

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) *NH* 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



QMP 7.1 D/F

Department of Electronics & Communication Engineering

VLSI Laboratory

15ECL77

B.E - VII Semester

Lab Manual 2019-20

Name :_____

USN :_____

Batch : _____ Section : _____



(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



Department of Electronics & Communication Engineering

VLSI Lab Manual

2019-20

Prepared by:

Mrs. Lakshmidevi P

Mr. Madhu R S

Mrs. Sowmya M R

Assistant Professors

Dept of ECE

Reviewed by:

Mr.Gavisiddappa Assistant Professor Dept of ECE

Approved by:

Dr. Rajagopala R Professor & Head, Dept. of ECE



(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



Vision

To create globally competent Electronics and Communication Engineering professionals with ethical and moral values for the betterment of the society

Mission

- To impart quality technical education in the field of electronics and communication engineering to meet over the current/future global industry requirements.
- To create the centres of excellence in the field of electronics and communication in collaboration with industry and universities
- To nurture the technical/professional/engineering and entrepreneurial skills for overall self and societal upliftment
- To orient the student community towards the higher education, research and development activities
- To provide a platform for equipping the students with necessary skills through co-curricular and extra-curricular events.
- To have Industrial collaboration for strengthening the Teaching-Learning Process/Academics
- To associate with industries for training the faculty on the latest technologies through continuous education programmes.

QMP 7.1 D/F



Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



B.E: Electronics & Communication Engineering

Program Outcomes (POs)

At the end of the B.E program, students are expected to have developed the following outcomes.

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialisation to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change



(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



Program Educational Objectives (PEO's)

After four Years of Graduation, our graduates are able to:

- Provide technical solutions to real world problems in the areas of electronics and communication by developing suitable systems.
- Pursue engineering career in Industry and/or pursue higher education and research.
- Acquire and follow best professional and ethical practices in Industry and Society.
- Communicate effectively and have the ability to work in team and to lead the team.

Program Specific Outcomes (PSOs)

At the end of the B.E Electronics & Communication Engineering program, students are expected to have developed the following program specific outcomes.

PSO1: Specify, design, build and test analog and digital systems for signal processing including multimedia applications, using suitable components or simulation tools.

PSO2: Understand and architect wired and wireless analog and digital communication systems as per specifications and determine their performance.

SYLLABUS

VLSI Laboratory

Sub code: 15ECL77 Hrs/Week: 03 Total Hrs: 42 IA Marks: 20 Exam Hours: 03 Exam marks:80

Experiments can be conducted using any of the following or equivalent designtools: Cadence/Synopsis/Mentor Graphics/Microwind

PART-A ASIC-DIGITAL DESIGN

1. Write Verilog Code for the following circuits and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given constraints*. Do the initial timing verification with gate level simulation.

i. An inverter

ii. A Buffer

iii. Transmission Gate

iv. Basic/universal gates

v. Flip flop -RS, D, JK, MS, T

vi. Serial & Parallel adder

vii. 4-bit counter [Synchronous and Asynchronous counter]

viii. Successive approximation register [SAR]

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 (i) Common source and Common Drain amplifier and (ii) A 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

d. Extract RC and back annotate the same and verify the Design.

3. Design an op-amp with given specification** using given differential amplifier Common source and Common Drain amplifier in library*** and 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.

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

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



* An appropriate constraint should be given.

** Appropriate specification should be given.

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



(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



QMP 7.1 D/D

Course Objectives and outcomes of VLSI Laboratory

Course Objectives: This course will enable students to:

- 1)Explore the CAD tool and understand the flow of the Full Custom IC design cycle.
- 2) Learn DRC, LVS and Parasitic Extraction of the various designs.
- 3)Design and simulate the various basic CMOS analog circuits and use them in higher circuits like data converters using design abstraction concepts.
- 4) Design and simulate the various basic CMOS digital circuits and use them in higher circuits like adders and shift registers using design abstraction concepts.

Course Outcomes: On the completion of this laboratory course, the students will be able to:

- 1) Write test bench to simulate various digital circuits.
- 2) Interpret concepts of DC Analysis, AC Analysis and Transient Analysis in analog circuits.
- 3) Design and simulate basic CMOS circuits like inverter, common source amplifier and differential amplifiers.
- 4) Use basic amplifiers and further design higher level circuits like operational amplifier and analog/digital converters to meet desired parameters.
- 5) Use transistors to design gates and further using gates realize shift registers and adders to meet desired parameters.

QMP 7.1 D/D



Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi) (NAAC Accredited & ISO 9001:2015 Certified Institution) NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.



TABLE OF CONTENTS

Sl.N0	Particulars	Page No					
	Part-A						
1.	Inverter	6					
2.	Buffer	9					
3.	Transmission Gate	12					
4.	Basic/universal gates	14					
5.	Flip flop -RS, D, JK, MS, T	18					
6.	Parallel & Serial adder	26					
7.	4-bit counter [Synchronous and Asynchronous counter]	29					
8.	Successive approximation register [SAR]	31					
	Part-B						
1.	Inverter	35					
2.	Differential Amplifier	39					
3.	Common Source Amplifier	44					
4.	Common Drain Amplifier	47					
5.	R-2R based DAC	50					
6.	Additional Experiment 1:Synchronous FIFO	55					
7.	Additional Experiment 2:Single port RAM synchronous Read/Write	56					

Instructions to the Students

- 1. Students should come with thorough preparation for the experiment to be conducted.
- 2. Students will not be permitted to attend the laboratory unless they bring the practical record

fully completed in all respects pertaining to the experiments conducted in the previous class.

- 3. Practical record should be neatly maintained.
- 4. They should obtain the signature of the staff-in –charge in the observation book after completing each experiment.
- 5. Theory regarding each experiment should be written in the practical record before procedure in own words.
- 6. Ask lab technician for assistance for any problem.
- 7. Save your class work assignments in system.
- 8. Do not download or install software without the assistance of laboratory technician.
- 9. Do not alter the configuration of system.
- 10. Turn off the systems after use.

History of Verilog

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo as well as from traditional computer language such as C.

Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation. The time was late 1990. Cadence Design System, whose primary product at that time included Thin film process simulator, decided to acquire Gateway Automation System. Along with other Gateway product, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator. At the same time, Synopsys was marketing the topdown design methodology, using Verilog. This was a powerful combination.

In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language. OVI did a considerable amount of work to improve the Language Reference Manual (LRM), clarifying things and making the language specification as vendor-independent as possible.In 1990.

The standard, which combined both the Verilog language syntax and the PLI in a single volume, was passed in May 1995 and now known as IEEE Std. 1364-1995. After many years, new features have been added to Verilog, and new version is called Verilog 2001. This version seems to have fixed lot of problems that Verilog 1995 had. This version is called 1364-2000. Only waiting now is that all the tool vendors implementing it.

Introduction to Verilog

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL.

HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits.

Verilog can be used to describe designs at four levels of abstraction:

(i) Algorithmic level (much like c code with if, case and loop statements).

(ii) Register transfer level (RTL uses registers connected by Boolean equations).

(iii) Gate level (interconnected AND, NOR etc.).

(iv) Switch level (the switches are MOS transistors inside gates).

The language also defines constructs that can be used to control the input and output of simulation.

Design and Tool Flow

For designing digital systems using programmable logic devices (PLDs), such as FPGA devices (Spartan families), computer aided design (CAD) software packages (Xilinx ISE 9.2i) are used. These software packages assist the designer through all the stages of the

design process. Therefore, most of the CAD packages for programmable logic devices provide the following main functions:

□ *Description* of the digital system;

 \Box Synthesis of the description, which means transforming the description into a netlist containing basic logic gates and their interconnections;

 \Box Functional simulation of the system based on the netlist obtained, before the implementation in a specific device;

□ *Implementation* of the system in a device, by adapting the netlist for an efficient usage of the device's available resources;

□ *Configuration* (programming) of the device in order to perform the desired function.

System Description: During the design entry you will describe your FPGA design with a high-level hardware description language. In this lab course we will use Verilog for this purpose.

Behavioral Simulation: The behavioral simulation is the most important step in verifying the correct functionality of your circuit description. First the simulator compiles your Verilog files and indicates syntax errors. If the compilation runs without errors, the behavior of the circuit design can be investigated in a waveform showing signal traces of external and internal signals.

Synthesis, Place & Route: During the synthesis step your Verilog circuit description is analyzed by the synthesis tool and transformed into a gate-level netlist. The netlist elements can then be placed on the FPGA's configurable cells and connected by suitable interconnect lines. Finally a bitstream file is generated containing the configuration information for the FPGA.

Post-layout Simulation: The post-layout simulation is performed on the netlist generated in the synthesis step. Additionally timing information, which are prodived by the Place&Route-tool can be used for simulating the final circuit with correct signal timings. A post-layout simulation is useful for resolving timing problems in your circuit. For simple FPGA designs this type of simulation is usually not necessary.

Download: The download transfers the bitstream data to the target board for configuring the FPGA.

Test & HW Debugging : Finally, the configured FPGA can be tested. In some cases it might be necessary to attach a logic analyzer to the FPGA for verifying the correct functionality of the HW circuit or for debugging errors, which can not be reproduced during simulation.

The complete design flow of synthesizing using Verilog code is given in Figure 1.

VLSI LAB MANUAL (15ECL77)

2019 - 20



Figure 1: VLSI Design Flow

Introduction to MEMSPRO Software

The interdisciplinary nature of Micro-Electro-Mechanical Systems (MEMS) and the expertise required to develop the technology is a significant bottleneck in the timely design of new products incorporating MEMS technology.

This issue calls for a new generation of design tools that combines aspects of EDA and mechanical, thermal, fluidic, optical, and magnetic CAD.

The SoftMEMS approach to solving this design bottleneck is based on the following principles and features:

 \Box Supporting multiple flows: for the component engineer, multi-physics circuit designer and for the system engineer;

□ □ Allowing data exchange between the different description levels: the structural level (FEM/BEM), the system/behavioral level (SPICE, HDL-A, VHDL-AMS, Verilog-AMS), and the physical level (mask layout);

□ □ Targeting key features for MEMS specific design.

MEMS Pro, in combination with ANSYS Multiphysics or other 3D analysis programs, enables system designers, MEMS circuit designers, IC designers, process engineers, MEMS specialists, and packaging engineers to share critical design and process information in the most relevant language for each contributor.

The MEMS Pro package includes a schematic entry tool, an analog and mixed analog / digital circuit level behavioral simulator, a statistical analyzer, an optimizer, a waveform viewer, a full-custom mask-level layout editing tool, an automatic layout generator, an automatic standard cell placement and routing tool, a design rule checking feature, an automatic netlist extraction tool (from layout or schematic), a comparison tool between netlists extracted from layout and schematics (Layout Versus Schematic), and libraries of MEMS examples.

MEMS specific key features also available in MEMS Pro encompass a 3D solid model generator using mask layout and process information, a 3D solid model viewer with cross-section capability, a process description editor, true curved drawing tools and automation of time consuming tasks using the MEMS Pro Easy MEMS features.

Embedded features within ANSYS allow automatic mask layout generation from an ANSYS 3D Model and process description (ANSYS to Layout), as well as automatic MEMS behavioral Model generation in hardware description languages (MEMS Modeler).

Tool Flow

Each stage of the MEMS design process is addressed by a different component of the MEMS Pro tool suite.



Figure 1.33: MEMS Design flow

S-Edit :S-Edit is a fully hierarchical schematic capture program for MEMS and IC applications. The program also serves as a schematic entry front end to the T-Spice simulator, L-Edit/ SPR automatic standard cell placement and router, and layout vs. Schematic (LVS) netlist comparison programs.

T-Spice simulator :The T-Spice simulator provides full-chip analysis of analog, mixed analog/digital and MEMS designs using an extremely fast simulation engine that has been proven in designs of over 300,000 devices. For large circuits, the T-Spice simulator can be ten times faster than typical SPICE simulators.

L-Edit: L-Edit is an interactive, graphical layout editor for MEMS and IC design. This full-custom editor is fast, easy-to-use, and fully hierarchical. Primitives include boxes, polygons, circles, lines, wires, labels, arcs, splines, ellipses, tori, fillet and arbitrary curve generator. Drawing modes include 90°, 45°, and all-angle layout.

User-programmable Interface: L-Edit/UPI is a powerful tool for automating, customizing, and extending L-Edit.s command and function set. The heart of L-Edit/UPI is the macro interface. Macros are user-programmed routines, written in the C language, that describe automated actions or sets of actions.

Layout vs. Schematic (LVS): LVS compares the SPICE netlist generated from S-Edit or another schematic editor with the netlist generated from layout by L-Edit Extract. LVS is a check to ensure that both netlists represent the same multiphysics "circuit". Should any inconsistencies be found between the two lists, LVS can be used to identify and resolve the ambiguity.

MEMS Library (MEMSLib): MEMSLib provides MEMS designers with schematics, simulation models, and parameterized layout generators for a set of MEMS components. MEMSLib includes several types of suspension elements, electro-mechanical transducers and test structures for extracting material properties. Various example elements can be assembled to produce a single MEMS device.

DIGITAL DESIGN

LAB 1: INVERTER

Objective: To Compile and simulate the Verilog Code for an inverter circuit and observe the waveform.

Inverter:

`resetall `timescale 1 ns / 1 ns

//Define
our own
Inverter,
module inv
(y, x);
// Declarations of I/O, Power and Ground Lines
Output y;
reg y;
input x;
always @(x)
y=~x;
endmodule



Inverter_test:

`resetall `timescale 1 ns / 1 ns // Testbench for Inverter Module module inv_test; wire out; reg in; // Instantiate inverter Module inv i (out, in); // Display task display; begin \$display ("time=%0d" , \$time , " ns" , " Input=" , in , " Output=", out); end endtask

// Apply Stimulus

initial
 begin
 in = 1'b0; #10; display;
 in = 1'b1; #10; display;
 in = 1'bx; #10; display;
 in = 1'bz; #10; display;
 end

endmodule

Note: The task "display" is used to display the output on the monitor, in the form of a truth table.

Constraints file for Synthesis: Constraints_Inverter.g

Truth Table

Input	Output
А	Y
0	1
1	0
Х	Х
Z	Х

Result

Wave - Default	Nave - Default :								
\$ 1.	Msgs								
<pre>/inv_test/out</pre>	1'h0 1'h1								
	101								

For synthesis, follow the steps that are mentioned previously. After synthesis, the synthesis window will be shown as follows –



LAB 2: BUFFER

Objective: To write Verilog Code for the Buffer circuit and Test Bench for Verification, observe the waveform.

Buffer:

resetall timescale 1 ns / 1 ns



//Define our own Inverter,

module buf (r,a);

// Declarations of I/O, Power and Ground Lines

output r; input a; reg r; always@(*) r=a; endmodule

```
Buffer_test:

`resetall

`timescale 1 ns / 1 ns

// Testbench for Buffer Module

module buf_test;

wire out;

reg in;
```

// Instantiate Buffer Module

buf b1(out,in);

// Display

```
task display ;
begin
$display ( "time=%0d" , $time , " ns", " Input=" , in , " Output=", out ) ;
end
endtask
```

// Apply Stimulus

```
initial
  begin
    in = 1'b0; #10; display;
    in = 1'b1; #10; display;
```

in = 1'bx ; #10 ; display ; in = 1'bz ; #10 ; display ; end

endmodule

Constraints file for Synthesis: Constraints_Buffer.g

Truth Table

Input	Output
A	Y
0	0
1	1
Х	Х
Z	Х

Result

Wave - Default	Wave - Default					 				
\$.	Msgs									
 ✓ /buf_test/out ✓ /buf_test/in 	-No Data- -No Data-	L								

The snapshot obtained after synthesis is as shown below -



When double-clicked inside the blocks shown, the subsystem is displayed as shown -



LAB 3: TRANSMISSION GATE

Objective: To write Verilog Code for the Transmission gate circuit and Test Bench For Verification, observe the waveform.

TG:

`resetall `timescale 1 ns / 1 ns

//Define our own Transmission Gate,

module tg (out , in , cntrl1, cntrl2);

// Declarations of I/O and Control Lines

output out; input in; input cntrl1,cntrl2;

// Instantiate pmos and nmos switches

pmos (out,in,cntrl1); nmos (out,in,cntrl2);

endmodule

TG_test:

resetall timescale 1 ns / 1 ns

// Testbench for Inverter Module

module trangate_test;

wire out ;
reg in ;
reg cntrl1,cntrl2;

// Instantiate trangate Module

trangate t1 (out, in, cntrl1, cntrl2);

// Display

task display;



2019 - 20

end

endmodule

Truth Table

Input	Cntrl1	Cntrl2	Output
0	0	1	0
0	1	0	Ζ
1	0	1	1
1	1	0	Ζ

Result

-	Wave - Default				 ". <u> </u>	
4	ù.	Msgs				
•	 /tg_test/out /tg_test/in /tg_test/cntrl1 /tg_test/cntrl2 	1'h0 1'h0 1'h0 1'h1				

LAB 4: BASIC/ UNIVERSAL GATES

Objective: To write Verilog Code for the Basic/ Universal gates and Test Bench For Verification, observe the waveform.

UG:

`resetall
`timescale 1 ns / 1 ns
module allg (not1,or1,and1,nor1,nand1,xor1,xnor1,a,b);

```
// Declarations of I/O ,Power and Ground Lines
output not1,or1,and1,nor1,nand1,xor1,xnor1;
input a,b;
reg not1,or1,and1,nor1,nand1,xor1,xnor1;
always @(a,b)
begin
not1 = \sima;
or1 = a|b;
and1 = a&b;
nor1 = \sim(a|b);
nand1 = \sim(a&b);
xor1 = a^b;
xnor1 = \sim(a^b);
end
endmodule
```

UG_test:

`resetall `timescale 1 ns / 1 ns

// Testbench for universal gates Module module allg_t; wire not1,or1,and1,nor1,nand1,xor1,xnor1; reg a,b;

// Instantiate universal gates Module
allg uut(not1,or1,and1,nor1,nand1,xor1,xnor1,a,b);

```
// Apply Stimulus
initial
begin
a=1'b0;b=1'b0;
#10 a=1'b0;b=1'b1;
#10 a=1'b1;b=1'b0;
#10 a=1'b1;b=1'b1;
#200 $finish;
end
endmodule
```

Logic diagram

a. AND gate



Truth Table

In	put	Output
А	В	Y
0	0	0
0	1	0
1	0	0
1	1	1

b. OR gate



-

In	Output	
А	В	Y
0	0	0
0	1	1
1	0	1
1	1	1

c. NAND gate



In	put	Output
Α	В	Y
0	0	1
0	1	1
1	0	1
1	1	0

d. NOR gate



In	Input				
Α	В	Y			
0	0	1			
0	1	0			
1	0	0			
1	1	0			

e. XOR gate



In	put	Output
А	В	Y
0	0	0
0	1	1
1	0	1
1	1	0

f. XNOR gate



In	Output	
Α	В	Y
0	0	1
0	1	0
1	0	0
1	1	1

4. Result

B Wave - Default						
- 💫 -	Msgs					
/allg_t/not1	1'h1					
✓ /allg_t/or1	1'h0					
✓ /allg_t/and1	1'h0					
/allg_t/nor1	1'h1					
🔶 /allg_t/nand1	1'h1					
/allg_t/xor1	1'h0					
/allg_t/xnor1	1'h1					
🔶 /allg_t/a	1'h0					
🔶 /allg_t/b	1'h0					

Constraints file for Synthesis: Constraints_gates.g

1. The synthesis result snapshot using RTL description is as shown -



LAB 5: FLIP FLOPS

Objective: To write Verilog Code for various Flip flop circuits and Test Bench For Verification, observe the waveform.

1) **D-FF:**

module d_ff(d, rst, clk, q, qb); input d, rst, clk; output q, qb; reg q,qb; always@(posedge clk) begin if (rst=1)begin q=0; qb=1;end else begin q=d; qb=~d; end end endmodule



D-FF_test:

```
module d_ff_test;
reg clk, d, rst;
wire q,qb;
d_ff df1 (d, rst, clk, q, qb);
initial
clk = 1'b0;
always
#5 clk = ~clk;
initial
        begin
               d = 1'b0;
               rst = 1'b1;
               #30 rst = 1'b0;
               #40 d = 1'b1;
               #20 d = 1'b0;
               #40 d = 1'b1;
               #20 d = 1'b0;
```

end

endmodule

Dept. Of ECE ,CIT-Gubbi

Truth Table

RESET	D	Q	QBAR
1	Х	0	1
0	0	0	1
0	1	1	0

Result

Wave - Default					
- 💫 🗸	Msgs				
<pre>/d_ff_test/dk /d_ff_test/d /d_ff_test/rst /d_ff_test/q /d_ff_test/q /d_ff_test/qb</pre>	1'h0 1'h0 1'h0 1'h0 1'h1				

2) JK-ff:

```
module jk_ff(jk, clk, rst, q, qb);
input [1:0]jk;
input clk,rst;
output q, qb;
reg q, qb;
always @ (posedge clk)
begin
       if(rst==1)
       begin
              q=0;
              qb=1;
       end
       else
       begin
              case (jk)
                      2'b00: begin q=q; qb=qb; end
                      2'b01: begin q=0; qb=1; end
                     2'b10: begin q=1; qb=0; end
                      2'b11: begin q=(q); qb=(qb); end
              endcase
       end
end
endmodule
```

JK-FF_test:

```
module jk_ff_test;
reg clk,rst;
reg [1:0] jk;
wire q,qb;
wire clk1;
wire [1:0] jk1;
jk_ff uut (jk, clk, rst, q, qb);
assign clk1=clk;
assign jk1=jk;
initial
clk = 1'b0;
always
      #5 \ clk = \sim clk;
initial
        begin
          jk = 2'b00; rst = 1'b1;
          #20 rst = 1'b0;
          #20 jk = 2'b01;
          #20 jk = 2'b10;
          #20 jk = 2'b11;
          #20 rst = 1'b1;
          #10;
        end
```

endmodule

Constraints file for Synthesis: Constraints_JK_ff.g



Truth Table

RESET	J	K	Q	QBAR
1	Х	Х	0	1
0	0	0	Q	QBAR
0	0	1	0	1
0	1	0	1	0
0	1	1	~Q	~QBAR

Result

💶 Wave - Default 🚃	Wave - Default :====================================						
💫 🗸	Msgs						
↓jk_ff_test/dk	1'h0						
/jk_ff_test/rst	1'h1						
	2'h3						
↓jk_ff_test/q	1'h0						
/jk_ff_test/qb	1'h1						
✓ /jk_ff_test/dk1	1'h0						
	2'h3						
P							

3) MS-FF:

```
module ms_ff(q,q_bar,clk,j,k);
       output q,q_bar;
       input clk,j,k;
       reg tq,q,q_bar;
       always @(clk)
       begin
               if (!clk)
               begin
                      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 <= \simtq;
               end
       end
     endmodule
```

MS-FF_test:

module ms_ff_test; reg clk,j,k; wire q,q_bar; wire clk2,j2,k2; ms_ff inst(q,q_bar,clk,j,k); assign clk2=clk; assign j2=j; assign k2=k; initial clk = 1'b0;always #10 $clk = \sim clk;$ initial begin j = 1'b0; k = 1'b0;#60 j = 1'b0; k = 1'b1; #40 j = 1'b1; k = 1'b0; #20 j = 1'b1; k = 1'b1; #40 j = 1'b1; k = 1'b0; #5 j = 1'b0; #20 j = 1'b1; #10; end always #5 \$display(\$time," clk=%b j=%b k=%b ",clk,j,k); initial #200 \$finish; endmodule

Truth Table

RESET	J	K	Q	QBAR
0	Х	Х	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

Result

💶 Wave - Default 🚞		
🌮 🗧	Msgs	
/ms_ff_test/clk	1'h1	
/ms_ff_test/j	1'h1	
/ms_ff_test/k	1'h0	
/ms_ff_test/q	1'h1	
/ms_ff_test/q_bar	1'h0	
/ms_ff_test/clk2	1'h1	
/ms_ff_test/j2	1'h1	
/ms_ff_test/k2	1'h0	
P		

4) T-FF:

module t_ff(t, clk, rst, q, qb); input t, clk, rst; output q, qb; reg q,qb; always @ (posedge clk) begin if (rst==1) begin q=1'b0; qb=1'b1; end else begin case (t) 1'b0:begin q=q; qb=qb; end 1'b1:begin q=~(q); qb=~(qb); end endcase end end endmodule



T-FF_test:

```
module t_ff_t;
     reg clk,t,rst;
     wire q,qb;
     t_ff t1(t, clk, rst, q, qb);
     initial
          clk = 1'b0;
     always
          #10 clk = ~clk;
     initial
     begin
       rst = 1'b1; t = 1'b0;
               #30 rst = 1'b0;
               #10 t = 1'b1;
               #30 t = 1'b0;
               #20 rst = 1'b1;
               #20;
       end
     endmodule
```

Truth Table

RESET	Т	Q	QBAR
1	Х	0	1
0	0	Q	QBAR
0	1	~Q	~QBAR

Result

Wave - Default					
💫 🗸	Msgs				
✓ /t_ff_t/dk	1'h0	LUUU	uluu	111	
🧇 /t_ff_t/t	1'h0				
🔶 /t_ff_t/rst	1'h1				
🔶 /t_ff_t/q	1'h0				
✓ /t_ff_t/qb	1'h1				
P					

5) **RS-FF:**

module sr_ff(sr, clk, rst, q, qb); RST input [1:0] sr; input rst, clk; output q,qb; S reg q,qb; S-R always @ (posedge clk) CLK-FLIPbegin FLOP Rif (rst = 1) begin q=0; qb=1; end else begin case (sr) 2'b00: begin q=q; qb=qb; end 2'b01: begin q=0; qb=1; end 2'b10: begin q=1; qb=0; end 2'b11: begin q=1'bx; qb=1'bx; end endcase end end endmodule



RS-FF_test:

```
module sr_ff_test;
       reg clk,rst;
       wire q,qb;
       reg [1:0] sr;
       wire [1:0] sr1;
wire clk1;
       sr_ff uut (sr, clk, rst, q, qb);
       assign sr1=sr;
       assign clk1=clk;
       initial
       clk = 1'b0;
       always
       #5 clk = \sim clk;
       initial
       begin
          sr = 2'b00; rst=1'b1;
          #30 rst =1'b0;
          #30 sr = 2'b01;
          #40 sr = 2'b10;
          #30 sr = 2'b11;
         end
         endmodule
```

Truth Table

INPUTS			OUTPUTS			
RST	S	R	Q	QBAR		
1	Х	Х	0	1		
0	0	0	Q	QBAR		
0	0	1	0	1		
0	1	0	1	0		
0	1	1	Х	Х		

Result

📰 Wave - Default 🚞									
💫 🗸	Msgs								
↓sr_ff_test/dk	1'h0	LUUL	பா	LUU	nn	LUL	பா	nn	ப்பி
/sr_ff_test/rst	1'h1	1							
🔶 /sr_ff_test/q	1'hx								
✓ /sr_ff_test/qb	1'hx	_				٦			
💶 🥠 /sr_ff_test/sr	2'h0	0		1		2	3		
💶 🥠 /sr_ff_test/sr1	2'h0	0		1		2	3		
/sr_ff_test/dk1	1'h0	Innr	மா	Inn	nnr	nnr	மா	INNE	und
¢									
LAB 6: PARALLEL AND SERIAL ADDERS

Objective: To write Verilog Code for the Serial and Parallel Adder and Test Bench For Verification, observe the waveform.

1) i) Parallel Adder

module adder4 (carryin,x,y,sum,carryout); input carryin; input [3:0] x,y; output [3:0] sum; output carryout; fulladd stage0 (carryin,x[0],y[0],sum[0],c1); fulladd stage1 (c1,x[1],y[1],sum[1],c2); fulladd stage2 (c2,x[2],y[2],sum[2],c3); fulladd stage3 (c3,x[3],y[3],sum[3],carryout); endmodule

module fulladd (cin,x,y,s,cout); input cin,x,y; output s,cout; assign s = x^y^cin; assign cout =(x & y) | (x & cin) |(y & cin); endmodule

ii) Parallel Adder_test

```
module adder4_t;
reg [3:0] x,y;
reg carryin;
wire [3:0] sum;
wire carryout;
adder4 a1 (carryin,x,y,sum,carryout);
initial
begin
x = 4'b0000; y= 4'b0000;carryin = 1'b0;
#20 x =4'b1111; y = 4'b1010;
#40 x =4'b1011; y = 4'b0110;
#40 x =4'b1111; y=4'b0110;
#40 x =4'b1111; y=4'b1111;
#50 $finish;
end
endmodule
```

Result

📲 Wave - Default 🚃		
💫 🗸	Msgs	
=	4'hf	OXF XB XF
	1	
	1	
	1	
L-🔶 [0]	1	
🖃 🔷 /adder4_t/y	4'hf	0 (a)6 (f
	1	
	1	
	1	
└─� [0]	1	
🧇 /adder4_t/carryin	1'h0	
	4'he	0 <u>(9</u>]1 [e
	1'h1	
	1'h1	
[↓ [1]	1'h1	
[0]	1'h0	
/adder4_t/carryout	1'h1	
·		

2) i) SERIAL ADDER

module adder_serial(input clk,rst,input en,input a,input b,output [3:0] result); reg [3:0] y; reg carry; always@(posedge rst or posedge clk) begin if (rst) begin y = 4'b0; carry = 1'b0; end else if (en) begin y[3] = y[2];y[2] = y[1];y[1] = y[0]; $\{carry, y[0]\} = a + b + carry;$ end end

assign result = y; endmodule

ii) SERIAL ADDER_TEST

```
module serial_adder_test;
reg clk,rst,en,a,b;
wire [3:0] result;
adder_serial U1 (clk,rst,en,a,b,result); //instantiation
initial
clk=1'b0;
always
#5 clk=~clk;
initial
begin
  rst =1'b1;en=1'b0;a=0;b=0;
#10 rst =1'b0;en=1'b1;a=1;b=0;
#10 rst =1'b0;en=1'b1;a=0;b=1;
#10 rst =1'b0;en=1'b1;a=1;b=1;
#10 rst =1'b0;en=1'b1;a=0;b=1;
#10 rst =1'b0;en=1'b1;a=1;b=0;
#200 $finish;
end
endmodule
```

	💶 Wave - Default 🚃		
	- 🏖 -	Msgs	
l	♦ /serial_adder_test/clk	1'h1	
l	/serial_adder_test/rst	1'h0	
l	/serial_adder_test/en	1'h1	
l	♦ /serial_adder_test/a	1'h1	
l	/serial_adder_test/b	1'h0	
l	/serial_adder_test/r	4'h0	0 11 13 16 10 18 10
l		1'h0	
l		1'h0	
l		1'h0	
	L [0]	1'h0	

LAB 7: 4-BIT COUNTERS

Objective: To write Verilog Code for the 4-bit Synchronous and Asynchronous counter And Test Bench for Verification, observe the waveform.

1) Asynchronous counter:

```
module async_counter ( count,reset,clk);
input wire reset, clk;
output reg [3:0] count;
always @(posedge clk or posedge rst)
begin
if (reset)
count <= 4'b0000;
else
count <= count + 4'b0001;
end
endmodule
```

Asynchronous counter_test

```
module async_counter_t ;
wire [3:0] count;
reg reset, clk;
initial
clk = 1'b0;
always
#5 clk = \sim clk;
async_counter m1 (count,reset,clk);
initial
begin
reset = 1'b0;
#15 reset =1'b1;
#30 reset =1'b0;
#300 $finish;
end
initial
$monitor ($time, "Output count = %d ",count );
endmodule
```

Wave - Default	-	
	1	
_ <mark>}</mark> ≁	Msgs	
	4'he	<u>{0) [] </u>
	1'h1	
	1'h1	
	1'h1	
L-🔶 [0]	1'h0	
/async_counter_t/r	1'h0	
/async_counter_t/ck	1'h0	

2) Synchronous Counter

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

Synchronous counter_test

```
module sync_counter_t ;
wire [3:0] count;
reg reset,clk;
initial
clk = 1'b0;
always
#5 clk = \sim clk;
Sync_counter m1 ( count,reset,clk);
initial
begin
 reset = 1'b0;
#15 reset =1'b1;
#30 reset =1'b0;
#300 $finish;
end
initial
$monitor ($time, "Output count = %d ",count );
endmodule
```

wave - Default		
<u>*</u> -	Msgs	
	4'he	<u>) (1) (0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (a) (b) (c) (d) (e) (f) (0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (a) (b) (c) (d) (e)</u>
	1'h1	
	1'h1	
	1'h1	
└-� [0]	1'h0	
/sync_counter_t/reset	1'h0	
/sync_counter_t/clk	1'h0	
,		

LAB 8: SUCCESSIVE APPROXIMATION REGISTER

Aim: To compile and to simulate the Verilog code for the successive approximation register.

Design Files:

I Main Design Module: sar.v

module sar (digitalout,done,comp,start,reset,clk); output [3:0] digitalout; output done; input clk, start, reset, comp; reg [3:0]ring_count; reg [3:0]digital; wire D4, set0, set1, set2, set3; assign D4 = ring_count[0]; assign done = !D4;always @(posedge clk or negedge reset) begin if (~reset) ring_count <= 4'b1000; else begin if (start) ring_count <= 4'b1000; else ring_count <= (ring_count>>1); end end assign set3 = ring_count[3]; assign set2 = ring_count[2]; assign set1 = ring_count[1]; assign set0 = ring count[0]; always @(posedge clk or negedge reset) begin if(~reset) digital[3] <= 1'b1; else if(start) digital[3] <= 1'b1; else if(set3) digital[3] <= comp;</pre> end always @(posedge clk or negedge reset) begin if(~reset) digital[2] <= 1'b1; else if(start)

digital[2] ≤ 1 'b1; else if(set2) digital[2] <= comp; end always @(posedge clk or negedge reset) begin if(~reset) digital[1] <= 1'b1; else if(start) digital[1] <= 1'b1; else if(set1) digital[1] <= comp; end always @(posedge clk or negedge reset) begin if(~reset) digital[0] ≤ 1 'b1; else if(start) digital[0] <= 1'b1; else if(set0) digital[0] <= comp; end assign digitalout = (digital) | (ring_count); endmodule

II Sub Design Module: dac.v

module dac (comp,sar_out,vref_d,vin_d,clk,start); output comp; input clk, start; input [3:0]sar_out; input [63:0]vref_d; input [63:0]vin_d; reg comp; real v dac, vref, vin; always @ (vin_d or start) begin vref = \$bitstoreal(vref_d); vin = \$bitstoreal(vin_d); end always @* begin if(start) comp = 1'b0;else begin $v_dac = (vref/15)*(sar_out);$ if (vin<v_dac)

```
comp = 1'b0;
else
comp = 1'b1;
end
end
endmodule
```

III Test Bench Module: sar_tb.v

```
module sar_tb;
reg clk, reset, start;
reg [63:0] vref_d,vin_d;
wire done, comp;
wire [3:0] digitalout;
real vref real = 7.5;
sar s1 (digitalout,done,comp,start,reset,clk);
dac d1 (comp,digitalout,vref_d,vin_d,clk,start);
initial
begin
clk = 1'b1;
start = 1'b1;
#4000 $finish;
end
always \#10 \text{ clk} = \text{~clk};
initial
begin
#1;reset = 1'b1;
#10; reset = 1'b0;
#1; reset = 1'b1;
end
initial
begin
#10;
stimulus (0.0,0.5, vref_real,8'd5);
end
task stimulus (input analog, input step, input reference, input [7:0]delay);
real analog, step;
real reference;
begin
while(analog <= reference)</pre>
begin
repeat(delay)
@(posedge clk);
start \leq 1'b0;
vref_d = $realtobits (reference);
vin_d = $realtobits (analog);
@(posedge done)
analog = analog + step;
@(posedge clk);
start <= 1'b1;
```

VLSI LAB MANUAL (15ECL77)

end end endtask endmodule

Name 🔻	Cursor 🔻	IS 6	00ns	700ns	800ns	900ns	1000ns	1100ns	1200ns	1300ns	1400ns	1500ns	1600ns	1700ns	1800ns	1900ns	2000ns	2100ns	2200ns	2300ns
	0	חחנ			MM					הההנ								NNN		MM
comp	0	\Box																		
🕂 👘 digitalout(3:0)	'h F	1	P	3	F	3	F	3	F	3	F	7	F	1	F	7	F	7	F	1
done	1	Г		ſ		ſ		ſ		J		J		J		J		J		J
	1																			
I start	1]]		l]				
⊕¶o vin_d(63:0)	'h 401a0000_00000000	3FF000	100_00	3FF8000	0_000000)	4000000	000000	40040000	_0000000	40080000	000000)	400c0000	_000000)	4010000	_000000)	4012000	000000)	4014000	000000)	40160000
🗄 📠 vref_d[63:0]	'h 40120000_00000000	401E00	100_000	00000																
		t.																		

ANALOG DESIGN

LAB 1: INVERTER

Objective: To design an Inverter with given specifications and verifying the following

- 1. Schematic:
- i) DC Analysis ii) Transient Analysis 2. Layout:

i) DRC ii) LVS iii) PEX

Design:

Device	Cutoff	Non saturation	Saturation
	Vgsp>Vtp	Vgsp <vtp< td=""><td>Vgsp<vtp< td=""></vtp<></td></vtp<>	Vgsp <vtp< td=""></vtp<>
P device		Vin < Vtp+Vdd	Vin <vtp+vdd< td=""></vtp+vdd<>
	Vin > Vtp+Vdd	Vdsp>Vgsp-Vtp	Vdsp <vgsp-vtp< td=""></vgsp-vtp<>
		Vout> Vin-Vtp	Vout <vin-vtp< td=""></vin-vtp<>
	Vgsn <vtn< td=""><td>Vgsn>Vtn</td><td>Vgsn>Vtn</td></vtn<>	Vgsn>Vtn	Vgsn>Vtn
		Vin> Vtn	Vin>Vtn
n device			
	Vin <vtn< td=""><td>Vdsn<vgs-vtn< td=""><td>Vdsn>Vgs-Vtn</td></vgs-vtn<></td></vtn<>	Vdsn <vgs-vtn< td=""><td>Vdsn>Vgs-Vtn</td></vgs-vtn<>	Vdsn>Vgs-Vtn
		Vout <vin-vtn< td=""><td>Vout>Vin-Vtn</td></vin-vtn<>	Vout>Vin-Vtn

Region A:
$$0 \le Vin \le Vin \le Vin$$
, $Vout=Vdd$;
Region B: $Vtn \le Vin \le Vin$

Region E: Vin>= Vdd-Vtp, Vout = 0

VLSI LAB MANUAL (15ECL77)

2019 - 20

Region	Condition	P device	n device
А	0≤Vin< Vtn	Nonsaturation	Cutoff
В	$Vtn \le Vin \le Vdd/2$	Nonsaturation	Saturation
С	Vin=Vdd/2	Saturation	Saturation
D	Vdd/2 <vin≤vdd -="" td="" vtp<=""><td>Saturation</td><td>Nonsaturation</td></vin≤vdd>	Saturation	Nonsaturation
Е	Vin>Vdd- Vtp	Cutoff	Nonsaturation

Inverter Schematic:

Specifications:

Library name	Cell name	Properties
Generic_250nm.lib	pmos	W=1.5µ, L=250n
Generic_250nm.lib	nmos	W=1.5µ, L=250n

							15						
						14	1						
						E.							
				20103		: %						2,490	
						·	Car Marca		-				
						- 42			12221-02				
						- 32	DMO		29.23				
							TTX-		1200				
						IL-r		1.54	1520.05				
			-				TW	1.50					
			122		3.50								
			205				15Z3	50m					
			12				fine	gers					
			136	5520			m=1					2008	
			1.00										
			33										
		A Press											
		#	1						-				
			1.55										
			18										
			1										
			- 64	 •>>> 		• • •	NUMER	175				+	
			12			1000	The	514					
			3				****	i Sui					
			<u> </u>				1.00	A - 214	i				
						100	1=24	STIN.	12,533				
						· ·	100	TON	100				
						🔺	LIN	der >.					
							10-1		-				
						- 24							

Tanner T-Spice 2016.2.10 C:\Users\Admin\AppData\Local\Temp\inverter_n.sp 15:57:50 10/13/18 A:GND:V 5.000 4.500 4.000 3.500 3.000 12.500 2.000 1.500 1.000 0.500 0.000 Y:GND:V 5.000 4.500 4.000 3.500 3.000 2.500 2.000 1.500 1.000 0.500 0.000 50.0n Seconds 10.0n 20.0n 30.0n 40.0n 60.0n 70.0n 80.0n 90.0n 100.0r 0.0n

Transient analysis:

DC Analysis



Spice Commands :

// INVERTER TRANSIENT ANALYSIS

vsource vdd gnd 5 vsource1 A gnd BIT ({10010001} pw=10n lt=10n ht=10n on=5 rt=.1n ft=.1n delay=0) .tran .1n 100n start=0 .print tran v(A,gnd) v(Y,gnd) .include "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\SpecialDevices.md" .lib "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib"TT .dc lin source vsource1 0 5 0.2 .print dc v(A) v(Y) .end

Inverter Layout:



Objective: To design a single stage Differential Amplifier with given specifications and verifying the following

Schematic:

 DC Analysis
 AC Analysis
 Transient Analysis

 Layout:

i) DRC ii) LVS iii) PEX

Design:



Iout =
$$\frac{gm_1gm_3rp_1}{1+gm_3rp_1}Vgs_1 - gm_2Vgs_2$$

Where $gm_1=gm_2=gmd$, $rp_1=rds_1||rds_3|$

rout =
$$\frac{1}{gds_2 + gds_4}$$
 or $\frac{2}{(\Box n + \Box p)I_5}$
Av = $\frac{gmd}{gds_2 + gds_4}$

 $(W/L)_{1=}(W/L)_2$

Av =
$$\frac{\sqrt{k'_1 Iss w_1 / l_1}}{(\Box_2 + \Box_4)(Iss / 2)}$$

(W/L) _{3 = (W/L)}
⁴
Vgs₃ = $\sqrt{\frac{2I_5}{k'_p (w / l)_3}} + Vtn$
(W/L) ₅ = $\sqrt{\frac{2I_5}{k'_n Vds \frac{2}{5}(sat)}}$
Vds ₅ (sat) = V _{Ic} min - Vss - Vgs₁

Differential Amplifier Schematic:



Transient analysis:



DC Analysis:



AC Analysis:

Dept. Of ECE ,CIT-Gubbi

VLSI LAB MANUAL (15ECL77)

15:33:14 10/13/18 Tanner T-Spice 2016.2.10 C:\Users\Admin\AppData\Local\Temp\Diff_amp.sp out:VdB 30.00 25.00 20.00 15.00 מ 10.00 5.00 0.00 -5.00 out:Vp 180.0 170.0 160.0 150.0 140.0 130.0 n120.0 110.0 100.0 90.0 80.0 70.0 60.0 50.0 10.00 100.0 1.000k 10.00k 1.000M 10.00M 100.0M 1.000G 1.000 10.000 100.0k

SPICE Commands :

.tran 10n 100m or .tran 10n 5m .ac dec 10 1 10G .dc lin source v1 -5 5 .1 v1 Vinp Vinm SIN (0 5m 1k) AC 1 v2 vdd GND 5 v3 Vbias GND 0.8 v4 Vinm GND 1.3 .print tran v(Vinp) v(out) .print dc v(out) .print ac vdb(out) vp(out) .lib "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT

2019 - 20

Differential Amplifier Layout



DRC results: No DRC errors found, DRC results verified

Objective: To design a Common Source Amplifier with given specifications and Verify the following

- 1. Schematic:
- i) DC Analysis ii) AC Analysis iii) Transient Analysis
- 2. Layout: i) DRC

Design:



Av =Vout/Vin= - [resistance in drain/resistance in source]

$$= - \frac{r_{01} \| r_{02}}{\frac{1}{gm_1}}$$

Av = -gm₁($r_{01} || r_{02}$)

CS Amplifier Schematic:



Transient results:



AC results:



Spice Commands Common Source Amplifier:

v1 vinm vinp SIN(0 100m 1k) AC 1 180 v2 vinp Gnd 3.2 .dc lin source v2 0 10 1 V4 Vdd Gnd 5 v3 vbias Gnd 3.87 .tran .1m 5m .ac dec 10 10 1G .print ac vdb(out) vp(out) .print tran v(vinp,vinm) v(out) *.include "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\SpecialDevices.md" .lib "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT .end .op **Objective:** To design a Common Drain Amplifier with given specifications and verify the following

- 1. Schematic:
- i) DC Analysis ii) AC Analysis iii) Transient Analysis 2. Layout:
 - i) DRC

Design:



I1=0; I2= - $gm_1 Vgs_1 + (gmb_1+gds_1)Vo$

 $= -gm_1 (V_1 - V_2) + (gmb_1 + gds_1)Vo$

 $Av = -\frac{gm_1}{gm_1 + gmb_1 + gds_1 + gds_2}$

CD Amplifier Schematic:



Transient results:



AC Results:



Coursed

SPICE Commands:

```
v1 vinm vinp SIN(0 5m 1k) AC 1
v2 vinp gnd 3.1
.dc lin source v1 -5 5 .1
v3 vdd gnd 5
v4 vbias gnd 3.78
.ac dec 10 1k 105
.tran .1m 5m
.print tran v(out) v(vinm)
.print ac vp(out) vdb(out)
.print dc v(out)
.op
.lib "C:\Users\Admin\Documents\Tanner EDA\Tanner Tools v2016.2\Process\Generic_250nm\Generic_250nm_Tech\Generic_250nm.lib" TT
.end
```

LAB 5: R2R_DAC ladder

Schematic of R2R DAC ladder:



Figure: Schematic of R2R_DAC ladder

The final output voltage VOUT depends on the value of B(0 to 15), following the given formula:

Table below gives the value of VOUT versus the input code, with Vdac equal to 1v.

Vout =
$$V_{\text{DAC}}(2^N - B)/2^N$$

Table: Output voltage produced by the 4-bit R-2R DAC versus input code B

B3	B2	B1	B0	Vout
0	0	0	0	1
0	0	0	1	0.9
0	0	1	0	0.83
0	0	1	1	0.78
0	1	0	0	0.65
0	1	0	1	0.61
0	1	1	0	0.56
0	1	1	1	0.54
1	0	0	0	0.38
1	0	0	1	0.34
1	0	1	0	0.31
1	0	1	1	0.29
1	1	0	0	0.25
1	1	0	1	0.23
1	1	1	0	0.2
1	1	1	1	0.2

// Spice Commands (4 bit DAC)

.tran 100n 8500n .print tran v(VOUT) .lib "C:\IMS\VTU_legacy\4bitdac\Generic_025.lib" TT .end

Result Analysis:



Figure 1.64: Result Analysis of R2R_DAC ladder // Spice Commands for R2R_DAC

V2 vdd GND 5 V3 VIN GND 2.5 .tran 10n 8500n .lib "C:\IMS\vtuexperiments\commondrain\Generic_025.lib" TT .print tran v(B4) v(B5) v(B6) v(B7) v(VOUT) V8 B4 GND PULSE(5 0 0 1n 1n 500n 1000n) V9 B5 GND PULSE(5 0 0 1n 1n 1000n 2000n) V10 B6 GND PULSE(5 0 0 1n 1n 2000n 4000n) V11 B7 GND PULSE(5 0 0 1n 1n 4000n 8000n) .END

Additional Digital Designs

LAB 6: Synchronous FIFO

Objective: To design synchronous FIFO (single Clock) using verilog.

```
module syn_fifo (
clk
      , // Clock input
     , // Active high reset
rst
wr_cs , // Write chip select
rd_cs , // Read chipe select
data_in , // Data input
rd_en , // Read enable
wr_en , // Write Enable
data_out , // Data Output
empty , // FIFO empty
full
       // FIFO full
);
// FIFO constants
parameter DATA_WIDTH = 8;
parameter ADDR_WIDTH = 8;
parameter RAM_DEPTH = (1 << ADDR_WIDTH);
// Port Declarations
input clk;
input rst;
input wr_cs ;
input rd_cs ;
input rd_en ;
input wr_en ;
input [DATA_WIDTH-1:0] data_in;
output full;
output empty;
output [DATA_WIDTH-1:0] data_out;
```

//-----Internal variables------

```
reg [ADDR_WIDTH-1:0] wr_pointer;
reg [ADDR_WIDTH-1:0] rd_pointer;
reg [ADDR_WIDTH :0] status_cnt;
reg [DATA_WIDTH-1:0] data_out ;
wire [DATA_WIDTH-1:0] data_ram;
//-----Variable assignments------
assign full = (status_cnt == (RAM_DEPTH-1));
assign empty = (status_cnt == 0);
//-----Code Start-----
always @ (posedge clk or posedge rst)
begin : WRITE_POINTER
if (rst) begin
  wr_pointer <= 0;
 end else if (wr_cs && wr_en ) begin
  wr_pointer <= wr_pointer + 1;</pre>
end
end
always @ (posedge clk or posedge rst)
begin : READ_POINTER
if (rst) begin
```

rd_pointer <= 0; end else if (rd_cs && rd_en) begin rd_pointer <= rd_pointer + 1; end end

always @ (posedge clk or posedge rst)
begin : READ_DATA
if (rst) begin
 data_out <= 0;
end else if (rd_cs && rd_en) begin
 data_out <= data_ram;
end</pre>

end

end

```
ram_dp_ar_aw #(DATA_WIDTH,ADDR_WIDTH)DP_RAM (
```

```
.address_0 (wr_pointer), // address_0 input
```

```
.data_0 (data_in) , // data_0 bi-directional
```

.cs_0 (wr_cs) , // chip select

.we_0 (wr_en) , // write enable

 $.oe_0$ (1'b0) , // output enable

.address_1 (rd_pointer) , // address_q input

.data_1 (data_ram), // data_1 bi-directional

.cs_1 (rd_cs) , // chip select

.we_1 (1'b0) , // Read enable

.oe_1 (rd_en) // output enable

```
);
```

Endmodule

LAB 7: Single Port RAM Synchronous Read/Write

Objective: To design Single Port RAM Synchronous Read/Write using Verilog

```
module ram_sp_sr_sw (
```

clk	, // Clock Input
address	, // Address Input
data	, // Data bi-directional
cs	, // Chip Select
we	, // Write Enable/Read Enable
oe	// Output Enable
);	

parameter DATA_WIDTH = 8 ;
parameter ADDR_WIDTH = 8 ;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;</pre>

//-----Input Ports----input clk ;
input [ADDR_WIDTH-1:0] address ;
input cs ;
input we ;
input we ;

//-----Inout Ports-----inout [DATA_WIDTH-1:0] data ;

//-----Internal variables----reg [DATA_WIDTH-1:0] data_out ;
reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1];
reg oe_r;

//-----Code Starts Here-----

VLSI LAB MANUAL (15ECL77)

// Tri-State Buffer control
// output : When we = 0, oe = 1, cs = 1
assign data = (cs && oe && !we) ? data_out : 8'bz;

// Memory Write Block // Write Operation : When we = 1, cs = 1always @ (posedge clk) begin : MEM_WRITE if (cs && we) begin mem[address] = data; end end // Memory Read Block // Read Operation : When we = 0, oe = 1, cs = 1always @ (posedge clk) begin : MEM_READ if (cs && !we && oe) begin data_out = mem[address]; $oe_r = 1;$ end else begin oe_r = 0;end end

 $endmodule \, /\!/ \, End \; of \; Module \; ram_sp_sr_sw$

APPENDIX DESIGN OF AMPLIFIERS:

Single Stage Diffrential Amplifier



Common source amplifier:

VLSI LAB MANUAL (15ECL77)



<u>NMOS</u>	PMOS
□ □ =0.0278	□ □ =0.0105
$T_{ox=} 5.7 \times 10^{-9}$	$T_{ox=} 5.7 \times 10^{-9}$
$V_{Tn} = 0.385$	$V_{Tn} = 0.545$
Cox=6.058x10 ⁻³	Cox=6.058x10 ⁻³

For Transistor M1 & M8 (PMOS): $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{p} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x80x10^{-6}}{0.0105x6.058x10^{-3}x(0.545)^{2}}$ =8.45 Take W=2.5 L = 0.29For Transistor M2 & M3 (PMOS) :(differential pair) $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{p} C_{ox} \left(V_{DSAT} \right)^{2} \right]} = \frac{2x40x10^{-6}x10^{3}}{0.0105x6.058x10^{-3}x(0.545)^{2}} = 4.23$ Take W=2.5 L = 0.60For Transistor M9 & M7 (NMOS) : $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{n} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x80x10^{-6}}{0.0278x6.058x10^{-3}x(0.385)^{2}} = 6.4$ Take W=2.5 L = 0.4For Transistor M4& M5 (NMOS) : $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{n} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x40x10^{-6}x10^{3}}{0.0278x6.058x10^{-3}x(0.385)^{2}} = 3.2$ Take W=2.5 L = 0.77

2019 - 20

For Transistor M6 (NMOS):

 $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{n} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x80x10^{-6}}{0.0278x6.058x10^{-3}x(0.385)^{2}} = 6.4$ Assume W=2.5 L = 0.385

Common Drain amplifier:

PMOS NMOS □ □ =0.0278 **0 0** =0.0105 $T_{ox=} 5.7 \times 10^{-9}$ $T_{ox=} 5.7 \times 10^{-9}$ $V_{Tn} = 0.35$ $V_{Tn} = 0.565$ Cox=6.058x10⁻³ $Cox=6.058x10^{-3}$ For Transistor M1, M2 & M3 (PMOS): $W = 2I_{DSAT} = 2x40x10^{-6}$ =3.9 $\overline{L}^{-} \left[\int_{p} C_{ox} (V_{DSAT})^{2} - \overline{0.0105x6.058x10^{-3}x(0.565)^{2}} \right]$ Take W=1.5 L = 0.4For Transistor M4 & M5 (PMOS): $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{p} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x20x10^{-6}}{0.0105x6.058x10^{-3}x(0.565)^{2}} = 2.0$ Take W=1.5 L = 0.74For Transistor M8 & M9 (NMOS): $\frac{W}{L} = \frac{2I_{DSAT}}{\left[\int_{n} C_{ox} (V_{DSAT})^{2} \right]} = \frac{2x20x10^{-6}}{0.0278x6.058x10^{-3}x(0.35)^{2}} = 1.95$

VLSI LAB MANUAL (15ECL77)

Assume W=1.5 L = 0.75 For Transistor M6 & M7 (NMOS): $2x40x10^{-6}$ =3.79 $W = 2I_{DSAT} =$ $L = \left[\int_{n} C_{ox} (V_{DSAT})^{2} - 0.0278x6.058x10^{-3}x(0.35)^{2} \right]$ А S S u m e W = 1 . 5 ; L =0 • 4
QUESTION BANK

- 1. a) Design an Inverter with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for Parallel / Serial Adder and observe the waveform.

- 2. a) Design an Inverter with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.
 - b) Write Verilog code and test bench for verification, for 4-bit Synchronous / Asynchronous Counter and observe the waveform
- 3. a) Design a Common Source Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code, their test bench for verification and observe the waveform for RS Flip Flop

- 4. a) Design a Common Drain Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for an NAND Gate and observe the waveform.

- 5. a) Design a Differential Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for a transmission gate and observe the waveform.

- 6. a) Design an Operational Amplifier (Diff amp + CS amp) with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification and observe the waveform for D FF.

- 7. a) Design a 4-bit R-2R based DAC with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify the Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for an inverter and observe the waveform

8. a) Design a Common Source Amplifier with given specifications, completing the design flow mentioned below.

VLSI LAB MANUAL (15ECL77)

- i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
- ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code, their test bench for verification and observe the waveform for OR gate.

- 9. a) Design a Common Drain Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for a JK FF and observe the waveform.

- 10. a) Design a Differential Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for a buffer and observe the waveform.

- 11. a) Design a Common Source Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code, their test bench for verification and observe the waveform for AND gate

- 12. a) Design a Common Drain Amplifier with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify AC analysis, DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code, their test bench for verification and observe the waveform for XOR gate.

- 13. a) Design an Inverter with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for SAR based ADC and observe the waveform.

- 14. a) Design an Inverter with given specifications, completing the design flow mentioned below.
 - i. Draw the schematic and verify DC analysis and Transient analysis.
 - ii. Draw the Layout, verify DRC, check for LVS and extract RC.

b) Write Verilog code and test bench for verification, for an XNOR gate and observe the waveform.

VIVA – VOCE MODEL QUESTIONS

- 1. What is CMOS technology?
- 2. Which is the technology used in VLSI lab?
- 3. Explain ASIC design flow?
- 4. Explain the operation of NMOS and PMOS MOSFETS.
- 5. Explain the regions of operations for both NMOS and PMOS.
- 6. Which are majority carriers in NMOS and PMOS?
- 7. Explain the operation of CMOS inverter.
- 8. What is the difference between CMOS and BIPOLAR technology?
- 9. What are advantages and disadvantages of CMOS technology?
- 10. Explain 5 regions of operation for an inverter.
- 11. What is transient, DC and AC analysis?
- 12. What is noise margin?
- 13. What is channel length modulation?
- 14. What is HOT electron effect?
- 15. What is routing?
- 16. What is floor plan?
- 17. What is placement?
- 18. What is DRC, LVS?
- 19. Explain the fabrication process of both NMOS and PMOS.
- 20. What is an amplifier?
- 21. Write the design of common source and common drain amplifier.
- 22. What is differential amplifier? Explain the design.
- 23. What are CMRR, SNR, and PSRR?
- 24. What is operational amplifier and explain the basic building block of an OP-AMP?
- 25. Explain the design procedure of an OP-AMP.
- 26. What is test bench?
- 27. What is the difference between simulation and synthesis?
- 28. What is a buffer?
- 29. Explain the CMOS circuits for basic gates.
- 30. What is the difference between stick diagram and layout?
- 31. Which is the layer used to connect the components?
- 32. What are the different styles of programming?

- 33. What is user defined packages?
- 34. What is the function of task and display?
- 35. Explain the operation of transmission gates.
- 36. Explain D flip-flop with truth table?
- 37. Explain T flip-flop with truth table?
- 38. Explain JK flip-flop with truth table?
- 39. Explain SR flip-flop with truth table?
- 40. Explain MS flip-flop with truth table?
- 41. What are glitches?
- 42. What is pinch of region?
- 43. What is setup and hold time?
- 44. What is race around condition?
- 45. Explain latch up problem with circuit.
- 46. How to overcome from latch up problem?
- 47. What is the difference between serial adder and parallel adder?
- 48. Explain the circuits for half adder and full adder.
- 49. What is the difference between asynchronous and synchronous counter?
- 50. Design ripple counter?
- 51. Design a synchronous counter for given mod.
- 52. Write the layout for the basic gates and given expression.
- 53. Write the stick diagram for the basic gates and given expression.
- 54. Write the color code for n+ and p+ diffusion?
- 55. What is lambda based design rules?
- 56. What is metallization?
- 57. What are Vias and guard rings?
- 58. What is a function?
- 59. What are diffusion, oxidation, ion implantation and etching?
- 60. What is SOI?

REFERENCES

- [1] Design of analog CMOS integrated circuits, B Razavi, First Edition, Mcgraw Hill 2001
- [2] Design, Layout, simulation, R.jacob Baker, Harry W Li, David E Boyci, CMOS Circuit, PHI edition, 2005.
- [3] CMOS Mixed Signal Circuit Design (Vol II of CMOS: Circuit design, Layout and simulation), R. Jacob. Baker, CMOS-IEEE press and wiley interscience 2002.
- [4] CMOS analog circuit design, P E Allen and D R Holberg, Second Edition, oxford university press 2002.
- [5] Fundamentals of logic design with VHDL, Stephen Brown & Zvonko vranesic, Tata McGraw-Hill, New Delhi, Second Edition, 2007.
- [6] Verilog HDL: A guide to digital design and synthesis, Samir palnitkar, Second edition