysis Programmed for Analogue Integrated Circuits. Tanner tool consists of the following Engine Machines:

1. S-EDIT (Schematic Edit)
2. T-EDIT (Simulation Edit)
3. W-EDIT (Waveforms Edit)
4. L-EDIT (Layout Edit)

Using these engine tools, spice program provides facility to the use to design & simulate new ideas in Analogue Integrated Circuits before going to the time consuming & costly process of chip fabrication.

- **SCHEMATIC EDIT TOOL (S-EDIT)**

S-Edit is hierarchy of files, modules & pages. It introduces symbol & schematic modes. S-Edit provides the facility of:

1. Beginning a design.
2. Viewing, drawing & editing of objects.
3. Design connectivity.
4. Properties, net lists & simulation.
5. Instance & browse schematic & symbol mode.

Beginning a design: It explains the design process in detail in terms of file module operation and module.

Browser: Effective schematic design requires a working knowledge of the S-Edit design hierarchy of files & modules. S-Edit design files consist of modules. A module is a functional unit of design such as a transistor, a gate and an amplifier.

Modules contain two components:

1) Primitives: Geometrical objects created with drawing tools.

2) Instances: References to other modules in file. The instanced module is the original.

S-Edit has two viewing modes:

1. Schematic Mode: to create or view a schematic, we operate in schematic mode.
2. Symbol Mode: it represents symbol of a larger functional unit such as operational amplifier.

T-SPICE PRO CIRCUIT ANALYSIS

An introduction to the integrated components of the T- Spice Pro circuit analysis suite:

Schematic data files (.sdb): describes the circuits to be analyzed in graphical form, for display and editing by S- Edit" Schematic Editor.

Simulation input files (.sp): describes the circuits to be analyzed in textual form, for editing and simulation by T- Spice" Circuit Simulator.

Simulation output files (.out): containing the numerical results of the circuit analyses, for manipulation and display by W- Edit" Waveform Viewer.

➢ CIRCUIT SIMULATOR (T-SPICE)

T- Spice Pro's waveform probing feature integrates S- Edit, T- Spice, and W- Edit to allow individual points in a circuit to be specified and analyzed. A few analysis is described below:

The heart of T-Spice operation is the input file (also known as the circuit description, the net list & the input deck). This is a plain text file that contains the device statement & simulation commands, drawn from the SPICE circuit description language with which T-Spice constructs a model of the circuit to be simulated. Input files can be created and modified with any text editor.

T-Spice is a tool used for simulation of the circuit. It provides the facility of

1. Design Simulation
2. Simulation Commands
3. Device Statements
4. User-Designed External Models
5. Small Signal & Noise Models

T-Spice uses Kirchhoff's Current Law (KCL) to solve circuit problems. To T-Spice, a circuit is a set of devices attached to nodes. The voltage at all nodes represents the circuit state. T-Spice solves for a set of node voltage that satisfied KCL (implying that sum of currents flowing into each node is zero). In order to evaluate whether a set of node voltages is a solution, T-Spice computers and sums all the current flowing out of each device into nodes connected to it (its terminals). The relationship between the voltages at device terminals and the currents through the terminal is determined by the device model for a resistor of resistance R is

$$I=\Delta V/R$$

Where, ΔV represents the voltage difference across the device. A few analyses are discussed below:

DC Operating Point Analysis

DC operating point analysis finds a circuit's steady- state condition, obtained (in principle) after the input voltages have been applied for an infinite amount of time. The .include command causes T- Spice to read in the contents of the model file for the evaluation of NMOS and PMOS transistors.

The technology file assigns values to MOSFET model parameters for both n - and p -type devices. When read by the input file, these parameters are used to evaluate MOSFET model equations, and the results are used to construct internal tables of current and charge values. Values read or interpolated from these tables are used in the computations called for by the simulation. Following each transistor name are the names of its terminals. The required order of terminal names is: drain -gate -source -bulk. Then the model name (NMOS or PMOS in this example), and physical characteristics such as length and width, are specified. The .op command performs a DC operating point calculation and writes the results to the file specified in the Simulate > Start Simulation dialog. The output file lists the DC operating point information for the circuit described by the input file.

DC Transfer Analysis

DC transfer analysis is used to study the voltage or current at one set of points in a circuit as a function of the voltage or current at another set of points. This is done by sweeping the source variables over specified ranges, and recording the output. A list of sources to be swept, and the voltage ranges across which the sweeps are to take place follow the .dc command, indicating transfer analysis. The transfer analysis will be performed as follows: vdd will be set at 5 volts and vin will be swept over its specified range; vdd will then be incremented and vin will be reswept over its range; and so on, until vdd reaches the upper limit of its range. The .dc command ignores the values assigned to the voltage sources vdd and vin in the voltage source statements, but they must still be declared in those statements. The results for nodes in and out are reported by the .print dc command to the specified destination.

Transient Analysis

Transient analysis provides information on how circuit elements vary with time. The basic T- Spice command for transient analysis has three modes. In the default mode, the DC operating point is computed, and T- Spice uses this as the starting point for the transient simulation. The .tran command specifies the characteristics of the transient analysis to be performed.

AC Analysis

AC analysis characterizes the circuit's behavior dependence on small- signal input frequency. It involves

three steps: (1) calculating the DC operating point; (2) linearizing the circuit; and (3) solving the linearized circuit for each frequency. When ac voltage source is to be applied, then vdiff sets the DC voltage difference between nodes the two nodes to -0. 0007 volts; its AC magnitude is 1 volt and its AC phase is 180 degrees. The .ac command performs an AC analysis. Following the .ac keyword is information concerning the frequencies to be swept during the analysis. In case, the frequency is to be swept logarithmically, by decades (DEC); 5 data points are to be included per decade is considered to be the standard The two .print commands write the voltage magnitude (in decibels) and phase (in degrees), respectively, for the node out to the specified file. The .acmodel command writes the small- signal model parameters and operating point voltages and currents for all circuit devices

Noise Analysis

Real circuits, of course, are never immune from small, random fluctuations in voltage and current levels. In T- Spice, the influence of noise in a circuit can be simulated and reported in conjunction with AC analysis. The purpose of noise analysis is to compute the effect of the noise associated with various circuit devices on an output voltage or voltages as a function of frequency. Noise analysis is performed in conjunction with AC analysis; if the .ac command is missing, then the .noise command is ignored. With the .ac command present, the .noise command causes noise analysis to be performed at the same frequencies. The .noise command takes two arguments: the output at which the effects of noise are to be computed, and the input at which the .noise can be considered to be concentrated for the purposes of estimating the equivalent noise spectral density. The print command is used to print results.

➢ WAVEFORM EDIT

The ability to visualize the complex numerical data resulting from VLSI circuit simulation is critical to testing, understanding & improving these circuits. W-Edit is a waveform viewer that provides ease of use, power & speed in a flexible environment designed for graphical data representation. The advantages of W-Edit include:

1. Tight Integration with T-spice, Tanner EDA_s circuit level simulator. W-Edit can chart data generated by T-spice directly, without modification of the output text data files. The data can also be charted dynamically as it is produced during the simulation.

2. Charts can automatically configure for the type of data being presented.

3. A data is treated by W-Edit as a unit called a trace. Multiple traces from different output files can be viewed simultaneously in single or several windows; traces can be copied and moved between charts & windows. Trace arithmetic can be performed on existed tracing to create new ones.

4. Chart views can be panned back & forth and zoomed in & out, including specifying the exact X-Y co-ordinate range.

5. Properties of axes, traces, rides, charts, text & colors can be customized.

Numerical data is input to W-Edit in the form of plain or binary text files. Header & Comment information supplied by T-Spice is used for automatic chart configuration. Runtime update of results is made possible by linking W-Edit to a running simulation in T-Spice. W-Edit saves data with chart, trace, axis & environment settings in files with the WDB (W-Edit Database).

➢ LAYOUT(L-EDIT)

It is a tool that represents the masks that are used to fabricate an integrated circuit. It describes a layout design in terms of files, cells & mask primitives. On the layout level, the component parameters are totally different from schematic level. So it provides the facility to the user to analyze the response of the circuit before forwarding it to the time consuming & costly process of fabrication. There are rules for designing layout diagram of a schematic circuit using which user can compare the output response with the expected one.

L- Edit: An Integrated Circuit Layout Tool

In L- Edit, layers are associated with masks used in the fabrication process. Different layers can be conveniently represented by different colors and patterns. L- Edit describes a layout design in terms of files, cells, instances, and mask primitives. You may load as many files as desired into memory. A file may be composed of any number of cells. A file may be composed of any number of cells. These cells may be hierarchically related, as in a typical design, or they may be independent, as in a library file. Cells may contain any number or combination of mask primitives and instances of other cells.

Cells: The Basic Building Blocks

The basic building block of the integrated circuit design in L- Edit is a cell. Design layout occurs within cells. A cell can:

- ❖ Contain part or all of the entire design.
- ❖ Be referenced in other cells as a sub- cell, or instance.
- ❖ Be made up entirely of instances of other cells.
- ❖ Contain original drawn objects, or primitives.
- ❖ Be made up entirely of primitives or a combination of primitives and instances of other cells.

Hierarchy

L- Edit supports fully hierarchical mask design. Cells may contain instances of other cells. An instance is a reference to a cell; should you edit the instanced cell, the change is reflected in all the instances of that cell. Instances simplify the process of updating a design, and also reduce data storage requirements, because an instance does not need to store all the data within the instanced cell instead, only a reference to the instanced cell is stored, along with information on the position of the instance and on how the instance may be rotated and mirrored.

L- Edit does not use a "separated" hierarchy: instances and primitives may coexist in the same cell at any level in the hierarchy. Design files are self- contained. The pointer to a cell contained in an instance always points to a cell within the same design file. When cells are copied from one file to another, L- Edit automatically copies across any cells that are instanced by the copied cell, to maintain the self- contained nature of the destination file.

Design Rules

Manufacturing constraints can be defined in L- Edit as design rules. Layouts can be checked against these design rules.

Design Features

L- Edit is a full- custom mask editor. Manual layout can be accomplished more quickly because of L Edit's intuitive user interface. In addition, one can construct special structures to utilize a technology without, worrying about problems caused by automatic transformations. Phototransistors, guard bars, vertical and horizontal bipolar transistors, static structures, and Schottky diodes, for example, are as easy to design in CMOS- Bulk technology as are conventional MOS transistors.

Floor plans

L- Edit is a manual floor planning tool. You have the choice of displaying instances in outline, identified only by name, or as fully fleshed- out mask geometry. When you display your design in outline, you can manipulate the arrangement of the cells in your design quickly and easily to achieve the desired floor plan. One can manipulate instances at any level in the hierarchy, with insides hidden or displayed, using the same graphical move/ select operations or rotation/ mirror commands that you use on primitive mask geometry.

Memory Limits

In L- Edit, one can make your design files as large as one like, given available RAM and disk space.

Hard Copy

L- Edit provides the capability to print hard copy of the design. A multiage option allows very large plots to

be printed to a specific scale on multiple 8 1/ 2 x 11 inch pages. An L- Edit macro is available to support large- format, high- resolution, color plotting on inkjet plotters.

Variable Grid

L- Edit's grid options support lambda- based design as well as micron- based and mil- based design.

Error Recovery

L- Edit's error- trapping mechanism catches system errors and in most cases provides a means to recover without losing or damaging data.

L- Edit Modules

- L- Edit ™: a layout editor
- L- Edit ¤ Extract ™: a layout extractor
- L- Edit ¤ DRC ™: a design rule checker

L- Edit is a full- featured, high-performance, interactive, graphical mask layout editor. L- Edit generates layouts quickly and easily, supports fully hierarchical designs, and allows an unlimited number of layers, cells, and levels of hierarchy. It includes all major drawing primitives and supports 90°, 45°, and all- angle drawing modes.

L- Edit ¤ Extract creates SPICE- compatible circuit netlists from L- Edit layouts. It can recognize active and passive devices, sub circuits, and the most common device parameters, including resistance, capacitance, device length, width, and area, and device source and drain area.

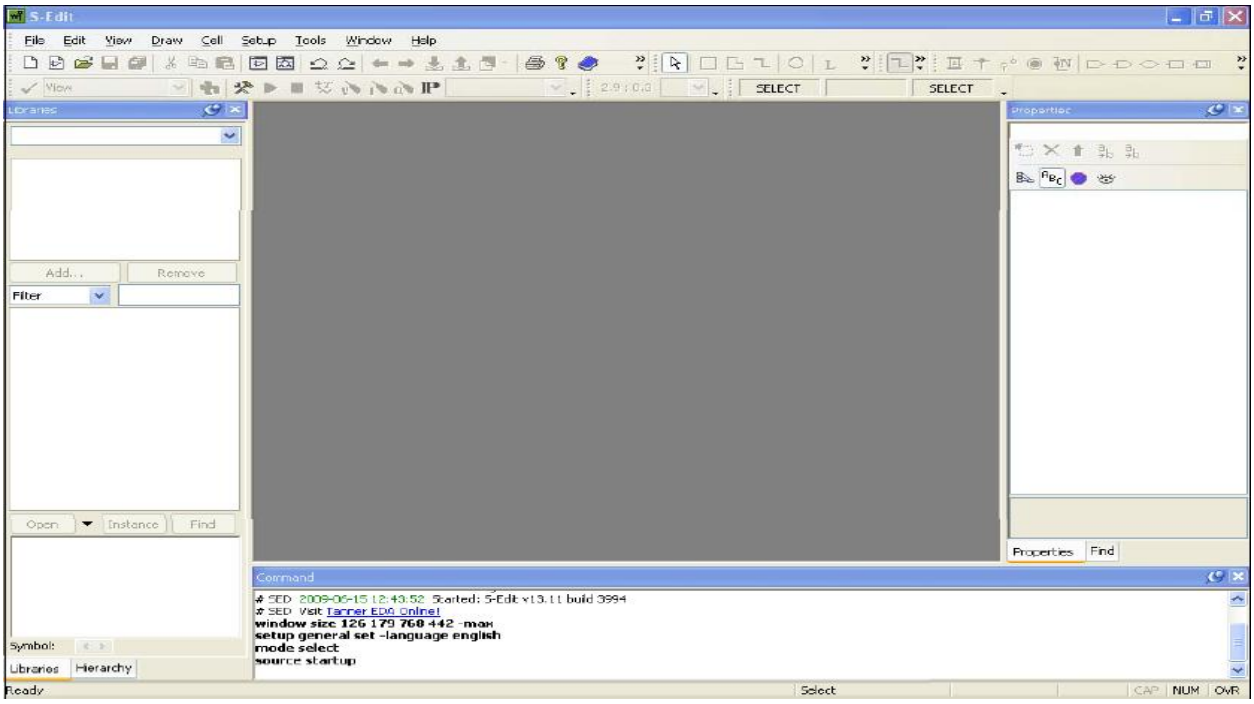
L- Edit ¤ DRC features user- programmable rules and handles minimum width, exact width, minimum space, minimum surround, non- exist, overlap, and extension rules. It can handle full chip and region- only DRC. DRC offers Error Browser and Object Browser functions for quickly and easily cycling through rule-checking errors.

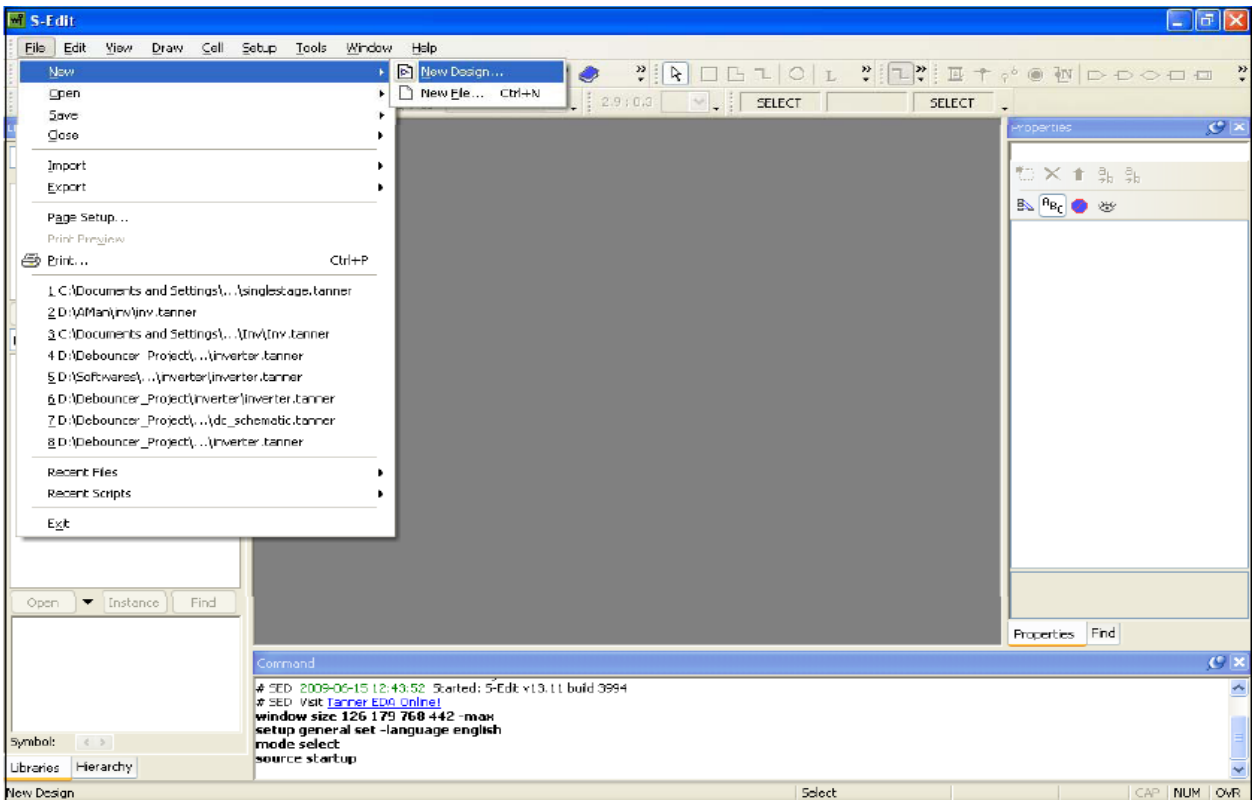
Steps to use Tanner tool:

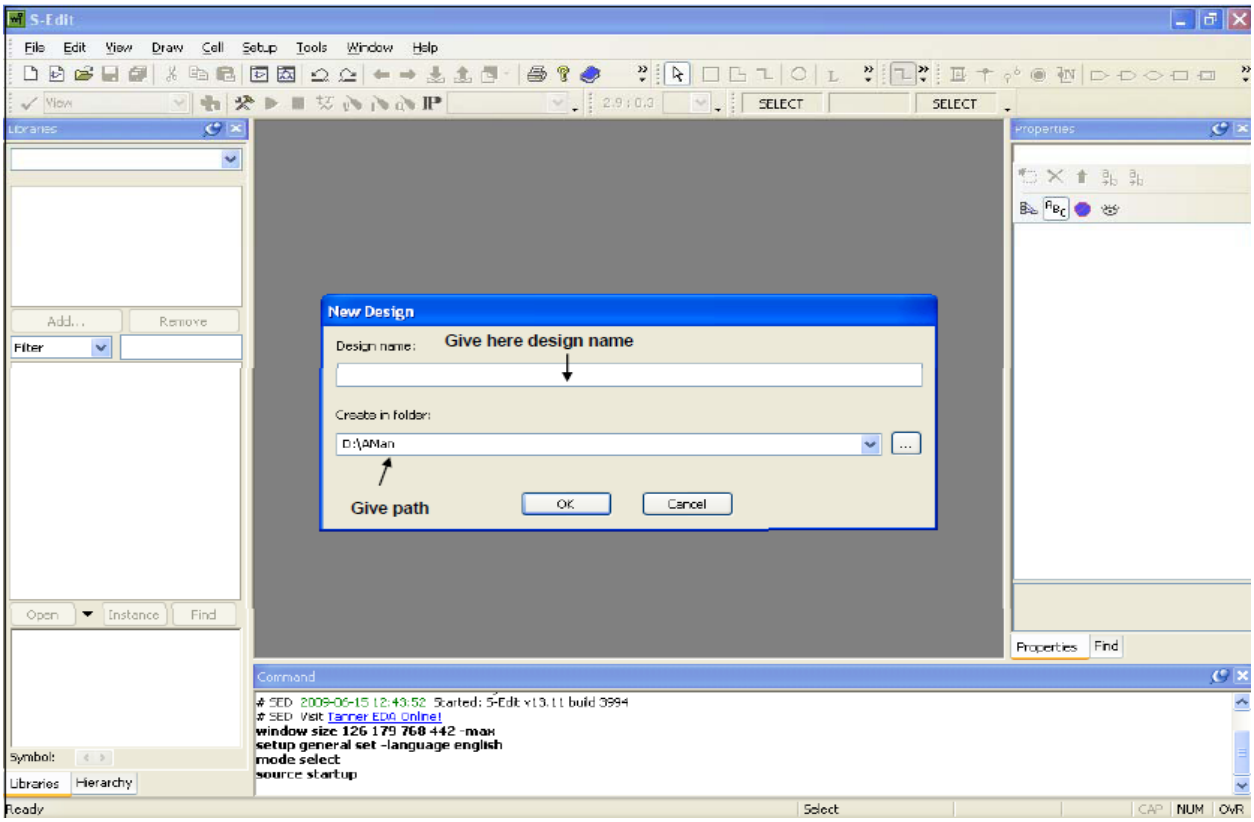
i.SCHEMATIC (S-edit):

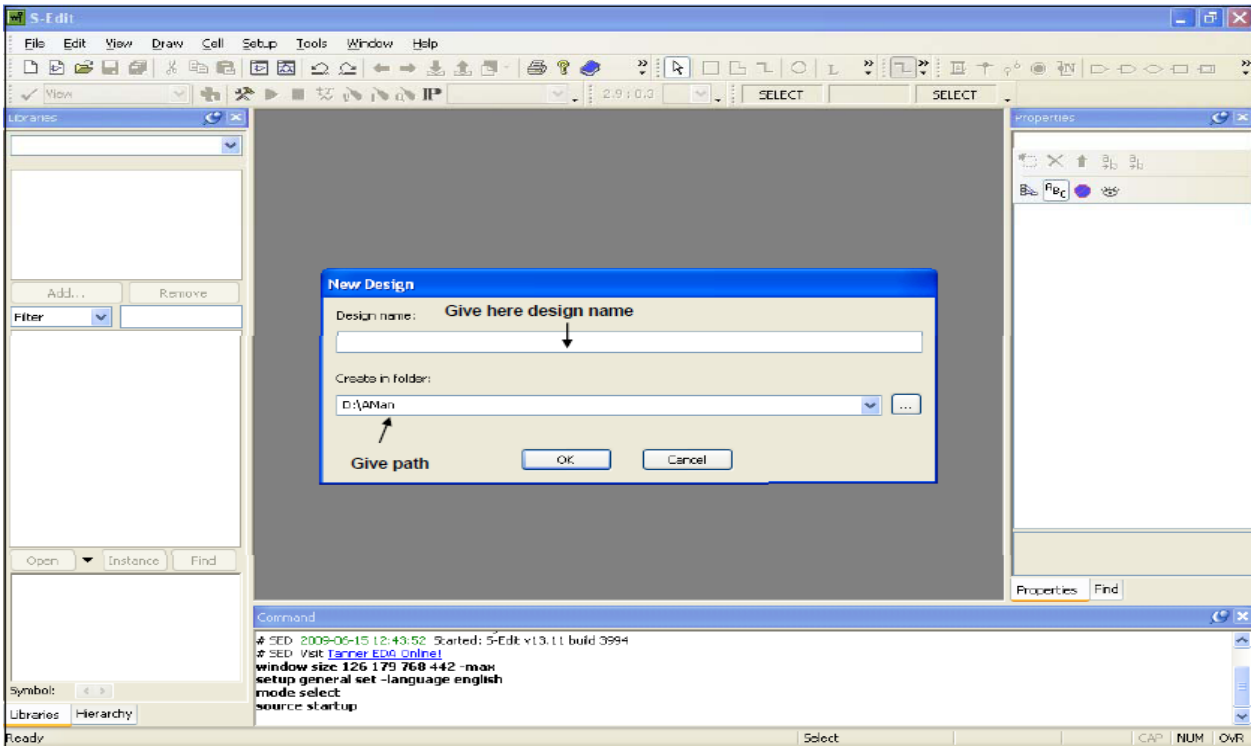
Start the tanner EDA by using the desktop shortcut or by using the

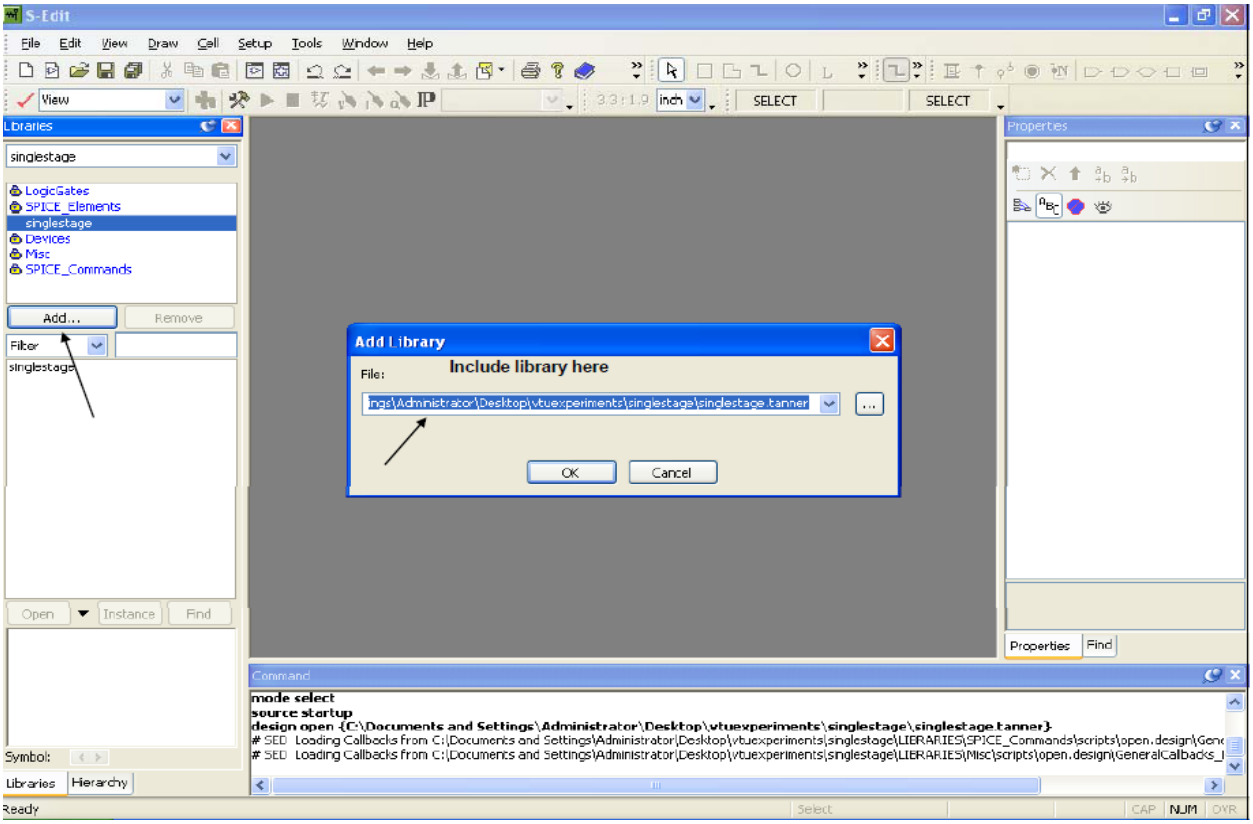
- Start → Programs → tanner EDA →tanner tool v13.0 →S-edit.

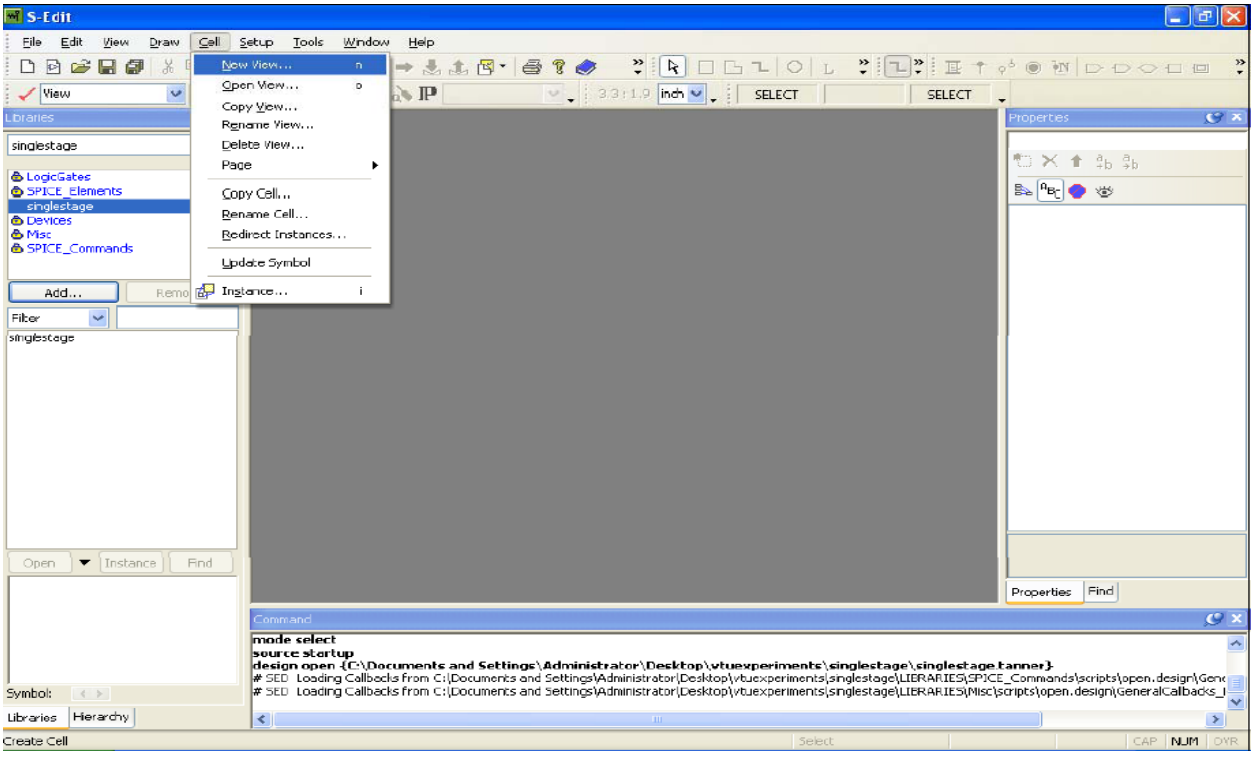


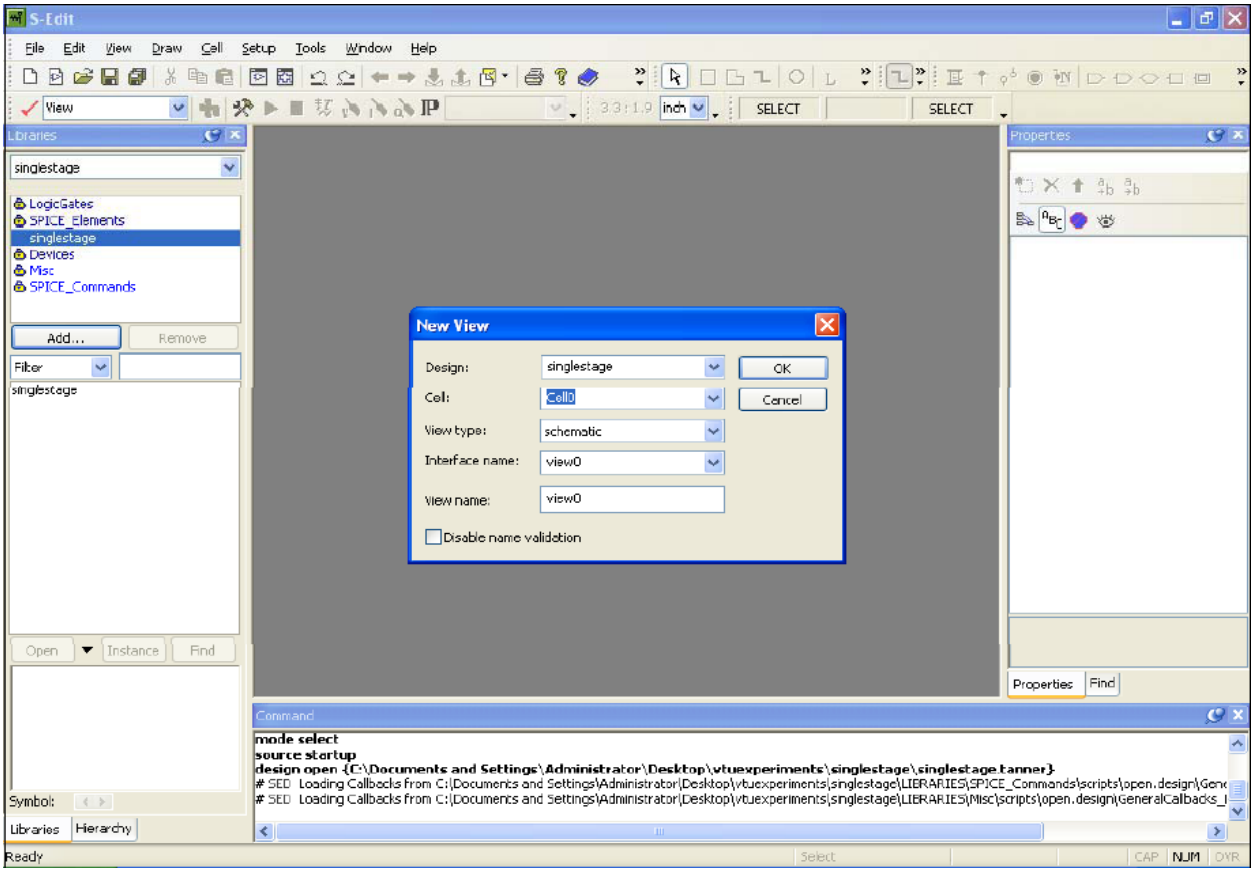


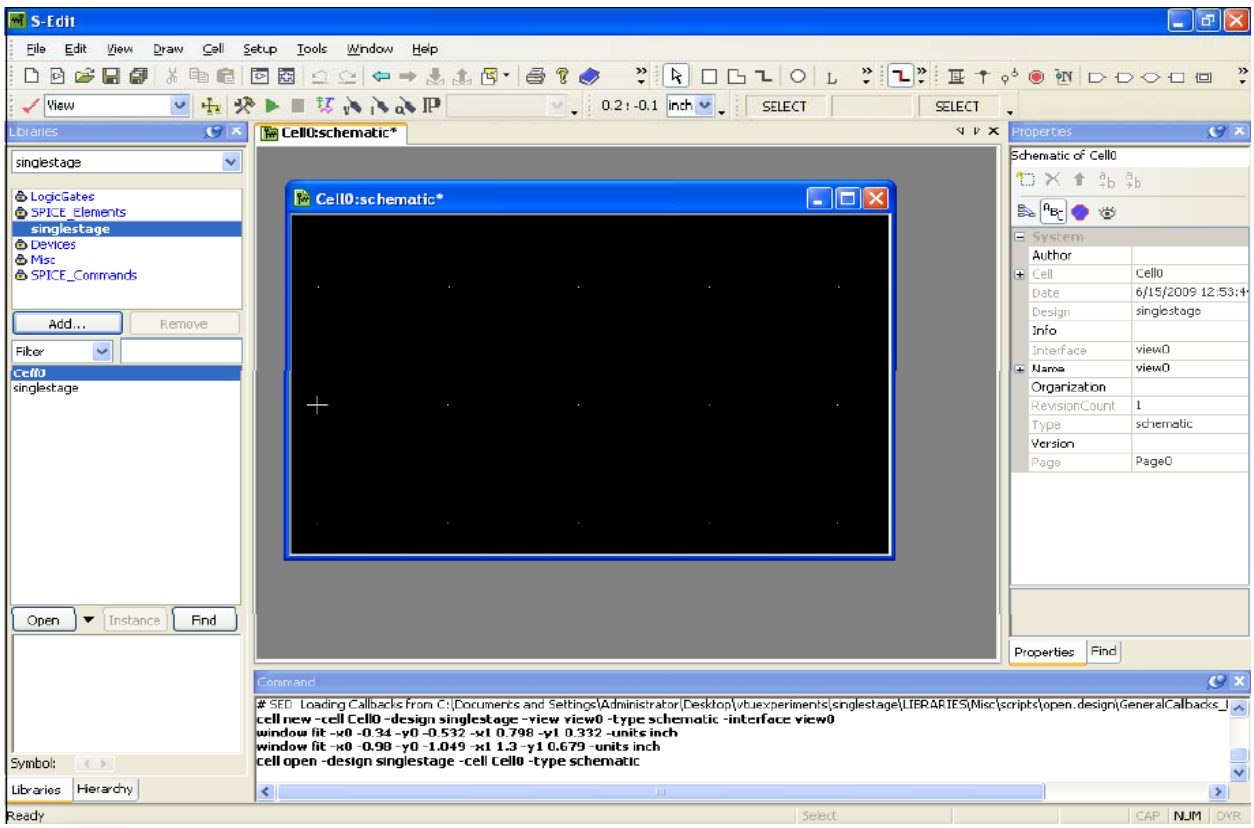

S-Edit
File Edit View Draw Cell Setup Tools Window Help
SELECT
SELECT
Libraries
Add...
Remove
Filter
Open
Instance
Find
Symbol:
Libraries
Hierarchy
Properties
Properties
Find
New Design
Design name:
Give here design name
Create in folder:
D:\AMan
Give path
OK
Cancel
Command
SED 2009-06-15 12:43:52 Started: S-Edit v13.11 build 3994
SED Visit Tanner EDA Online!
window size 126 179 768 442 -max
setup general set -language english
mode select
source startup
Ready
Select
CAP
NUM
OVR

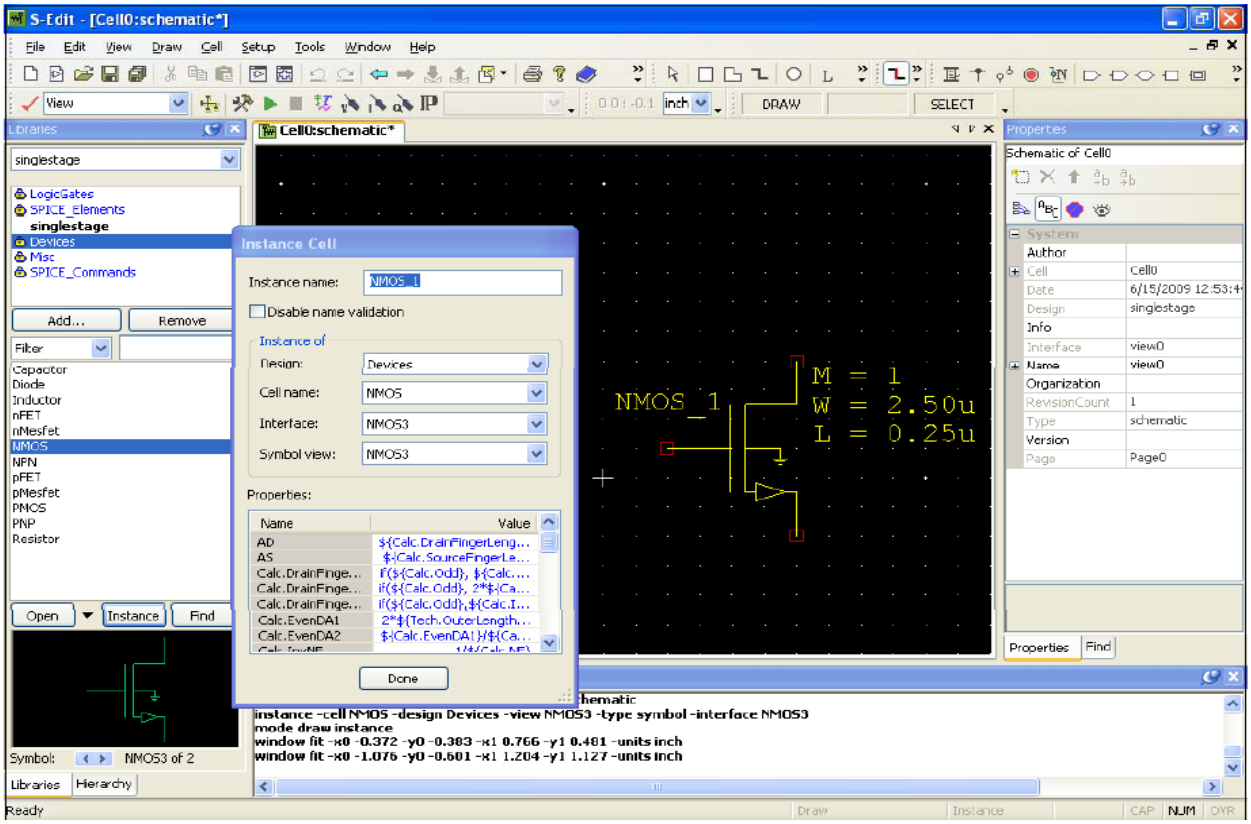

S-Edit
File Edit View Draw Cell Setup Tools Window Help
SELECT
SELECT
Libraries
Add...
Remove
Filter
Open
Instance
Find
Symbol:
Libraries
Hierarchy
Properties
Properties
Find
New Design
Design name:
Give here design name
Create in folder:
D:\AMan
Give path
OK
Cancel
Command
SED 2009-06-15 12:43:52 Started: S-Edit v13.11 build 3994
SED Visit Tanner EDA Online!
window size 126 179 768 442 -max
setup general set -language english
mode select
source startup
Ready
Select
CAP
NUM
OVR

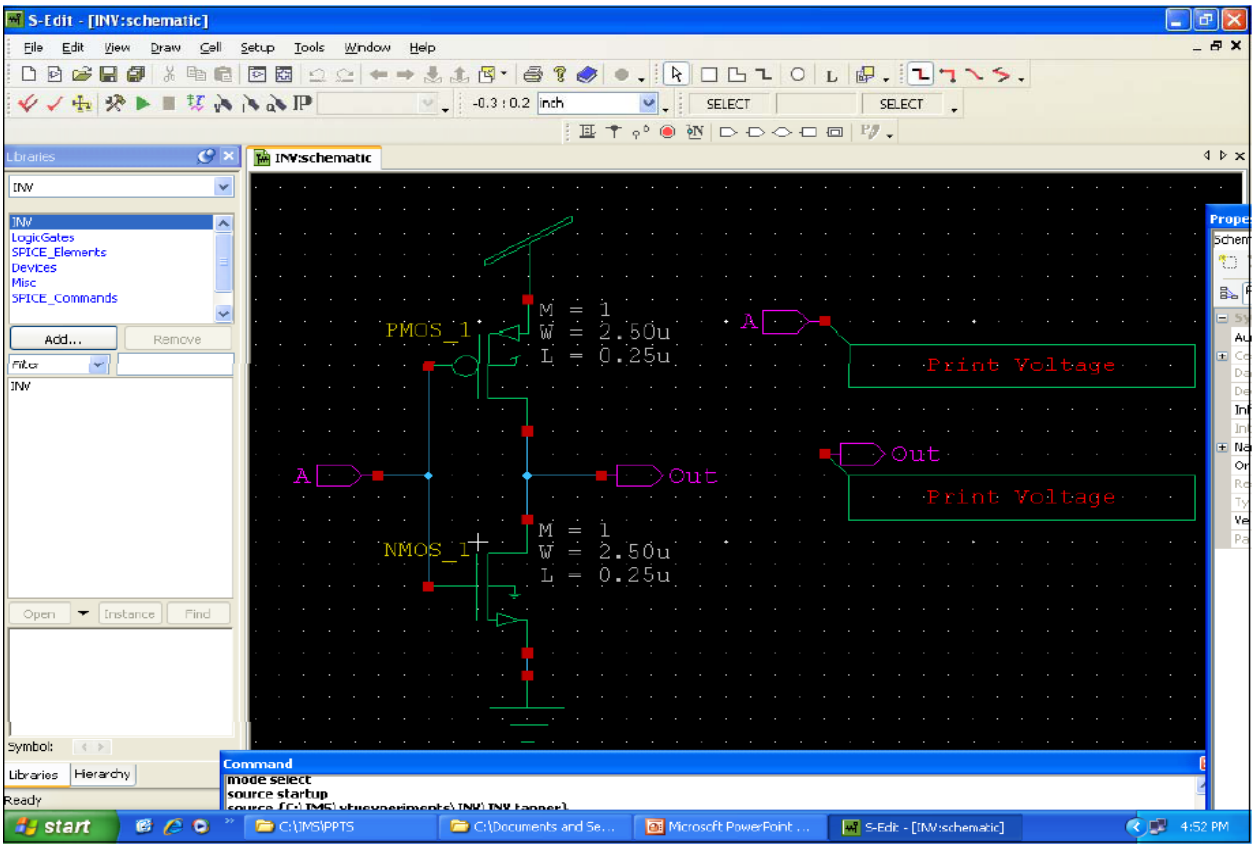

S-Edit
File Edit View Draw Cell Setup Tools Window Help
Libraries
singlestage
LogicGates
SPICE_Elements
singlestage
Devices
Misc
SPICE_Commands
Add...
Remove
Filter
singlestage
Open
Instance
Find
Symbol:
Libraries
Hierarchy
Add Library
File:
Include library here
ings\Administrator\Desktop\vtuexperiments\singlestage\singlestage.tanner
OK
Cancel
Properties
Properties
Find
Command
mode select
source startup
design open {C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\singlestage.tanner}
Ready


S-Edit
File Edit View Draw Cell Setup Tools Window Help
New View... n
Open View... o
Copy View...
Rename View...
Delete View...
Page
Copy Cell...
Rename Cell...
Redirect Instances...
Update Symbol
Instance... i
Libraries
singlestage
LogicGates
SPICE_Elements
singlestage
Devices
Misc
SPICE_Commands
Add...
Filter
singlestage
Open
Instance
Find
Symbol:
Libraries
Hierarchy
Properties
Find
Command
mode select
source startup
design open {C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\singlestage.tanner}
Create Cell

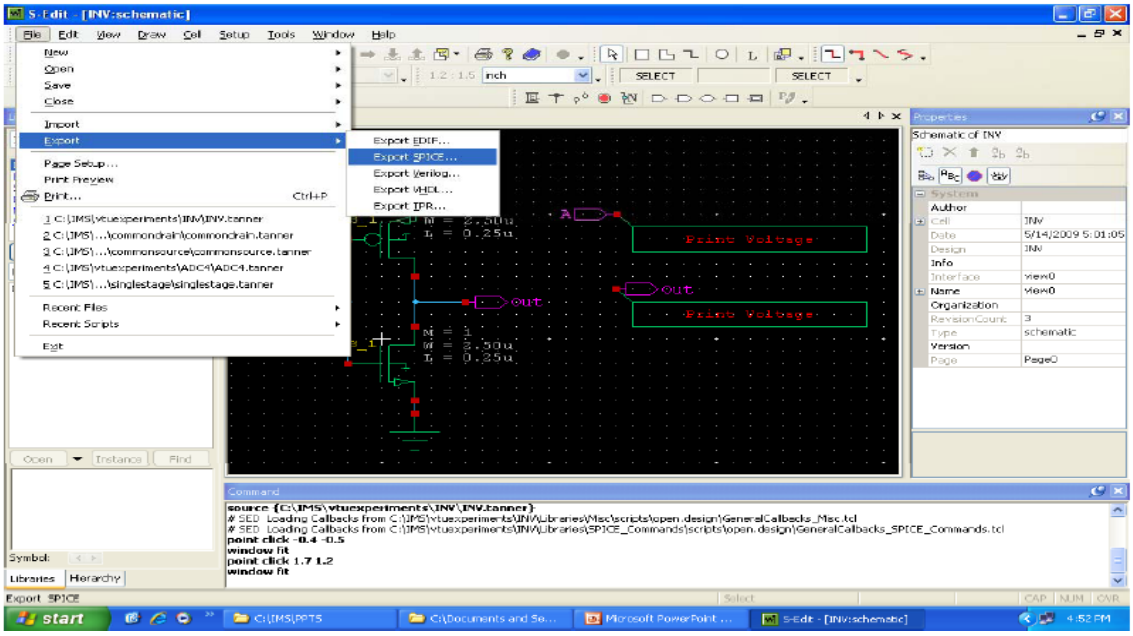

S-Edit
File Edit View Draw Cell Setup Tools Window Help
Libraries
singlestage
LogicGates
SPICE_Elements
singlestage
Devices
Misc
SPICE_Commands
Add...
Remove
Filter
singlestage
Open
Instance
Find
Symbol:
Libraries
Hierarchy
New View
Design:
singlestage
Cell:
Cell0
View type:
schematic
Interface name:
view0
View name:
view0
Disable name validation
OK
Cancel
Properties
Find
Command
mode select
source startup
design open -{C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\singlestage.tanner}
SED Loading Callbacks from C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\LIBRARIES\SPICE_Commands\scripts\open.design\Gene
SED Loading Callbacks from C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\LIBRARIES\Misc\scripts\open.design\GeneralCallbacks_I
Ready
Select
CAP
NUM
OVR

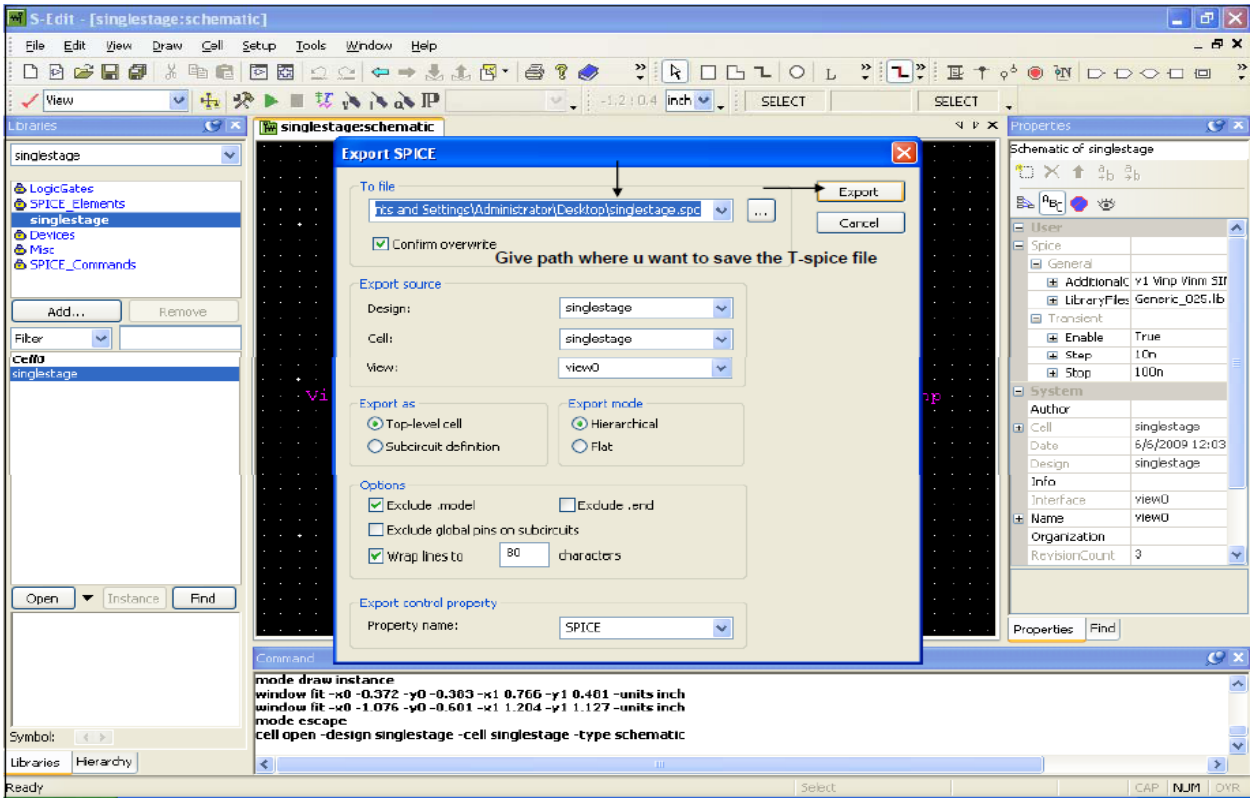

S-Edit
File Edit View Draw Cell Setup Tools Window Help
Libraries
singlestage
LogicGates
SPICE_Elements
singlestage
Devices
Misc
SPICE_Commands
Add...
Remove
Filter
Cell0
singlestage
Open
Instance
Find
Symbol:
Libraries
Hierarchy
Cell0:schematic*
Properties
Schematic of Cell0
System
Author
Cell
Cell0
Date
6/15/2009 12:53:4
Design
singlestage
Info
Interface
view0
Name
view0
Organization
RevisionCount
1
Type
schematic
Version
Page
Page0
Properties
Find
Command
SED Loading Callbacks from C:\Documents and Settings\Administrator\Desktop\vtuexperiments\singlestage\LIBRARIES\Misc\scripts\open.design\GeneralCallbacks_I
cell new -cell Cell0 -design singlestage -view view0 -type schematic -interface view0
window fit -x0 -0.34 -y0 -0.532 -x1 0.798 -y1 0.332 -units inch
window fit -x0 -0.98 -y0 -1.049 -x1 1.3 -y1 0.679 -units inch
cell open -design singlestage -cell Cell0 -type schematic
Ready
Select
CAP
NUM
OVR

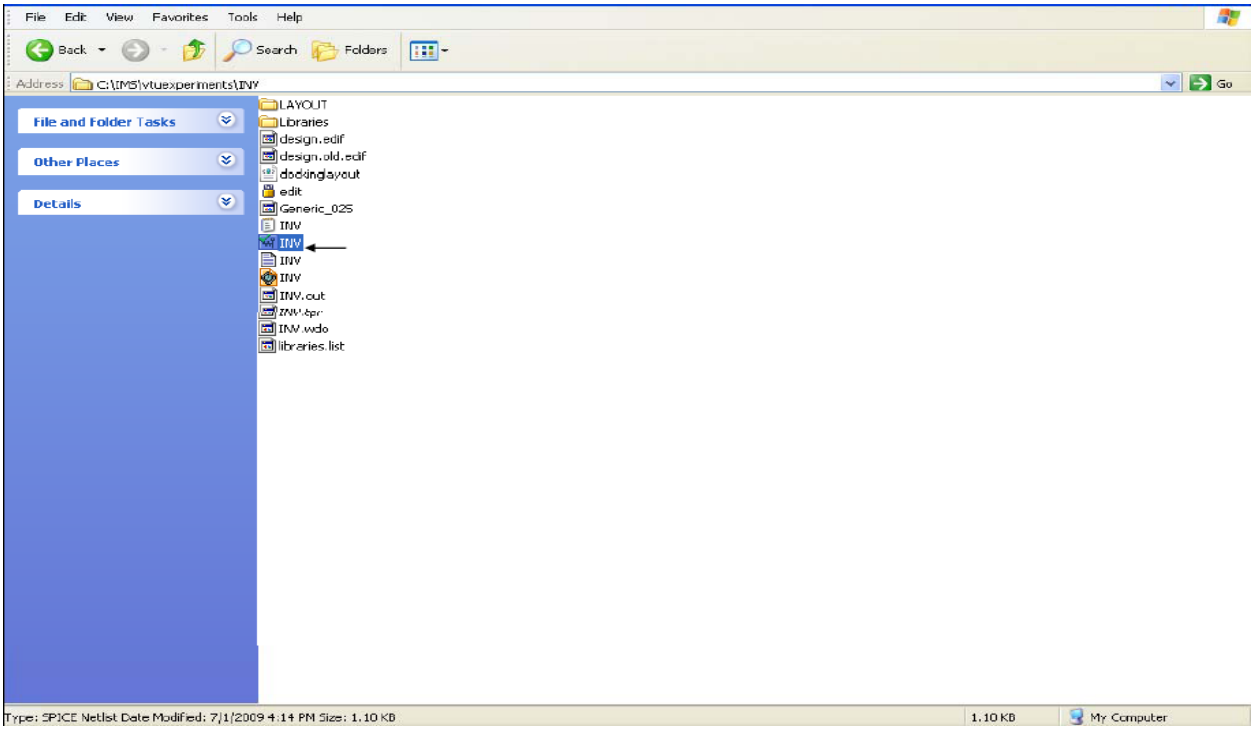

S-Edit - [Cell0:schematic*]
Instance Cell
Instance name:
NMOS_1
Disable name validation
Instance of
Design:
Devices
Cell name:
NMOS
Interface:
NMOS3
Symbol view:
NMOS3
Properties:
Done
NMOS_1
M = 1
W = 2.50u
L = 0.25u

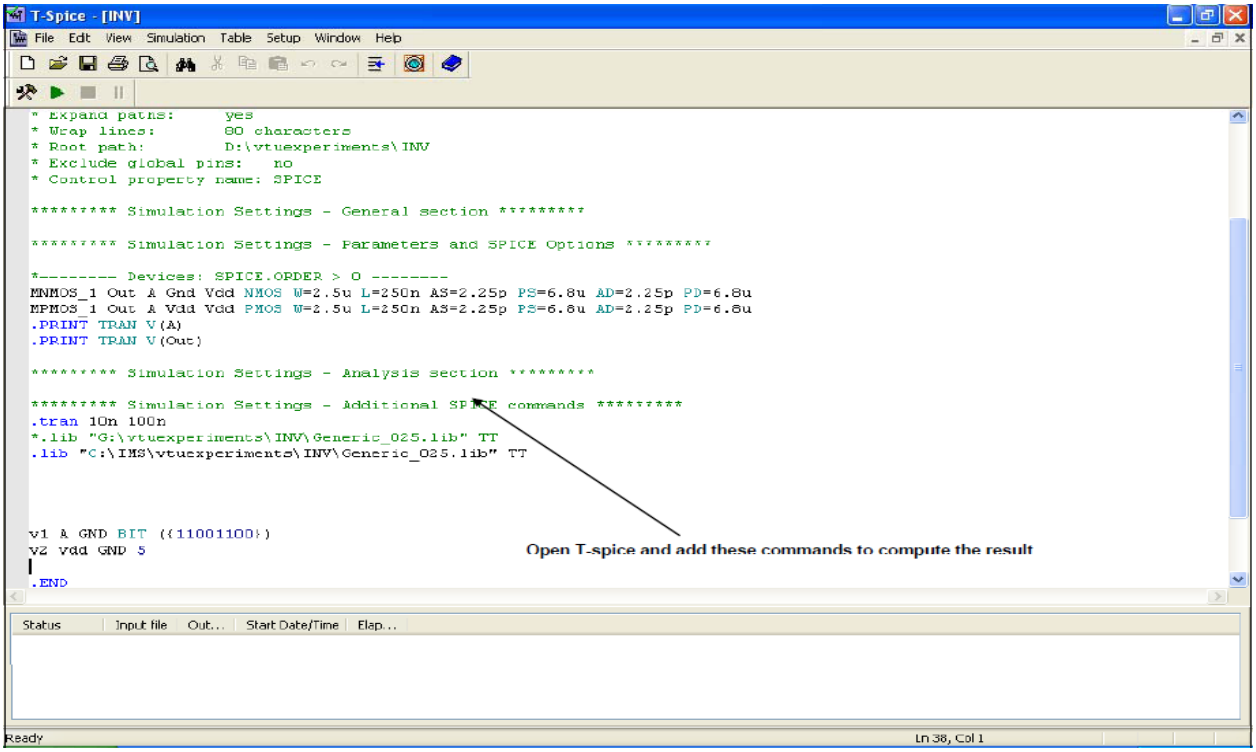

S-Edit - [INV:schematic]
INV:schematic
PMOS_1
NMOS_1
M = 1
W = 2.50u
L = 0.25u
A
Out
Print Voltage
Command
mode select
source startup

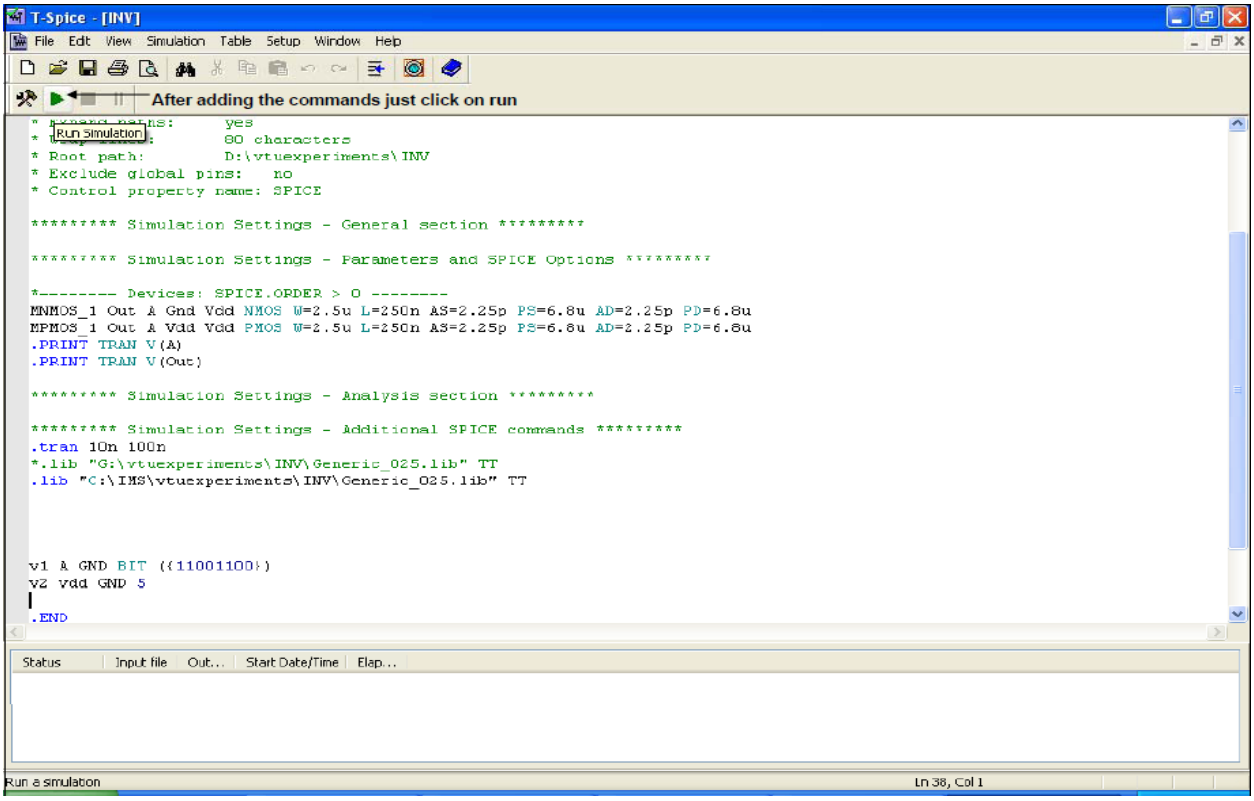
After completion of the design just export T-spice

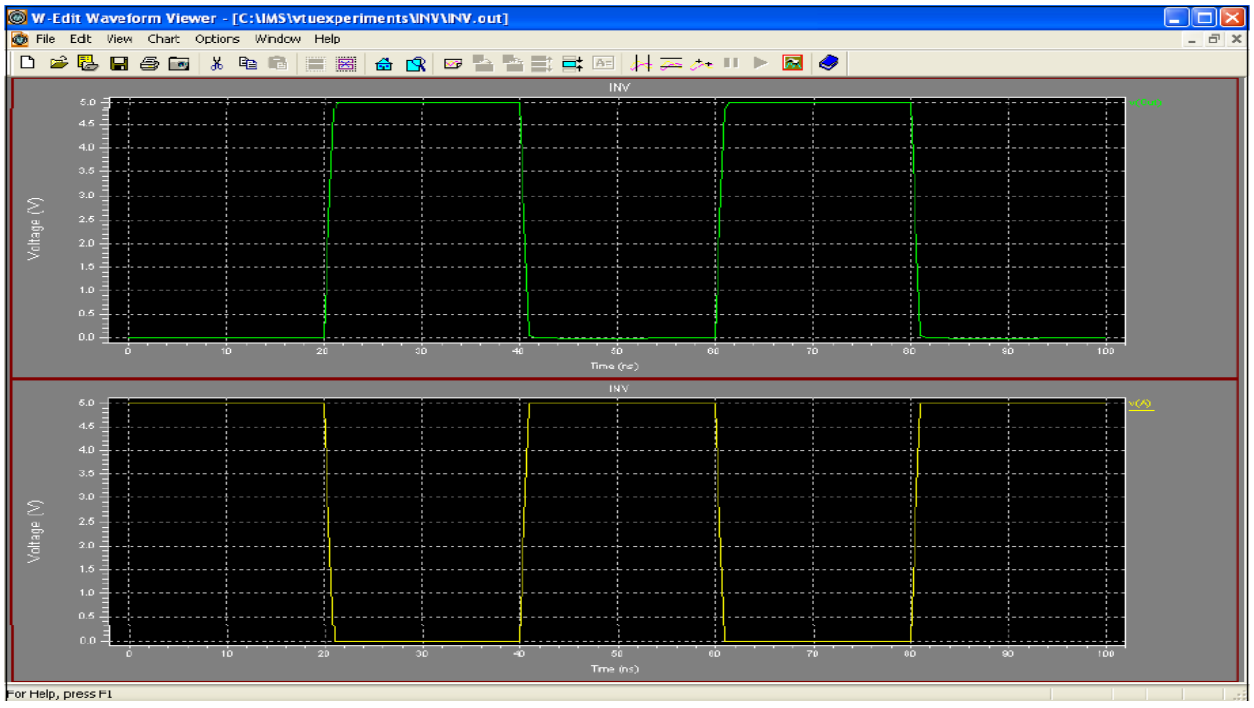





File Edit View Favorites Tools Help
Back Search Folders
Address C:\IMS\vtuexperiments\INV
Go
File and Folder Tasks
Other Places
Details
LAYOUT
Libraries
design.edif
design.old.edif
dockinglayout
edit
Generic_025
INV
INV
INV
INV
INV.out
INV.tpr
INV.wdb
libraries.list
Type: SPICE Netlist Date Modified: 7/1/2009 4:14 PM Size: 1.10 KB
1.10 KB
My Computer

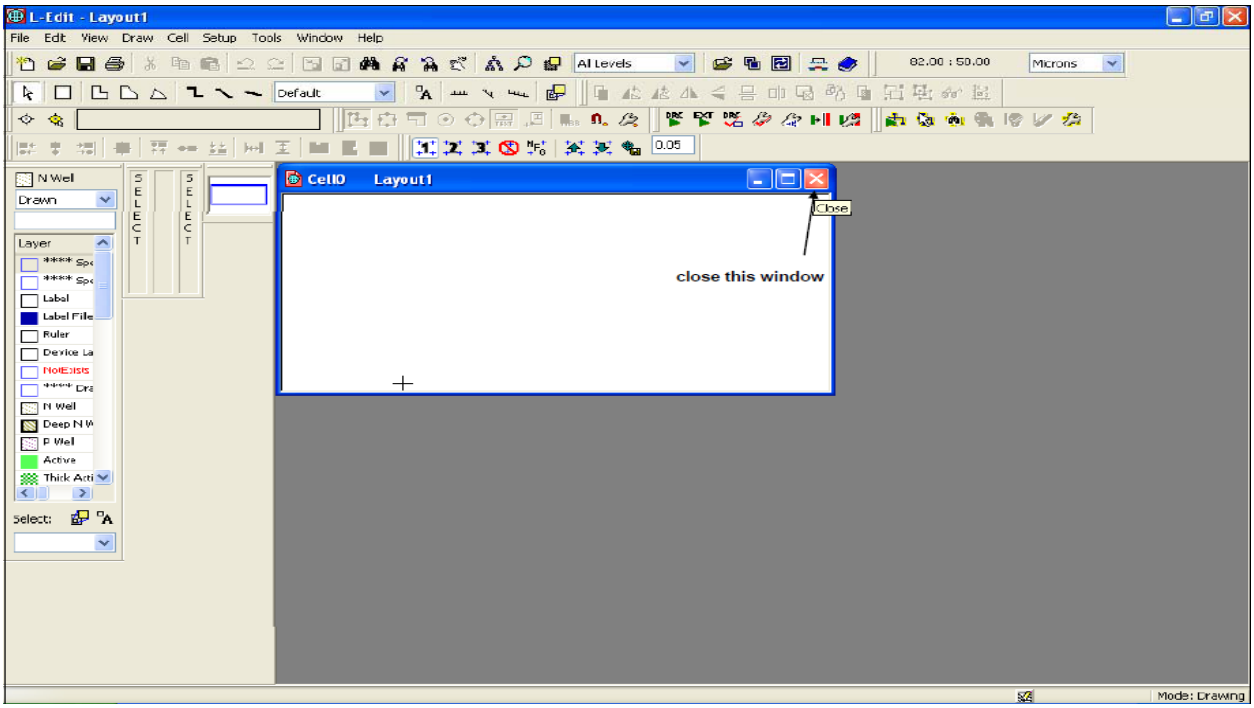

T-Spice - [INV]
File Edit View Simulation Table Setup Window Help
* Expand paths: yes
* Wrap lines: 80 characters
* Root path: D:\vtuexperiments\INV
* Exclude global pins: no
* Control property name: SPICE
********* Simulation Settings - General section *********
********* Simulation Settings - Parameters and SPICE Options *********
*-------- Devices: SPICE.ORDER > 0 --------
MNMOS_1 Out A Gnd Vdd NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_1 Out A Vdd Vdd PMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
.PRINT TRAN V(A)
.PRINT TRAN V(Out)
********* Simulation Settings - Analysis section *********
********* Simulation Settings - Additional SPICE commands *********
.tran 10n 100n
*.lib "G:\vtuexperiments\INV\Generic_025.lib" TT
.lib "C:\IMS\vtuexperiments\INV\Generic_025.lib" TT
v1 A GND BIT ({11001100})
v2 Vdd GND 5
.END
Open T-spice and add these commands to compute the result
Status Input file Out... Start Date/Time Elap...
Ready
Ln 38, Col 1

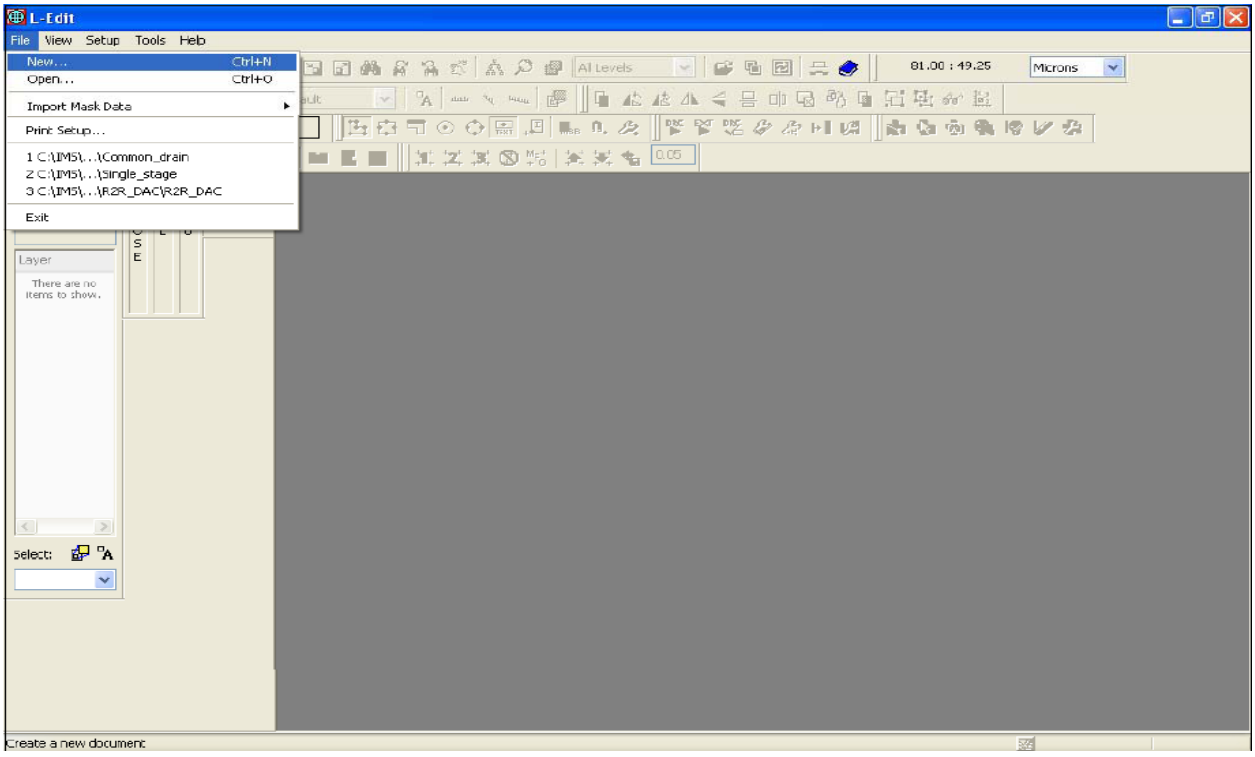

T-Spice - [INV]
File Edit View Simulation Table Setup Window Help
After adding the commands just click on run
Run Simulation
* Expand paths: yes
* Root path: D:\vtuexperiments\INV
* Exclude global pins: no
* Control property name: SPICE
********* Simulation Settings - General section *********
********** Simulation Settings - Parameters and SPICE Options **********
*-------- Devices: SPICE.ORDER > 0 --------
MNMOS_1 Out A Gnd Vdd NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_1 Out A Vdd Vdd PMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
.PRINT TRAN V(A)
.PRINT TRAN V(Out)
********** Simulation Settings - Analysis section **********
********* Simulation Settings - Additional SPICE commands *********
.tran 10n 100n
*.lib "G:\vtuexperiments\INV\Generic_025.lib" TT
.lib "C:\IMS\vtuexperiments\INV\Generic_025.lib" TT
v1 A GND BIT ({11001100})
v2 Vdd GND 5
.END
Status Input file Out... Start Date/Time Elap...
Run a simulation
Ln 38, Col 1

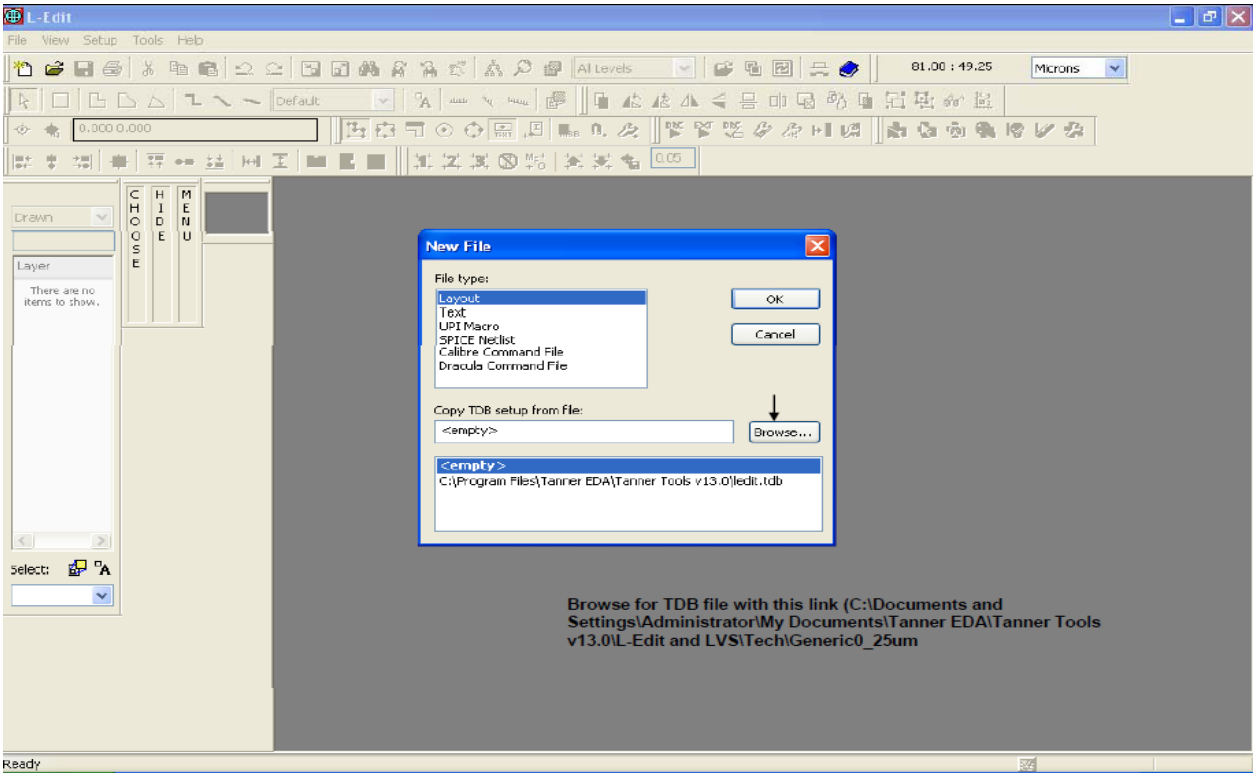

W-Edit Waveform Viewer - [C:\IMS\vtuexperiments\INV\INV.out]
File Edit View Chart Options Window Help
INV
Voltage (V)
Time (ns)
INV
Voltage (V)
Time (ns)
For Help, press F1

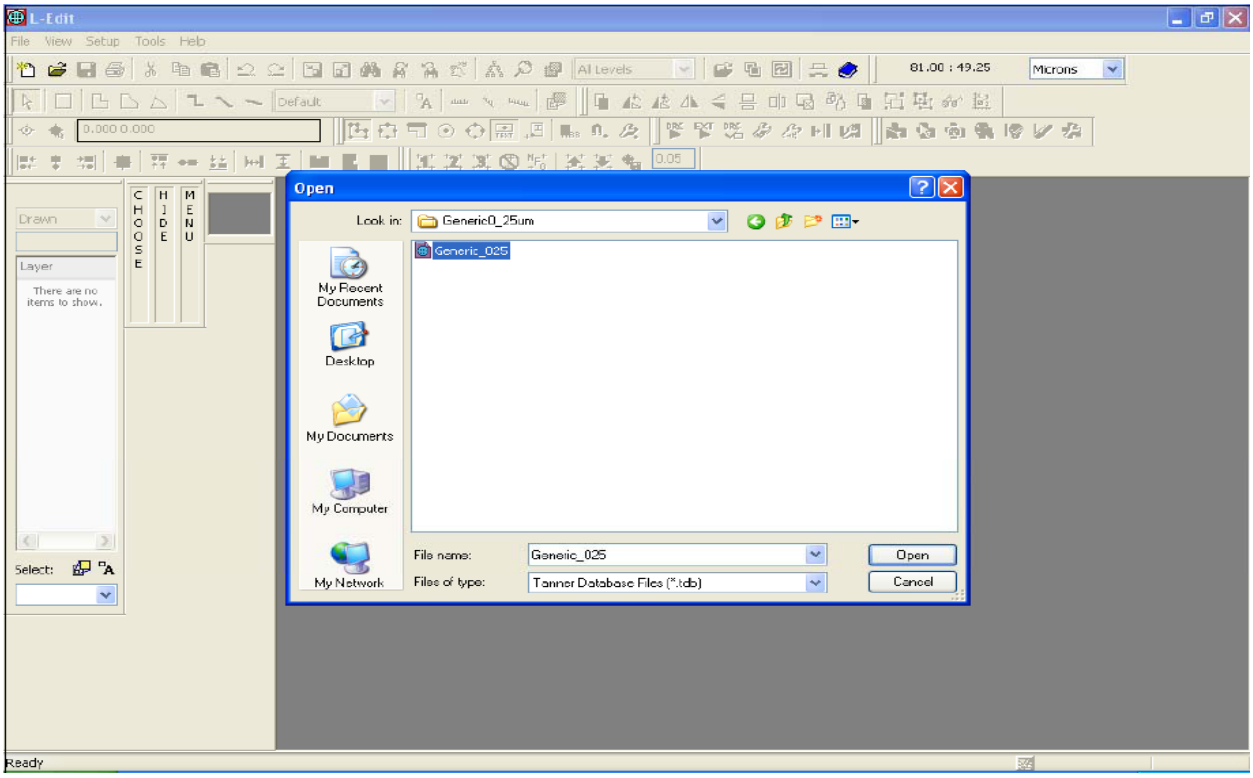
ii) Layout (L-edit):

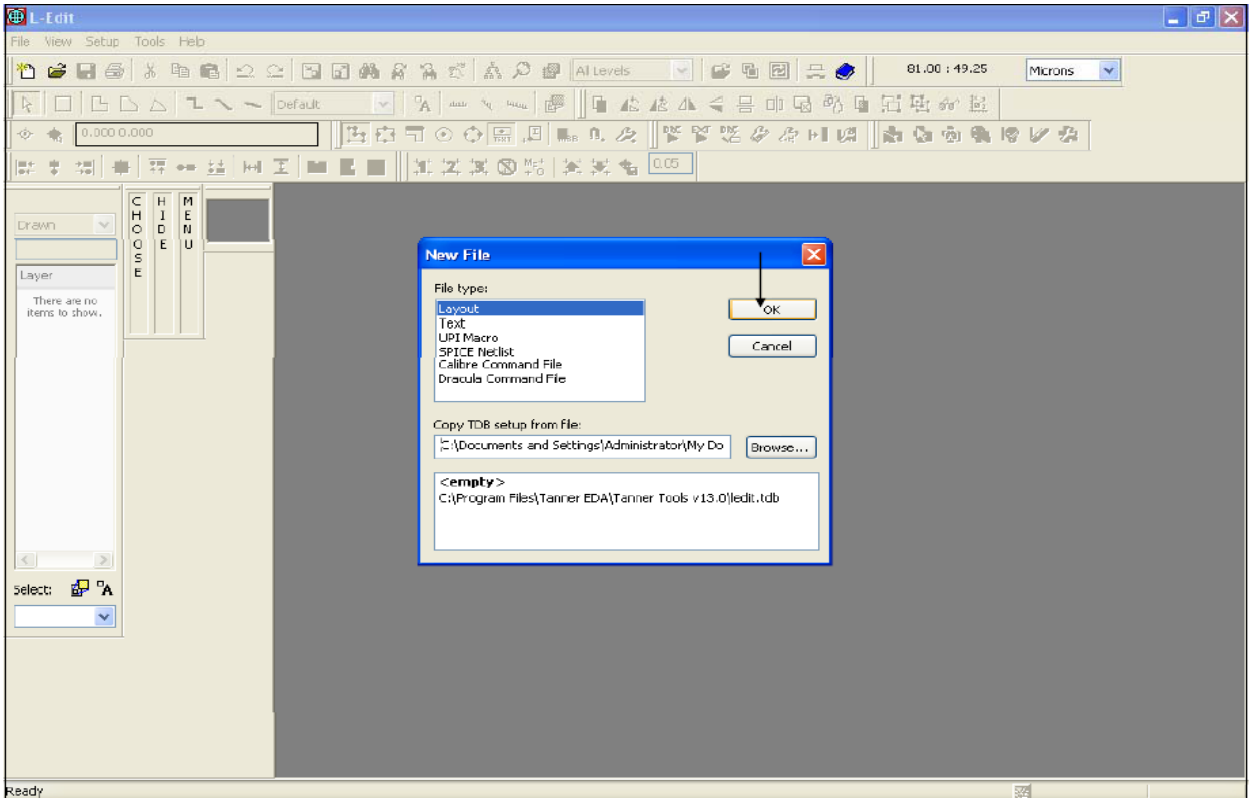
Start → Programs → tanner EDA →tanner tool v13.0 →L-edit

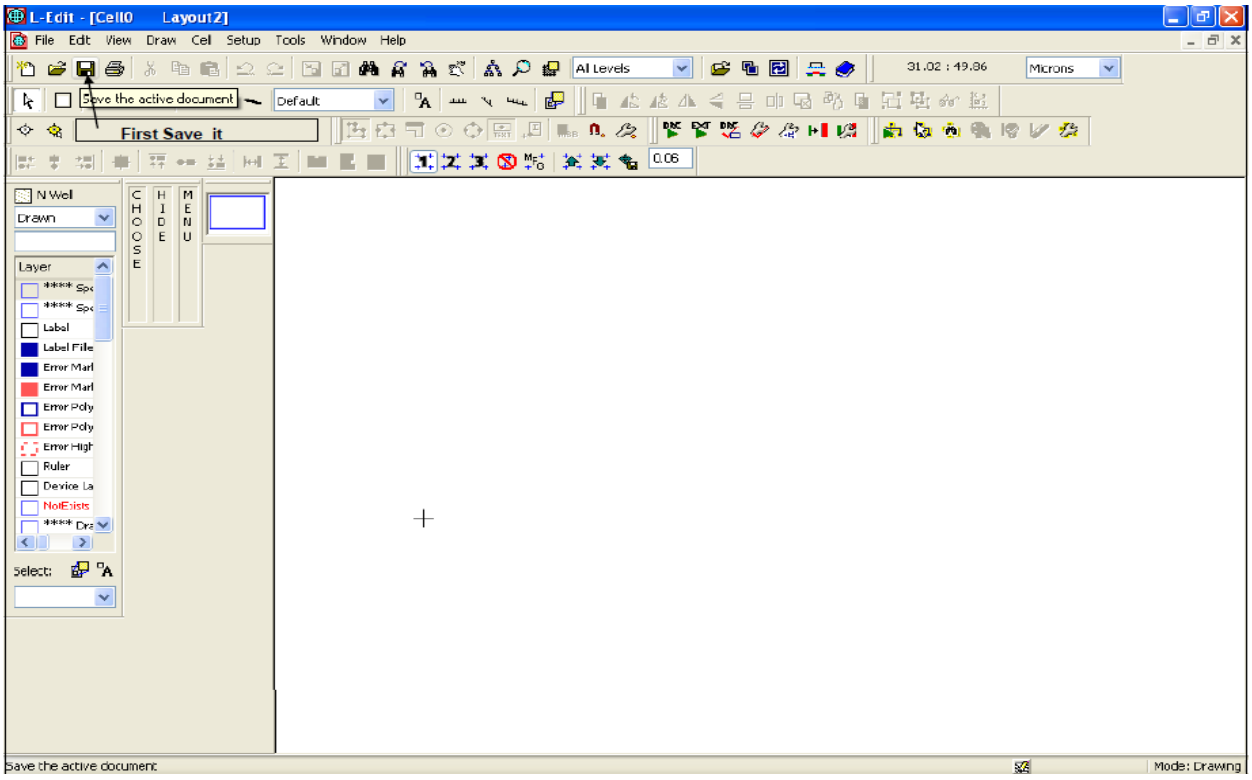


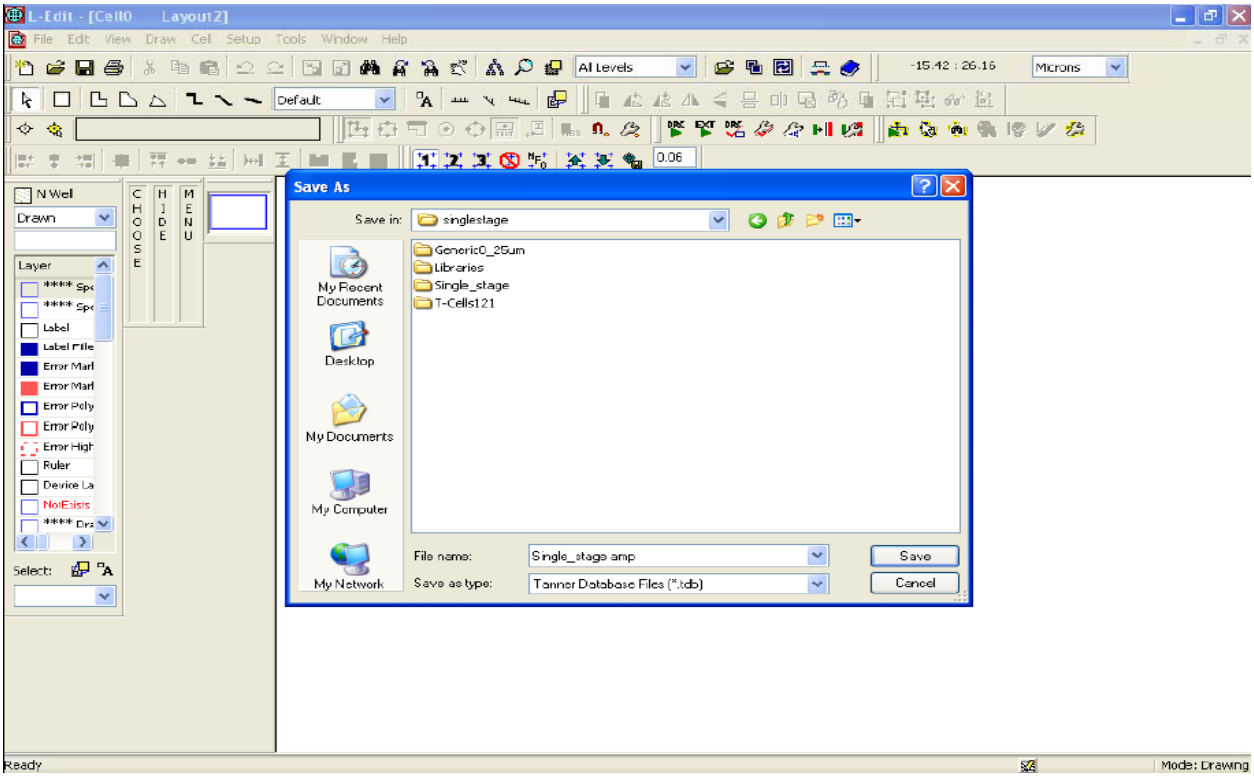


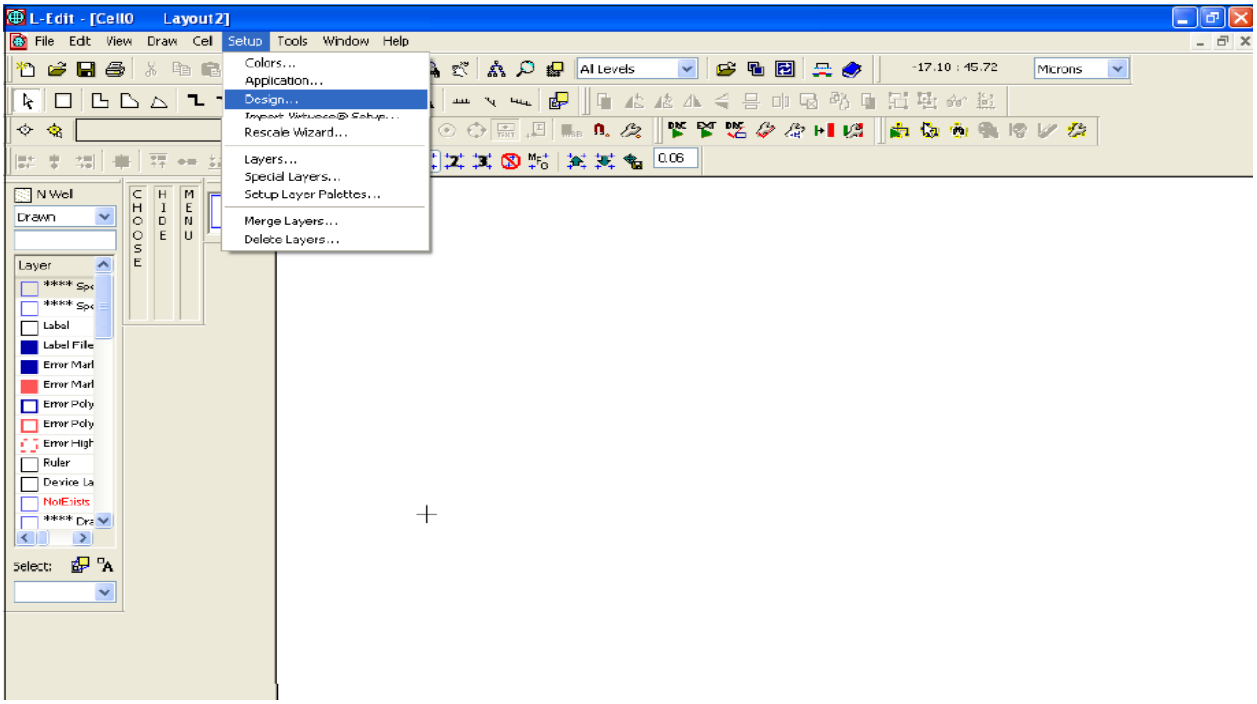

L-Edit
File View Setup Tools Help
81.00 : 49.25
Microns
New File
File type:
Layout
Text
UPI Macro
SPICE Netlist
Calibre Command File
Dracula Command File
OK
Cancel
Copy TDB setup from file:
<empty>
Browse...
<empty>
C:\Program Files\Tanner EDA\Tanner Tools v13.0\ledit.tdb
Browse for TDB file with this link (C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools v13.0\L-Edit and LVS\Tech\Generic0_25um
Ready

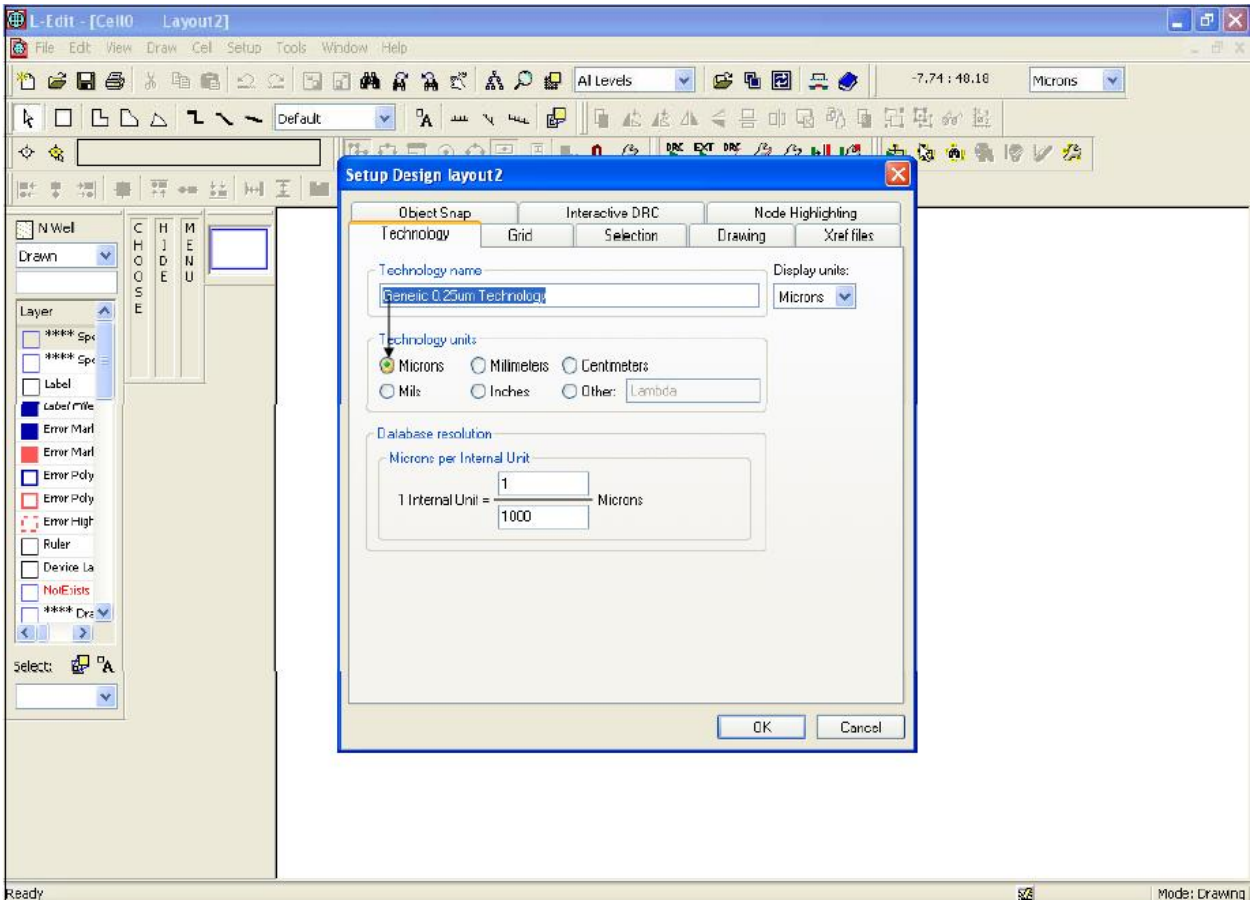

L-Edit
File View Setup Tools Help
81.00 : 49.25
Microns
Open
Look in:
Generic0_25um
Generic_025
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name:
Generic_025
Files of type:
Tanner Database Files (*.tdb)
Open
Cancel
Ready

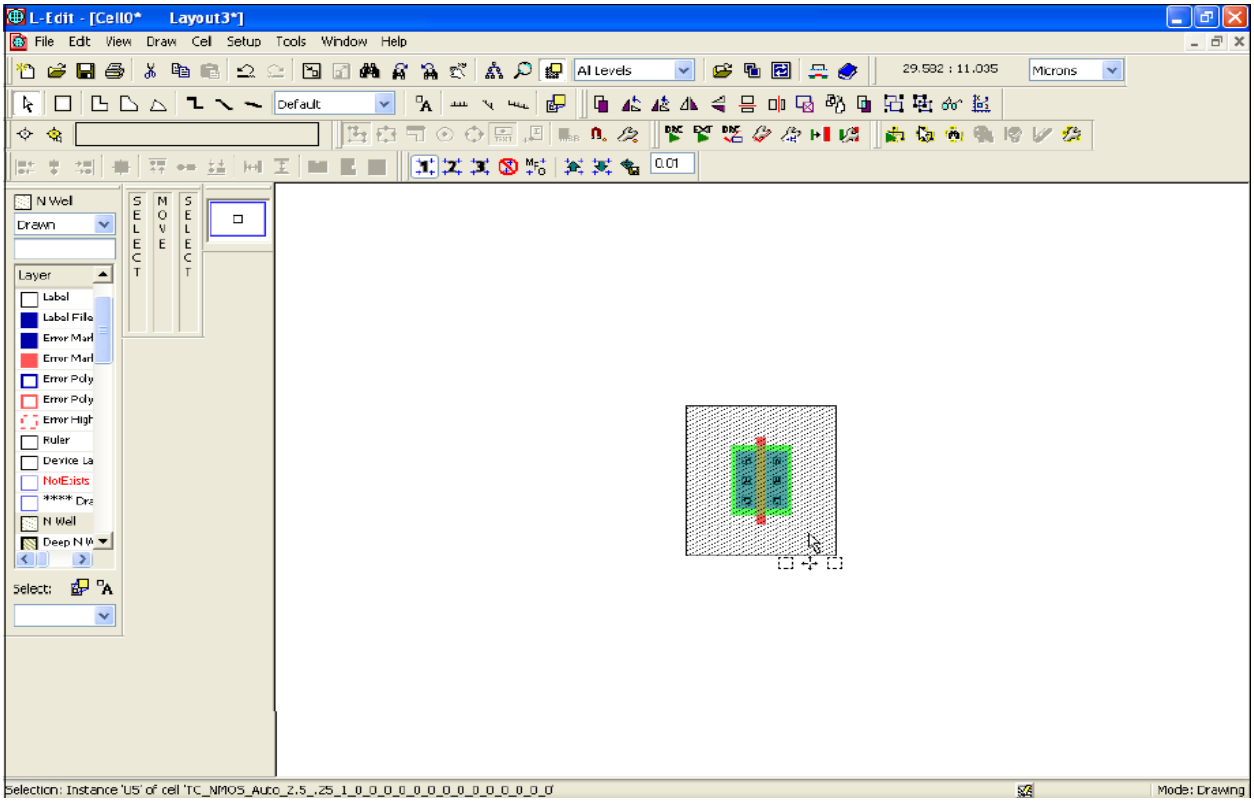

L-Edit
File View Setup Tools Help
New File
File type:
Layout
Text
UPI Macro
SPICE Netlist
Calibre Command File
Dracula Command File
OK
Cancel
Copy TDB setup from file:
C:\Documents and Settings\Administrator\My Do
Browse...
<empty>
C:\Program Files\Tanner EDA\Tanner Tools v13.0\ledit.tdb
CHOOSE
HIDE
MENU
Drawn
Layer
There are no items to show.
Select:
Ready

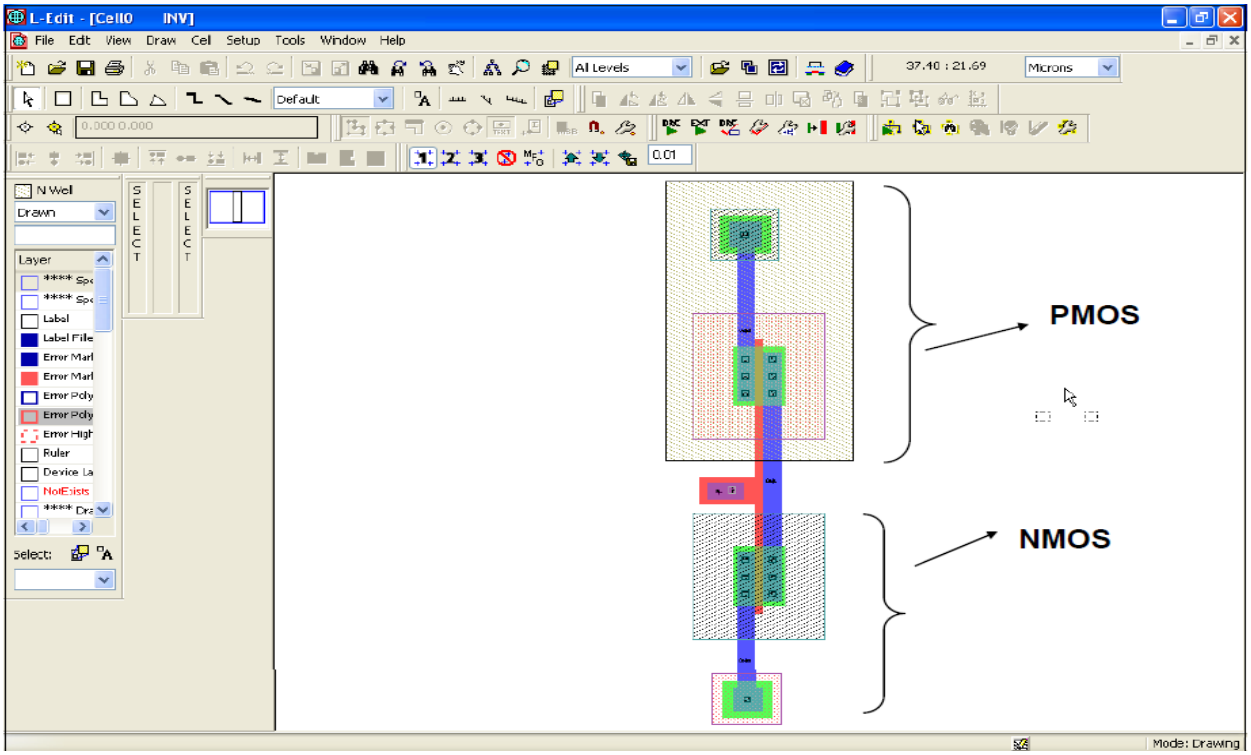

L-Edit - [Cell0 Layout2]
File Edit View Draw Cell Setup Tools Window Help
Save the active document
First Save it
Default
All Levels
Microns
N Well
Drawn
Layer
Label
Label File
Error Mark
Error Poly
Ruler
Device La
NotExists
Select:
Save the active document
Mode: Drawing

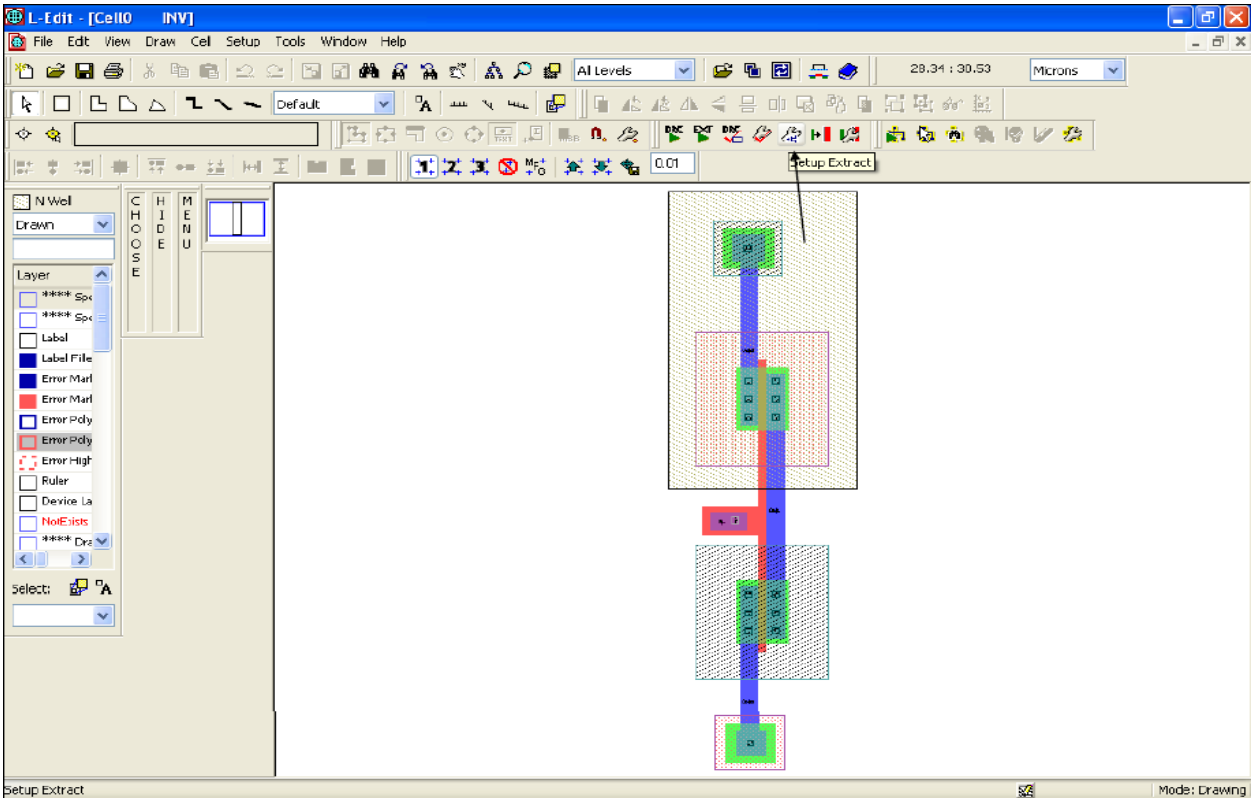

L-Edit - [Cell0 Layout2]
Save As
Save in: singlestage
Generic0_25um
Libraries
Single_stage
T-Cells121
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name: Single_stage amp
Save as type: Tanner Database Files (*.tdb)
Save
Cancel
Ready
Mode: Drawing

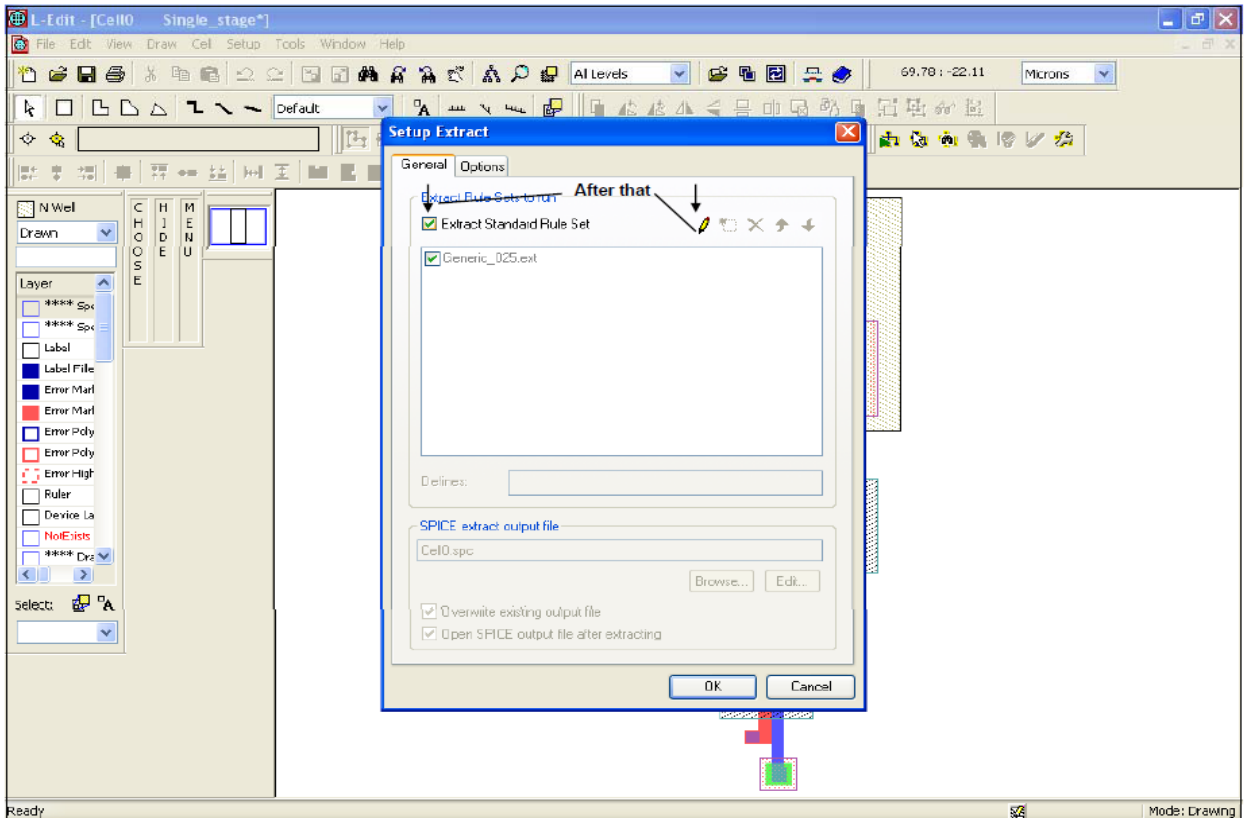

L-Edit - [Cell0 Layout2]
File Edit View Draw Cel Setup Tools Window Help
Colors...
Application...
Design...
Rescale Wizard...
Layers...
Special Layers...
Setup Layer Palettes...
Merge Layers...
Delete Layers...
Microns

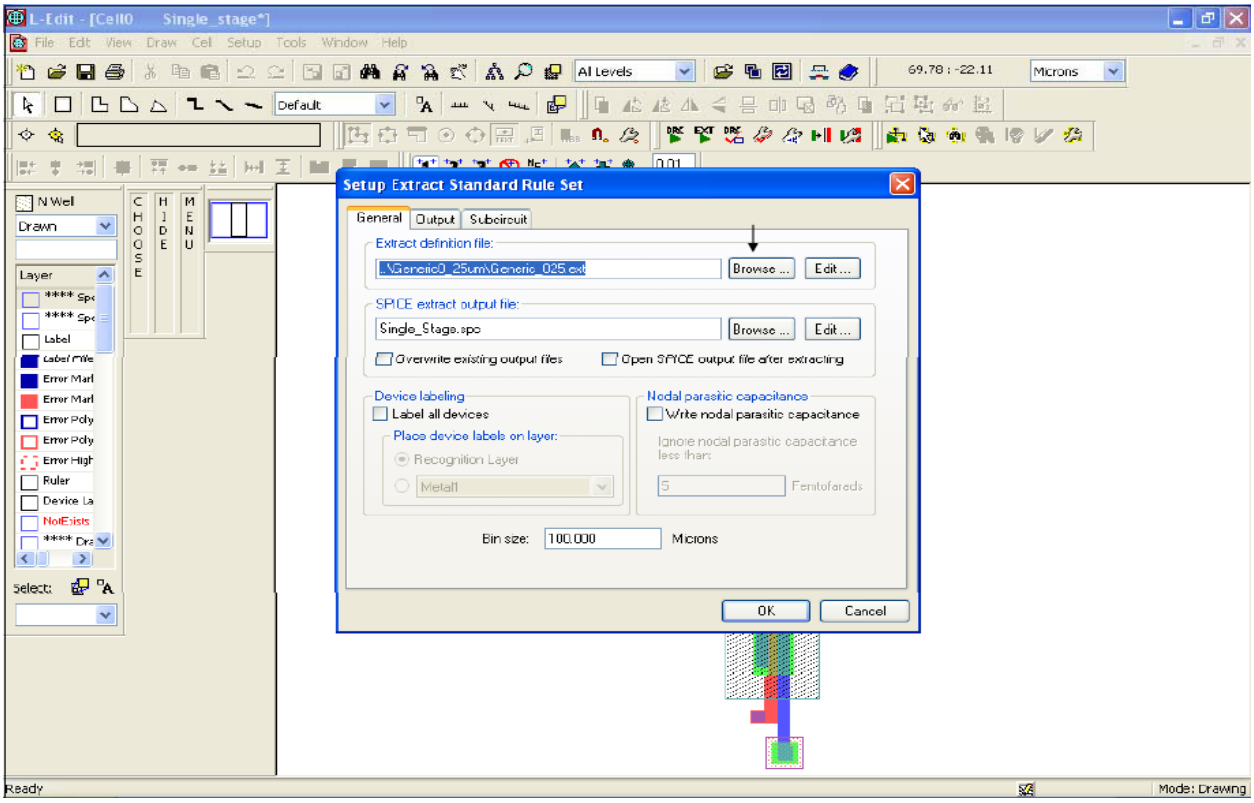

L-Edit - [Cell0 Layout2]
File Edit View Draw Cel Setup Tools Window Help
All Levels
-7.74 : 48.18
Microns
Default
Setup Design layout2
Object Snap
Interactive DRC
Node Highlighting
Technology
Grid
Selection
Drawing
Xref files
Technology name
Generic 0.25um Technolog
Display units:
Microns
Technology units
Microns
Millimeters
Centimeters
Mils
Inches
Other:
Lambda
Database resolution
Microns per Internal Unit
1 Internal Unit =
1
1000
Microns
OK
Cancel
N Well
Drawn
Layer
Ruler
Select:
Ready
Mode: Drawing

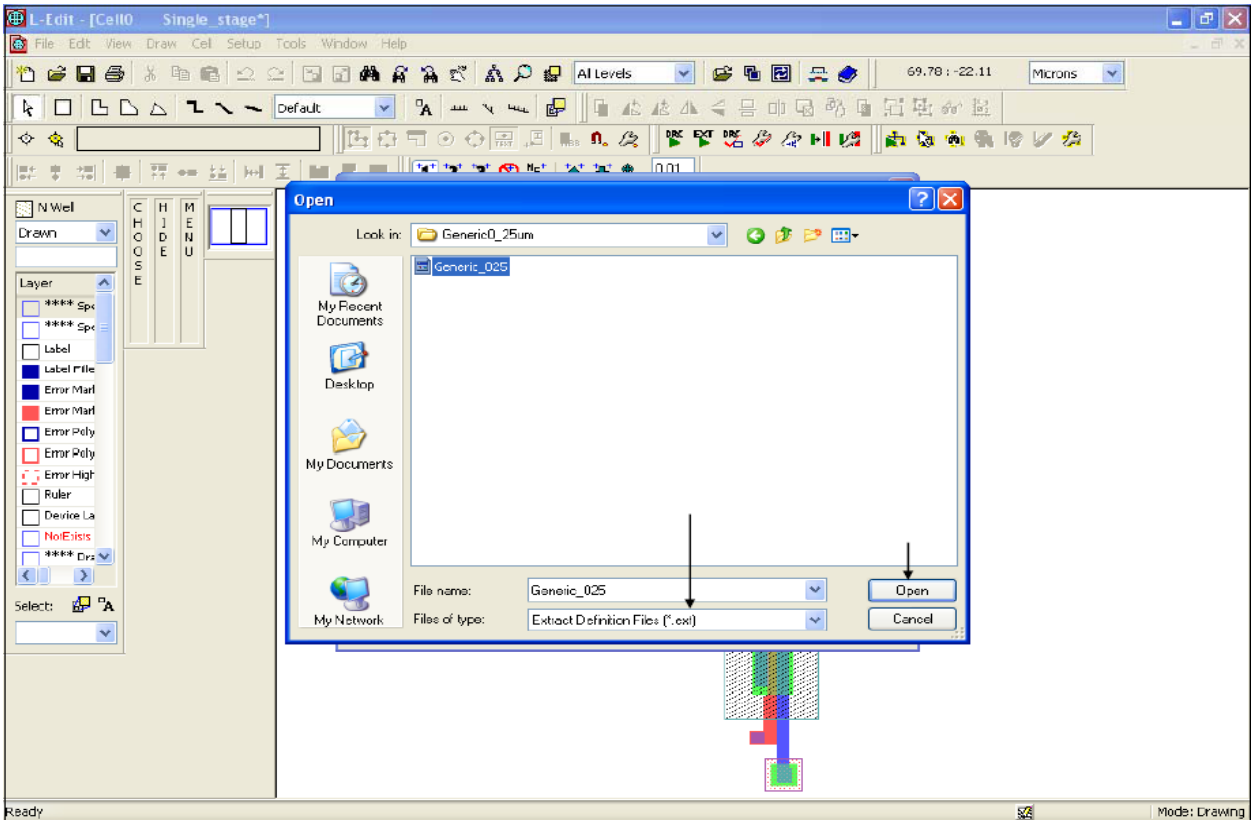

L-Edit - [Cell0* Layout3*]
File Edit View Draw Cel Setup Tools Window Help
All Levels
29.532 : 11.035
Microns
Default
0.01
N Well
Drawn
Layer
Ruler
Select:
Selection: Instance 'U5' of cell 'TC_NMOS_Auto_2.5_.25_1_0_0_0_0_0_0_0_0_0_0_0_0_0'
Mode: Drawing

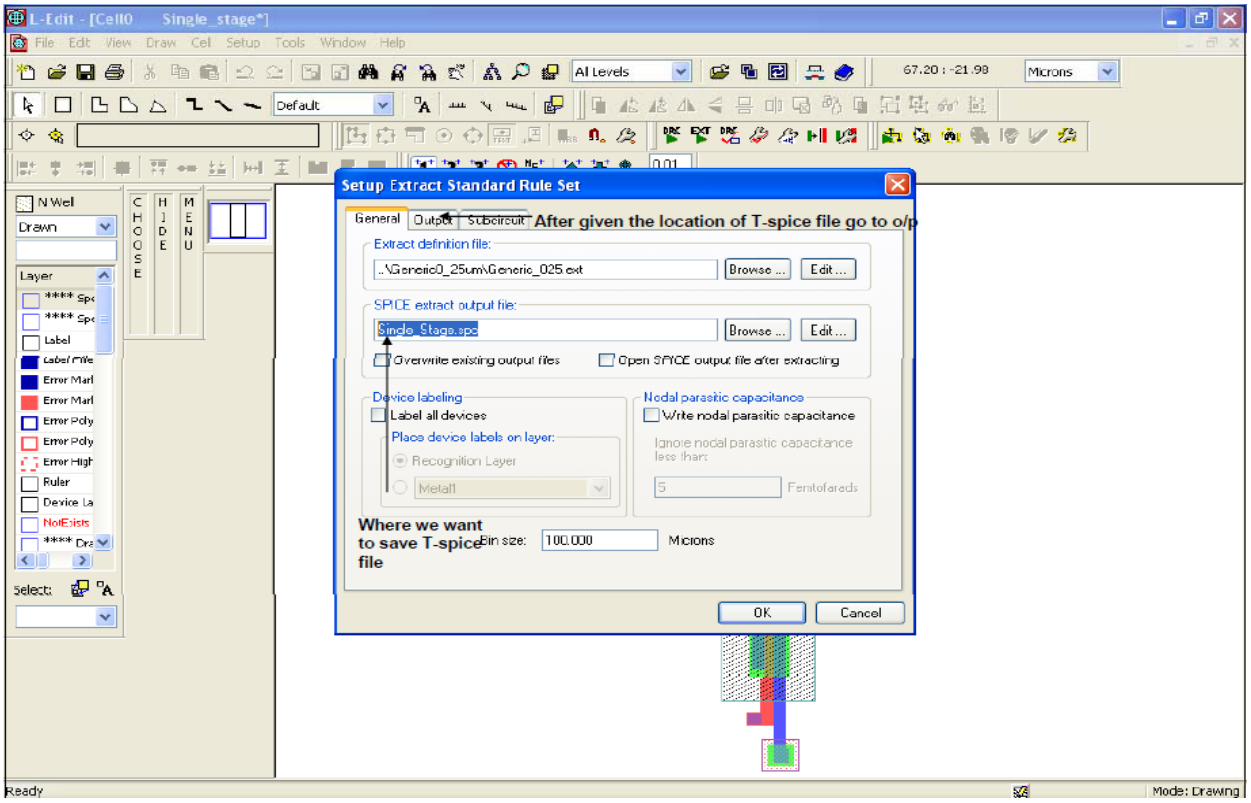

L-Edit - [Cell0 INV]
File Edit View Draw Cel Setup Tools Window Help
PMOS
NMOS
Mode: Drawing

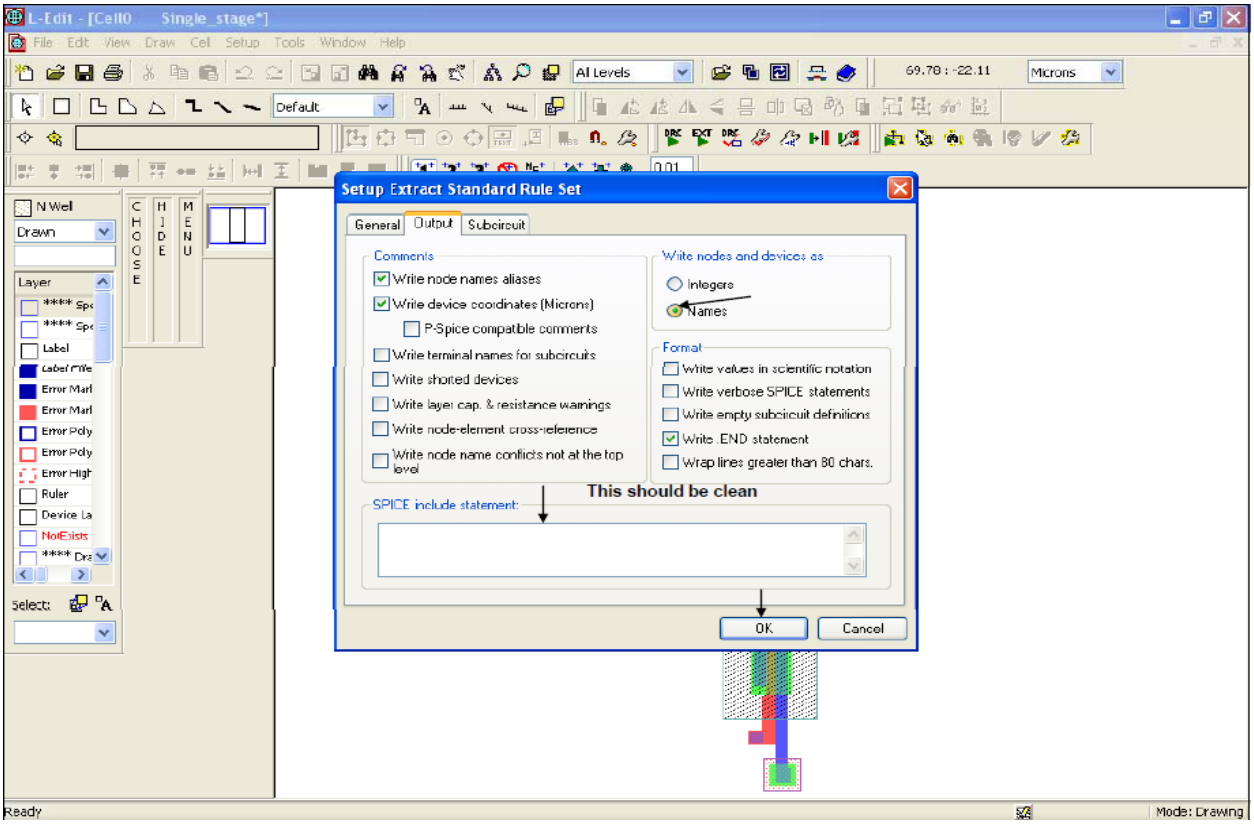

L-Edit - [Cell0 INV]
File Edit View Draw Cel Setup Tools Window Help
Setup Extract
Setup Extract
Mode: Drawing

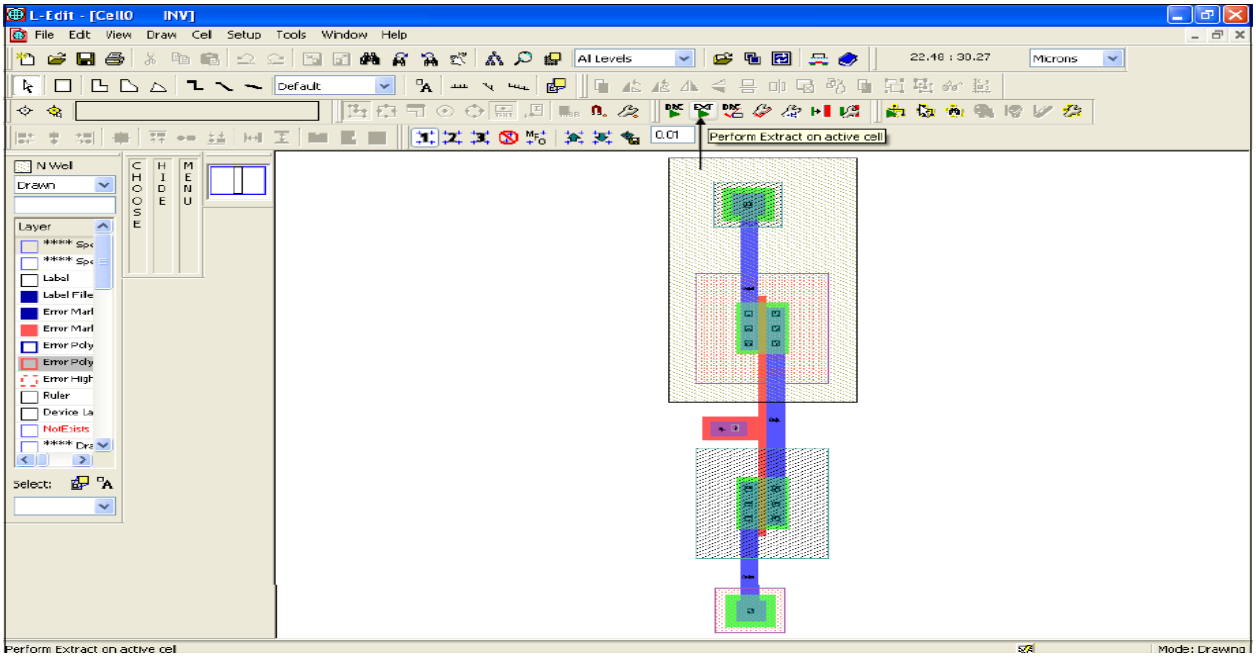

L-Edit - [Cell0 Single_stage*]
Setup Extract
General
Options
After that
Extract Standard Rule Set
Generic_025.ext
SPICE extract output file
Cell0.spc
Overwrite existing output file
Open SPICE output file after extracting
OK
Cancel
Ready
Mode: Drawing

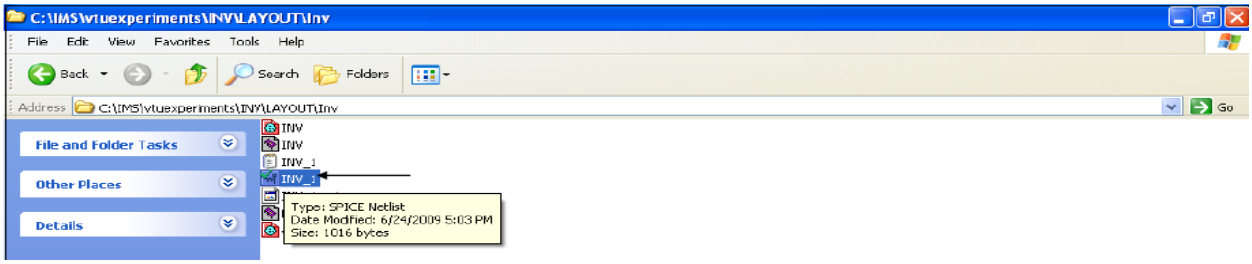

L-Edit - [Cell0 Single_stage*]
Setup Extract Standard Rule Set
General
Output
Subcircuit
Extract definition file:
Browse ...
Edit ...
SPICE extract output file:
Single_Stage.spc
Overwrite existing output files
Open SPICE output file after extracting
Device labeling
Label all devices
Nodal parasitic capacitance
Write nodal parasitic capacitance
Bin size:
100.000
Microns
OK
Cancel
Ready
Mode: Drawing

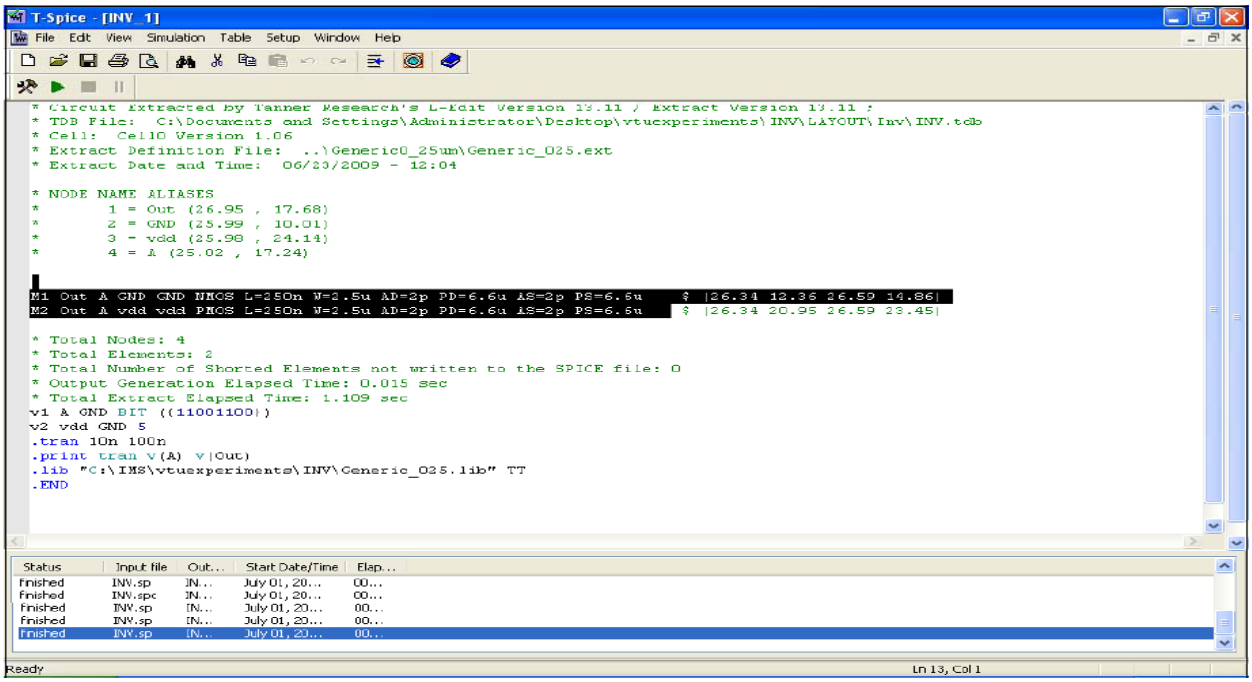

L-Edit - [Cell0 Single_stage*]
Open
Look in:
Generic0_25um
Generic_025
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name:
Generic_025
Files of type:
Extract Definition Files (*.ext)
Open
Cancel
Ready
Mode: Drawing

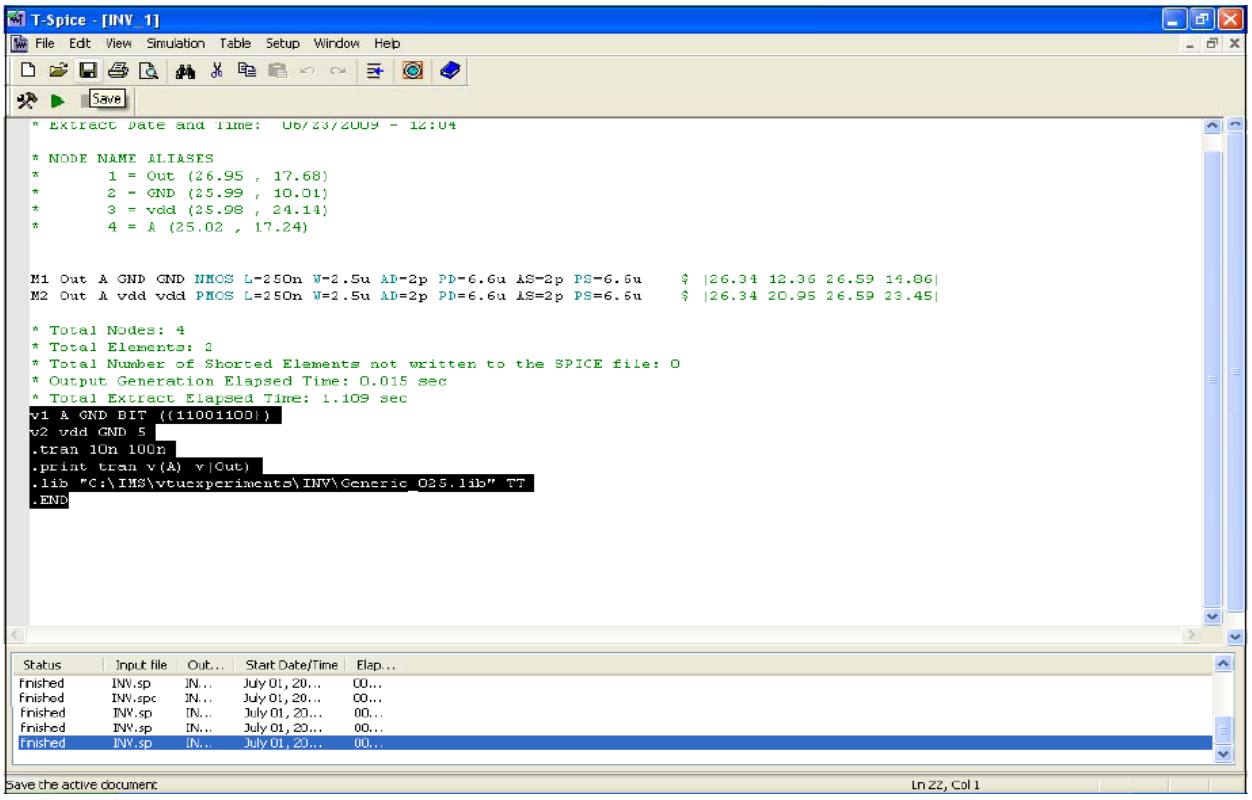

L-Edit - [Cell0 Single_stage*]
Setup Extract Standard Rule Set
General
Output
Subcircuit
After given the location of T-spice file go to o/p
Extract definition file:
..\Generic0_25um\Generic_025.ext
Browse ...
Edit ...
SPICE extract output file:
Single_Stage.spc
Overwrite existing output files
Open SPICE output file after extracting
Device labeling
Label all devices
Place device labels on layer:
Recognition Layer
Metal1
Nodal parasitic capacitance
Write nodal parasitic capacitance
Ignore nodal parasitic capacitance less than:
5
Femtofarads
Where we want to save T-spice file
Bin size:
100.000
Microns
OK
Cancel
Ready
Mode: Drawing

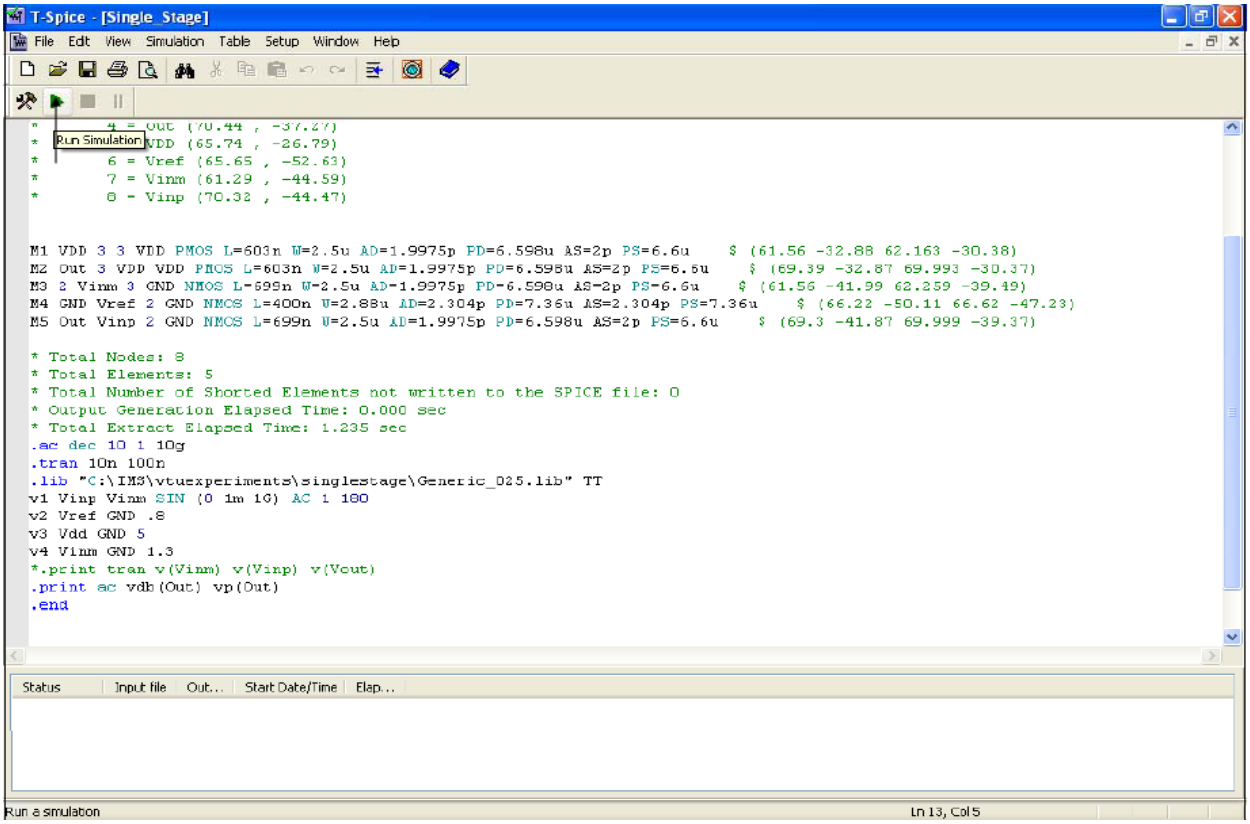

L-Edit - [Cell0 Single_stage*]
Setup Extract Standard Rule Set
General
Output
Subcircuit
Comments
Write node names aliases
Write device coordinates (Microns)
P-Spice compatible comments
Write terminal names for subcircuits
Write shorted devices
Write layer cap. & resistance warnings
Write node-element cross-reference
Write node name conflicts not at the top level
Write nodes and devices as
Integers
Names
Format
Write values in scientific notation
Write verbose SPICE statements
Write empty subcircuit definitions
Write .END statement
Wrap lines greater than 80 chars.
This should be clean
SPICE include statement:
OK
Cancel
Ready
Mode: Drawing

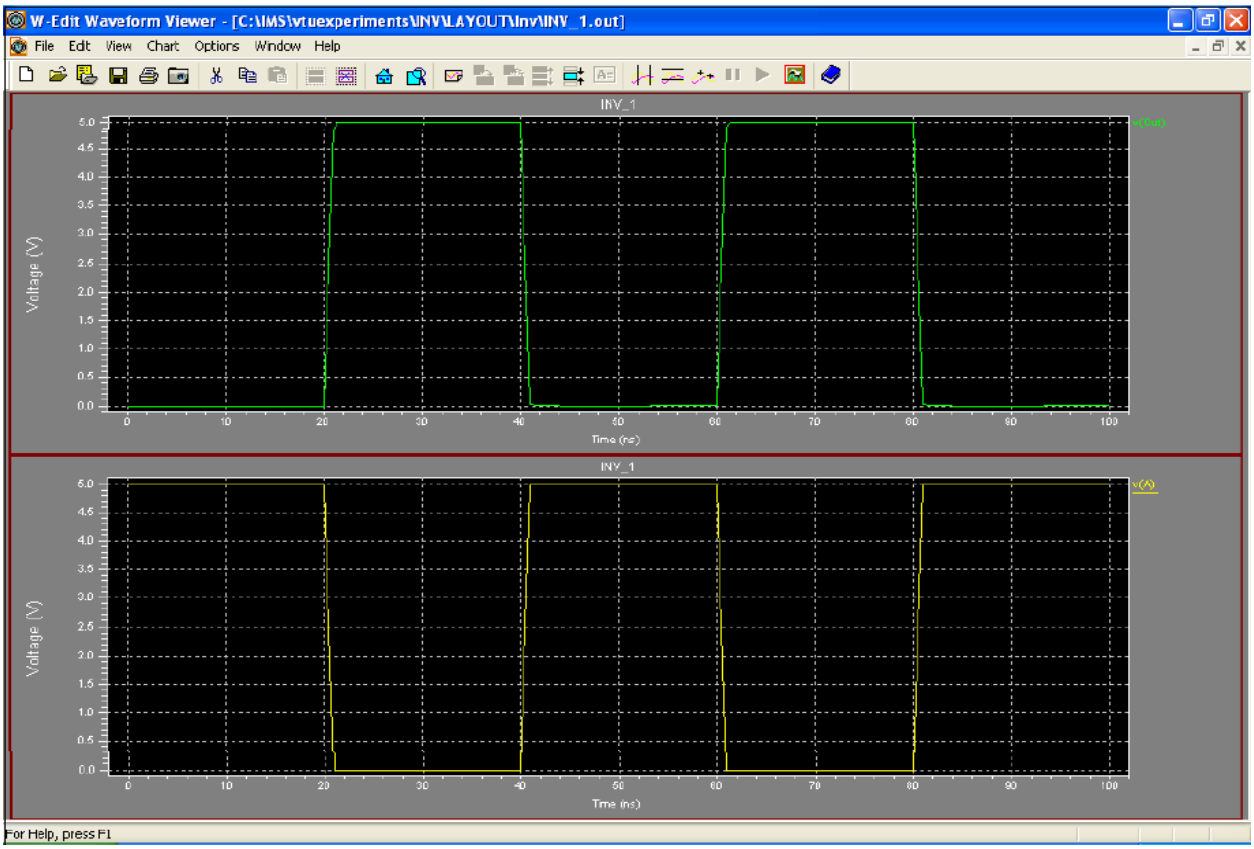

L-Edit - [Cell0 INV]
Perform Extract on active cell
Perform Extract on active cell
Mode: Drawing


C:\IMS\vtuexperiments\INV\LAYOUT\Inv
File and Folder Tasks
Other Places
Details
INV
INV
INV_1
INV_1
Type: SPICE Netlist
Date Modified: 6/24/2009 5:03 PM
Size: 1016 bytes


T-Spice - [INV_1]
File Edit View Simulation Table Setup Window Help
* Circuit Extracted by Tanner Research's L-Edit Version 13.11 / Extract Version 13.11 :
* TDB File: C:\Documents and Settings\Administrator\Desktop\vtuexperiments\INV\LAYOUT\Inv\INV.tdb
* Cell: Cell0 Version 1.06
* Extract Definition File: ..\Generic0_25um\Generic_025.ext
* Extract Date and Time: 06/23/2009 - 12:04
* NODE NAME ALIASES
* 1 = Out (26.95 , 17.68)
* 2 = GND (25.99 , 10.01)
* 3 = vdd (25.98 , 24.14)
* 4 = A (25.02 , 17.24)
M1 Out A GND GND NMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u $ |26.34 12.36 26.59 14.86|
M2 Out A vdd vdd PMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u $ |26.34 20.95 26.59 23.45|
* Total Nodes: 4
* Total Elements: 2
* Total Number of Shorted Elements not written to the SPICE file: 0
* Output Generation Elapsed Time: 0.015 sec
* Total Extract Elapsed Time: 1.109 sec
v1 A GND BIT ((11001100))
v2 vdd GND 5
.tran 10n 100n
.print tran v(A) v|Out)
.lib "C:\IMS\vtuexperiments\INV\Generic_025.lib" TT
.END
Status Input file Out... Start Date/Time Elap...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.spc IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Ready
Ln 13, Col 1


T-Spice - [INV_1]
File Edit View Simulation Table Setup Window Help
Save
* Extract Date and Time: 06/23/2009 - 12:04
* NODE NAME ALIASES
* 1 = Out (26.95 , 17.68)
* 2 = GND (25.99 , 10.01)
* 3 = vdd (25.98 , 24.14)
* 4 = A (25.02 , 17.24)
M1 Out A GND GND NMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u $ |26.34 12.36 26.59 14.86|
M2 Out A vdd vdd PMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u $ |26.34 20.95 26.59 23.45|
* Total Nodes: 4
* Total Elements: 2
* Total Number of Shorted Elements not written to the SPICE file: 0
* Output Generation Elapsed Time: 0.015 sec
* Total Extract Elapsed Time: 1.109 sec
v1 A GND BIT ((11001100))
v2 vdd GND 5
.tran 10n 100n
.print tran v(A) v|Out)
.lib "C:\IMS\vtuexperiments\INV\Generic_025.lib" TT
.END
Status Input file Out... Start Date/Time Elap...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.spc IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Finished INV.sp IN... July 01, 20... 00...
Save the active document
Ln 22, Col 1


T-Spice - [Single_Stage]
File Edit View Simulation Table Setup Window Help
Run Simulation
* 4 = Out (70.44 , -37.27)
* 5 = VDD (65.74 , -26.79)
* 6 = Vref (65.65 , -52.63)
* 7 = Vinm (61.29 , -44.59)
* 8 = Vinp (70.32 , -44.47)
M1 VDD 3 3 VDD PMOS L=603n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u $ (61.56 -32.88 62.163 -30.38)
M2 Out 3 VDD VDD PMOS L=603n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u $ (69.39 -32.87 69.993 -30.37)
M3 2 Vinm 3 GND NMOS L=699n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u $ (61.56 -41.99 62.259 -39.49)
M4 GND Vref 2 GND NMOS L=400n W=2.88u AD=2.304p PD=7.36u AS=2.304p PS=7.36u $ (66.22 -50.11 66.62 -47.23)
M5 Out Vinp 2 GND NMOS L=699n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u $ (69.3 -41.87 69.999 -39.37)
* Total Nodes: 8
* Total Elements: 5
* Total Number of Shorted Elements not written to the SPICE file: 0
* Output Generation Elapsed Time: 0.000 sec
* Total Extract Elapsed Time: 1.235 sec
.ac dec 10 1 10g
.tran 10n 100n
.lib "C:\IMS\vtuexperiments\singlestage\Generic_025.lib" TT
v1 Vinp Vinm SIN (0 1m 1G) AC 1 180
v2 Vref GND .8
v3 Vdd GND 5
v4 Vinm GND 1.3
*.print tran v(Vinm) v(Vinp) v(Vout)
.print ac vdb(Out) vp(Out)
.end
Status Input file Out... Start Date/Time Elap...
Run a simulation
Ln 13, Col 5


W-Edit Waveform Viewer - [C:\IMS\vtuexperiments\INV\LAYOUT\Inv\INV_1.out]
File Edit View Chart Options Window Help
INV_1
Voltage (V)
Time (ns)
INV_1
Voltage (V)
Time (ns)
For Help, press F1

EXP.NO:	DIFFERENTIAL AMPLIFER
DATE:	

AIM:

To calculate the gain, bandwidth and CMRR of a differential amplifier through schematic entry using Tanner EDA tool.

FACILITIES REQUIRED AND PROCEDURE

a) Facilities required to do the experiment

S.No.	SOFTWARE REQUIREMENTS	Quantity
1	S-Edit, W-Edit, T-Edit using Tanner Tool.	1

b) Procedure for doing the experiment

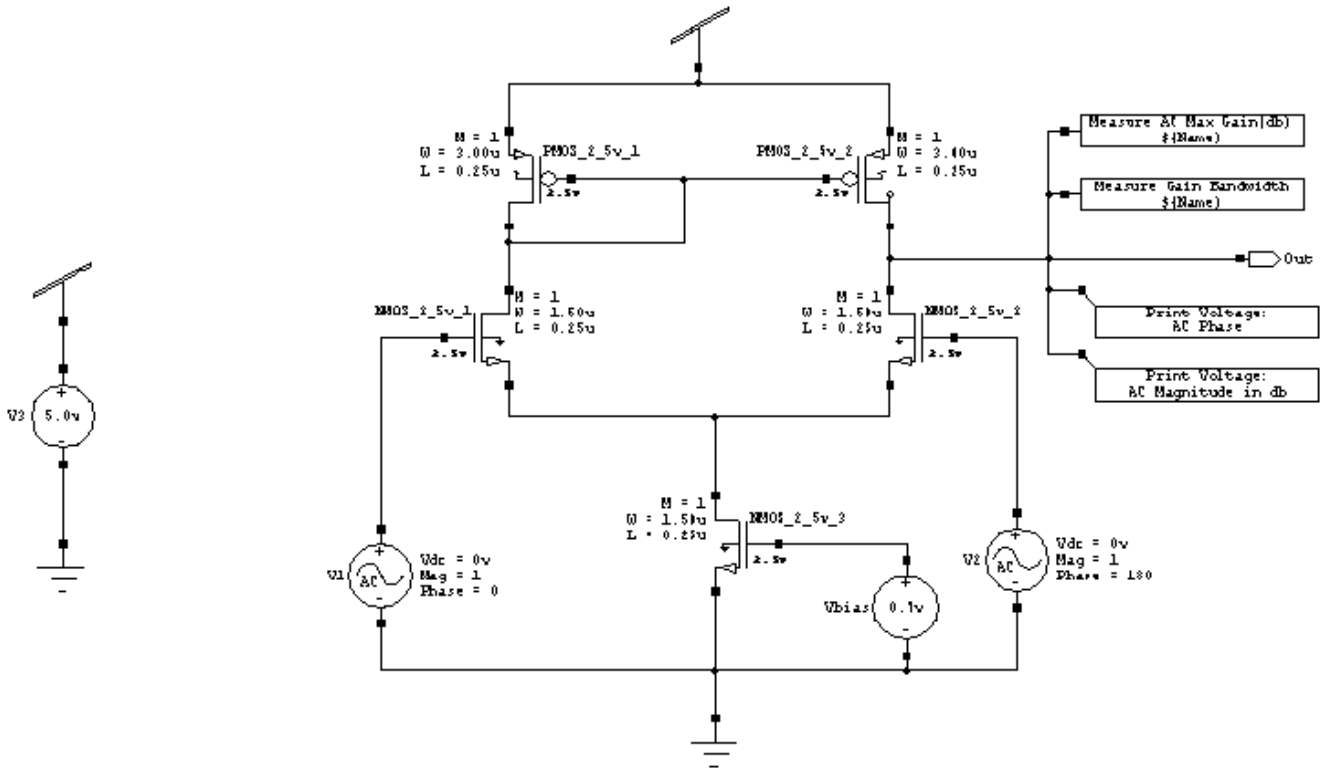
S.No	Details of the step
1	Draw the schematic of differential amplifier using S-edit and generate the Symbol.
2	Draw the schematic of differential amplifier circuit using the generated Symbol.
3	Perform AC Analysis of the differential amplifier.
4	Obtain the frequency response from W-edit.
5	Obtain the spice code using T-edit.

PROCEDURE:

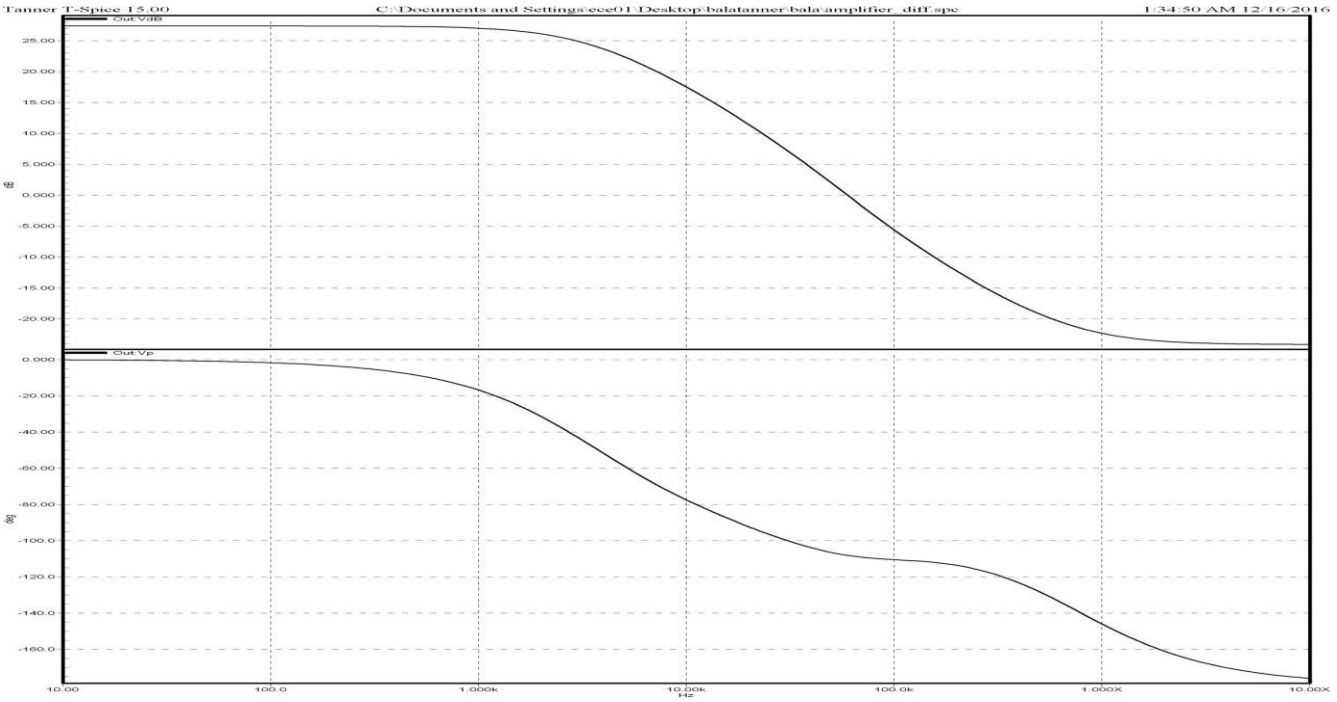
- Enter the schematic of differential amplifier using S-Edit.
- Perform AC Analysis of the differential amplifier.
- Go to ‘setup’ in that select ‘spice simulation’. Choose ‘ac analysis’ and give the following values.
- Set ‘Start frequency =10’,’ Stop frequency=10meg’, ‘No. of frequency=25’, ‘Sweep type = dec’. Click on ‘general’ type and give path to Generic_250nm.lib.Then Click OK.
- RUN Simulation to get output.
- Obtain the frequency response from W-Edit.
- Obtain the spice code using T-Edit.

SCHEMATICDIAGRAM:

DIFFERENTIAL MODE:



DIFFERENTIAL MODE OUTPUT:

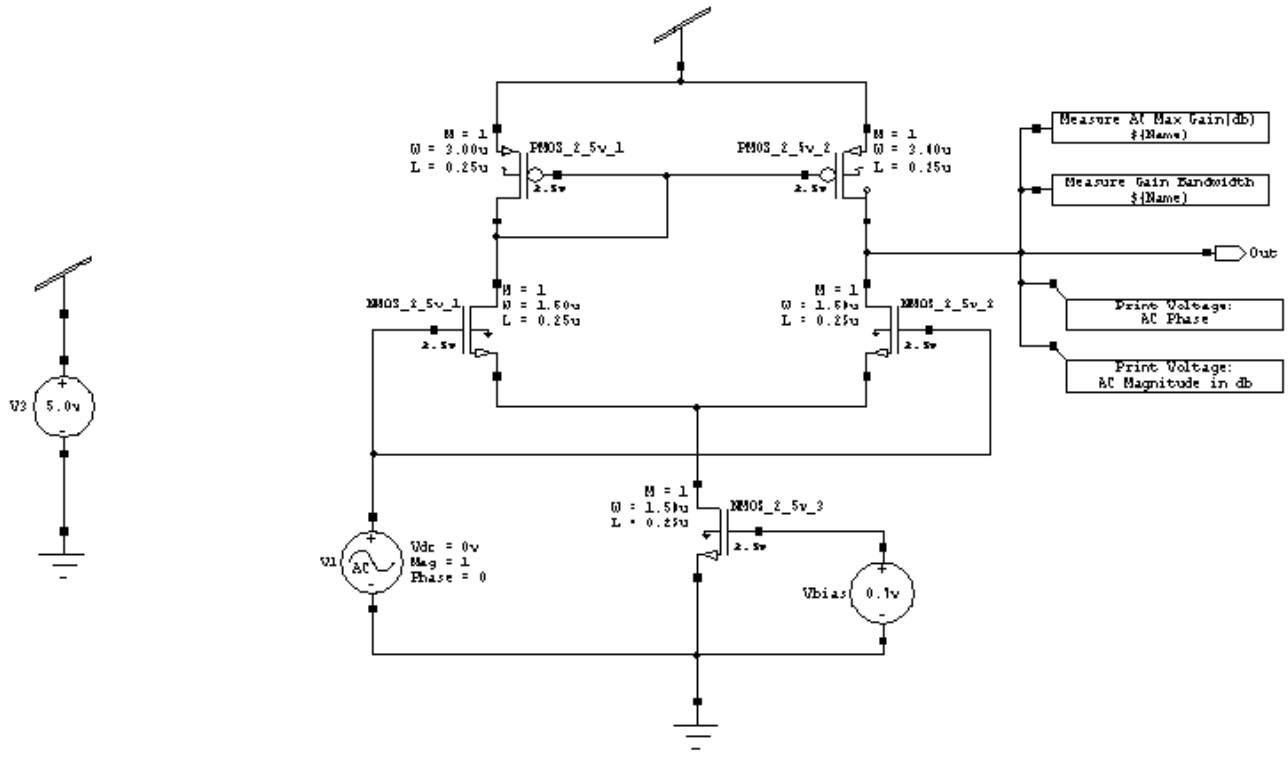


NETLIST:

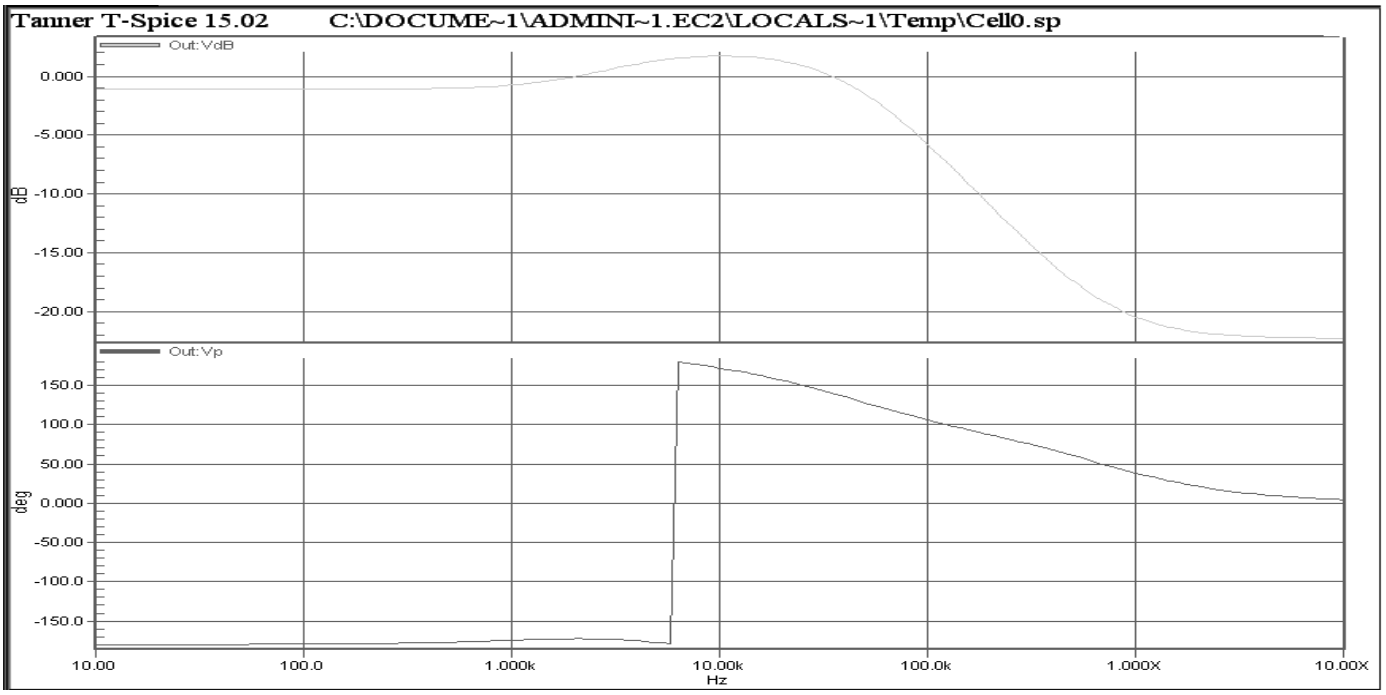
```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT
*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
MNMOS_2_5v_1 N_1 N_3 N_4 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u
+$ $x=3793 $y=4300 $w=414 $h=600
MNMOS_2_5v_2 Out N_5 N_4 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u
+$ $x=6607 $y=4300 $w=414 $h=600 $m
MNMOS_2_5v_3 N_4 N_6 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u
+$ $x=5507 $y=3000 $w=414 $h=600 $m
MPMOS_2_5v_1 N_1 N_1 Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u
+$ $x=4207 $y=5300 $w=414 $h=600 $m
MPMOS_2_5v_2 Out N_1 Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u
+$ $x=6193 $y=5300 $w=414 $h=600

*-------- Devices With SPICE.ORDER > 0.0  --------
VV3 Vdd Gnd  DC 5 $ $x=1200 $y=3800 $w=400 $h=600
VVbias N_6 Gnd  DC 700m $ $x=6500 $y=2600 $w=400 $h=600
VV1 N_3 Gnd  DC 0 AC 1 0 $ $x=3200 $y=2800 $w=400 $h=600
VV2 N_5 Gnd  DC 0 AC 1 180 $ $x=7200 $y=2900 $w=400 $h=600
.PRINT AC Vdb(Out) $ $x=8350 $y=4050 $w=1500 $h=300
.PRINT AC Vp(Out) $ $x=8350 $y=4450 $w=1500 $h=300
.MEASURE AC AC_Measure_Gain_1 MAX vdb(Out) ON $ $x=8250 $y=5600 $w=1500 $h=200
.MEASURE AC AC_Measure_GainBandwidthProduct_1_Gain MAX vdb(Out) OFF
.MEASURE AC AC_Measure_GainBandwidthProduct_1_UGFreq WHEN Vdb(Out)=0 OFF
.MEASURE AC AC_Measure_GainBandwidthProduct_1
PARAM='AC_Measure_GainBandwidthProduct_1_Gain*AC_Measure_GainBandwidthProduct_1_UGFr
eq' +ON $ $x=8250 $y=5200 $w=1500 $h=200
********* Simulation Settings - Analysis Section *********
.ac dec 25 10 10X
********* Simulation Settings - Additional SPICE Commands *********
.end
```

SCHEMATICDIAGRAM:

COMMON MODE:



COMMON MODE OUTPUT:



NETLIST:

********* Simulation Settings - General Section *********

.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*-------- Devices With SPICE.ORDER == 0.0 --------

***** Top Level *****

MNMOS_2_5v_1 N_1 N_2 N_3 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u

+$ $x=3793 $y=4300 $w=414 $h=600

MNMOS_2_5v_2 Out N_2 N_3 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u

+$ $x=6607 $y=4300 $w=414 $h=600 $m

MNMOS_2_5v_3 N_3 N_5 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u

+$ $x=5507 $y=3000 $w=414 $h=600 $m

MPMOS_2_5v_1 N_1 N_1 Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u

+$ $x=4207 $y=5300 $w=414 $h=600 $m

MPMOS_2_5v_2 Out N_1 Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u

+$ $x=6193 $y=5300 $w=414 $h=600

*-------- Devices With SPICE.ORDER > 0.0 --------

VV3 Vdd Gnd DC 5 $ $x=1200 $y=3800 $w=400 $h=600

VVbias N_5 Gnd DC 700m $ $x=6500 $y=2600 $w=400 $h=600

VV1 N_2 Gnd DC 0 AC 1 0 $ $x=3200 $y=2800 $w=400 $h=600

.PRINT AC Vdb(Out) $ $x=8350 $y=4050 $w=1500 $h=300

.PRINT AC Vp(Out) $ $x=8350 $y=4450 $w=1500 $h=300

.MEASURE AC AC_Measure_Gain_1 MAX vdb(Out) ON $ $x=8250 $y=5600 $w=1500 $h=200

.MEASURE AC AC_Measure_GainBandwidthProduct_1_Gain MAX vdb(Out) OFF

.MEASURE AC AC_Measure_GainBandwidthProduct_1_UGFreq WHEN Vdb(Out)=0 OFF

.MEASURE AC AC_Measure_GainBandwidthProduct_1 PARAM='AC_Measure_GainBandwidthProduct_1_Gain*AC_Measure_GainBandwidthProduct_1_UGFreq'

+ON $ $x=8250 $y=5200 $w=1500 $h=200

********* Simulation Settings - Analysis Section *********

.ac dec 25 10 10X

********* Simulation Settings - Additional SPICE Commands *********

.end

MEASUREMENT RESULT SUMMARY:

DIFFERNTIAL AMPLIFIER		
	COMMOM MODE	**DIFFERNTIAL MODE**
AC_Measure_Gain	Ac=	Ad=
AC_Measure_GainBandwidthProduct		

$$CMRR = A_d / A_c$$

=

RESULT

EXP.NO:	CMOS GATES
DATE:	

AIM:

To perform the functional verification of the CMOS gates through schematic entry.

FACILITIES REQUIRED AND PROCEDURE

a) Facilities required to do the experiment

S.No.	SOFTWARE REQUIREMENTS	QUANTITY
1	S-Edit, W-Edit, T-Edit using Tanner Tool.	1

b) Procedure for doing the experiment

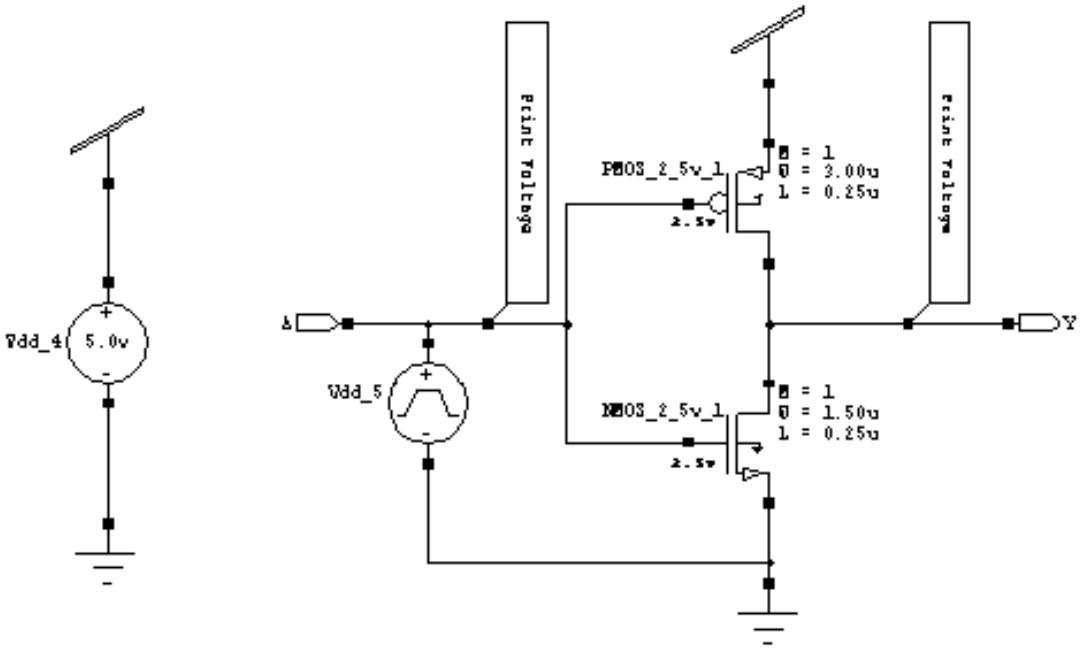
S.NO	DETAILS OF THE STEP
1	Draw the schematic of CMOS Gates using S-edit
2	Perform Transient Analysis of the CMOS Inverter
3	Obtain the output wave form from W-edit
4	Obtain the spice code using T-edit

c) THEORY: (CMOS NOT)

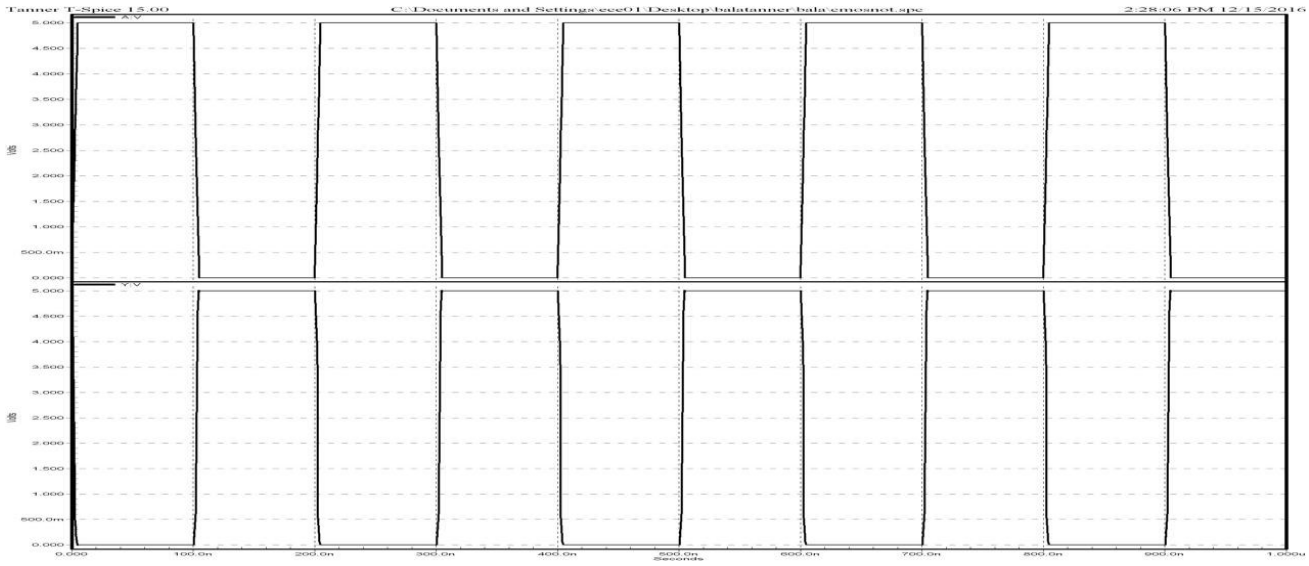
- Inverter consists of nMOS and pMOS transistor in series connected between VDD and GND.
- The gate of the two transistors are shorted and connected to the input. When the input to the inverter A =0, nMOS transistor is OFF and pMOS transistor is ON. The output is pull-up to VDD.When the input A=1, nMOS transistor is ON and pMOS transistor is OFF. The Output is Pull-down to GND.

CMOS NOT

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:

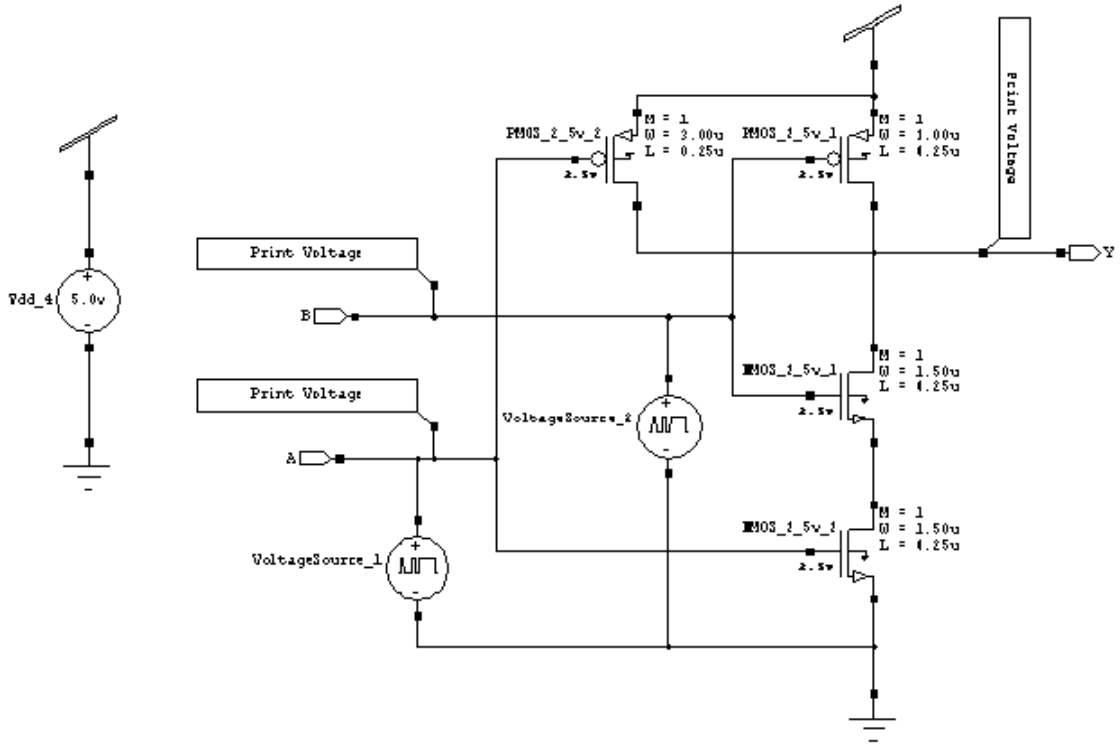


NETLIST:

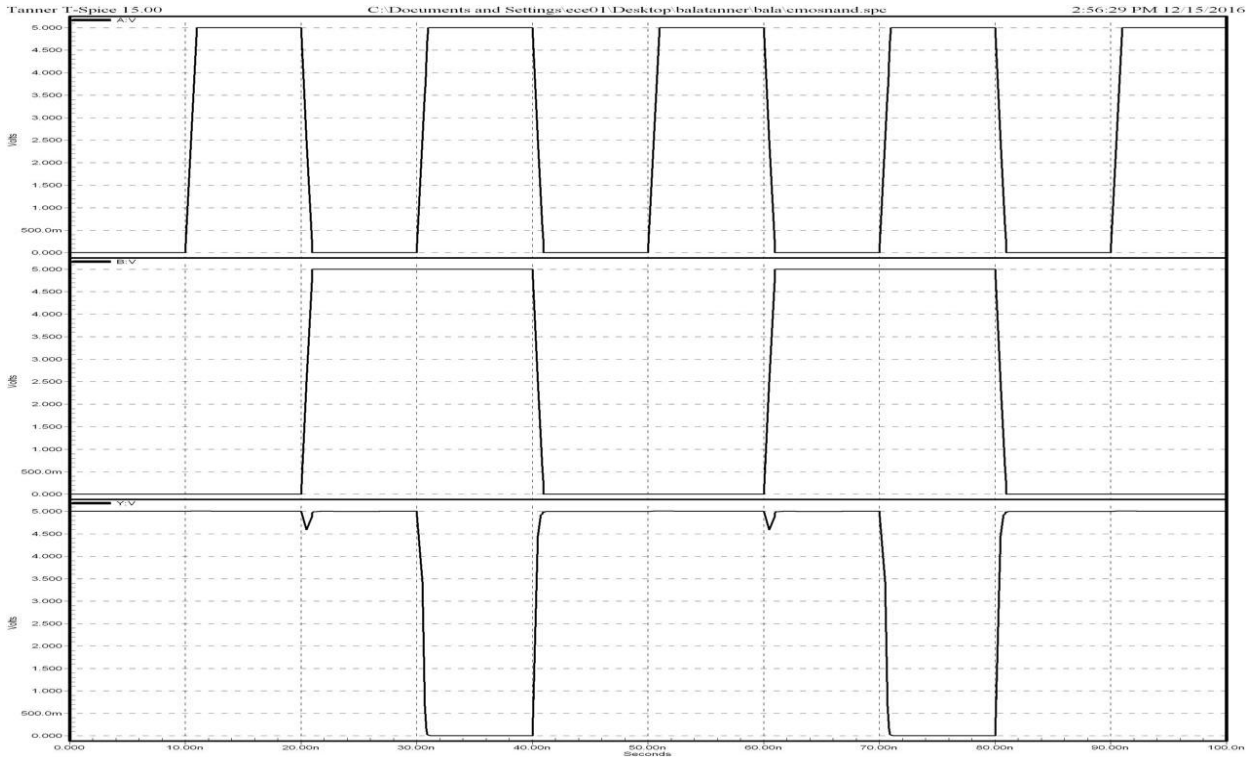
```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
MNMOS_2_5v_1 Y A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $
$x=4293
+$y=3300 $w=414 $h=600
MPMOS_2_5v_1 Y A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
+$x=4293 $y=4500 $w=414 $h=600

*-------- Devices With SPICE.ORDER > 0.0  --------
VVdd_4 Vdd Gnd  DC 5 $ $x=1200 $y=3800 $w=400 $h=600
VVdd_5 A Gnd  PULSE(0 5 0 5n 5n 95n 200n) $ $x=2800 $y=3500 $w=400 $h=600
.PRINT TRAN V(A) $ $x=3250 $y=4650 $w=300 $h=1500 $r=270
.PRINT TRAN V(Y) $ $x=5350 $y=4650 $w=300 $h=1500 $r=270

********* Simulation Settings - Analysis Section *********
.tran 10ns 1000ns
********* Simulation Settings - Additional SPICE Commands *********
.end
```

CMOS NAND

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:

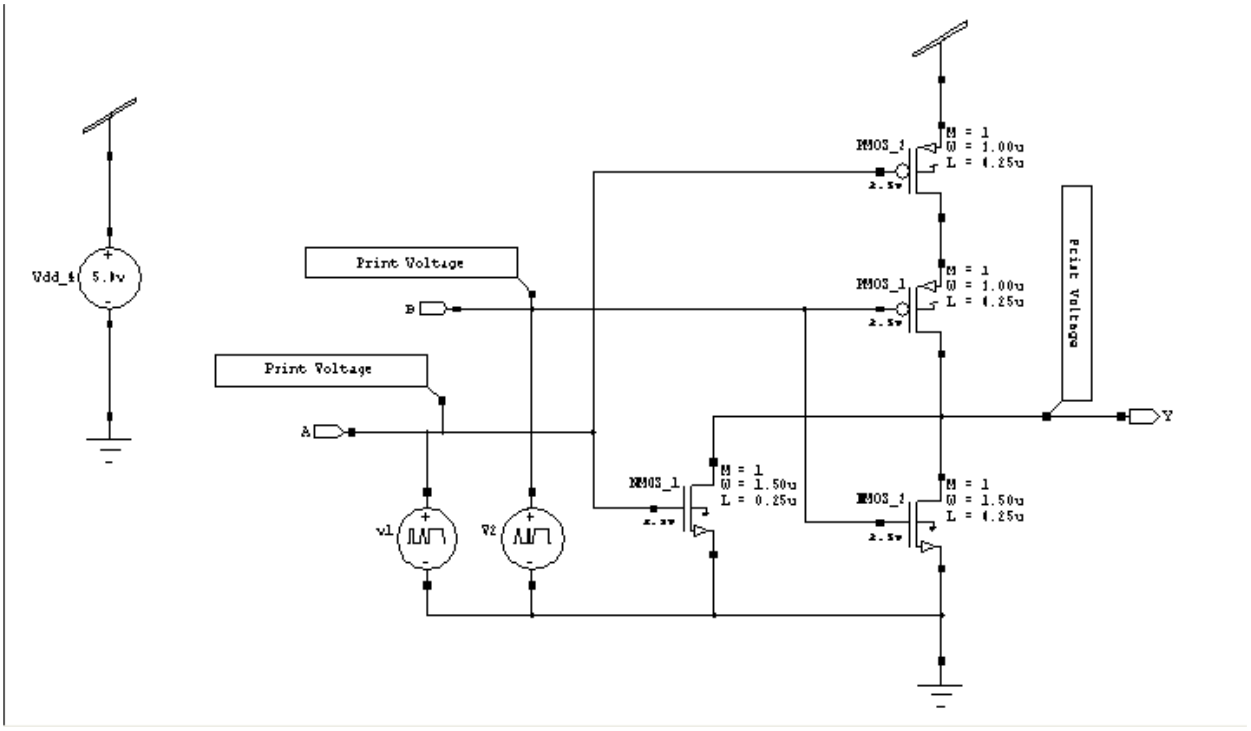


NETLIST:

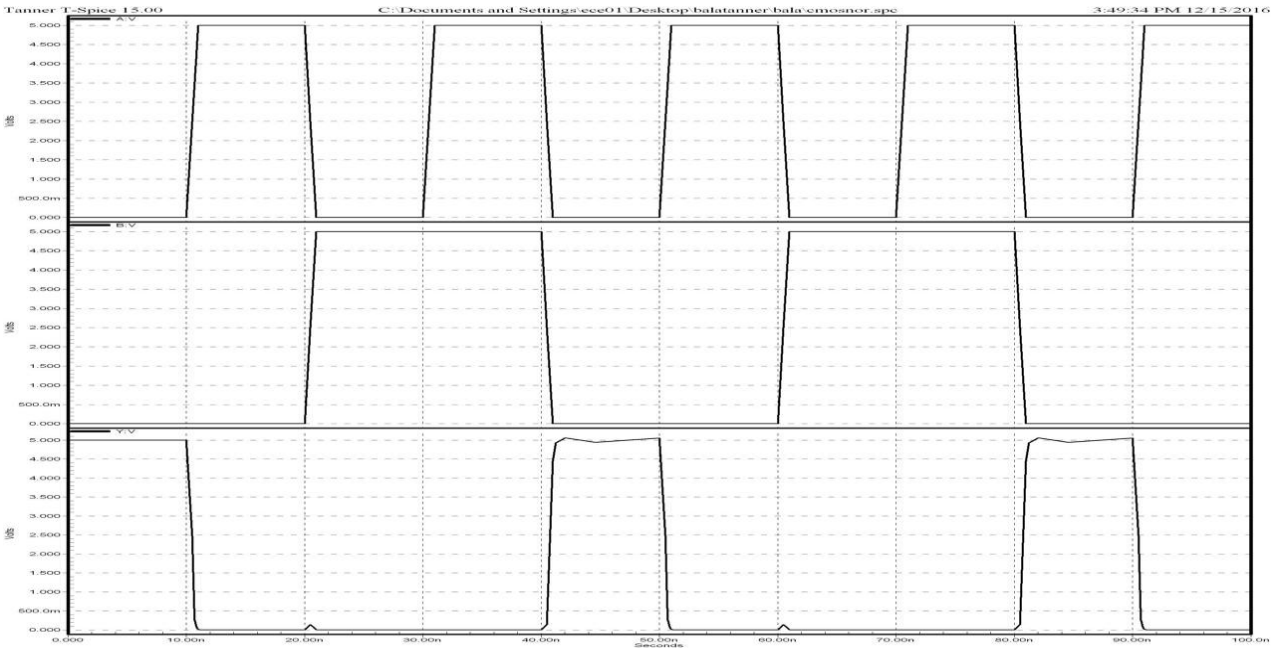
```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
MNMOS_2_5v_1 Y B N_1 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $
$x=5993
+$y=3200 $w=414 $h=600
MNMOS_2_5v_2 N_1 A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $
+$x=5993 $y=2200 $w=414 $h=600
MPMOS_2_5v_1 Y B Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
+$x=5993 $y=4700 $w=414 $h=600
MPMOS_2_5v_2 Y A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
+$x=4493 $y=4700 $w=414 $h=600

*-------- Devices With SPICE.ORDER > 0.0  --------
VVdd_4 Vdd Gnd  DC 5 $ $x=1200 $y=3800 $w=400 $h=600
VVoltageSource_1 A Gnd  BIT({0101} ) $ $x=3300 $y=2100 $w=400 $h=600
VVoltageSource_2 B Gnd  BIT({0011} ) $ $x=4900 $y=3000 $w=400 $h=600
.PRINT TRAN V(A) $ $x=2650 $y=3150 $w=1500 $h=300 $r=180
.PRINT TRAN V(B) $ $x=2650 $y=4050 $w=1500 $h=300 $r=180
.PRINT TRAN V(Y) $ $x=7050 $y=4850 $w=300 $h=1500 $r=270
********* Simulation Settings - Analysis Section *********
.tran 10ns 100ns
********* Simulation Settings - Additional SPICE Commands *********
.end
```

CMOS NOR

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:



NETLIST:

```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
MNMOS_1 Y A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=4493
+$y=2300 $w=414 $h=600
MNMOS_2 Y B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5993
+$y=2200 $w=414 $h=600
MPMOS_1 Y B N_3 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=5993
+$y=3600 $w=414 $h=600
MPMOS_2 N_3 A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=5993
+$y=4500 $w=414 $h=600

*-------- Devices With SPICE.ORDER > 0.0  --------
VVdd_4 Vdd Gnd  DC 5 $ $x=700 $y=3800 $w=400 $h=600
Vv1 A Gnd  BIT({0101} ) $ $x=2800 $y=2100 $w=400 $h=600
VV2 B Gnd  BIT({0011} ) $ $x=3500 $y=2100 $w=400 $h=600
.PRINT TRAN V(A) $ $x=2150 $y=3150 $w=1500 $h=300 $r=180
.PRINT TRAN V(B) $ $x=2750 $y=3850 $w=1500 $h=300 $r=180
.PRINT TRAN V(Y) $ $x=7050 $y=3650 $w=300 $h=1500 $r=270

********* Simulation Settings - Analysis Section *********
.tran 10ns 100ns
********* Simulation Settings - Additional SPICE Commands *********
.end
```

RESULT

EXP.NO:	HALF ADDER AND FULL ADDER
DATE:	

AIM:

To perform the functional verification of the Full and Half adder circuit through schematic entry.

FACILITIES REQUIRED AND PROCEDURE

a) Facilities required to do the experiment

S.No.	SOFTWARE REQUIREMENTS	QUANTITY
1	S-Edit, W-Edit, T-Edit using Tanner Tool.	1

b) Procedure for doing the experiment

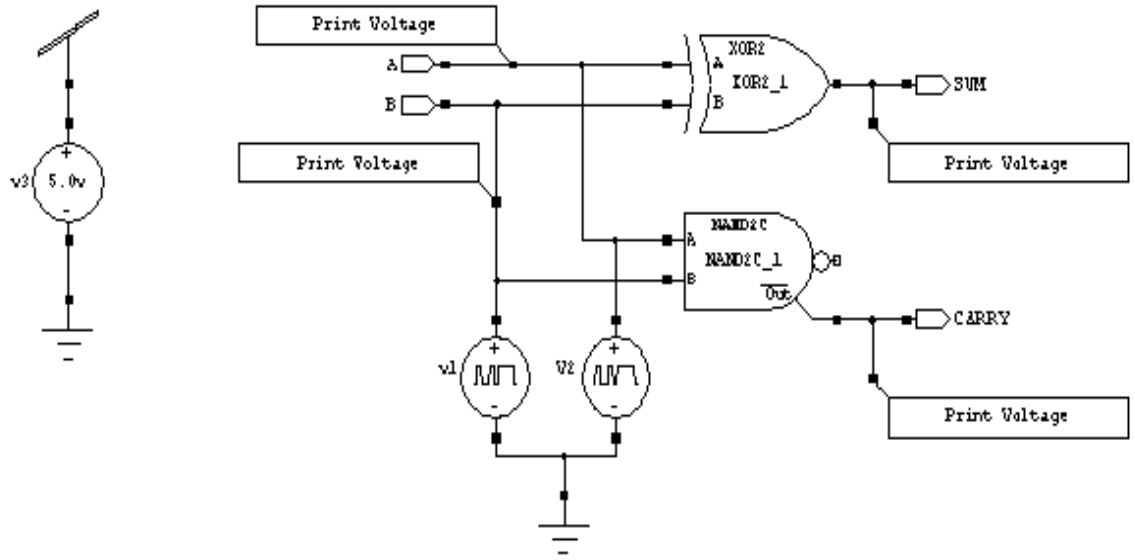
S.NO	DETAILS OF THE STEP
1	Draw the schematic of Full and Half adder Gates using S-edit
2	Perform Transient Analysis of the Full and Half adder circuit
3	Obtain the spice code using T-edit
4	Obtain the output wave form from W-edit

c) THEORY: (HALF ADDER)

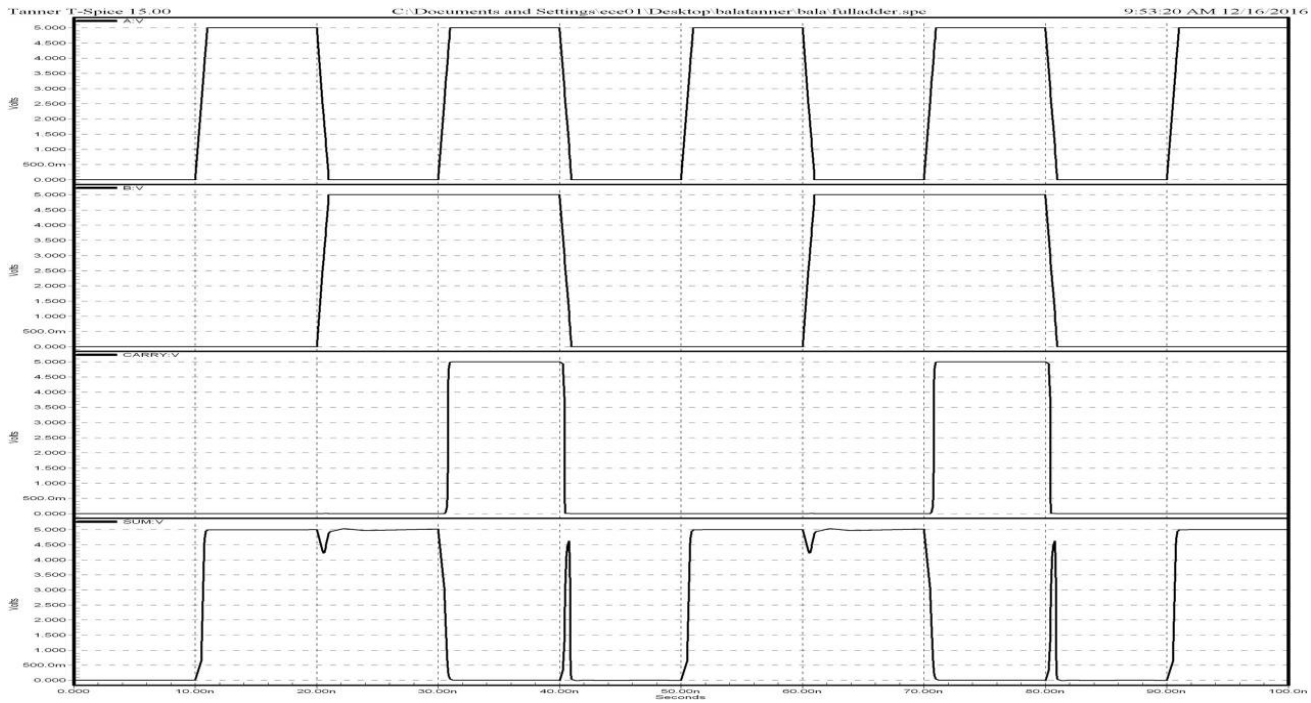
The **half adder** adds two single binary digits *A* and *B*. It has two outputs, sum (*S*) and carry (*C*). The carry signal represents an overflow into the next digit of a multi-digit addition. The value of the sum is $2C + S$. The simplest half-adder design, pictured on the right, incorporates an XOR gate for *S* and an AND gate for *C*. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder.The half adder adds two input bits and generates a carry and sum, which are the two outputs of a half adder. The input variables of a half adder are called the augend and addend bits. The output variables are the sum and carry.

HALF ADDER

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:



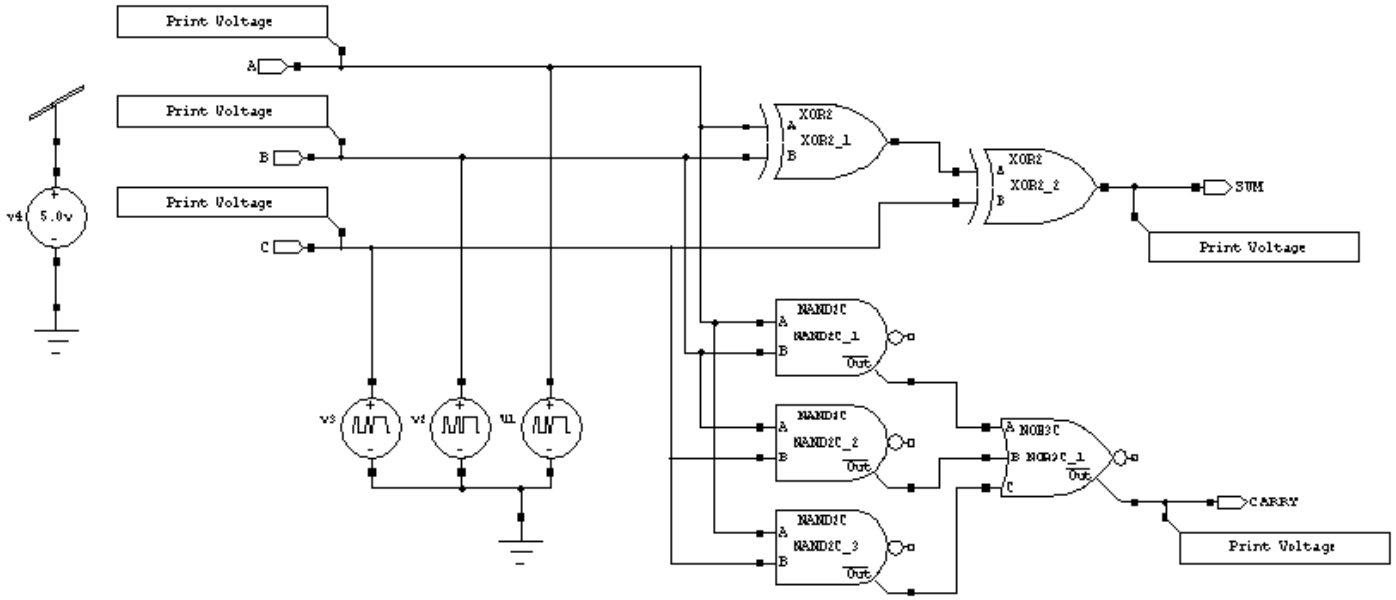
NETLIST:

```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*************** Subcircuits ******************
.subckt NAND2C A B Out Outbar Gnd Vdd

*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: NAND2C / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: 2 Input NAND with complementary output.
* Date: 5/30/2008 6:42:21 PM
* Revision: 10 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1n 1 B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3493 $y=2800
+$w=414 $h=600
MM2n Out A 1 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3493 $y=3600
+$w=414 $h=600
MM3p Out A Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $
+$x=3493 $y=4400 $w=414 $h=600
MM4p Out B Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $
+$x=4693 $y=4400 $w=414 $h=600
MM5n Outbar Out Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5893
+$y=3600 $w=414 $h=600
MM6p Outbar Out Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u
+$ $x=5893 $y=4400 $w=414 $h=600
.ends

.subckt XOR2 A B Out Gnd Vdd

*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: XOR2 / View: Main / Page:
* Designed by: Tanner CES Design Team
* Organization: Tanner Research, Inc.
* Info: TSMC 0.25u Digital Standard Cell Library
* Date: 10/13/2008 3:48:47 PM
* Revision: 3 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1p 1 A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=2907
+$y=5000 $w=414 $h=600 $m
MM2p 2 A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=4393
+$y=5000 $w=414 $h=600
MM3p 4 B 1 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=2493 $y=4100
+$w=414 $h=600
MM4p 3 B Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=6207
+$y=5000 $w=414 $h=600 $m
MM5p Out 4 2 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=4393
+$y=4100 $w=414 $h=600
MM6p Out 4 3 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=6207
+$y=4100 $w=414 $h=600 $m
MM7n Out 4 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=7393
```

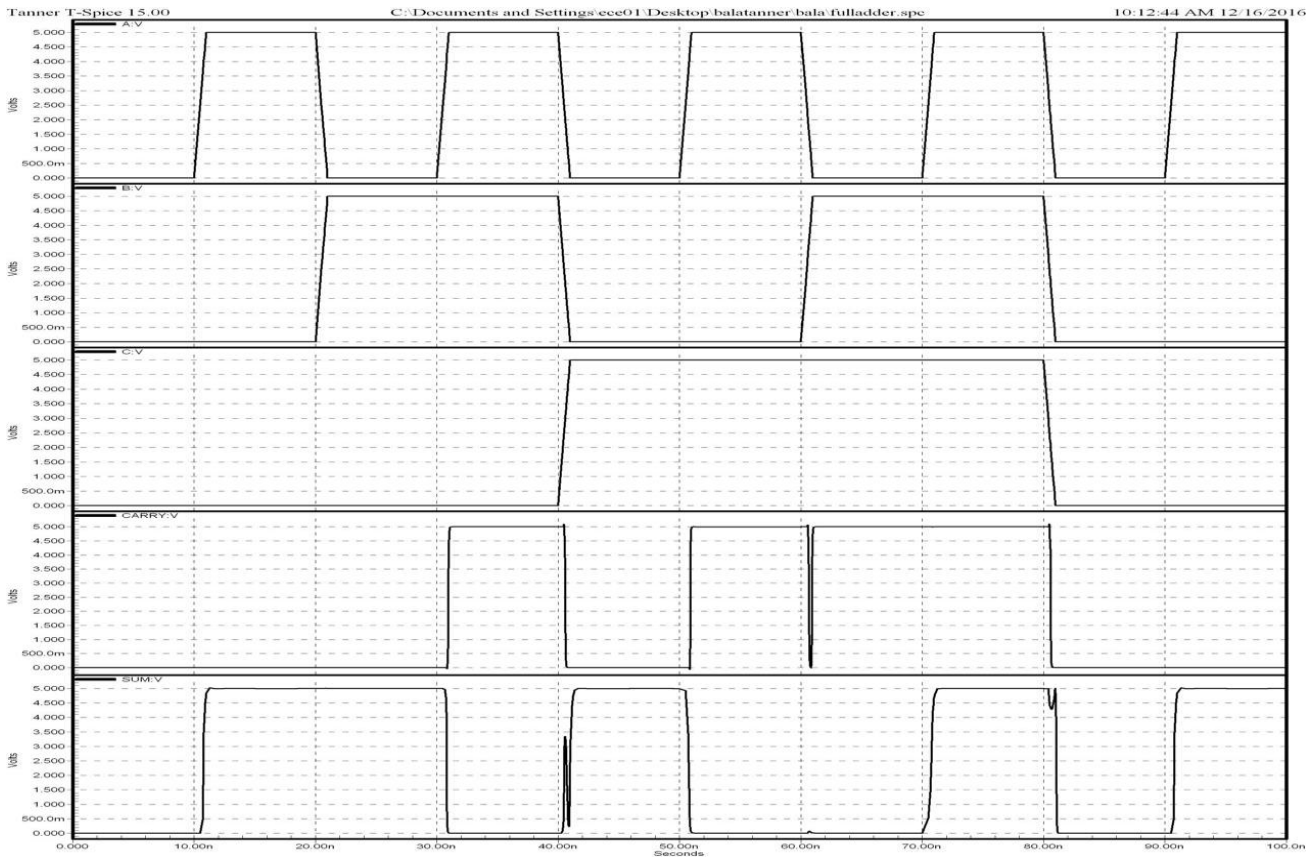
```
+$y=4100 $w=414 $h=600
MM8n 4 B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1893 $y=2800
+$w=414 $h=600
MM9n 4 A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3507 $y=2800
+$w=414 $h=600 $m
MM10n Out B 5 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5507 $y=2800
+$w=414 $h=600 $m
MM11n 5 A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5093 $y=2000
+$w=414 $h=600
.ends

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
XNAND2C_1 A B N_1 CARRY Gnd Vdd NAND2C $ $x=4700 $y=3975 $w=1000 $h=550
XXOR2_1 A B SUM Gnd Vdd XOR2 $ $x=4700 $y=4900 $w=1000 $h=500

*-------- Devices With SPICE.ORDER > 0.0  --------
Vv3 Vdd Gnd  DC 5 $ $x=700 $y=4400 $w=400 $h=600
Vv1 B Gnd  BIT({0011} ) $ $x=3200 $y=3400 $w=400 $h=600
VV2 A Gnd  BIT({0101} ) $ $x=3900 $y=3400 $w=400 $h=600
.PRINT TRAN V(A) $ $x=2550 $y=5150 $w=1500 $h=300 $r=180
.PRINT TRAN V(B) $ $x=2450 $y=4450 $w=1500 $h=300 $r=180
.PRINT TRAN V(CARRY) $ $x=6150 $y=3250 $w=1500 $h=300
.PRINT TRAN V(SUM) $ $x=6150 $y=4550 $w=1500 $h=300
********* Simulation Settings - Analysis Section *********
.tran 10n 100n
********* Simulation Settings - Additional SPICE Commands *********
.end
```

FULL ADDER

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:



NETLIST:

```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

*************** Subcircuits *****************
.subckt NAND2C A B Out Outbar Gnd Vdd

*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: NAND2C / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: 2 Input NAND with complementary output.
* Date: 5/30/2008 6:42:21 PM
* Revision: 10 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1n 1 B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3493 $y=2800
+$w=414 $h=600
MM2n Out A 1 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3493 $y=3600
+$w=414 $h=600
MM3p Out A Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $
+$x=3493 $y=4400 $w=414 $h=600
MM4p Out B Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $
+$x=4693 $y=4400 $w=414 $h=600
MM5n Outbar Out Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5893
+$y=3600 $w=414 $h=600
MM6p Outbar Out Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u
+$ $x=5893 $y=4400 $w=414 $h=600
.ends

.subckt NOR3C A B C Out Outbar Gnd Vdd

*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: NOR3C / View: Main / Page:
* Designed by: Author
* Organization: Organization
* Info: Info
* Date: 5/30/2008 6:42:21 PM
* Revision: 53 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1n Out A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=2093
+$y=2100 $w=414 $h=600
MM2p Out C 2 Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $ $x=2093
+$y=2800 $w=414 $h=600
MM3p 2 B 1 Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $ $x=2093
+$y=3600 $w=414 $h=600
MM4p 1 A Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $ $x=2093
+$y=4400 $w=414 $h=600
MM5n Out B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3593
+$y=2100 $w=414 $h=600
MM6n Out C Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5093
+$y=2100 $w=414 $h=600
MM7n Outbar Out Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=6393
+$y=2100 $w=414 $h=600
MM8p Outbar Out Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u
```

```
+$ $x=6393 $y=2900 $w=414 $h=600
.ends

.subckt XOR2 A B Out Gnd Vdd

*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: XOR2 / View: Main / Page:
* Designed by: Tanner CES Design Team
* Organization: Tanner Research, Inc.
* Info: TSMC 0.25u Digital Standard Cell Library
* Date: 10/13/2008 3:48:47 PM
* Revision: 3 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1p 1 A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=2907
+$y=5000 $w=414 $h=600 $m
MM2p 2 A Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=4393
+$y=5000 $w=414 $h=600
MM3p 4 B 1 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=2493 $y=4100
+$w=414 $h=600
MM4p 3 B Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=6207
+$y=5000 $w=414 $h=600 $m
MM5p Out 4 2 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=4393
+$y=4100 $w=414 $h=600
MM6p Out 4 3 Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=6207
+$y=4100 $w=414 $h=600 $m
MM7n Out 4 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=7393
+$y=4100 $w=414 $h=600
MM8n 4 B Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1893 $y=2800
+$w=414 $h=600
MM9n 4 A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3507 $y=2800
+$w=414 $h=600 $m
MM10n Out B 5 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5507 $y=2800
+$w=414 $h=600 $m
MM11n 5 A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5093 $y=2000
+$w=414 $h=600
.ends

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
XNAND2C_1 A B N_2 N_1 Gnd Vdd NAND2C $ $x=5700 $y=3575 $w=1000 $h=550
XNAND2C_2 B C N_5 N_3 Gnd Vdd NAND2C $ $x=5700 $y=2875 $w=1000 $h=550
XNAND2C_3 A C N_7 N_6 Gnd Vdd NAND2C $ $x=5700 $y=2175 $w=1000 $h=550
XNOR3C_1 N_1 N_3 N_6 N_8 CARRY Gnd Vdd NOR3C $ $x=7202 $y=2775 $w=1004 $h=550
XXOR2_1 A B N_4 Gnd Vdd XOR2 $ $x=5600 $y=4900 $w=1000 $h=500
XXOR2_2 N_4 C SUM Gnd Vdd XOR2 $ $x=7000 $y=4600 $w=1000 $h=500

*-------- Devices With SPICE.ORDER > 0.0  --------
Vv4 Vdd Gnd  DC 5 $ $x=500 $y=4400 $w=400 $h=600
VV1 A Gnd  BIT({01010101} ) $ $x=3800 $y=3000 $w=400 $h=600
Vv2 B Gnd  BIT({00110011} ) $ $x=3200 $y=3000 $w=400 $h=600
Vv3 C Gnd  BIT({00001111} ) $ $x=2600 $y=3000 $w=400 $h=600
.PRINT TRAN V(A) $ $x=1650 $y=5650 $w=1500 $h=300 $r=180
.PRINT TRAN V(B) $ $x=1650 $y=5050 $w=1500 $h=300 $r=180
.PRINT TRAN V(C) $ $x=1650 $y=4450 $w=1500 $h=300 $r=180
.PRINT TRAN V(CARRY) $ $x=8650 $y=2250 $w=1500 $h=300
```

```
.PRINT TRAN V(SUM) $ $x=8450 $y=4250 $w=1500 $h=300
********* Simulation Settings - Analysis Section *********
.tran 10n 100n
********* Simulation Settings - Additional SPICE Commands *********
.end
```

RESULT:

EXP.NO:	4-BIT COUNTER
DATE:	

AIM:

To perform the functional verification of the 4-bit counter circuit through schematic entry.

FACILITIES REQUIRED AND PROCEDURE

a) Facilities required to do the experiment

S.No.	SOFTWARE REQUIREMENTS	QUANTITY
1	S-Edit, W-Edit, T-Edit using Tanner Tool.	1

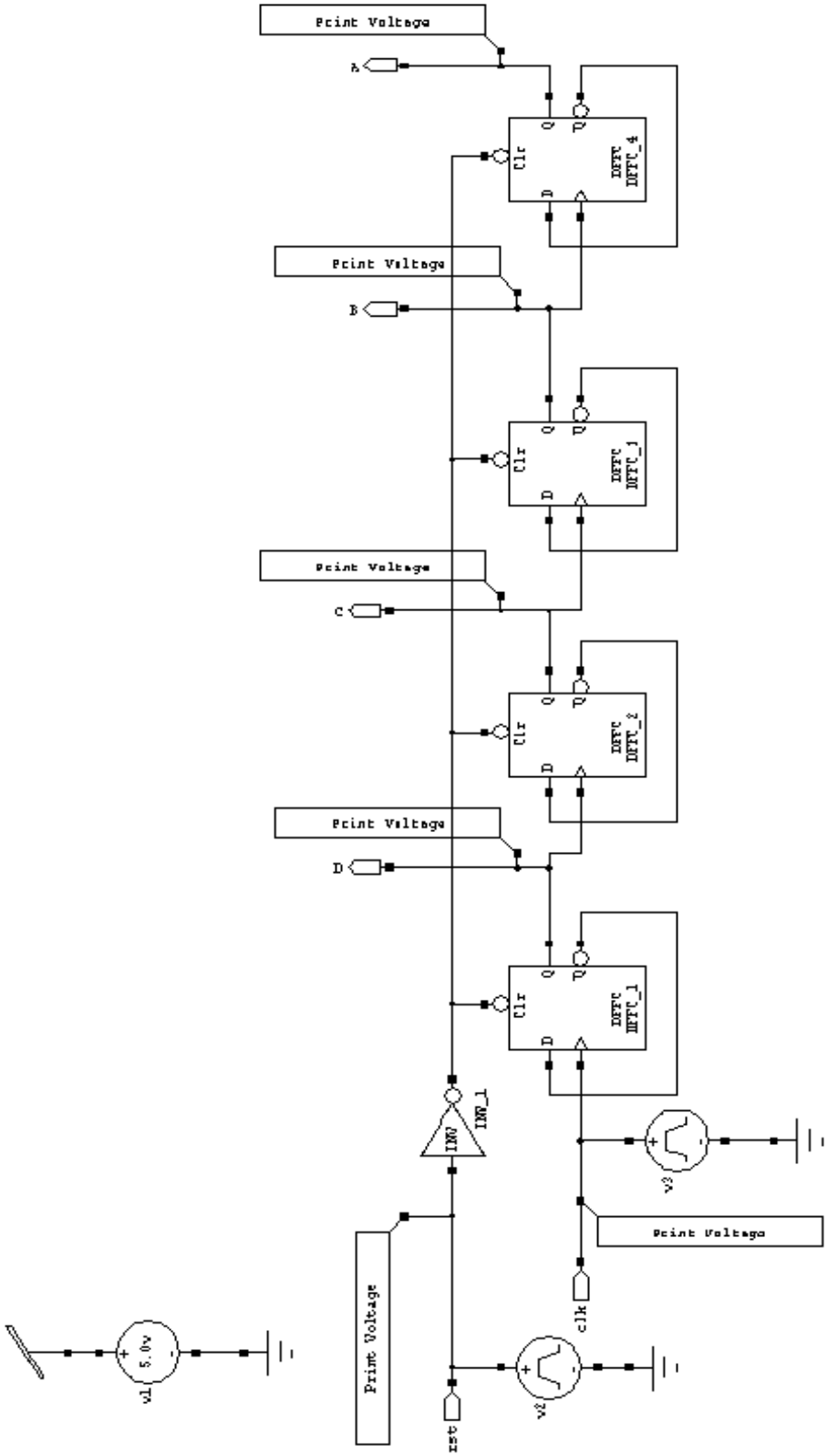
b) Procedure for doing the experiment

S.NO	DETAILS OF THE STEP
1	Draw the schematic of 4-bit counter using S-edit
2	Perform Transient Analysis of the 4-bit counter
3	Obtain the spice code using T-edit
4	Obtain the output wave form from W-edit

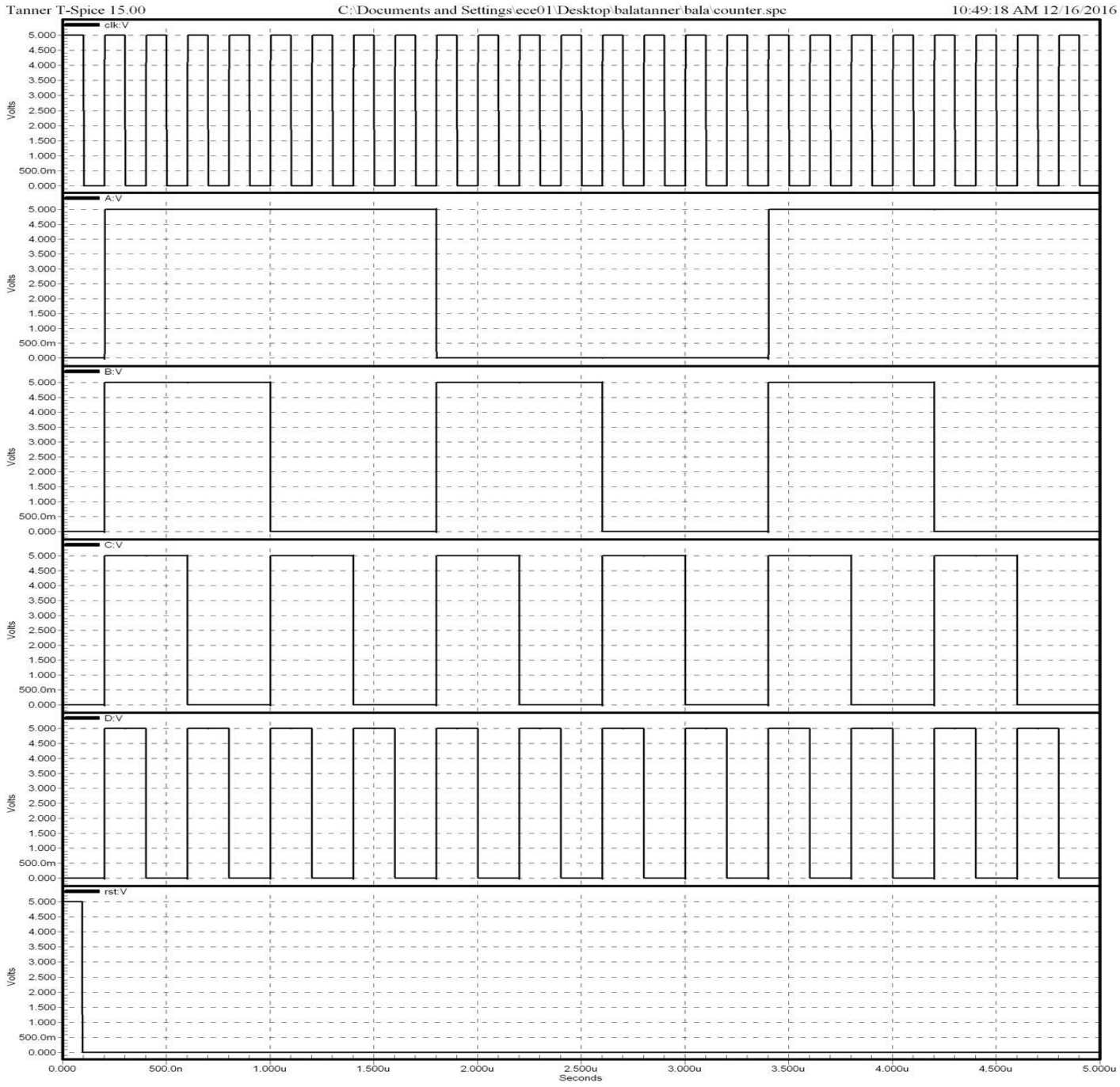
c) THEORY: (COUNTER)

4-BIT COUNTER

SCHEMATIC DIAGRAM:



OUTPUT WAVEFORM:



NETLIST:

```
********* Simulation Settings - General Section *********
.lib "C:\Documents and Settings\ece01\My Documents\Tanner EDA\Tanner Tools
v15.0\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT
*************** Subcircuits *****************
.subckt DFFC Clk Clr Data Q QB Gnd Vdd
*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: DFFC / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: D Flip-Flop with Clear
* Date: 10/15/2008 12:04:43 PM
* Revision: 144 $ $x=7600 $y=600 $w=3600 $h=1200
* Design: Generic_250nm_LogicGates / Cell: DFFC / View: Main / Page:
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: D Flip-Flop with Clear
* Date: 10/15/2008 12:04:43 PM
* Revision: 144 $ $x=7600 $y=600 $w=3600 $h=1200

*-------- Devices With SPICE.ORDER == 0.0  --------
MM1p CB Clk Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=1293
+$y=1800 $w=414 $h=600
MM2n CB Clk Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1293
+$y=1000 $w=414 $h=600
MM3p C CB Vdd Vdd PMOS25 W=3u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=2693
+$y=1800 $w=414 $h=600
MM4n C CB Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=2693
$y=1000
+$w=414 $h=600
MM5p 3 Data Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=3993
+$y=5600 $w=414 $h=600
MM6p 4 C 3 Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=3993
+$y=4700 $w=414 $h=600
MM7n 4 CB 5 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3993
$y=3900
+$w=414 $h=600
MM8n 5 Data Gnd 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3993
+$y=3100 $w=414 $h=600
MM9p 6 10 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=5693
+$y=5600 $w=414 $h=600
MM10p 4 CB 6 Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=6107
+$y=4700 $w=414 $h=600 $m
MM11n 4 C 7 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5693
$y=3900
+$w=414 $h=600
MM12n 7 10 8 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5693
$y=3100
+$w=414 $h=600
MM13n 8 Clr Gnd 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5693
+$y=2300 $w=414 $h=600
MM14p 9 Clr Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
```

```
$x=7293
+$y=5600 $w=414 $h=600
MM15p 4 CB 9 Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $ $x=7293
+$y=4700 $w=414 $h=600
MM16p 10 4 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=7193
+$y=3400 $w=414 $h=600
MM17n 10 4 Gnd 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=7193
+$y=2600 $w=414 $h=600
MM18p 11 10 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=1393
+$y=5200 $w=414 $h=600
MM19p 12 CB 11 Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=1393
+$y=4400 $w=414 $h=600
MM20An 12 Clr 15 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1393
+$y=3600 $w=414 $h=600
MM20n 15 C 13 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1393
+$y=2800 $w=414 $h=600
MM21n 13 10 Gnd 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=1393
+$y=2000 $w=414 $h=600
MM22p 17 Clr Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=3093
+$y=5200 $w=414 $h=600
MM23p 12 C 17 Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=3093
+$y=4400 $w=414 $h=600
MM24n 12 CB 17 0 NMOS25 W=1.50u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=3093
+$y=3600 $w=414 $h=600
MM25Ap Q 17 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=5293
+$y=4400 $w=414 $h=600
MM25p 17 16 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=4493
+$y=5200 $w=414 $h=600
MM26An Q 17 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=5293
+$y=3600 $w=414 $h=600
MM26n 17 Clr 14 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=4493
+$y=2800 $w=414 $h=600
MM27n 14 16 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=4493
+$y=2000 $w=414 $h=600
MM28Ap QB 16 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=8193
+$y=4300 $w=414 $h=600
MM28p 16 12 Vdd Vdd PMOS25 W=3.00u L=250n AS=1.95p PS=7.3u AD=1.95p PD=7.3u $
$x=6693
+$y=4400 $w=414 $h=600
MM29An QB 16 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $
$x=8193
+$y=3500 $w=414 $h=600
MM29n 16 12 Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=6693
+$y=3600 $w=414 $h=600
.ends
.subckt INV A Out Gnd Vdd
*-------- Devices With SPICE.ORDER < 0.0  --------
* Design: Generic_250nm_LogicGates / Cell: INV / View: Main / Page:
```

```
* Designed by: Tanner EDA Library Development Team
* Organization: Tanner EDA - Tanner Research, Inc.
* Info: Inverter
* Date: 5/30/2008 4:06:39 PM
* Revision: 13 $ $x=7600 $y=600 $w=3600 $h=1200
*-------- Devices With SPICE.ORDER == 0.0  --------
MM1n Out A Gnd 0 NMOS25 W=1.5u L=250n AS=975f PS=4.3u AD=975f PD=4.3u $ $x=4593
+$y=2600 $w=414 $h=600
MM2p Out A Vdd Vdd PMOS25 W=3u L=250n M=2 AS=1.125p PS=3.75u AD=1.95p PD=7.3u $
+$x=4593 $y=3600 $w=414 $h=600
.ends

*-------- Devices With SPICE.ORDER == 0.0  --------
***** Top Level *****
XDFFC_1 clk N_1 N_2 D N_2 Gnd Vdd DFFC $ $x=3100 $y=3400 $w=800 $h=1000
XDFFC_2 D N_1 N_4 C N_4 Gnd Vdd DFFC $ $x=4900 $y=3400 $w=800 $h=1000
XDFFC_3 C N_1 N_5 B N_5 Gnd Vdd DFFC $ $x=6700 $y=3400 $w=800 $h=1000
XDFFC_4 B N_1 N_6 A N_6 Gnd Vdd DFFC $ $x=8700 $y=3400 $w=800 $h=1000
XINV_1 rst N_1 Gnd Vdd INV $ $x=2300 $y=4100 $w=600 $h=400

*-------- Devices With SPICE.ORDER > 0.0  --------
Vv1 Vdd Gnd  DC 5 $ $x=800 $y=6000 $w=400 $h=600
Vv2 rst Gnd  PULSE(0 5 0 1n 1n 95n 10u) $ $x=700 $y=3500 $w=400 $h=600
Vv3 clk Gnd  PULSE(0 5 0 5n 5n 95n 200n) $ $x=2200 $y=2700 $w=400 $h=600
.PRINT TRAN V(A) $ $x=9550 $y=4550 $w=300 $h=1500 $r=270
.PRINT TRAN V(B) $ $x=7950 $y=4450 $w=300 $h=1500 $r=270
.PRINT TRAN V(C) $ $x=5950 $y=4550 $w=300 $h=1500 $r=270
.PRINT TRAN V(clk) $ $x=1650 $y=2550 $w=300 $h=1500 $r=90
.PRINT TRAN V(D) $ $x=4250 $y=4450 $w=300 $h=1500 $r=270
.PRINT TRAN V(rst) $ $x=950 $y=4550 $w=1500 $h=300 $r=180
********* Simulation Settings - Analysis Section *********
.tran 1u 5u
********* Simulation Settings - Additional SPICE Commands *********
.end
```

RESULT:

EXP.NO:	LAYOUT CMOS INVERTOR
DATE:	

AIM:

To draw the layout of CMOS Inverter using L-Edit and extract the SPICE code.

FACILITIES REQUIRED AND PROCEDURE

a) Facilities required to do the experiment

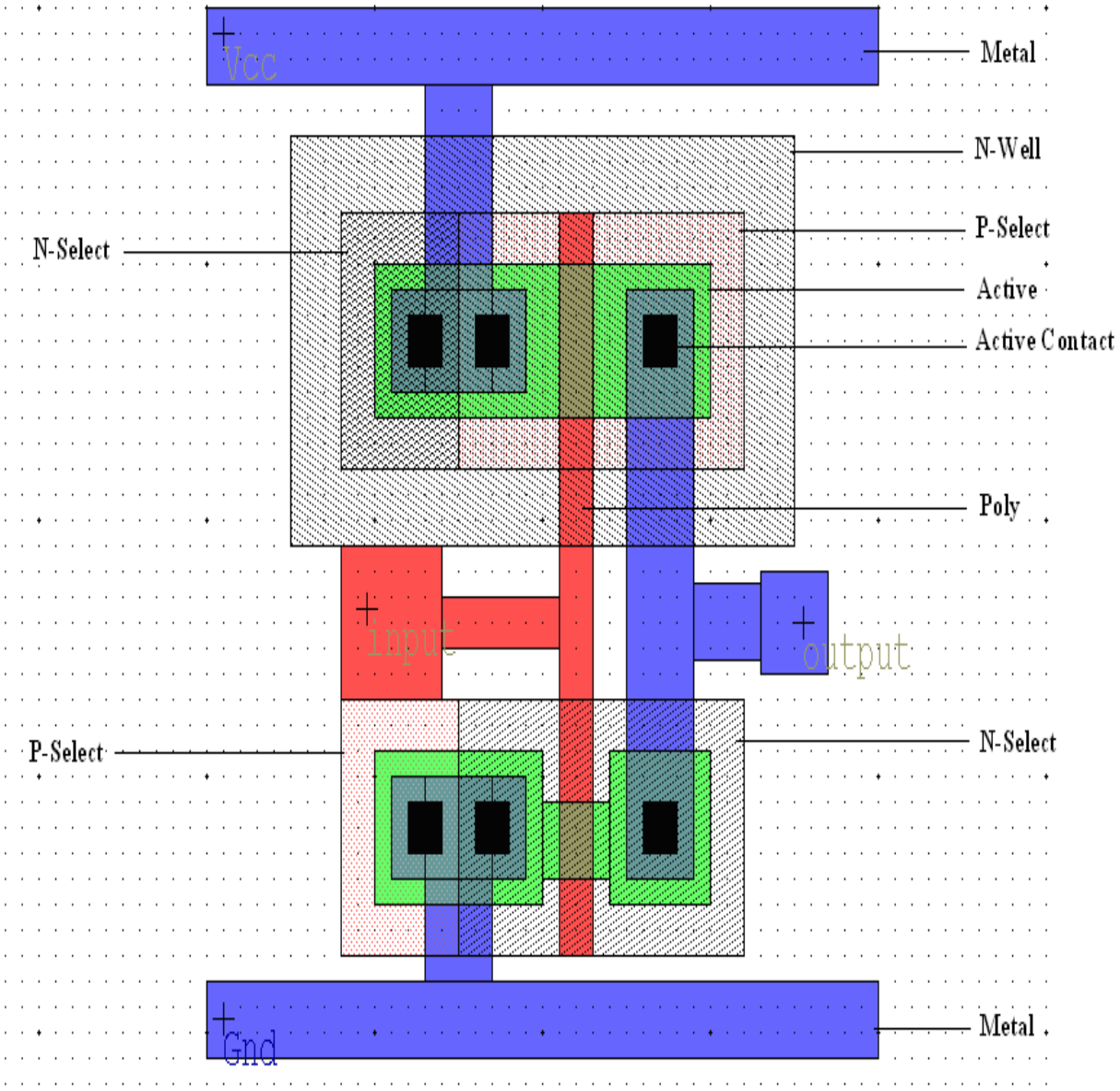
S.No.	SOFTWARE REQUIREMENTS	QUANTITY
1	L-Edit using Tanner Tool.	1

b) Procedure for doing the experiment

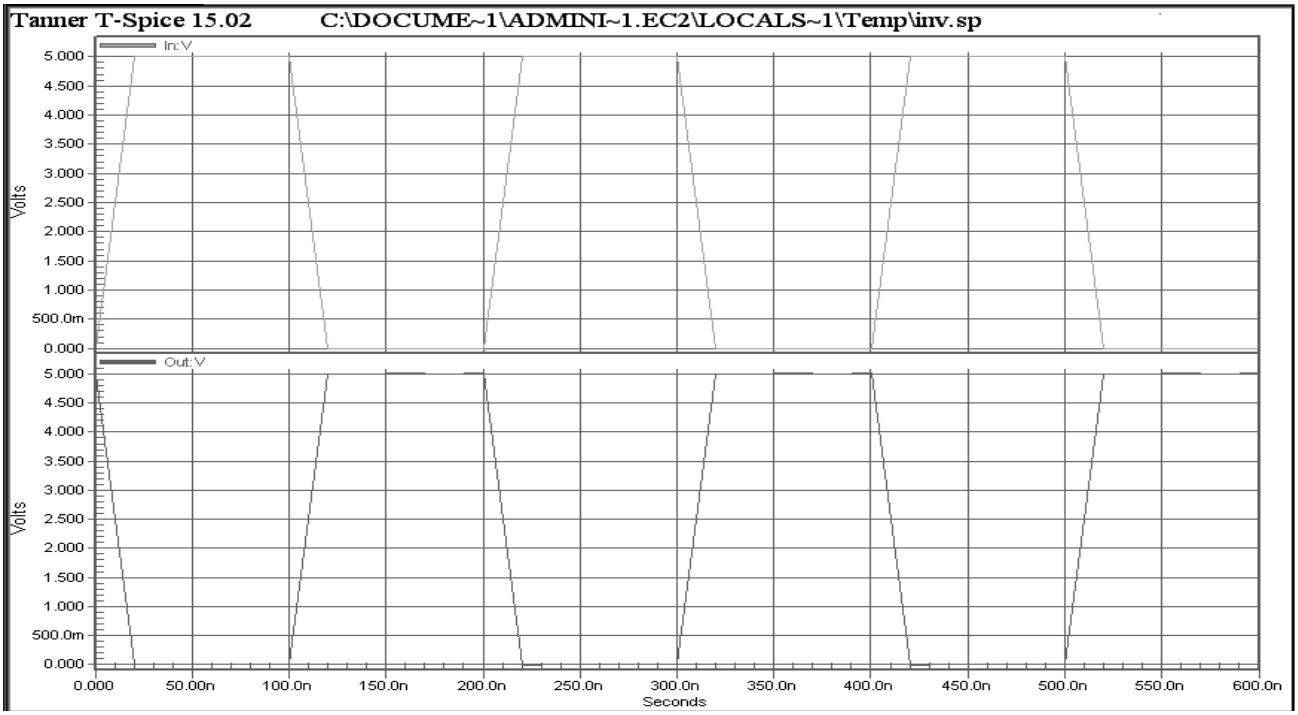
S.NO	DETAILS OF THE STEP
1	Draw the CMOS Inverter layout by obeying the Lamda Rules using L-edit.
2	i.Poly-2λ ii.Activecontact-2λ iii.ActiveContact–Metal-1λ iv.ActiveContact–Activeregion-2λ v.ActiveRegion–Pselect-3λ vi.Pselect–nWell-3λ
3	Check DRC to verify whether any region violate the Lamda rule
4	Setup the extraction and extract the spice code using T-spice.

CMOS INVERTER:

LAYOUT DAIGRAM:



OUTPUT WAVEFORM:



RESULT:

EXP.NO:	AUTOMATIC LAYOUT GENERATION
DATE:	

AIM:

To generate the automatic Layout from the schematic using the Tanner tool and verify the layout using simulation.

FACILITIES REQUIRED AND PROCEDURE

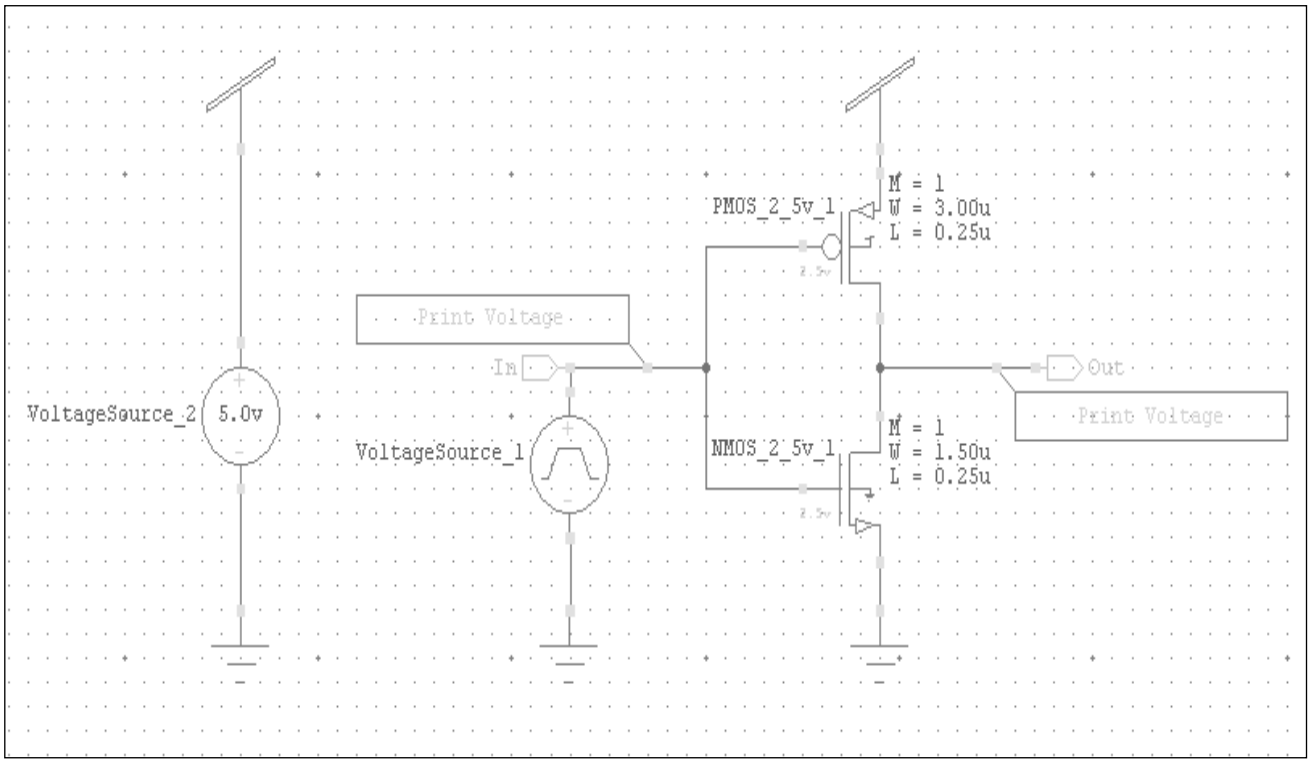
a) Facilities required to do the experiment

S.No.	SOFTWARE REQUIREMENTS	QUANTITY
1	S-Edit,L-Edit using Tanner Tool	1

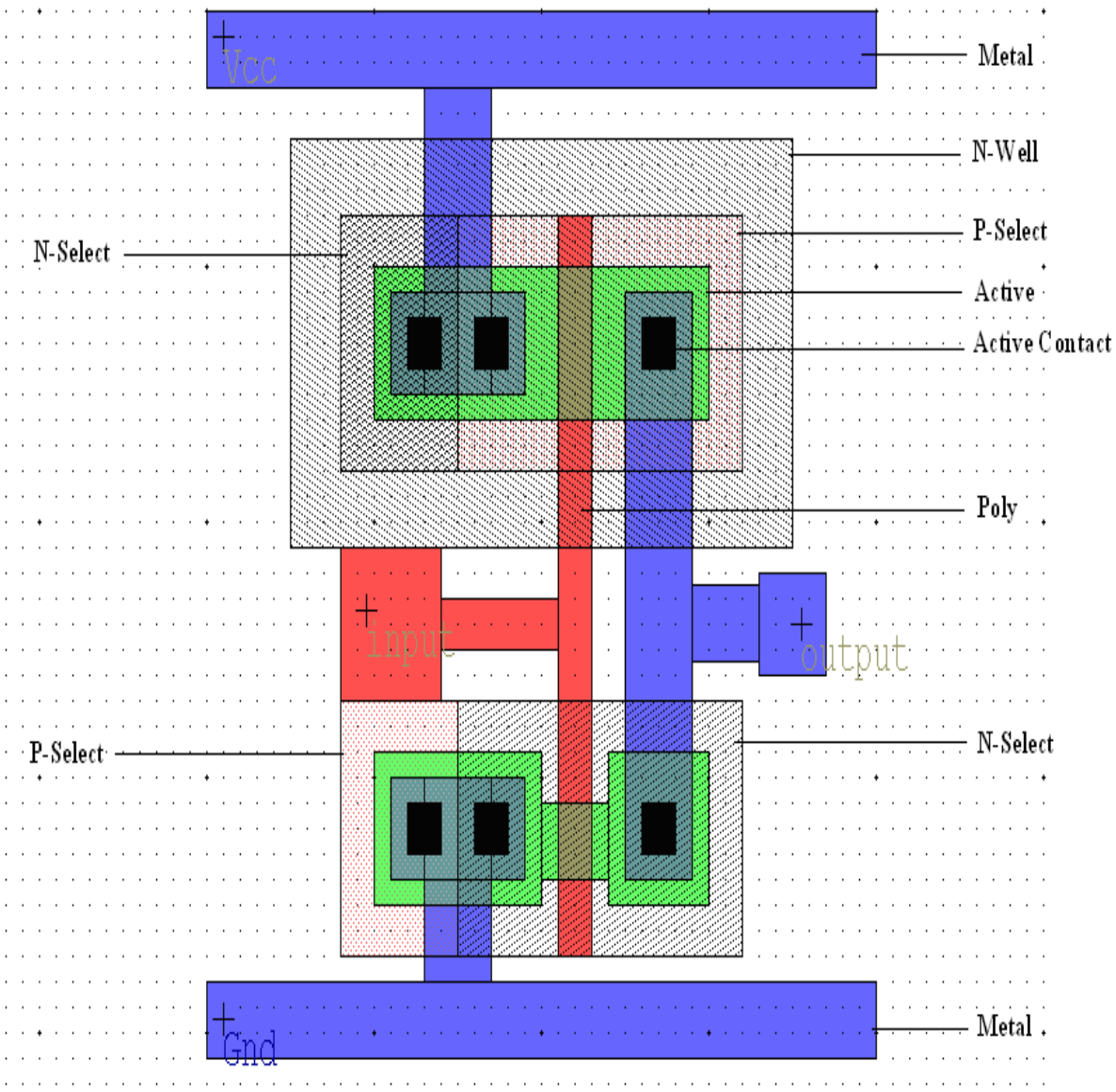
b) Procedure for doing the experiment

S.NO	DETAILS OF THE STEP
1	Draw the schematic using S -Edit and verify the output in W-Edit.
2	Extract the schematic and store it in another location
3	Open the L-Edit, open the design in Ring VCO
4	Create the new cell
5	Open the schematic file(.sdl) using the SDL Navigator
6	Do the necessary connections as per the design.
7	Name the ports and check the design for the DRC Rules
8	Locate the Destination file in the setup Extract window and extract the layout.
9	Include the Library and the print voltage statements in the net list which is obtained.
10	Verify the layout design using W-Edit.

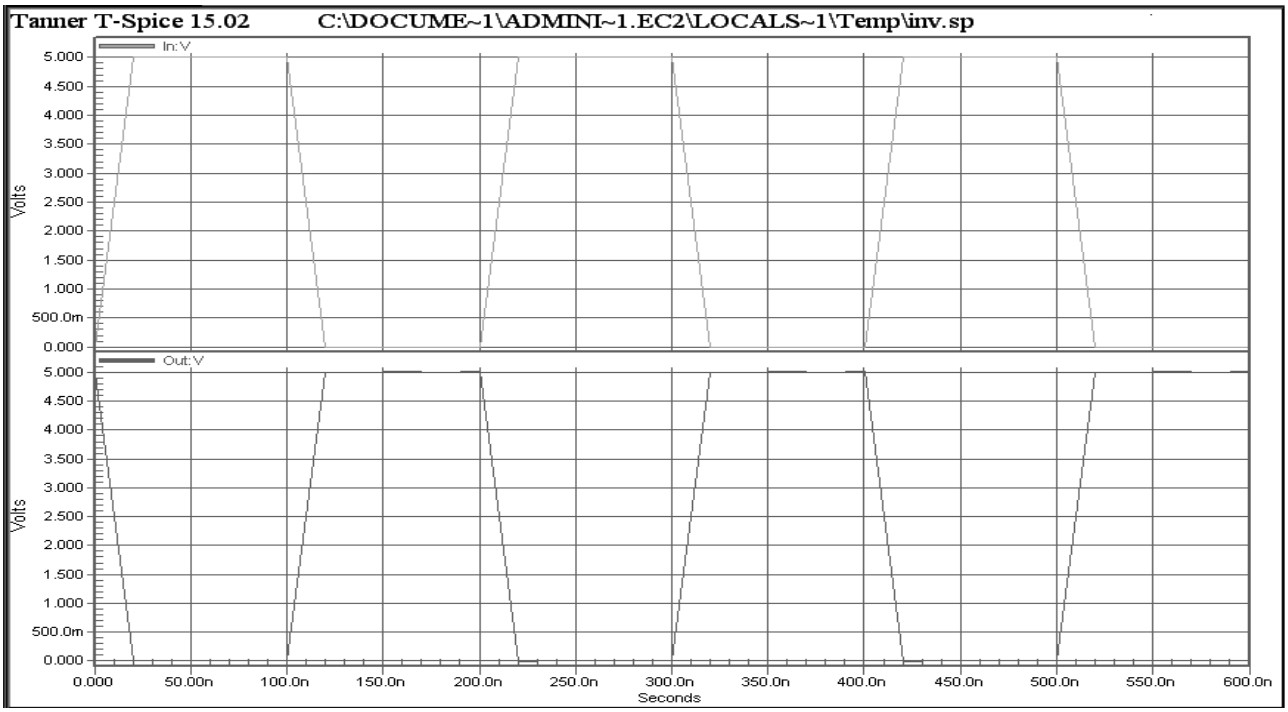
SCHEMATIC DIAGRAM:



LAYOUT GENERATION:



SIMULATED WAVEFORM:



RESULT:

APPENDIX (FPGA PIN DETAILS)

CLOCK SOURCE:

CLOCK INPUT	FPGA PIN
CLK	A8

APPENDIX (FPGA PIN DETAILS)

Slide Switch connections with FPGA

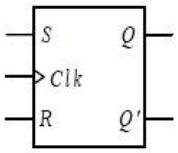
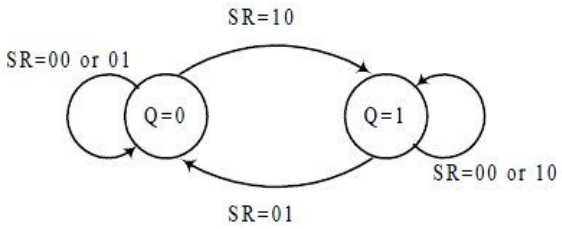
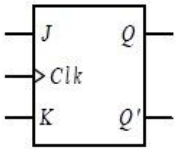
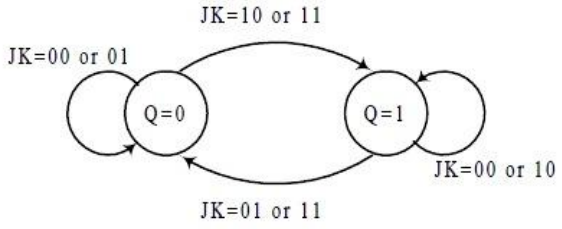
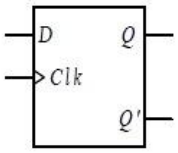
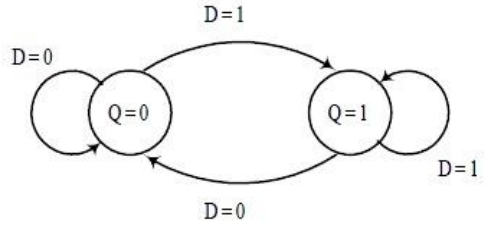
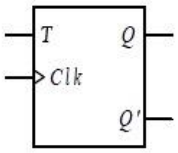
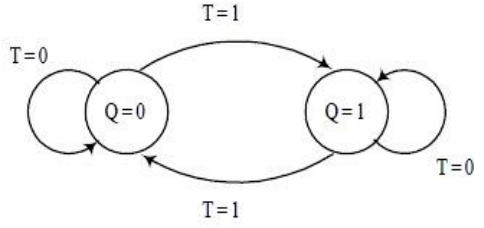
SWITCHES	FPGA PINS
SW4	T14
SW5	T12
SW6	T9
SW7	T7
SW8	T2
SW9	G12
SW10	H1
SW11	R3
SW12	N11
SW13	N3
SW14	M13

LED connections with FPGA

LEDS	FPGA PINS
L16	R1
L15	R2
L14	K3
L13	T4
L12	T5
L11	R6
L10	T8
L9	R10
L8	N10
L7	P12
L6	N9
L5	N12
L4	P13
L3	R13
L2	T13
L1	P14

Type of operation	Operator symbol	Description	Number of operands
Arithmetic	+	addition	2
	-	subtraction	2
	*	multiplication	2
	/	division	2
	%	modulus	2
	**	exponentiation	2
Shift	>>	logical right shift	2
	<<	logical left shift	2
	>>>	arithmetic right shift	2
	<<<	logical left shift	2
Relational	>	greater than	2
	<	less than	2
	>=	greater than or equal to	2
	<=	less than or equal to	2
Equality	==	equality	2
	!=	inequality	2
	===	case equality	2
	!==	case inequality	2
Bitwise	~	bitwise negation	1
	&	bitwise and	2
	|	bitwise or	2
	^	bitwise xor	2
Reduction	&	reduction and	1
	|	reduction or	1
	^	reduction xor	1
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2
Concatenation	{ }	concatenation	any
	{ { } }	replication	any
Conditional	? :	conditional	3

Operator	Precedence
! ~ + - (unary)	highest
**	
* / %	
+ - (binary)	
>> << >>> <<<	
< <= > >=	
== != === !==	
&	
^	
|	
&&	
||	
?:	lowest

Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table
SR 	$S\ R\ Q\ \ Q_{next}$ 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 × 1 1 1 ×	 $Q_{next} = S + R'Q$ $SR = 0$	$Q\ \ Q_{next}\ \ S\ \ R$ 0 0 0 × 0 1 1 0 1 0 0 1 1 1 × 0
JK 	$J\ K\ Q\ \ Q_{next}$ 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 0	 $Q_{next} = J'K'Q + JK' + JKQ'$ $= J'K'Q + JK'Q + JK'Q' + JKQ'$ $= K'Q(J'+J) + JQ'(K'+K)$ $= K'Q + JQ'$	$Q\ \ Q_{next}\ \ J\ \ K$ 0 0 0 × 0 1 1 × 1 0 × 1 1 1 × 0
D 	$D\ \ Q\ \ Q_{next}$ 0 × 0 1 × 1	 $Q_{next} = D$	$Q\ \ Q_{next}\ \ D$ 0 0 0 0 1 1 1 0 0 1 1 1
T 	$T\ \ Q\ \ Q_{next}$ 0 0 0 0 1 1 1 0 1 1 1 0	 $Q_{next} = TQ' + T'Q = T \oplus Q$	$Q\ \ Q_{next}\ \ T$ 0 0 0 0 1 1 1 0 1 1 1 0

ADDITIONAL PROBLEMS

Problems to be done by student

XILINX:

1. **NOR Gate Expression For Y=AB+CD**
2. **NOR GATE EXPRESSION FOR Y=AB+CD**
3. **1:4 Demux**
4. **Full Adder Using NAND Gate**
5. **4:1 mux using 2:1 mux**
6. **8:1 mux using 2:1 mux**
7. **1:8 demux (all 3 modelling)**
8. **BCD Encoder**
9. **Ripple Counter using T Flip Flop**
10. **T,JK, RS,D flip flop**

TANNER

11. **CMOS NAND (S-Edit)**
12. **CMOS NOR (S-Edit)**
13. **CMOS NAND (L-Edit)**
14. **CMOS NOR (L-Edit)**
15. **NOR Gate Expression For Y=AB+CD (S-Edit)**
16. **NOR GATE EXPRESSION FOR Y=AB+CD (S-Edit)**