

||Jai Sri Gurudev||

Sri Adichunchanagiri Shikshana Trust®

BGS INSTITUTE OF TECHNOLOGY

(Affiliated to VTU Belagavi, Approved by AICTE, New Delhi)

BG Nagara – 571448 (Bellur Cross)

Nagamangala Taluk, Mandya District.



VLSI LABORATORY MANUAL

15ECL77

For

VII Semester B.E. E&CE

2019-2020

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

Prepared by:

1. Dr. Naveen B, Asso. Prof
2. Mr. B.S. Balaji, Asst. Prof

Approved by:

Dr. M.B. Anandaraju
Head, Dept. ECE.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision:

To develop high quality engineers with technical knowledge, skills and ethics in the area of Electronics and Communication Engineering to meet industrial and societal needs.

Mission:

1. To provide high quality technical education with up-to-date infrastructure and trained human resources to deliver the curriculum effectively in order to impart technical knowledge and skills.
2. To train the students with entrepreneurship qualities, multidisciplinary knowledge and latest skill sets as required for industry, competitive examinations, higher studies and research activities.
3. To mould the students into professionally-ethical and socially-responsible engineers of high character, team spirit and leadership qualities.

Program Educational Objectives (PEO's):

After 3 to 5 years of graduation, the graduates of Electronics and Communication Engineering will –

1. Engage in industrial, teaching or any technical profession and pursue higher studies and research.
2. Apply the knowledge of Mathematics, Science as well as Electronics and Communication Engineering to solve social engineering problems.
3. Understand, Analyze, Design and Create novel products and solutions.
4. Display professional and leadership qualities, communication skills, team spirit, multidisciplinary traits and lifelong learning aptitude.

Program Specific Outcomes (PSO's):

1. To apply the knowledge of Electronics and Communication Engineering as well as automation tools to create electronic circuits, systems and solutions.
2. To collaborate effectively with Electronics and Information Technology industries through internship, induction, technical seminar, technical project, research, product design and development, industrial visit, staff training in order to provide the actual industrial exposure to students and faculties.

VLSI LABORATORY (15ECL77) SYLLABUS

Course Learning Objectives:

- To educate students with the knowledge of verilog coding and test bench, to write verilog code for all logic gates, flip-flops, counters and adders etc.
- Students will be able to compile, simulate and synthesize the verilog code.
- From this lab the students will be able to draw the schematic diagram and layout for the inverter and amplifiers and verify their functionality.

PART – A

DIGITAL DESIGN

1. Write Verilog Code for the following circuits and their Test Bench for verification, observe the waveform and synthesize the code with the technological library, with the given Constraints*. Do the initial timing verification with gate level simulation
 - 1.An inverter
 - 2.A Buffer
 - 3.Transmission Gate
 - 4.Basic/universal gates
 - 5.Flip flop -RS, D, JK, MS, T
 - 6.Serial & Parallel adder
 - 7.4-bit counter [Synchronous and Asynchronous counter]
 - 8.Successive approximation register [SAR]

* An appropriate constraint should be given

PART – B

ANALOG DESIGN

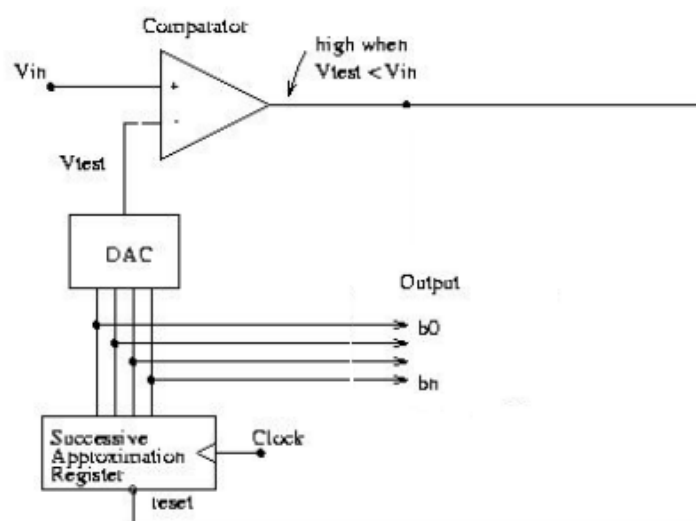
1. Design an Inverter with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following: i) DC Analysis ii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS
 - d. Extract RC and back annotate the same and verify the Design
 - e. Verify & Optimize for Time, Power and Area to the given constraint***

2. Design the following circuits with the given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following: i) DC Analysis
ii) AC Analysis iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS
 - d. Extract RC and back annotate the same and verify the Design.
 - i) A Single Stage differential amplifier
 - ii) Common source and Common Drain amplifier

3. Design an op-amp with the given specification* using given differential amplifier, Common source and Common Drain amplifier in library** and completing the design flow as mentioned below:
 - a. Draw the schematic and verify the following: i) DC Analysis ii) AC Analysis iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS
 - d. Extract RC and back annotate the same and verify the Design.

4. Design a 4 bit R-2R based DAC for the given specification and completing the design flow mentioned using given op-amp in the library**.
 - a. Draw the schematic and verify the following: i) DC Analysis ii) AC Analysis iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS
 - d. Extract RC and back annotate the same and verify the Design.

5. For the SAR based ADC mentioned in the figure below, draw the mixed signal schematic and verify the functionality by completing ASIC Designflow. [Specifications to GDS-II]



* Appropriate specification should be given.

** Applicable Library should be added & information should be given to the Designer.

*** An appropriate constraint should be given.

Beyond Syllabus:

Design layouts using Tanner L-Edit tool for the following circuits:

1. Nand gate
2. Nor gate
3. Transmission gate
4. Xor gate using Transmission gate
5. Xnor gate
6. 4x1 Multiplexer

Course Outcomes:

1. Develop the verilog programs for inverter, buffer, transmission gate and universal gates.
2. Test verilog programs for flip flops, serial, parallel adders and counters.
3. Design basic CMOS circuits like inverter, common source, common drain and differential amplifiers with neat layout.
4. Construct OP-AMP and SAR based DAC circuits with neat layout.

1.INVERTER

Aim: Write Verilog Code for Inverter and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

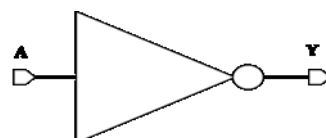
Theory:

CMOS circuits are constructed in such a way that all PMOS transistors must have either an input from the voltage source or from another PMOS transistor. Similarly, all NMOS transistors must have either an input from ground or from another NMOS transistor. The composition of a PMOS transistor creates low resistance between its source and drain contacts when a low gate voltage is applied and high resistance when a high gate voltage is applied. On the other hand, the composition of an NMOS transistor creates high resistance between source and drain when a low gate voltage is applied and low resistance when a high gate voltage is applied. CMOS accomplishes current reduction by complementing every nMOSFET with a pMOSFET and connecting both gates and both drains together. A high voltage on the gates will cause the nMOSFET to conduct and the pMOSFET to not conduct while a low voltage on the gates causes the reverse. This arrangement greatly reduces power consumption and heat generation. However, during the switching time both MOSFETs conduct briefly as the gate voltage goes from one state to another. This induces a brief spike in power consumption and becomes a serious issue at high frequencies.

Applications:

CMOS Inverter is used in digital logic devices for its fast switching operations to change the voltage from one state to another.

Logic Symbol:



Truth table:

| INPUT (A) | OUTPUT (Y) |
|-----------|------------|
| 0 | 1 |
| 1 | 0 |

Circuit diagram:**Programs:****Switch level description**

```
module inverter (a,y) ;  
input a;  
output y;  
supply1 vdd;  
supply0 gnd;  
pmos (y,vdd,a);  
nmos(y,gnd,a);  
endmodule
```

Dataflow description

```
module inverter (a,y) ;  
input a;  
output y;  
assign y = ~a;  
endmodule
```

Structural description

```
module inverter (a,y) ;  
input a;  
output y;  
not(y,a);  
endmodule
```

Behavioural description

```
module inverter (a,y) ;  
input a;  
output y;  
reg y;  
always @ (a)  
begin  
y = ~a;  
end  
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:**Result:**

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

2. BUFFER

Aim: Write Verilog Code for **Buffer** and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

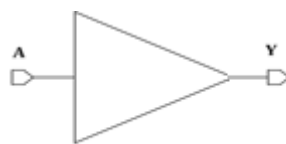
Theory:

Buffers serve to clean up noisy signals, provide delays to correct timing issues and interface small driver gates to large capacitive loads. A buffer is something that isolates or separates one circuit from another. A buffer is a unity-gain amplifier that has an extremely high input resistance and an extremely low output resistance. This means that the buffer can be modelled as a voltage controlled voltage source that has a gain of one. Since the buffer has an infinite input resistance, there is no loading effect. Moreover, we know that the output voltage produced by the buffer must be equal to since it has a gain of 1.

Applications:

Buffer is used to provide delays in on chip interconnects to correct timing issues and synchronise the signals and also to drive the large capacitive loads.

Logic Symbol:



Truth table:

| INPUT (A) | OUTPUT (Y) |
|-----------|------------|
| 0 | 0 |
| 1 | 1 |

Circuit diagram:**Programs:****Switch level description**

```
module buffer (a,y) ;  
input a;  
output y;  
wire s;  
supply1 vdd;  
supply0 gnd;  
pmos (s,vdd,a);  
pmos (y,vdd,s);  
nmos(s,gnd,a);  
nmos(y,gnd,s);  
endmodule
```

Dataflow description

```
module buffer (a,y) ;  
input a;  
output y;  
assign y=~(~a);  
endmodule
```

Structural description

```
module buffer (a,y) ;  
input a;  
output y;  
buf(y,a);  
endmodule
```

Behavioural description

```
module buffer (a,y) ;  
input a;  
output y;  
reg y;  
always @ (a)  
begin  
y=~(~a);  
end  
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:**Result:**

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

3. TRANSMISSION GATE

Aim: Write Verilog Code for **Transmission gate** and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

Theory:

A transmission gate, or analog switch, is defined as an electronic element that will selectively block or pass a signal level from the input to the output. This solid-state switch is comprised of a pMOS transistor and nMOS transistor. The control gates are biased in a complementary manner so that both transistors are either on or off.

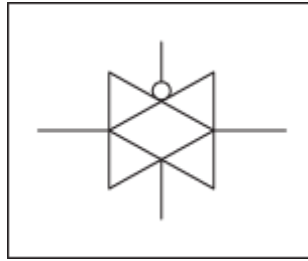
When the voltage on node A is a Logic 1, the complementary Logic 0 is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a Logic 0, the complementary Logic 1 is applied to node A, turning both transistors off and forcing a high-impedance condition on both the IN and OUT nodes. This high-impedance condition represents the third "state" (high, low, or high-Z) that is reflected as downstream.

The schematic diagram includes the arbitrary labels for IN and OUT, as the circuit will operate in an identical manner if those labels were reversed. This design provides true bidirectional connectivity without degradation of the input signal.

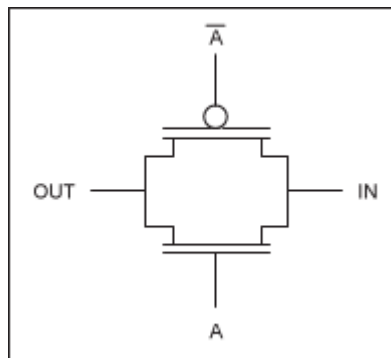
The common circuit symbol for a transmission gate depicts the bidirectional nature of the circuit's operation as shown in figure.

Applications:

Transmission gates are typically used as building blocks for logic circuitry, such as a D Latch or D Flip-Flop. As a stand-alone circuit, a transmission gate can isolate a component or components from live signals during hot insertion or removal. In a security application, they can selectively block critical signals or data from being transmitted without proper hardware-controlled authorization.

Logic Symbol:

Circuit symbol.

Circuit diagram:

Schematic representation of a transmission gate.

Truth table:

| Enable | Input | Output | Operation |
|--------|-------|--------|-----------|
| 0 | 0 | 1 | Inverter |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | Buffer |
| 1 | 1 | 1 | |

Programs:**Switch level description**

```
module tx_gate (a_in, y_out, enable);  
input a_in, enable;  
output y_out;  
nmos n1 (y_out, enable, a_in);  
pmos p1 (y_out, ~enable, a_in);  
endmodule
```

Mixed level description

```
module tx_gate (a_in, y_out, enable);  
input a_in, enable;  
output y_out;  
wire enbar;  
assign enbar = ~enable;  
nmos n1 (y_out, enable, a_in);  
pmos p1 (y_out, enbar, a_in);  
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:**Result:**

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

4. LOGIC GATES

Aim: Write Verilog Code for all the Logic gates and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

Theory:

A logic gate is an electronic circuit/device which makes the logical decisions alternatively a logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic normally performed is Boolean logic and is most commonly found in digital circuits. Logic gates are primarily implemented using diodes or transistors. The logic gates are broadly classified into 3 types:

Basic gates: AND, OR, NOT / INVERTER

Universal gates: NAND, NOR

Special gates: XOR, XNOR


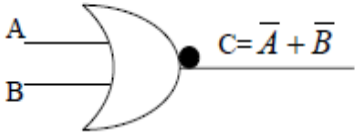
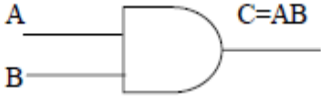
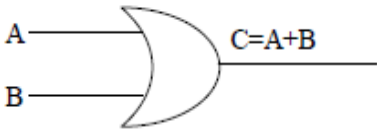
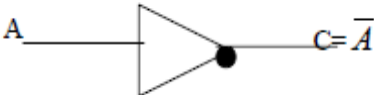
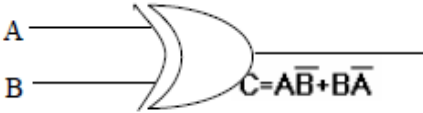
Application:

Data Storage

Logic gates can also be used to store data. A storage element can be constructed by connecting several gates in a "latch" circuit. More complicated designs that use clock signals and that change only on a rising or falling edge of the clock are called edge-triggered "flip-flops". The combination of multiple flip-flops in parallel, to store a multiple-bit value, is known as a register. When using any of these gate setups the overall system has memory; it is then called a sequential logic system since its output can be influenced by its previous state(s).

These logic circuits are known as computer memory. They vary in performance, based on factors of speed, complexity, and reliability of storage, and many different types of designs are used based on the application.

Truth table with symbols:

| S.NO | GATE | SYMBOL | INPUTS | | OUTPUT |
|------|------------------|---|--------|---|--------|
| | | | A | B | C |
| 1. | NAND IC 7400 |  | 0 | 0 | 1 |
| | | | 0 | 1 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 0 |
| 2. | NOR IC 7402 |  | 0 | 0 | 1 |
| | | | 0 | 1 | 0 |
| | | | 1 | 0 | 0 |
| | | | 1 | 1 | 0 |
| 3. | AND IC 7408 |  | 0 | 0 | 0 |
| | | | 0 | 1 | 0 |
| | | | 1 | 0 | 0 |
| | | | 1 | 1 | 1 |
| 4. | OR IC 7432 |  | 0 | 0 | 0 |
| | | | 0 | 1 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 1 |
| 5. | NOT IC 7404 |  | 1 | - | 0 |
| | | | 0 | - | 1 |
| 6. | EX-OR IC 7486 |  | 0 | 0 | 0 |
| | | | 0 | 1 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 0 |

Programs:**Dataflow Description**

```

module gates(a, b, y);
  input a;
  input b;
  output [6:0] y;
  assign y[0] = a & b;
  assign y[1] = a | b;
  assign y[2] = ~(a & b);
  assign y[3] = ~(a | b);
  assign y[4] = a ^ b;
  assign y[5] = ~(a ^ b);
  assign y[6] = ~a;
endmodule

```

Behavioural Description

```
module gates(a, b, y);
  input a;
  input b;
  output [6:0] y;
  reg [6:0] y;
always @(a,b)
begin
y[0] = a & b;
y[1] = a | b;
y[2] = ~(a & b);
y[3] = ~(a | b);
y[4] = a ^ b;
y[5] = ~(a ^ b);
y[6] = ~a;
end
endmodule
```

Structural Description

```
module gates(a, b, y);
  input a;
  input b;
  output [6:0] y;
  and (y[0], a, b);
  or (y[1], a, b);
  nand (y[2], a, b);
  nor (y[3], a, b);
  xor (y[4], a, b);
  xnor (y[5], a, b);
  not (y[6], a);
end
endmodule
```

Verilog Test Fixture or Test bench code:

Programs:**NAND GATE**

```
module nandgate ( out , a , b );  
output out;  
input a,b;  
supply1 vdd;  
supply0 gnd;  
wire contact;  
pmos (out,vdd,a);  
pmos (out,vdd,b);  
nmos (out,contact,a);  
nmos (contact,gnd,b);  
endmodule
```

NOR GATE

```
module norgate ( out , a , b );  
output out;  
input a,b;  
supply1 vdd;  
supply0 gnd;  
wire contact;  
pmos (contact,vdd,a);  
pmos (out,contact,b);  
nmos (out,gnd,a);  
nmos (out,gnd,b);  
endmodule
```

AND GATE

```
module andgate ( out ,a , b );  
output out;  
input a,b;  
supply1 vdd;  
supply0 gnd;  
wire contact;  
wire nout  
pmos (nout,vdd,a);  
pmos (nout,vdd,b);  
nmos (nout,contact,a);  
nmos (contact,gnd,b);  
pmos (out,vdd,nout);  
nmos (out,gnd,nout);  
endmodule
```

OR GATE

```
module orgate ( out , a , b );  
output out;
```

```
input a,b;  
supply1 vdd;  
supply0 gnd;  
wire contact;  
wire nout;  
pmos (contact,vdd,a);  
pmos (nout,contact,b);  
nmos (nout,gnd,a);  
nmos (nout,gnd,b);  
pmos (out,vdd,nout);  
nmos (out,gnd,nout);  
endmodule
```

XNOR GATE

```
module xnorgate ( out , a , b );  
output out;  
input a,b;  
wire bbar;  
assign bbar = ~b;  
pmos (out,bbar,a);  
nmos (out,b,a);  
pmos (out,a,bbar);  
nmos (out,a,b);  
endmodule
```

XOR GATE

```
module xorgate ( out , a , b );  
output out;  
input a,b;  
wire bbar;  
assign bbar = ~b;  
// Instantiate pmos and nmos switches :  
pmos (out,b,a);  
nmos (out,bbar,a);  
pmos (out,a,b);  
nmos (out,a,bbar);  
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:

Result:

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

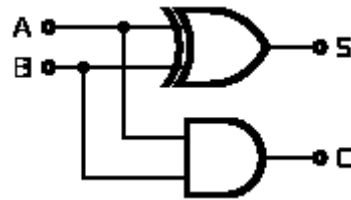
5. PARALLEL ADDER AND SERIAL ADDER

Aim: Write Verilog Code for 4 bit Parallel (Ripple-carry) adder and Serial adder circuits and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

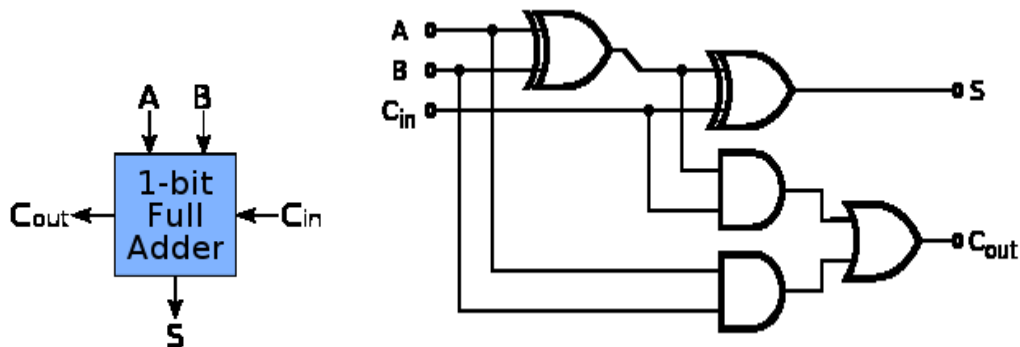
Theory:

Full 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 generate carry and sum which are the two outputs of half-adder.



Half Adder

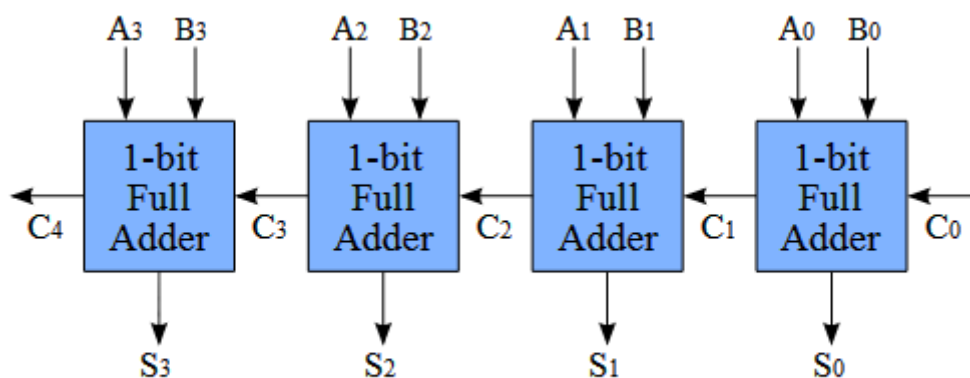


Full Adder

The one-bit full adder's truth table is:

| INPUTS | | | OUPUTS | |
|--------|---|-----|--------|------|
| A | B | Cin | SUM | Cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Ripple Carry Adder



It is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of adder

is called a ripple-carry adder, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

The layout of a ripple-carry adder is simple, which allows for fast design time; however, the ripple-carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit.

Serial Adder

Serial binary addition is done by a flip-flop and a full adder. The flip-flop takes the carry-out signal on each clock cycle and provides its value as the carry-in signal on the next clock cycle. After all of the bits of the input operands have arrived, all of the bits of the sum have come out of the sum output.

Example

Decimal $5+9=14$

$X=5, Y=9, \text{Sum}=14$

Binary $0101+1001=1110$

Addition of each step

| Inputs | | | Outputs | |
|--------|---|---|---------|------|
| Cin | X | Y | Sum | Cout |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |

**addition starts from lowest*

Result=1110 or 14

Programs:**Half adder**

```
module HA(x,y,s,c);
input x,y;
output s,c;
xor (s,x,y);
and (c,x,y);
endmodule
```

Full adder

```
module FA(a,b,ci,sum,cout);
input a,b,ci;
output sum,cout;
wire c0,c1,c2;
HA h1 (a,b,c0,c1);
HA h2 (ci,c0,sum,c2);
or (cout,c1,c2);
endmodule
```

Parallel adder

```
module PA ( carryin, x, y, sum, carryout);
input carryin;
input [3:0] x, y;
output [3:0] sum;
output carryout;
FA s0 (carryin, x[0], y[0], sum[0], c1);
FA s1 (c1, x[1], y[1], sum[1], c2);
FA s2 (c2, x[2], y[2], sum[2], c3);
FA s3 (c3, x[3], y[3], sum[3], carryout);
endmodule
```

Serial Adder

```
module sradd(a,b,start,clock,ready,result);
input a,b,start,clock;
output ready;
output [3:0] result;
reg [3:0] result;
reg sum,carry,ready;
integer count;
initial count = 4;
always @(negedge clock)
begin
if (start)
begin
count =0;
carry = 0;
```

```
result = 0;
end
else
begin
if (count <4)
begin
count = count + 1;
sum = a ^ b ^ carry ;
carry = (a&b)|(a & carry)|(b& carry);
result = {sum,result[3:1]};
end
end
if(count == 4)
ready = 1;
else
ready = 0;
end
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:

Result:

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

6. FLIPFLOPS

Aim: Write Verilog Code for Flipflops – SR, JK, D, T and MS-JK circuits and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

Theory:

SR flip-flop: A SR flip - flop is the simplest possible memory element. The SR flip flop has two inputs Set and Reset. The SR flip-flop is a basic building block for other flip-flops.

D flip-flop: This is a flip - flop with a delay (D) equal to exactly equal to one cycle of the clock. The defect with SR FF is the indeterminate output when the data inputs at S and R are 1. In order to avoid this input to R is through an inverter from S so that the input to R is always the complement of S and never same. The S input is redesignated as D.

JK flip-flop: The JK flip flop is called a “universal flip flop” because the other flip flops like D, SR, T can be derived from it. The “racing or race around condition” takes place in a JK FF when $J=1$ and $K=1$ and $clock=1$.

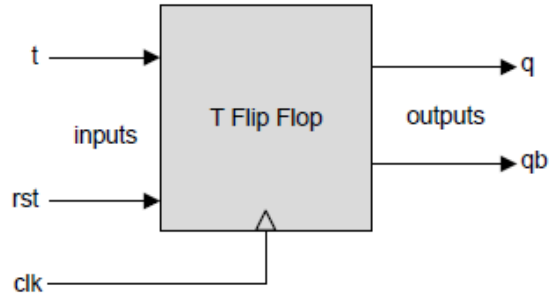
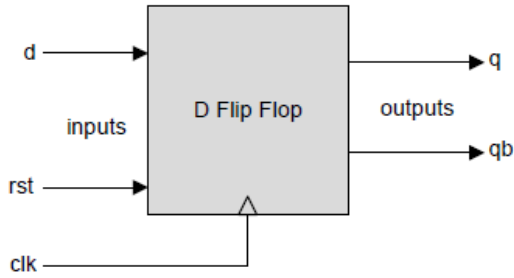
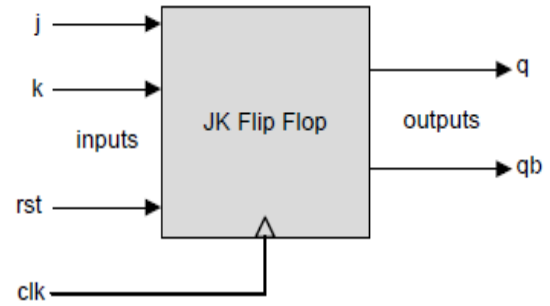
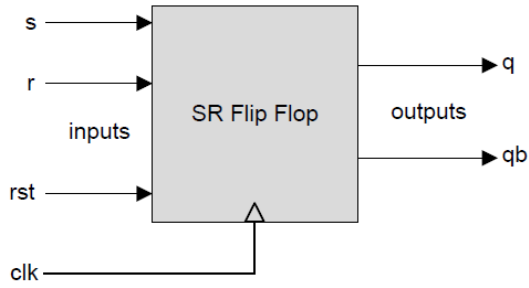
T flip-flop: T stands for toggling. It is obtained from JK FF by tying both the inputs J and K.

MS JK flip-flop: It consists of two flipflops (Master and Slave) sequentially connected to each other. Slave and Master flipflop perform the same operation but at inverted clock signal input or vice versa.

Applications:

- Event Detect
- Data Synchronizer
- Frequency Divider
- Shift Register
- Asynchronous (Ripple) Counter
- Synchronous (Parallel) Counter.

Logic Symbol:



Truth table:

| Inputs | | | | Outputs | | |
|--------|-----|---|---|---------|----|-----------|
| rst | clk | s | r | q | qb | Action |
| 1 | ↑ | X | X | q | qb | No Change |
| 0 | ↑ | 0 | 0 | q | qb | No Change |
| 0 | ↑ | 0 | 1 | 0 | 1 | Reset |
| 0 | ↑ | 1 | 0 | 1 | 0 | Set |
| 0 | ↑ | 1 | 1 | - | - | Illegal |

S R Flip Flop

| Inputs | | | | Outputs | | |
|--------|-----|---|---|---------|----|-----------|
| rst | clk | j | k | q | qb | Action |
| 1 | ↑ | X | X | q | qb | No Change |
| 0 | ↑ | 0 | 0 | q | qb | No Change |
| 0 | ↑ | 0 | 1 | 0 | 1 | Reset |
| 0 | ↑ | 1 | 0 | 1 | 0 | Set |
| 0 | ↑ | 1 | 1 | q' | q' | Toggle |

J K Flip Flop

| Inputs | | | | Outputs | |
|--------|-----|---|---|---------|-----------|
| rst | clk | d | q | qb | Action |
| 1 | ↑ | X | q | qb | No Change |
| 0 | ↑ | 0 | 0 | 1 | Reset |
| 0 | ↑ | 1 | 1 | 0 | Set |

D Flip Flop

| Inputs | | | | Outputs | |
|--------|-----|---|----|---------|-----------|
| rst | clk | t | q | qb | Action |
| 1 | ↑ | X | q | qb | No Change |
| 0 | ↑ | 0 | q | qb | No Change |
| 0 | ↑ | 1 | q' | q' | Toggle |

T Flip Flop

Programs:**SR Flipflop**

```
module sr_ff( sr , clk , reset , q ,qb );  
input [1:0] sr;  
input clk, reset ;  
output q,qb;  
reg q,qb;  
always @ ( posedge clk or posedge reset)  
if (reset)  
begin  
q = 1'b0;  
qb = ~q;  
end  
else  
begin  
case (sr)  
2'd0 : q = q;  
2'd1 : q = 1'b0;  
2'd2 : q = 1'b1;  
2'd3 : q = 1'bX;  
endcase  
qb = ~q;  
end  
endmodule
```

JK Flipflop

```
module jk_ff( jk , clk , reset , q ,qb );  
input [1:0] jk;  
input clk, reset ;  
output q,qb;  
reg q,qb;  
always @ ( posedge clk or posedge reset)  
if (reset)  
begin  
q = 1'b0;  
qb = ~q;  
end  
else  
begin  
case (jk)  
2'd0 : q = q;  
2'd1 : q = 1'b0;  
2'd2 : q = 1'b1;  
2'd3 : q = ~q;  
endcase  
qb = ~q;  
end  
endmodule
```

D Flipflop

```
module d_ff( d , clk , reset , q ,qb );
input d, clk, reset ;
output q,qb;
reg q,qb;
always @( posedge clk or posedge reset)
if (reset)
begin
q = 1'b0;
qb=~q;
end
else
begin
q = d;
qb=~q;
end
endmodule
```

T Flipflop

```
module t_ff( t, clk, reset, q, qb );
input t, clk, reset ;
output q,qb;
reg q,qb;
always @( posedge clk or posedge reset)
if (reset)
begin
q = 1'b0;
qb=~q;
end
else
if (t)
begin
q = ~q;
qb = ~q;
end
endmodule
```

MS JK Flipflop

```
module ms_jkff(q, q_bar, clk, j, k);
output q, q_bar;
input clk, j, k;
reg q, tq, q_bar;
always @(clk)
begin
if (!clk)
begin
if (j==1'b0 && k==1'b0)
tq = tq;
else if (j==1'b0 && k==1'b1)
```

```
tq=1'b0;
else if (j==1'b1 && k==1'b0)
tq = 1'b1;
else if (j==1'b1 && k==1'b1)
tq = ~tq;
end
if (clk)
begin
q = tq;
q_bar = ~tq;
end
end
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:

Result:

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

7. COUNTERS

Aim: Write Verilog Code 4 bit Synchronous (Parallel) and Asynchronous (Ripple) counter circuits and their Test Bench for **verification**, observe the waveform and **synthesise** the code.

Theory:

In electronics, counters can be implemented quite easily using register-type circuits such as the flip-flop, and a wide variety of classifications exist:

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

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

Decade counter – counts through ten states per stage

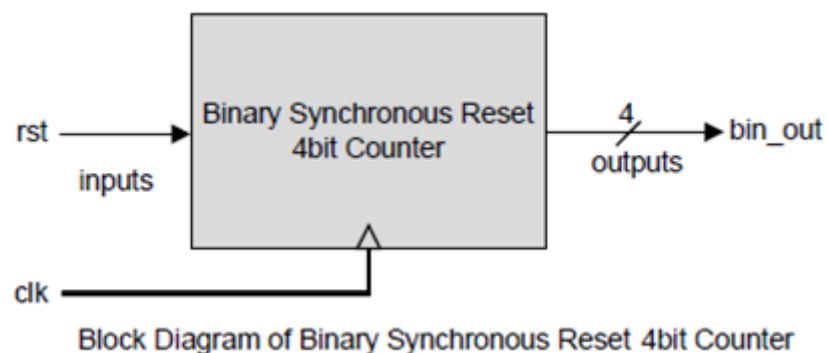
Up/down counter – counts both up and down, under command of a control input

Ring counter – formed by a shift register with feedback connection in a ring

Johnson counter – a twisted ring counter.

Application - Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

Logic Symbol:



Truth table:

| Clk | Rst | bin_out |
|-----|-----|---------|
| x | 1 | 0000 |
| 1 | 0 | 0001 |
| 1 | 0 | 0010 |
| 1 | 0 | 0011 |
| 1 | 0 | 0100 |
| 1 | 0 | 0101 |
| 1 | 0 | 0110 |
| 1 | 0 | 0111 |
| 1 | 0 | 1000 |
| 1 | 0 | 1001 |
| 1 | 0 | 1010 |
| 1 | 0 | 1011 |
| 1 | 0 | 1100 |
| 1 | 0 | 1101 |
| 1 | 0 | 1110 |
| 1 | 0 | 1111 |

Programs:**SYNCHRONOUS COUNTER**

```
module counter_behav ( count, reset, clk);  
input reset, clk;  
output [3:0] count;  
reg [3:0] count;  
always @(posedge clk)  
if (reset)  
count = 4'b0000;  
else  
count = count + 4'b0001;  
endmodule
```

Verilog Test Fixture or Test bench code:

Programs:**ASYNCHRONOUS COUNTER [RIPPLE COUNTER]**

```
module ripple_counter (clock, toggle, reset, count);  
input clock, toggle, reset;  
output [3:0] count;  
reg [3:0] count;  
wire c0, c1, c2;  
assign c0 = count[0], c1 = count[1], c2 = count[2];  
always @ (posedge reset or posedge clock)  
if (reset == 1'b1)  
count[0] = 1'b0;  
else if (toggle == 1'b1)  
count[0] = ~count[0];  
always @ (posedge reset or negedge c0)  
if (reset == 1'b1)  
count[1] = 1'b0;  
else if (toggle == 1'b1)  
count[1] = ~count[1];  
always @ (posedge reset or negedge c1)  
if (reset == 1'b1)  
count[2] = 1'b0;  
else if (toggle == 1'b1)  
count[2] = ~count[2];  
always @ (posedge reset or negedge c2)  
if (reset == 1'b1)  
count[3] = 1'b0;  
else if (toggle == 1'b1)  
count[3] = ~count[3];  
endmodule
```

Verilog Test Fixture or Test bench code:

Procedure:

1. Create a new project and a new Verilog source.
2. Write the Verilog code and save it.
3. Perform Check Syntax and check for errors. Synthesise the code.
4. Create the test bench code and provide the inputs.
5. Simulate the test bench code and realise the output.

Simulation Output:**Result:**

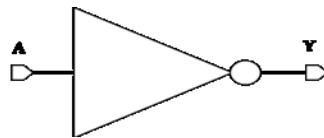
| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

1. INVERTER

Aim: Design an **Inverter** with given specifications*, completing the design flow mentioned below:

- a. **Draw the schematic** and verify the following
 - i) DC Analysis
 - ii) Transient Analysis
- b. **Draw the Layout** and verify the DRC, ERC
- c. **Check for LVS**

Logic Symbol:

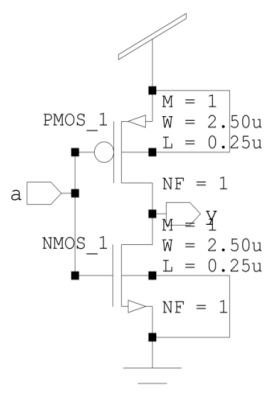


Truth table:

| INPUT (A) | OUTPUT (Y) |
|-----------|------------|
| 0 | 1 |
| 1 | 0 |

Circuit diagram:

Schematic design using S-Edit:



Layout using L-Edit:

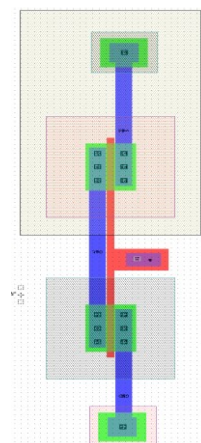


Figure 1(a) Inverter Schematic design using S-Edit.

Figure 1(b) Inverter Layout design using L-Edit.

T-Spice Netlist:**Using S-Edit:**

MPMOS_1 **Out In Vdd Vdd** PMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_1 **Out In Gnd Gnd** NMOS W=2.5u L=250n AS=2.25p PS=6.8u AD=2.25p PD=6.8u

Using L-Edit:

M1 **Out A vdd vdd** PMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M2 **Out A GND GND** NMOS L=250n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u

T-Spice Commands:

```
.tran 10n 100n
.dc lin source v1 0 5 0.1
.lib "C:\Tanner\vtuexperiments\INV\Generic_025.lib" TT
v1 a GND BIT {{10101}}
v2 vdd GND 5
.print tran v(a) v(y)
.print dc v(y)
.op
.end
```

Procedure:**S-Edit:**

1. Create New Schematic Design using Libraries in S-Edit.
2. Open New Cell and design the circuit.
3. Check for errors in status window.
4. Save the Design.
5. Export Spice file.

L-Edit:

1. Create New Layout Design using Libraries in L-Edit.
2. Run DRC to check errors in Layout.
3. Save the Design.
4. Extract the netlists of Layout and export Spice file using EXT.

T-Spice:

1. Open T-Spice file and insert the commands.
2. Save the file.
3. Run Simulation.

LVS:

1. Create new file.
2. Select for two spice file from Layout and Schematic design.
3. Run Verification.

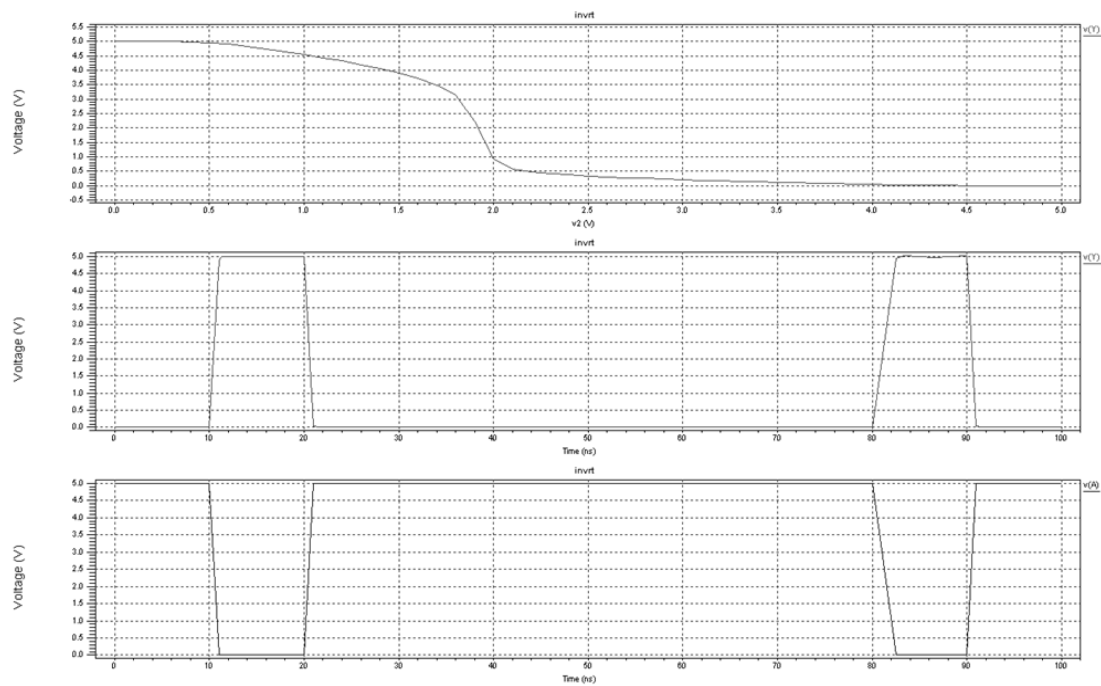
Waveform:

Figure 1.2 Simulation output waveform of Inverter.

Result:

Circuits are equal.

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

2. SINGLE STAGE DIFFERENTIAL AMPLIFIER

Aim: Design the **Single Stage Differential Amplifier** with given specifications*, completing the design flow mentioned below:

- a. **Draw the schematic** and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
- b. **Draw the Layout** and verify the DRC, ERC
- c. **Check for LVS**

Design:

NMOS

$$U_n = 0.0278$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = 0.3611$$

$$C_{ox} = 6.058 \times 10^{-3}$$

PMOS

$$U_n = 0.01047$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = -0.55186$$

$$C_{ox} = 6.058 \times 10^{-3}$$

$$\text{Let } I_D = 80 \mu\text{A}$$

For M_5

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 80 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.3611^2} \\ &= 7.286 \end{aligned}$$

$$\text{Let } L = 0.4 \mu$$

$$W = 2.88 \mu$$

For M_3, M_4

$$W/L = \frac{2 I_D}{U_n C_{ox} V_{dsat}^2}$$

$$= \frac{2 \times 40 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.3611^2}$$

$$= 3.6$$

Let $L = 0.69\mu$

$W = 2.5\mu$

For M_1, M_2

$$W/L = \frac{2 I_D}{U_n C_{ox} V_{dsat}^2}$$

$$= \frac{2 \times 40 \times 10^{-6}}{0.01047 \times 6.058 \times 10^{-3} \times 0.55816^2}$$

$$= 4.14$$

Let $L = 0.603\mu$

$W = 2.5\mu$

CALCULATION FOR GAIN

$$A_v = -g_{m1} (r_{o1} || r_{o2})$$

$$r_o = \frac{V_{An}}{I_D}$$

$$g_{m1} = \sqrt{2 \times k'_n \times \frac{W}{L} \times I_{ref}}$$

$$= \sqrt{\frac{2 \times 0.0278 \times 6.058 \times 10^{-3} \times 2.5 \times 40 \times 10^{-6}}{0.699}}$$

$$= 2.1 \times 10^{-4} \text{ A/V}$$

$$r_{o1} = \frac{V_{An}}{I_{D1}} = \frac{1.3}{40 \times 10^{-6}} = 32500 \text{ Ohm}$$

$$r_{o2} = \frac{V_{Ap}}{I_{D2}} = \frac{1.3}{40 \times 10^{-6}} = 32500 \text{ Ohm}$$

$$A_v = -g_{m1} (r_{o1} || r_{o2})$$

$$= 3.41$$

$$\text{Gain in dB} = \frac{20 \log 3.4125}{\log 2}$$

$$= 35 \text{ dB (Theoretical Gain)}$$

Practically $\rightarrow 30\text{dB}$

Circuit diagram:

Schematic Design using S-Edit:

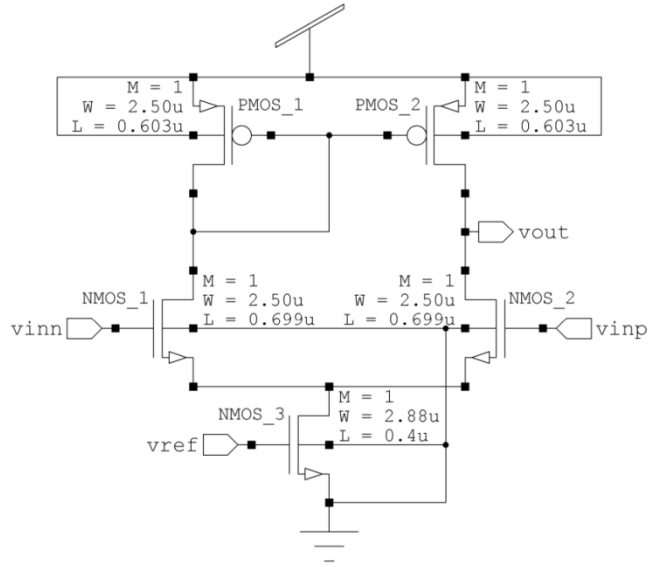


Figure 2.1 Single stage differential amplifier Schematic design using S-Edit.

Layout Design using L-Edit:

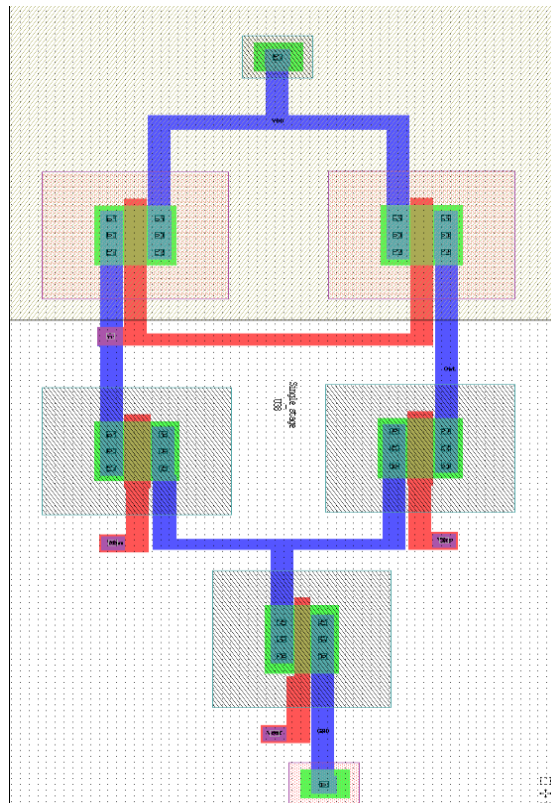


Figure 2.2 Single stage differential amplifier Layout design using L-Edit.

T-Spice Netlist:**Using S-Edit:**

```
MPMOS_1 N_1 N_1 Vdd Vdd PMOS W=2.5u L=603n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_2 Out N_1 Vdd Vdd PMOS W=2.5u L=603n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
NMNOS_1 N_1 vinn N_2 Gnd NMOS W=2.5u L=699n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
NMNOS_2 Out vinp N_2 Gnd NMOS W=2.5u L=699n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
NMNOS_3 N_2 vref Gnd Gnd NMOS W=2.88u L=400n AS=2.592p PS=7.56u AD=2.592p PD=7.56u
```

Using L-Edit:

```
M1 VDD 3 3 VDD PMOS L=603n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u
M2 Out 3 VDD VDD PMOS L=603n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u
M3 2 Vinn 3 GND NMOS L=699n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u
M4 GND Vref 2 GND NMOS L=400n W=2.88u AD=2.304p PD=7.36u AS=2.304p PS=7.36u
M5 Out Vinp 2 GND NMOS L=699n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u
```

T-Spice Commands:

```
.tran .5m 5m
.ac dec 10 10 10g
.dc lin source v4 -5 5 .1
.lib"C:\Users\Balaji\Documents\TannerEDA\TannerToolsv13.0\Libraries\Models\Generi
c_025.lib" tt
v1 Vdd Gnd 5
v2 vref Gnd 0.8
v3 vinn Gnd 1.3
v4 vinp vinn SIN (0 2m 1k) AC 1
.print ac vdb(out)
.print dc v(out)
.print tran v(out) v(vinp)
.op
.END
```

Procedure:**S-Edit:**

1. Create New Schematic Design using Libraries in S-Edit.
2. Open New Cell and design the circuit.
3. Check for errors in status window.
4. Save the Design.
5. Export Spice file.

L-Edit:

1. Create New Layout Design using Libraries in L-Edit.
2. Run DRC to check errors in Layout.
3. Save the Design.
4. Extract the netlists of Layout and export Spice file using EXT.

T-Spice:

1. Open T-Spice file and insert the commands.
2. Save the file.
3. Run Simulation.

LVS:

1. Create new file.
2. Select for two spice file from Layout and Schematic design.
3. Run Verification.

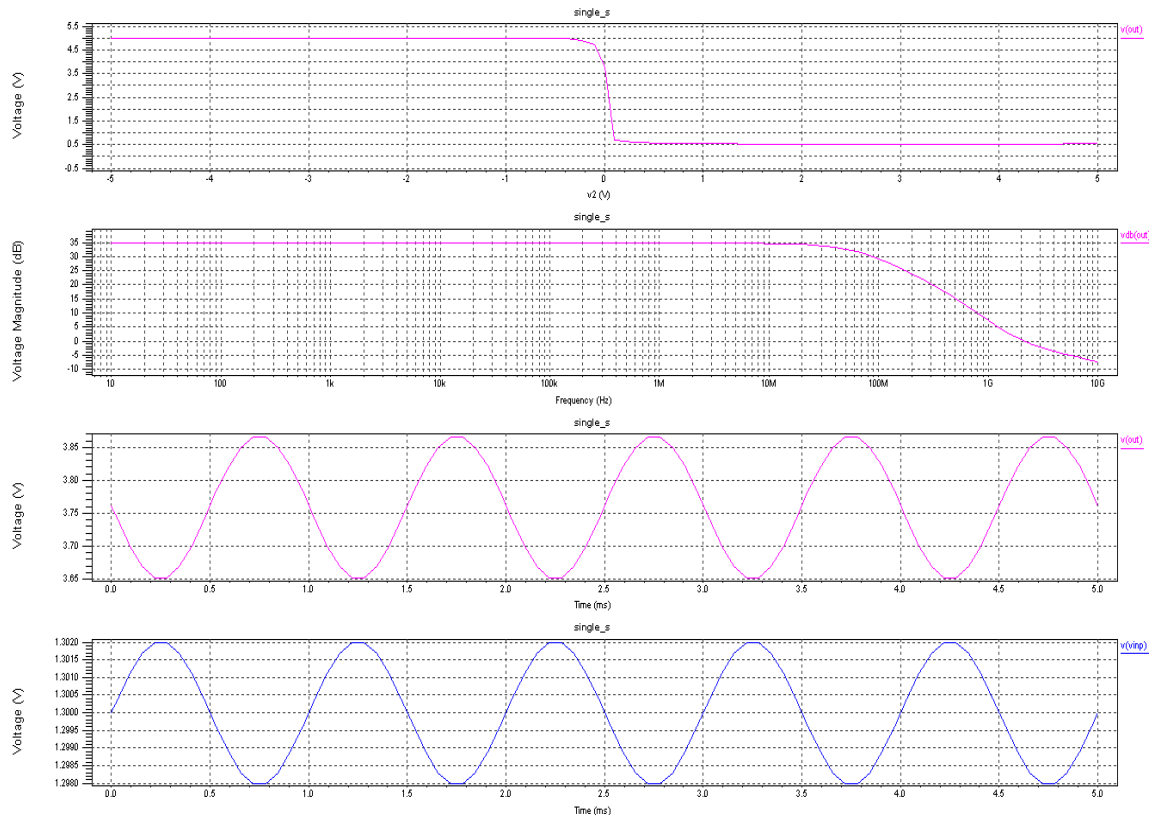
Waveform:

Figure 2.3 Simulation output waveform of Single stage differential amplifier.

Result:

Circuits are equal.

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

3. COMMON SOURCE AMPLIFIER

Aim: Design the **Common Source Amplifier** with given specifications*, completing the design flow mentioned below:

- a. **Draw the schematic** and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
- b. **Draw the Layout** and verify the DRC, ERC
- c. **Check for LVS**

Design:

COMMON SOURCE

NMOS

$$U_n = 0.0278$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = 0.385$$

$$C_{ox} = 6.058 \times 10^{-3}$$

PMOS

$$U_n = 0.0105$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = -0.545$$

$$C_{ox} = 6.058 \times 10^{-3}$$

Let $I_D = 80 \mu A$

For M1, M8

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 80 \times 10^{-6}}{0.0105 \times 6.058 \times 10^{-3} \times 0.545^2} \\ &= 8.45 \end{aligned}$$

Let $W=2.5\mu$

$L=.29\mu$

For M9, M7

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 80 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.385^2} \\ &= 6.4 \end{aligned}$$

Let $W=2.5\mu$

$L=.4\mu$

For M2, M3

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 40 \times 10^{-6}}{0.0105 \times 6.058 \times 10^{-3} \times 0.545^2} \\ &= 4.23 \end{aligned}$$

Let $W=2.5\mu$

$L=.6\mu$

For M4, M5

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 40 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.385^2} \\ &= 3.2 \end{aligned}$$

Let $W=2.5\mu$

$L=.77\mu$

For M6

$$\begin{aligned}
 W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\
 &= \frac{2 \times 80 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.385^2} \\
 &= 6.4
 \end{aligned}$$

Let $W=2.5\mu$ $L=.385\mu$ **CALCULATION FOR GAIN**

$$A_v = -g_{m1} (r_{o1} || r_{o2})$$

$$r_o = \frac{V_{An}}{I_D}$$

$$g_{m1} = \sqrt{2 \times k'_n \times \frac{W}{L} \times I_{ref}}$$

$$\begin{aligned}
 &= \sqrt{\frac{2 \times 0.0105 \times 6.058 \times 10^{-3} \times 1.5 \times 40 \times 10^{-6}}{0.6}} \\
 &= 1.457 \times 10^{-4} \text{ A/V}
 \end{aligned}$$

$$r_{o1} = \frac{V_{An}}{I_{D1}} = \frac{3.2}{40 \times 10^{-6}} = 80000 \text{ Ohm}$$

$$r_{o2} = \frac{V_{Ap}}{I_{D2}} = \frac{3.2}{40 \times 10^{-6}} = 80000 \text{ Ohm}$$

$$\begin{aligned}
 A_v &= -g_{m1} (r_{o1} || r_{o2}) \\
 &= 5.828
 \end{aligned}$$

$$\text{Gain in dB} = \frac{20 \log 5.828}{\log 2}$$

$$= 51 \text{ dB (Theoretical Gain)}$$

Practically \rightarrow 50 dB

Circuit diagram:

Schematic Design using S-Edit:

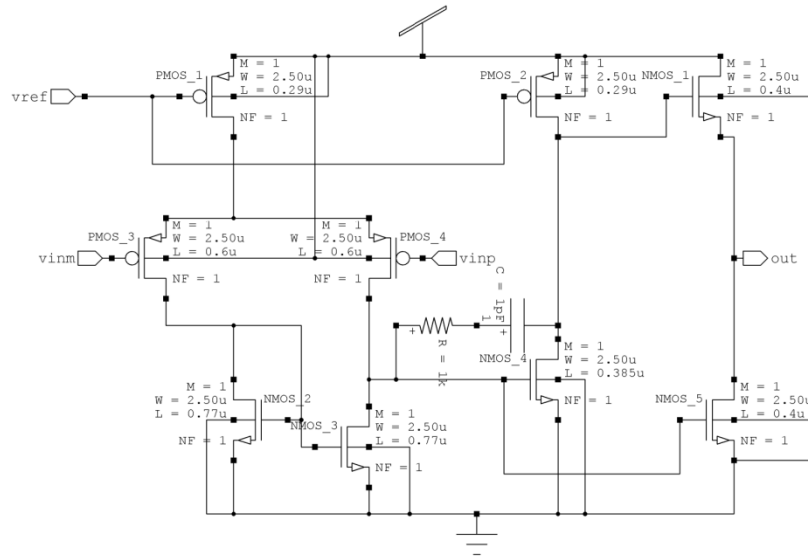


Figure3.1 Common source amplifier Schematic design using S-Edit.

Layout Design using L-Edit:

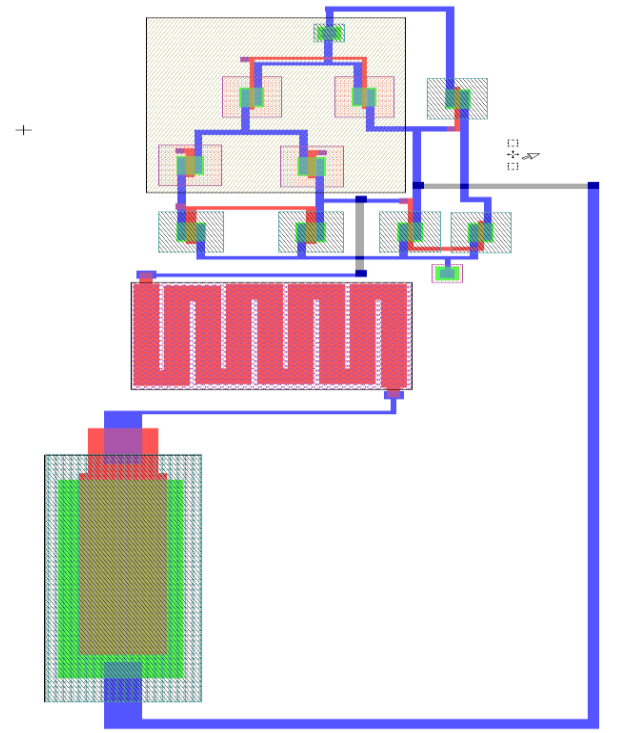


Figure 3.2 Common source amplifier Layout design using L-Edit.

T-Spice Netlist:**Using S-Edit:**

```

MPMOS_1 N_1 vref Vdd Vdd PMOS W=2.5u L=290n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
RResistor_1 N_4 N_5 R=1k
MPMOS_2 N_2 vref Vdd Vdd PMOS W=2.5u L=290n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_1 N_3 N_3 Gnd Gnd NMOS W=2.5u L=770n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_3 N_3 vinm N_1 Vdd PMOS W=2.5u L=600n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_2 N_4 N_3 Gnd Gnd NMOS W=2.5u L=770n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_4 N_4 vinp N_1 Vdd PMOS W=2.5u L=600n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_3 N_2 N_4 Gnd Gnd NMOS W=2.5u L=385n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_4 Out N_4 Gnd Gnd NMOS W=2.5u L=400n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_5 Vdd N_2 Out Gnd NMOS W=2.5u L=400n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
CCapacitor_1 N_5 N_2 1p

```

Using L-Edit:

```

M1 1 2 Gnd Gnd NMOS L=385n W=2.5u AD=1.9975p PD=6.598u AS=2p PS=6.6u
M2 Out 2 Gnd Gnd NMOS L=400n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M3 2 4 Gnd Gnd NMOS L=770n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M4 Gnd 4 4 Gnd NMOS L=770n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M5 Out 1 Vdd Gnd NMOS L=400n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
C1 5 1 1.0001444p
M6 11 Vinm 4 Vdd PMOS L=600n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M7 2 Vinp 11 Vdd PMOS L=600n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M8 Vdd Vbias 11 Vdd PMOS L=290n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
M9 1 Vbias Vdd Vdd PMOS L=290n W=2.5u AD=2p PD=6.6u AS=2p PS=6.6u
R1 5 2 1.0416k

```

T-Spice Commands:

```

.tran .01m 5m
.dc lin source v4 -5 5 .1
.ac dec 10 1 10g
.lib"C:\Users\Balaji\Documents\TannerEDA\TannerToolsv13.0\Libraries\Models\Generi
c_025.lib" tt
v1 vdd gnd 5
v2 vref gnd 3.87
v3 vinm gnd 3.2
v4 vinp vinm SIN (0 2m 1k) AC 1
.print ac vdb(out) vp(out)
.print dc v(out)
.print tran v(out) v(vinp)
.op
.end

```

Procedure:**S-Edit:**

1. Create New Schematic Design using Libraries in S-Edit.
2. Open New Cell and design the circuit.
3. Check for errors in status window.
4. Save the Design.
5. Export Spice file.

L-Edit:

1. Create New Layout Design using Libraries in L-Edit.
2. Run DRC to check errors in Layout.
3. Save the Design.
4. Extract the netlists of Layout and export Spice file using EXT.

T-Spice:

1. Open T-Spice file and insert the commands.
2. Save the file.
3. Run Simulation.

LVS:

1. Create new file.
2. Select for two spice file from Layout and Schematic design.
3. Run Verification.

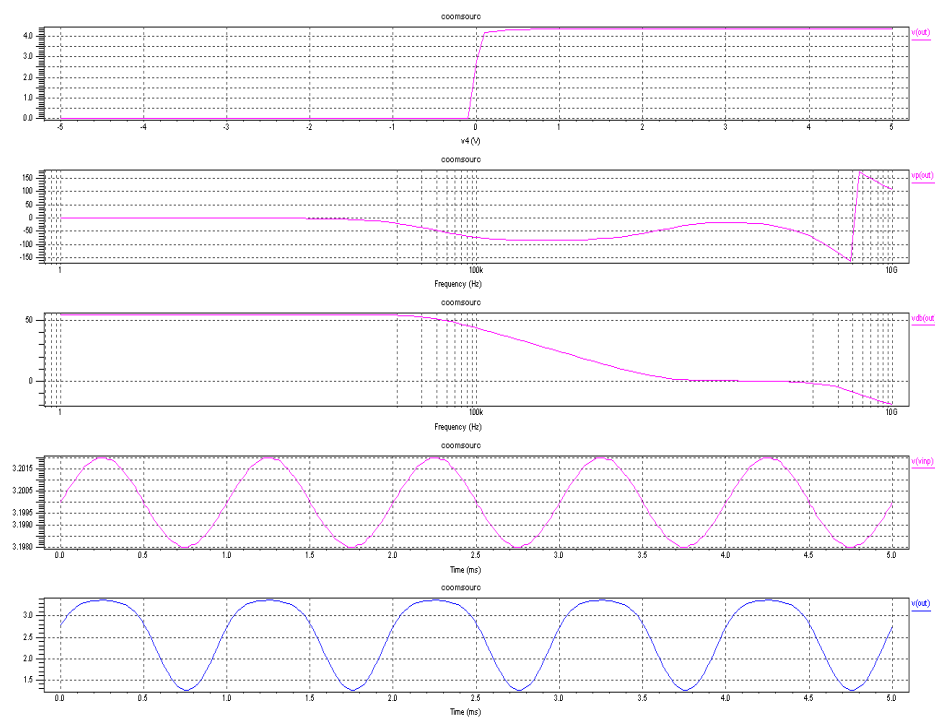
Waveform:

Figure 3.3 Simulation output waveform of Common source amplifier.

Result:

Circuits are equal.

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

4. COMMON DRAIN AMPLIFIER

Aim: Design the **Common Drain Amplifier** with given specifications*, completing the design flow mentioned below:

- a. **Draw the schematic** and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
- b. **Draw the Layout** and verify the DRC, ERC
- c. **Check for LVS**

Circuit diagram:

Schematic Design using S-Edit:

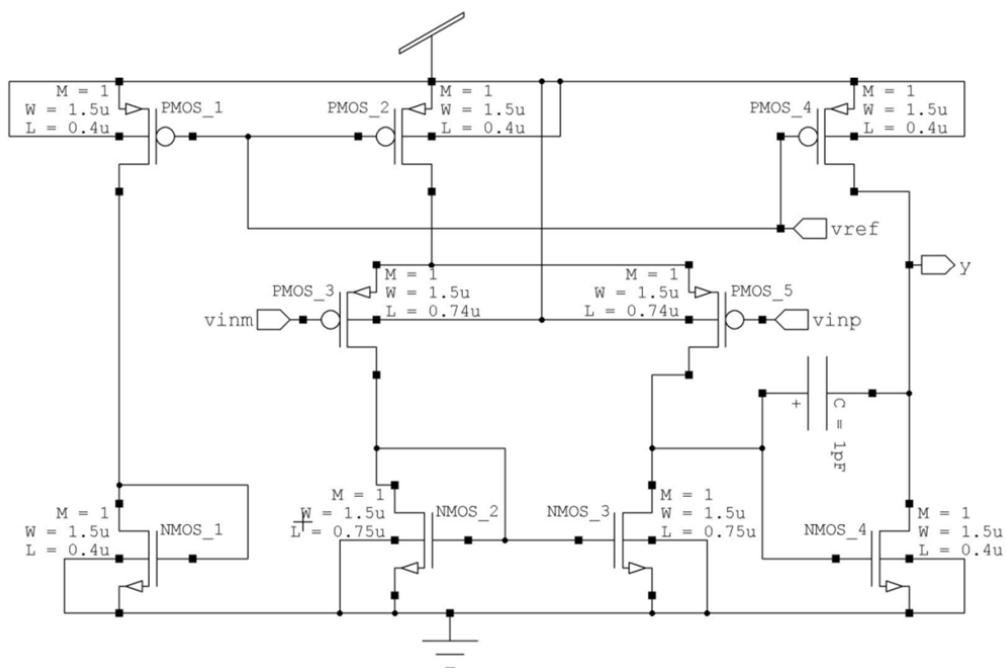


Figure 4.1 Common drain amplifier Schematic design using S-Edit.

Design:**COMMON DRAIN****NMOS**

$$U_n = 0.0278$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = 0.35$$

$$C_{ox} = 6.058 \times 10^{-3}$$

PMOS

$$U_n = 0.01047$$

$$T_{ox} = 5.7 \times 10^{-9}$$

$$V_{th} = -0.565$$

$$C_{ox} = 6.058 \times 10^{-3}$$

Let $I_D = 40 \mu A$

For M3, M2, M1

$$\begin{aligned}
 W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\
 &= \frac{2 \times 40 \times 10^{-6}}{0.0105 \times 6.058 \times 10^{-3} \times 0.565^2} \\
 &= 3.9
 \end{aligned}$$

Let $W=1.5\mu$

$L=.4\mu$

For M4, M5

$$\begin{aligned}
 W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\
 &= \frac{2 \times 20 \times 10^{-6}}{0.0105 \times 6.058 \times 10^{-3} \times 0.565^2} \\
 &= 2.0
 \end{aligned}$$

Let $W=1.5\mu$

$L=.74\mu$

For M8, M9

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 20 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.35^2} \\ &= 1.95 \end{aligned}$$

Let $W=1.5\mu$

$L=.75\mu$

For M6, M7

$$\begin{aligned} W/L &= \frac{2 I_D}{U_n C_{ox} V_{dsat}^2} \\ &= \frac{2 \times 40 \times 10^{-6}}{0.0278 \times 6.058 \times 10^{-3} \times 0.35^2} \\ &= 3.79 \end{aligned}$$

Let $W=1.5\mu$

$L=.4\mu$

CALCULATION FOR GAIN

$$A_v = -g_{m1} (r_{o1} || r_{o2})$$

$$r_o = \frac{V_{An}}{I_D}$$

$$g_{m1} = \sqrt{2 \times k'_n \times \frac{W}{L} \times I_{ref}}$$

$$= \sqrt{\frac{2 \times 0.0105 \times 6.058 \times 10^{-3} \times 1.5 \times 20 \times 10^{-6}}{0.74}}$$

$$= 7.2 \times 10^{-5} \text{ A/V}$$

$$r_{01} = \frac{V_{An}}{I_{D1}} = \frac{3.1}{20 \times 10^{-6}} = 155000 \text{ Ohm}$$

$$r_{02} = \frac{V_{Ap}}{I_{D2}} = \frac{3.1}{20 \times 10^{-6}} = 155000 \text{ Ohm}$$

$$A_v = -g_{m1} (r_{01} || r_{02})$$

$$= 5.6$$

$$\text{Gain in dB} = \frac{20 \log 5.6}{\log 2}$$

$$= 50 \text{ dB (Theoretical Gain)}$$

Practically \rightarrow 52 dB

Circuit diagram:

Layout Design using L-Edit:

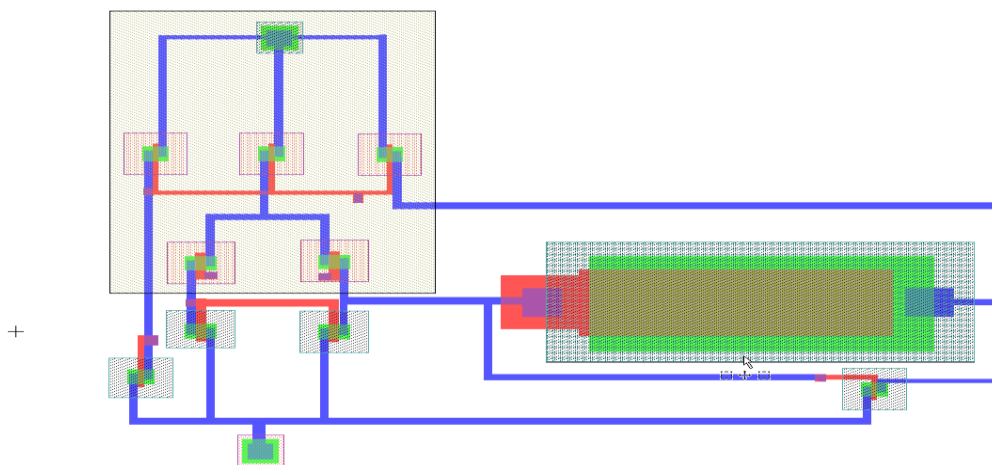


Figure 4.2 Common drain amplifier layout design using L-Edit.

T-Spice Netlist:**Using S-Edit:**

```

MPMOS_1 Out Vref Vdd Vdd PMOS W=1.5u L=400n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MPMOS_2 N_1 Vref Vdd Vdd PMOS W=1.5u L=400n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MNMOS_1 Vref Vref Gnd Gnd NMOS W=1.5u L=400n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MPMOS_3 Vref Vref Vdd Vdd PMOS W=1.5u L=400n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MNMOS_2 N_3 N_3 Gnd Gnd NMOS W=1.5u L=750n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MPMOS_4 N_3 Vinp N_1 Vdd PMOS W=1.5u L=740n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MNMOS_3 N_4 N_3 Gnd Gnd NMOS W=1.5u L=750n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MPMOS_5 N_4 Vinm N_1 Vdd PMOS W=1.5u L=740n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
MNMOS_4 Out N_4 Gnd Gnd NMOS W=1.5u L=400n AS=1.35p PS=4.8u AD=1.35p PD=4.8u
CCapacitor_1 N_4 Out 1p

```

Using L-Edit:

```

M1 Out 6 Gnd Gnd NMOS L=400n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M2 vref vref Gnd Gnd NMOS L=400n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M3 6 5 Gnd Gnd NMOS L=750n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M4 Gnd 5 5 Gnd NMOS L=750n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
C1 6 Out 1.0001444p
M5 Vdd vref vref Vdd PMOS L=400n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M6 Vdd vref 7 Vdd PMOS L=400n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M7 Out vref Vdd Vdd PMOS L=400n W=1.5u AD=1.2p PD=4.6u AS=1.2p PS=4.6u
M8 7 In+ 5 Vdd PMOS L=740n W=1.44u AD=1.152p PD=4.48u AS=1.152p PS=4.48u
M9 6 In- 7 Vdd PMOS L=740n W=1.44u AD=1.152p PD=4.48u AS=1.152p PS=4.48u

```

T-Spice Commands:

```

.tran .01m 5m
.ac dec 10 1 10meg
.dc lin source v4 -5 5 .1
.lib"C:\Users\Balaji\Documents\TannerEDA\TannerToolsv13.0\Libraries\Models\Generi
c_025.lib" TT
v1 Vdd GND 5
v2 vref GND 3.79
v3 vinm GND 3.1
v4 vinp vinm SIN (0 2m 1k) AC 1
.print ac vdb(Out) vp(Out)
.print dc v(Out)
.print tran v(Vinp) v(Out)
.op
.end

```

Procedure:**S-Edit:**

1. Create New Schematic Design using Libraries in S-Edit.
2. Open New Cell and design the circuit.
3. Check for errors in status window.
4. Save the Design.
5. Export Spice file.

L-Edit:

1. Create New Layout Design using Libraries in L-Edit.
2. Run DRC to check errors in Layout.
3. Save the Design.
4. Extract the netlists of Layout and export Spice file using EXT.

T-Spice:

1. Open T-Spice file and insert the commands.
2. Save the file.
3. Run Simulation.

LVS:

1. Create new file.
2. Select for two spice file from Layout and Schematic design.
3. Run Verification.

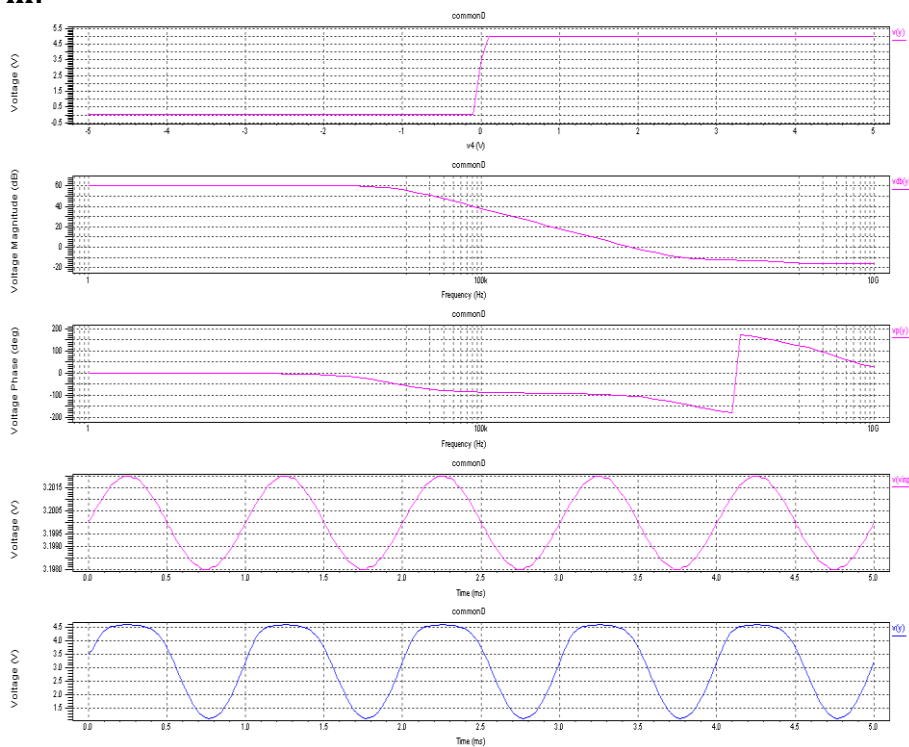
Waveform:

Figure 4.3 Simulation output waveform of Common drain amplifier.

Result:

Circuits are equal.

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

5. R2R DAC

Aim: Design a **4 bit R-2R based DAC** for the given specification and completing the design flow mentioned using

- a. **Draw the schematic** and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
- b. **Draw the Layout** and verify the DRC, ERC
- c. **Check for LVS**

Circuit diagram:

Schematic design using S-Edit:

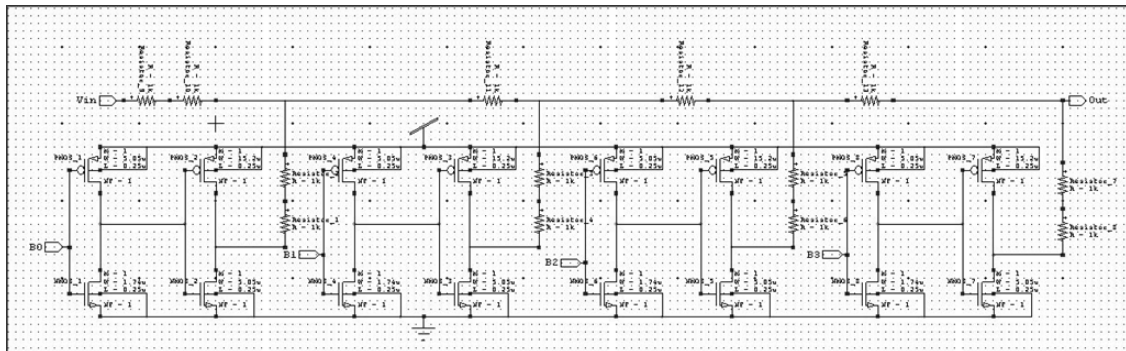


Figure 5.1 R2R DAC Schematic design using S-Edit.

Layout using L-Edit:

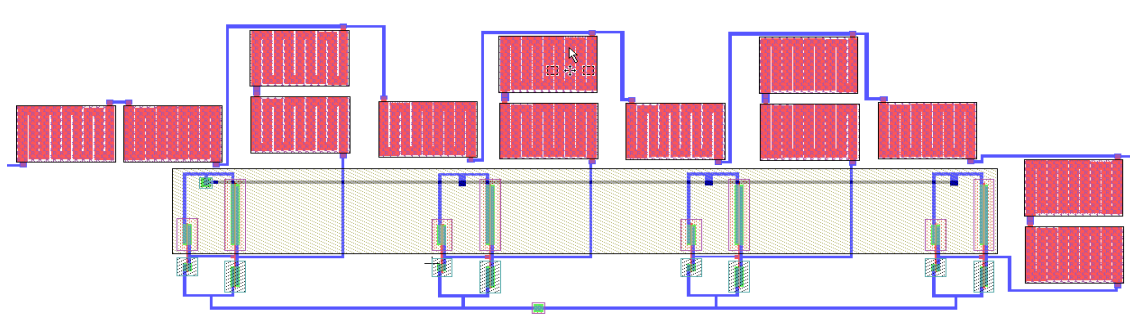


Figure 5.2 R2R DAC Layout design using L-Edit.

T-Spice Netlist:**Using S-Edit:**

```

MNMOS_3 N_3 N_4 Gnd Gnd NMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MNMOS_4 N_4 B1 Gnd Gnd NMOS W=1.74u L=250n AS=1.566p PS=5.28u AD=1.566p PD=5.28u
MNMOS_5 N_5 N_6 Gnd Gnd NMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MNMOS_6 N_6 B2 Gnd Gnd NMOS W=1.74u L=250n AS=1.566p PS=5.28u AD=1.566p PD=5.28u
MNMOS_7 N_7 N_8 Gnd Gnd NMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MNMOS_8 N_8 B3 Gnd Gnd NMOS W=1.74u L=250n AS=1.566p PS=5.28u AD=1.566p PD=5.28u
RResistor_1 N_9 N_2 R=1k
RResistor_2 N_10 N_9 R=1k
RResistor_3 N_11 N_12 R=1k
RResistor_4 N_12 N_3 R=1k
RResistor_5 N_13 N_14 R=1k
RResistor_10 N_16 N_10 R=1k
RResistor_6 N_14 N_5 R=1k
RResistor_11 N_10 N_11 R=1k
RResistor_7 Out N_15 R=1k
RResistor_12 N_11 N_13 R=1k
RResistor_8 N_15 N_7 R=1k
RResistor_13 N_13 Out R=1k
RResistor_9 Vin N_16 R=1k
MPMOS_1 N_1 B0 Vdd Vdd PMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MPMOS_2 N_2 N_1 Vdd Vdd PMOS W=15.2u L=250n AS=13.68p PS=32.2u AD=13.68p PD=32.2u
MPMOS_3 N_3 N_4 Vdd Vdd PMOS W=15.2u L=250n AS=13.68p PS=32.2u AD=13.68p PD=32.2u
MPMOS_4 N_4 B1 Vdd Vdd PMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MPMOS_5 N_5 N_6 Vdd Vdd PMOS W=15.2u L=250n AS=13.68p PS=32.2u AD=13.68p PD=32.2u
MPMOS_6 N_6 B2 Vdd Vdd PMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MPMOS_7 N_7 N_8 Vdd Vdd PMOS W=15.2u L=250n AS=13.68p PS=32.2u AD=13.68p PD=32.2u
MPMOS_8 N_8 B3 Vdd Vdd PMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u
MNMOS_1 N_1 B0 Gnd Gnd NMOS W=1.74u L=250n AS=1.566p PS=5.28u AD=1.566p PD=5.28u
MNMOS_2 N_2 N_1 Gnd Gnd NMOS W=5.05u L=250n AS=4.545p PS=11.9u AD=4.545p PD=11.9u

```

Using L-Edit:

```

M1 3 B3 vdd vdd PMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M2 1 3 vdd vdd PMOS L=250n W=15.2u AD=12.16p PD=32u AS=12.16p PS=32u
R1 1 4 1.0416k
R2 VOUT 4 1.0416k
R3 VOUT 6 1.0416k
M3 3 B3 Gnd Gnd NMOS L=250n W=1.7u AD=1.36p PD=5u AS=1.36p PS=5u
M4 1 3 Gnd Gnd NMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M5 10 B2 vdd vdd PMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M6 7 10 vdd vdd PMOS L=250n W=15.2u AD=12.16p PD=32u AS=12.16p PS=32u
M7 8 13 vdd vdd PMOS L=250n W=15.2u AD=12.16p PD=32u AS=12.16p PS=32u
R4 7 11 1.0416k
R5 6 11 1.0416k
R6 6 9 1.0416k
R7 8 14 1.0416k
R8 9 14 1.0416k
M8 10 B2 Gnd Gnd NMOS L=250n W=1.7u AD=1.36p PD=5u AS=1.36p PS=5u
M9 7 10 Gnd Gnd NMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M10 8 13 Gnd Gnd NMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M11 vdd B1 13 vdd PMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M12 21 B0 vdd vdd PMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u
M13 22 21 vdd vdd PMOS L=250n W=15.2u AD=12.16p PD=32u AS=12.16p PS=32u
R9 16 VIN 1.0416k
R10 18 9 1.0416k
R11 22 23 1.0416k

```

R12 18 23 1.0416k
R13 18 16 1.0416k
M14 Gnd B1 13 Gnd NMOS L=250n W=1.7u AD=1.36p PD=5u AS=1.36p PS=5u
M15 21 B0 Gnd Gnd NMOS L=250n W=1.7u AD=1.36p PD=5u AS=1.36p PS=5u
M16 22 21 Gnd Gnd NMOS L=250n W=5.05u AD=4.04p PD=11.7u AS=4.04p PS=11.7u

T-Spice Commands:

```
.tran 10n 8500n
.lib"C:\Users\Balaji\Documents\TannerEDA\TannerTools
v13.0\Libraries\Models\Generic_025.lib" tt
v1 vdd GND 5
v2 vin GND 2.5
v3 b0 GND PULSE (5 0 0 1n 1n 500n 1000n)
v4 b1 GND PULSE (5 0 0 1n 1n 1000n 2000n)
v5 b2 GND PULSE (5 0 0 1n 1n 2000n 4000n)
v6 b3 GND PULSE (5 0 0 1n 1n 4000n 8000n)
.print tran v(out)
.op
.end
```

Procedure:

S-Edit:

1. Create New Schematic Design using Libraries in S-Edit.
2. Open New Cell and design the circuit.
3. Check for errors in status window.
4. Save the Design.
5. Export Spice file.

L-Edit:

1. Create New Layout Design using Libraries in L-Edit.
2. Run DRC to check errors in Layout.
3. Save the Design.
4. Extract the netlists of Layout and export Spice file using EXT.

T-Spice:

1. Open T-Spice file and insert the commands.
2. Save the file.
3. Run Simulation.

LVS:

1. Create new file.
2. Select for two spice file from Layout and Schematic design.
3. Run Verification.

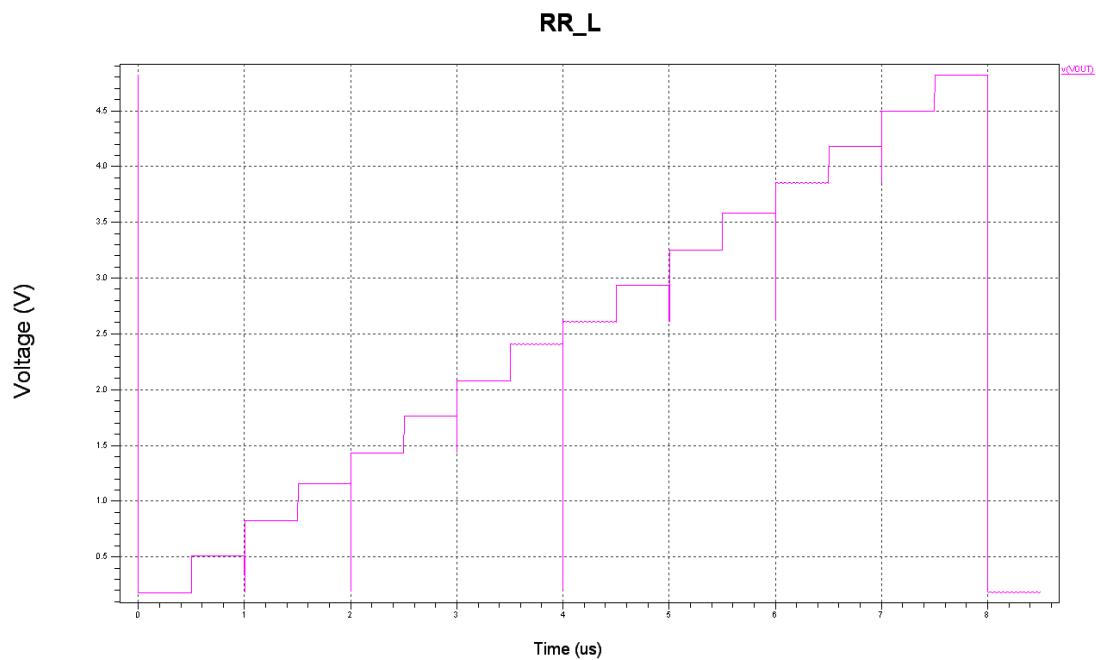
Waveform:

Figure 5.3 Simulation output of R2R DAC.

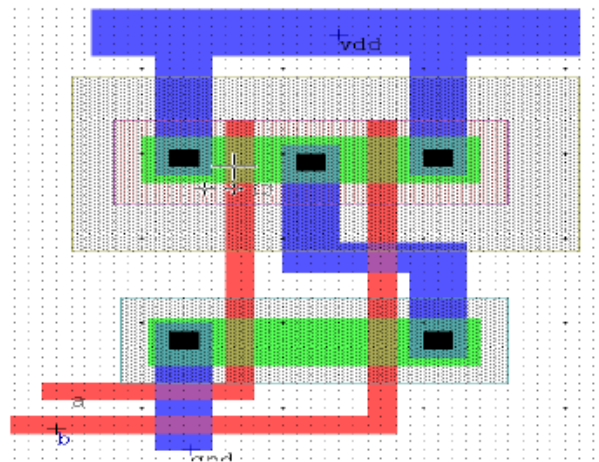
Result:

Circuits are equal.

| | |
|--------------|-------------------------|
| Date: | Lecturer's Sign: |
|--------------|-------------------------|

BEYOND THE SYLLABUS

1. NAND Gate:

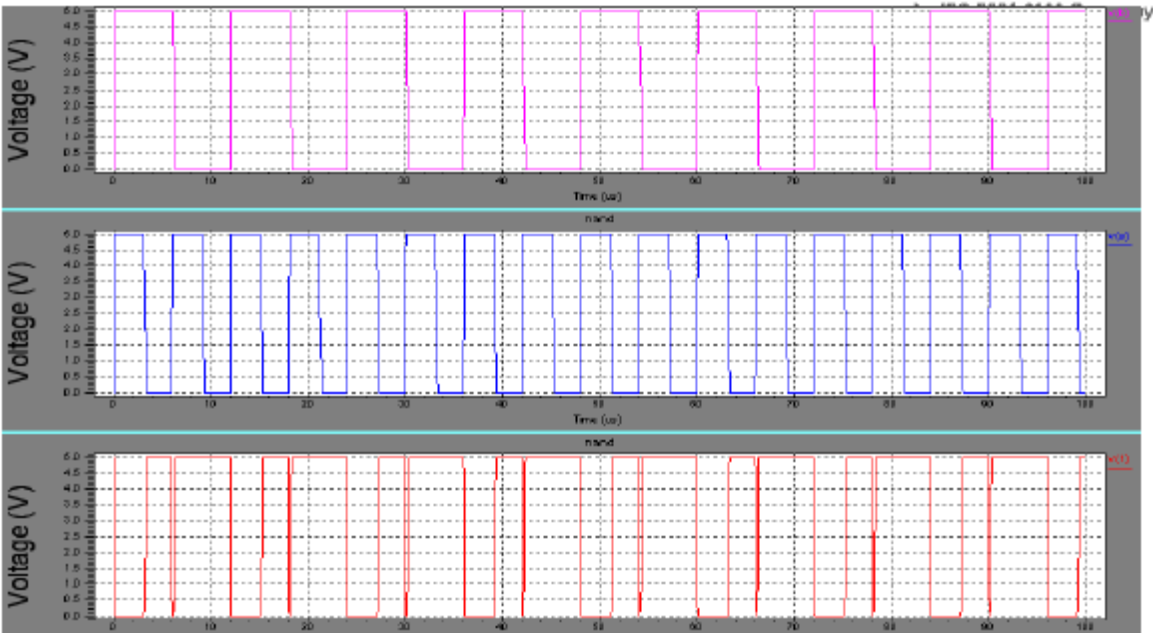


NAND gate layout

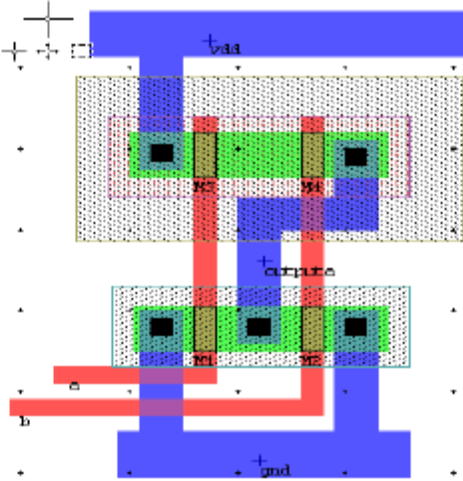
Spice File:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 outputa a gnd 7 NMOS L=1u W=2.75u
M2 gnd b outputa 7 NMOS L=1u W=2.75u
M3 5 a vdd 4 PMOS L=1u W=2.75u
M4 outputa b 5 4 PMOS L=1u W=2.75u
vdd vdd gnd 5
Va a gnd PULSE (0 5 0 100n 300n 3u 6u)
Vb b gnd PULSE (0 5 0 100n 300n 6u 12u)
.tran .1u 100u
.print tran v(outputa) v(a) v(b)
.END
```

Waveform:

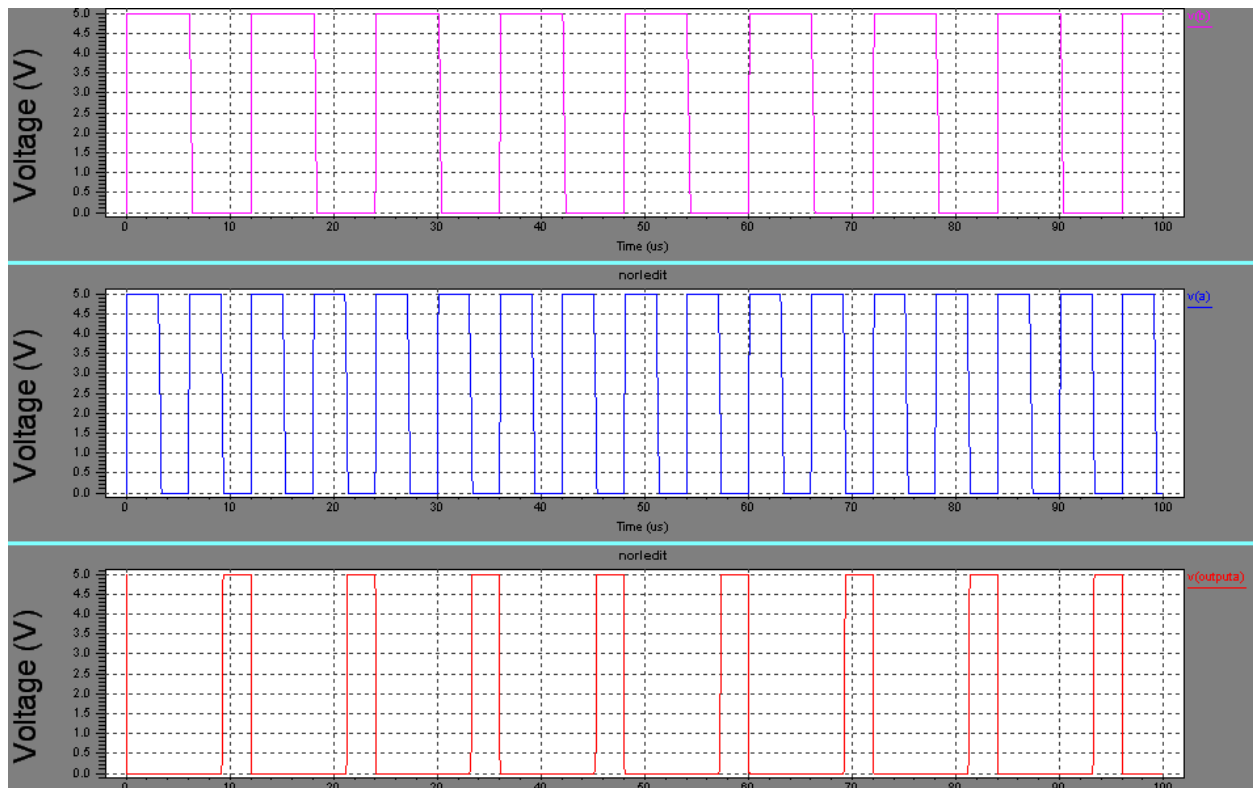


2. NOR Gate:

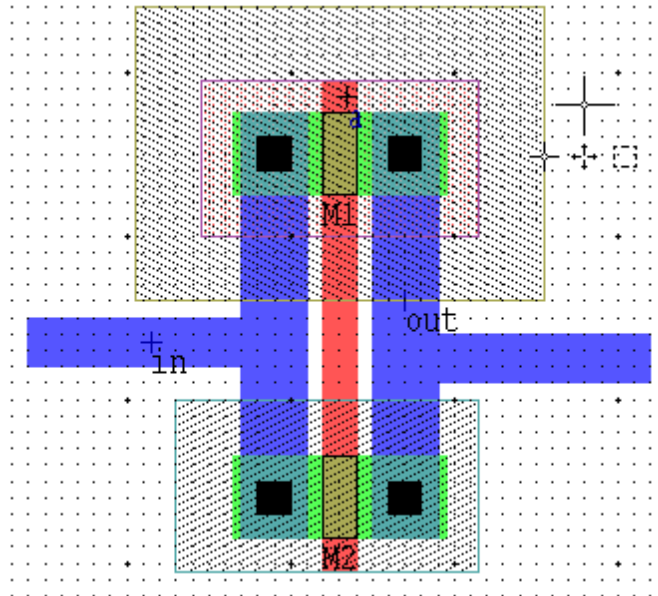


Spice File:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 outputa a gnd 7 NMOS L=1u W=2.75u
M2 gnd b outputa 7 NMOS L=1u W=2.75u
M3 5 a vdd 4 PMOS L=1u W=2.75u
M4 outputa b 5 4 PMOS L=1u W=2.75u
vdd vdd gnd 5
Va a gnd PULSE (0 5 0 100n 300n 3u 6u)
Vb b gnd PULSE (0 5 0 100n 300n 6u 12u)
.tran .1u 100u
.print tran v(outputa) v(a) v(b)
.END
```

Waveform:

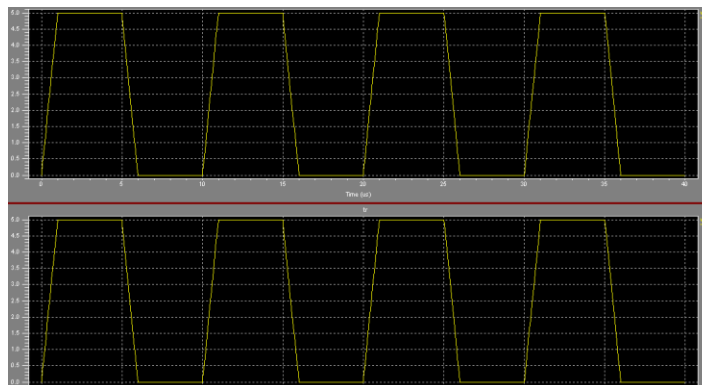
3. TRANSMISSION GATE



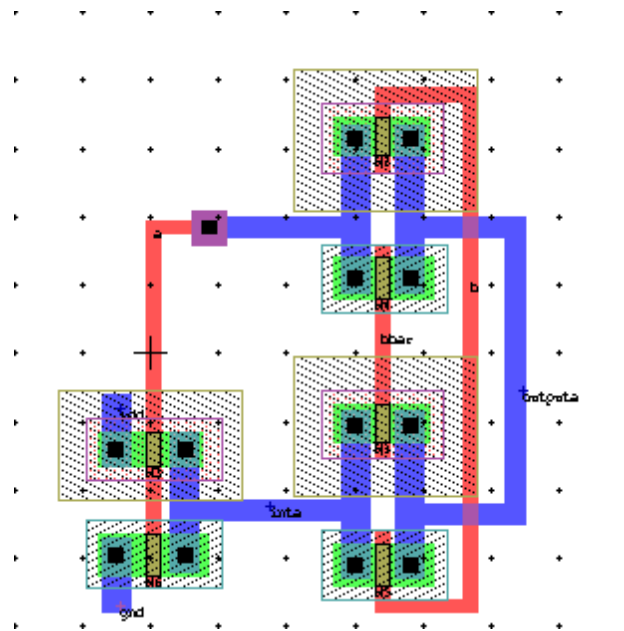
Spice file:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 out a in 4 PMOS L=1u W=2.5u
M2 out a in 3 NMOS L=1u W=2.5u
vdd vdd gnd 5v
Vin in gnd PULSE (0 5 0 1u 1u 4u 10u)
Va a gnd PULSE (0 5 0 1u 1u 4u 10u)
.tran .1u 40u
.print in out
.END
```

Waveform:



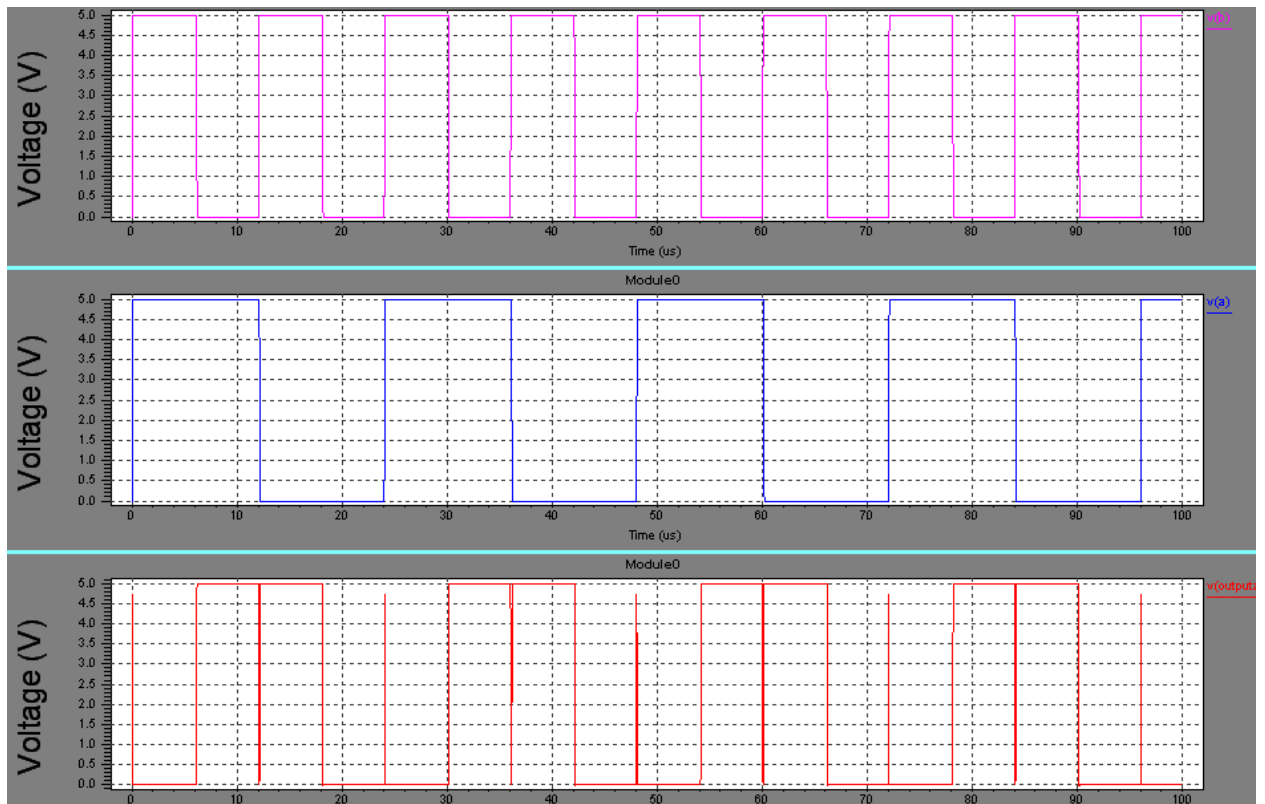
4. XOR GATE USING TRANSMISSION GATES



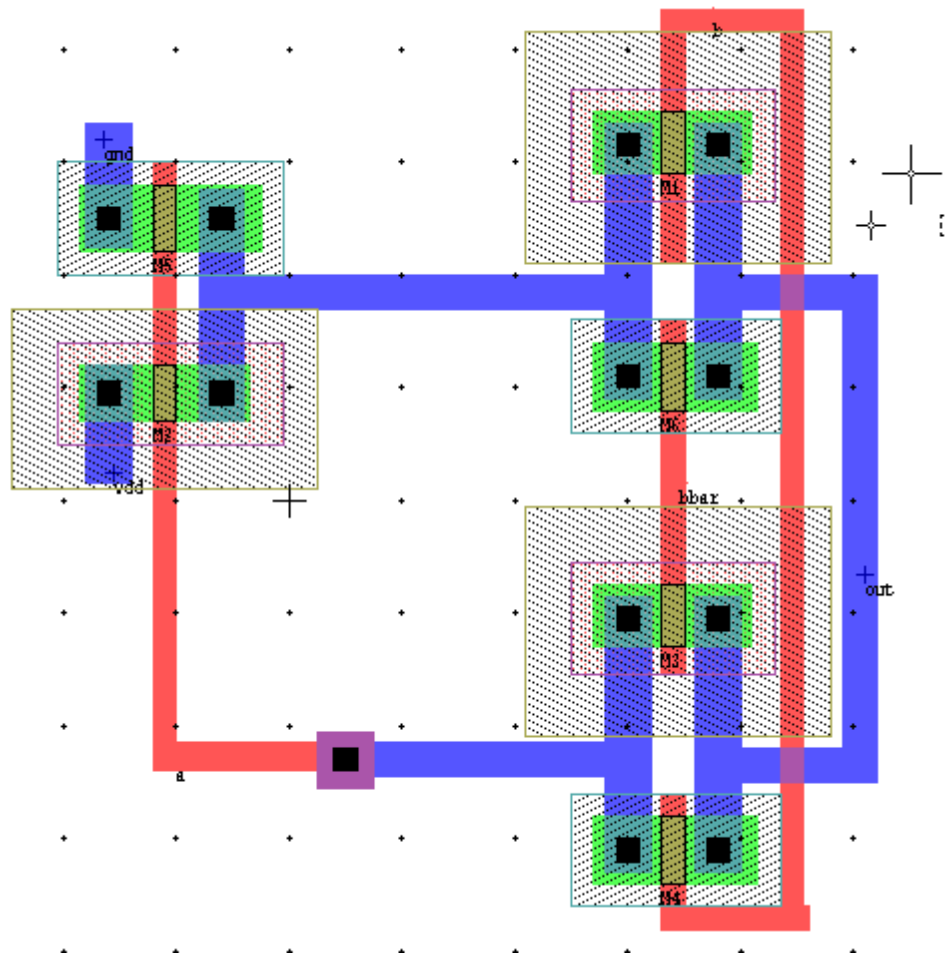
Spice file:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 outputa bbar inta 10 PMOS L=1u W=2.75u
M2 outputa b a 8 PMOS L=1u W=2.75u
M3 inta a vdd 7 PMOS L=1u W=2.5u
M4 outputa bbar a 6 NMOS L=1u W=3u
M5 outputa b inta 6 NMOS L=1u W=3u
M6 inta a 4 6 NMOS L=1u W=3u
vdd vdd gnd 5v
Va a gnd PULSE (0 5 0 1u 1u 4u 10u)
Vb b gnd PULSE (0 5 0 1u 1u 4u 10u)
Vbbar bbar gnd PULSE (5 5 0 1u 1u 4u 10u)
.tran .1u 40u
.print a bbar outputa
.END
```

Waveform:



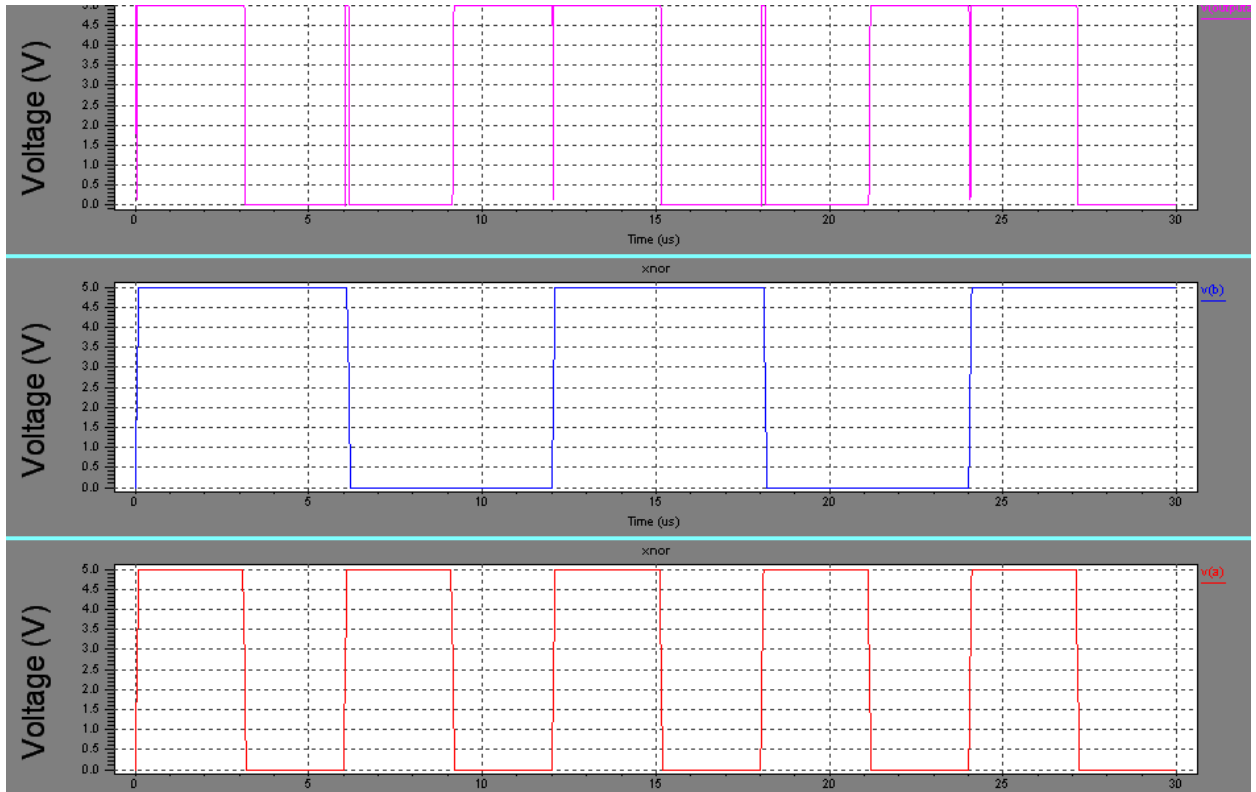
5. XNOR GATE USING TRANSMISSION GATES



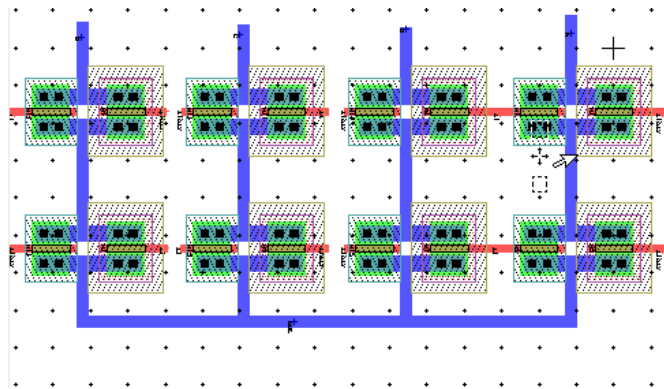
Spice file:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 out b 3 10 PMOS L=1u W=2.75u
M2 3 a vdd 9 PMOS L=1u W=2.5u
M3 out bbar a 7 PMOS L=1u W=2.75u
M4 out b a 6 NMOS L=1u W=3u
M5 3 a gnd 6 NMOS L=1u W=3u
M6 out bbar 3 6 NMOS L=1u W=3u
vdd vdd gnd 5
Va a gnd PULSE (0 5 0 1u 1u 4u 10u)
Vb b gnd PULSE (0 5 0 1u 1u 4u 10u)
Vbbar bbar gnd PULSE (5 5 0 1u 1u 4u 10u)
.tran .1u 40u
.print a bbar out
.END
```

Waveform:



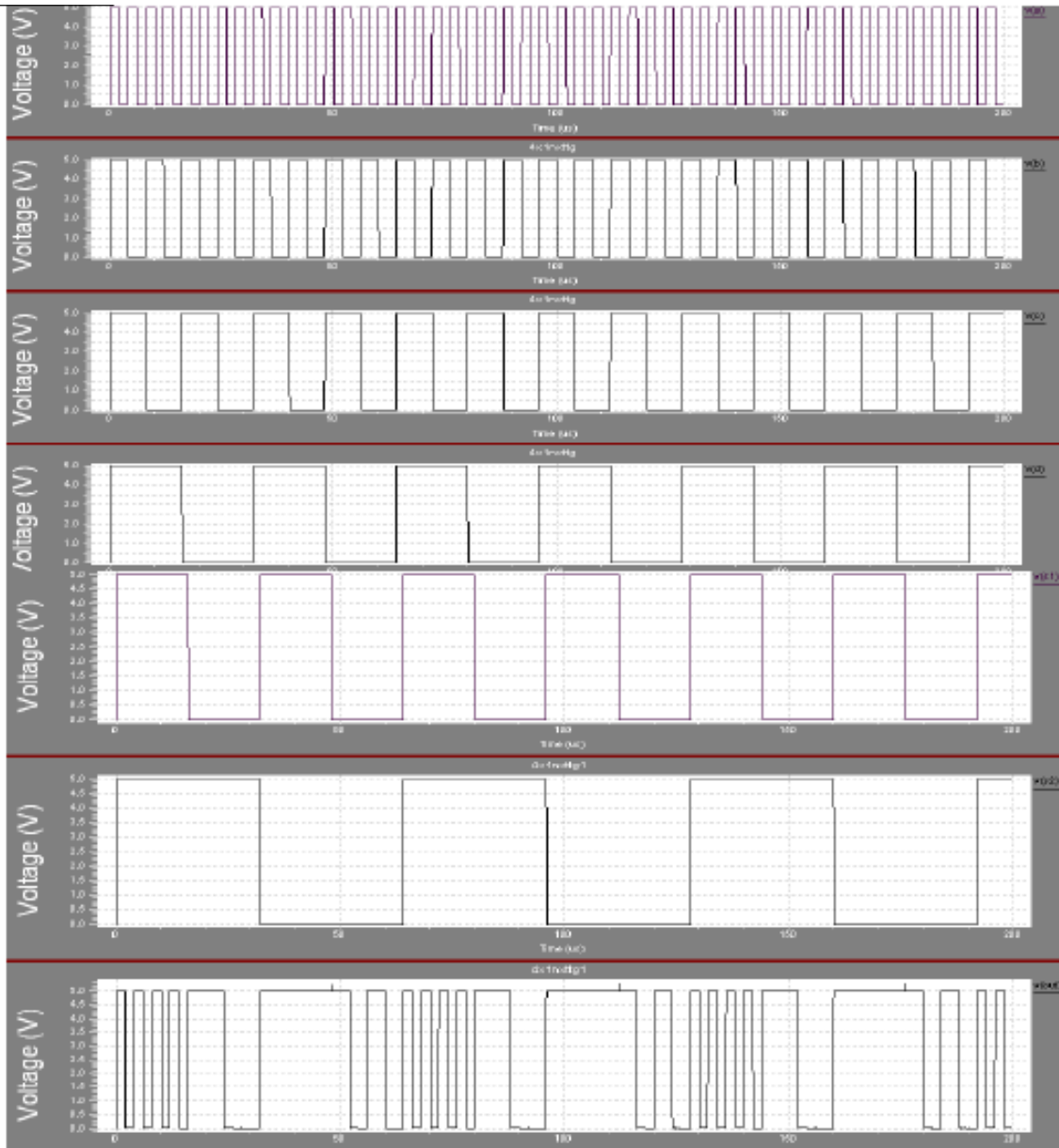
6. 4x1 MULTIPLEXER



Spice file:

```
.include "C:\Documents and Settings\Administrator\My Documents\Tanner EDA\Tanner Tools
v13.0\hp05.md"
M1 out s2 16 9 PMOS L=1u W=5u
M2 16 s1 B 7 PMOS L=1u W=5u
M3 out s2bar 1 5 PMOS L=1u W=5u
M4 1 s1bar A 3 PMOS L=1u W=5u
M5 out s2 1 23 NMOS L=1u W=5u
M6 1 s1 A 23 NMOS L=1u W=5u
M7 out s2 11 31 PMOS L=1u W=5u
M8 11 s1bar D 29 PMOS L=1u W=5u
M9 out s2bar 14 27 PMOS L=1u W=5u
M10 14 s1 C 25 PMOS L=1u W=5u
M11 out s2bar 11 23 NMOS L=1u W=5u
M12 11 s1 D 23 NMOS L=1u W=5u
M13 out s2 14 23 NMOS L=1u W=5u
M14 14 s1bar C 23 NMOS L=1u W=5u
M15 out s2bar 16 23 NMOS L=1u W=5u
M16 16 s1bar B 23 NMOS L=1u W=5u
Va a gnd PULSE (0 5 0 0 0 2u 4u)
Vb b gnd PULSE (0 5 0 0 0 4u 8u)
Vc c gnd PULSE (0 5 0 0 0 8u 16u)
Vd d gnd PULSE (0 5 0 0 0 16u 32u)
Vs1 s1 gnd PULSE (0 5 0 0 0 16u 32u)
Vs1bar s1bar gnd PULSE (0 5 16u 0 0 16u 32u)
Vs2bar s2bar gnd PULSE (0 5 32u 0 0 32u 64u)
Vs2 s2 gnd PULSE (0 5 0 0 0 32u 64u)
.tran .1u 200u
.print v(s1) v(s2) v(out)
.END
```

Waveform:



VIVA QUESTIONS

1. Why don't we use just one NMOS or PMOS transistor as a transmission gate?
2. What are set up time & hold time constraints? What do they signify?
3. Explain Clock Skew?
4. Why is NAND gate preferred over NOR gate for fabrication?
5. What is Body Effect?
6. Why is the substrate in NMOS connected to Ground and in PMOS to VDD?
7. What is the fundamental difference between a MOSFET and BJT ?
8. Why PMOS and NMOS are sized equally in a Transmission Gates?
9. What happens when the PMOS and NMOS are interchanged with one another in an inverter?
10. Why are PMOS transistor networks generally used to produce high signals, while NMOS networks are used to produce low signals?
11. What is Latch Up? Explain Latch Up with cross section of a CMOS Inverter. How do you avoid Latch Up?
12. Difference between Synchronous and Asynchronous reset.
13. What is DRC ?
14. What is LVS ?
15. What is RCX?
16. What are the differences between SIMULATION and SYNTHESIS?
17. What is a counter?
18. What are the differences between flipflop and latch?
19. How can you convert JK flipflop into JK Latch?
20. What are different types of adders?
21. Give the excitation table for JK flipflop?
22. Give the excitation table for SR flipflop?
23. Give the excitation table for D flipflop?
24. Give the excitation table for T flipflop?
25. What is the race around condition?
26. What is an amplifier?
27. What is an op-amp?
28. What is differential amplifier?
29. What is elaboration?
30. What is transient analysis?
31. What is DC analysis?
32. What is AC analysis?

VLSI LAB QUESTIONS

PART A – DIGITAL DESIGN

1. Write Verilog Code for Inverter using Switch level Description and their Test Bench for verification, observe the output waveform.
2. Write Verilog Code for Buffer using Switch level Description and their Test Bench for verification, observe the output waveform.
3. Write Verilog Code for Transmission gate using Switch level Description and their Test Bench for verification, observe the output waveform.
4. Write Verilog Code for all the Logic gates using Structural Description and their Test Bench for verification, observe the output waveform.
5. Write Verilog Code for AND gate using Switch level Description and their Test Bench for verification, observe the output waveform.
6. Write Verilog Code for NOR gate using Switch level Description and their Test Bench for verification, observe the output waveform.
7. Write Verilog Code for XOR gate using Switch level Description and their Test Bench for verification, observe the output waveform.
8. Write Verilog Code for OR gate using Switch level Description and their Test Bench for verification, observe the output waveform.
9. Write Verilog Code for NAND gate using Switch level Description and their Test Bench for verification, observe the output waveform.
10. Write Verilog Code for XNOR gate using Switch level Description and their Test Bench for verification, observe the output waveform.
11. Write Verilog Code for 4 bit Parallel (Ripple-carry) adder and their Test Bench for verification, observe the output waveform.
12. Write Verilog Code for SR Flipflop and their Test Bench for verification, observe the output waveform.
13. Write Verilog Code for JK Flipflop and their Test Bench for verification, observe the output waveform.
14. Write Verilog Code for D Flipflop and their Test Bench for verification, observe the output waveform.
15. Write Verilog Code for T Flipflop and their Test Bench for verification, observe the output waveform.
16. Write Verilog Code 4 bit Synchronous counter and their Test Bench for verification, observe the output waveform.

17. Write Verilog Code 4 bit Asynchronous counter and their Test Bench for verification, observe the output waveform.

PART B – ANALOG DESIGN

1. Design an Inverter with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS

2. Design the Single Stage Differential Amplifier with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS

3. Design the Common Source Amplifier with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS

4. Design the Common Drain Amplifier with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS

5. Design the Operational Amplifier with given specifications*, completing the design flow mentioned below:
 - a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS

6. Design a 4 bit R-2R based DAC for the given specification and completing the design flow mentioned using
- a. Draw the schematic and verify the following
 - i) DC Analysis
 - ii) AC Analysis
 - iii) Transient Analysis
 - b. Draw the Layout and verify the DRC, ERC
 - c. Check for LVS