

# DDVH COURSE FILE

**GEETHANJALI COLLEGE OF ENGINEERING AND  
TECHNOLOGY**

**DEPARTMENT OF *Electronics and communications Engineering***

(Name of the Subject / Lab Course) : Digital Design Using Verilog HDL

(JNTU CODE ) : A40410

Programme : UG

**Branch: ECE**

**Version No : 01**

**Year: II**

**Updated on : 17/11/2015**

**Semester: II**

**No. of pages :**

**Classification status (Unrestricted / Restricted )**

**Distribution List :**

**Prepared by :1) Name : B Sreelatha**

**1) Name : T Rama Krishna**

**2) Sign :**

**2) Sign :**

**3) Design : Assoc. professor**

**3) Design : Assoc. professor**

**4) Date : 17/11/2015**

**4) Date : 17/11/2015**

**Verified by : 1) Name :**

**\* For Q.C Only.1) Name :**

**2) Sign :**

**2) Sign :**

**3) Design :**

**3) Design :**

**4) Date :**

**4) Date :**

**Approved by : (HOD ) 1) Name :**

**2) Sign :**

**3) Date :**

# **Contents required for course file**

1. Cover page
2. Syllabus copy
3. Vision of the Department
4. Mission of the Department
5. PEOs and POs
6. Course objectives and outcomes
7. Brief importance of the course and how it fits into the curriculum
8. Prerequisites if any
9. Instructional Learning Outcomes
10. Course mapping with PEOs and POs
11. Class Time table
12. Individual Time table
13. Lecture schedule with methodology being used / adopted
14. Detailed notes
15. Additional topics
16. University previous Question papers of previous years
17. Question Bank
18. Assignment Questions
19. Unit-wise quiz questions and long answer questions
20. Tutorial Problems
21. Known curriculum Gaps (If any) and inclusion of the same in lecture schedule
22. Discussion topics, if any
23. References, Journals, websites and E-links if any
24. Quality measurement Sheets
  - a. Course end survey
  - b. Teaching Evaluation
25. Students List
26. Group-Wise students list for discussion topics

## 2. Syllabus copy

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

II Year B.Tech. ECE -II Sem

L T/P/D C  
4 -/- - 4

### Digital Design using Verilog HDL

#### UNIT I:

**Introduction to Verilog HDL:** Verilog as HDL, Levels of Design Description, Concurrency, Simulation and Synthesis, Functional Verification, System Tasks, Programming Language Interface (PLI), Module, Simulation and Synthesis Tools.

**Language Constructs and Conventions:** Introduction, Keywords, Identifiers, White Space Characters, Comments, Numbers, Strings, Logic Values, Strengths, Data Types, Scalars and Vectors, Parameters, Operators.

#### UNIT II:

**Gate Level Modeling:** Introduction, AND Gate Primitive, Module Structure, Other Gate Primitives, Illustrative Examples, Tri-State Gates, Array of Instances of Primitives, Design of Flip-flops with Gate Primitives, Delays, Strengths and Construction Resolution, Net Types, Design of Basic Circuits.

**Modeling at Dataflow Level:** Introduction, Continuous Assignment Structure, Delays and Continuous Assignments, Assignment to Vectors, Operators.

#### UNIT III:

**Behavioral Modeling:** Introduction, Operations and Assignments, Functional Bifurcation, *Initial* Construct, *Always* Construct, Assignments with Delays, *Wait* construct, Multiple Always Blocks, Designs at Behavioral Level, Blocking and Non-Blocking Assignments, The *case* statement, Simulation Flow *if* and *if-else* constructs, *Assign-De-Assign* construct, *Repeat* construct, *for* loop, the *Disable* construct, *While* loop, *Forever* loop, Parallel Blocks, *Force-Release* construct, Event.

#### UNIT IV:

**Switch Level Modeling:** Basic Transistor Switches, CMOS Switch, Bi-directional Gates, Time Delays with Switch Primitives, Instantiations with Strengths and Delays, Strength Contention with Trireg Nets, Exercises.

**System Tasks, Functions and Compiler Directives:** Parameters, Path Delays, Module Parameters, System Tasks and Functions, File-Based Tasks and Functions, Computer Directives, Hierarchical Access, User Defined Primitives.

#### UNIT V:

**Sequential Circuit Description:** Sequential Models – Feedback Model, Capacitive Model, Implicit Model, Basic Memory Components, Functional Register, Static Machine Coding, Sequential Synthesis.

**Components Test and Verification:** Test Bench- Combinational Circuit Testing, Sequential Circuit Testing, Test Bench Techniques, Design Verification, Assertion Verification.

#### TEXT BOOKS:

- 1.T.R. Padmanabhan, B. Bala Tripura Sundari , Design through Verilog HDL –, Wiley, 2009.
- 2.Zainalabdien Navabi, Verilog Digital System Design, TMH, 2<sup>nd</sup> Edition.

#### **REFERENCE BOOKS:**

- 1.Fundamentals of Logic Design with Verilog Design– Stephen. Brown and Zvonko Vranesic, TMH, 2<sup>nd</sup> Edition 2010.
- 2.Advanced Digital Logic Design using Verilog, State Machine & Synthesis for FPGA – Sunggu Lee, Cengage Learning , 2012.
- 3.Verilog HDL – Samir Palnitkar, 2<sup>nd</sup> Edition, Pearson Education, 2009.
- 4.Advanced Digital Design with Verilog HDL – Michael D. Ciletti, PHI, 2009.

### **3. Vision of the Department**

To impart quality technical education in Electronics and Communication Engineering emphasizing analysis, design/synthesis and evaluation of hardware/embedded software using various Electronic Design Automation (EDA) tools with accent on creativity, innovation and research thereby producing competent engineers who can meet global challenges with societal commitment.

### **4. Mission of the Department**

- i. To impart quality education in fundamentals of basic sciences, mathematics, electronics and communication engineering through innovative teaching-learning processes.
- ii. To facilitate Graduates define, design, and solve engineering problems in the field of Electronics and Communication Engineering using various Electronic Design Automation (EDA) tools.
- iii. To encourage research culture among faculty and students thereby facilitating them to be creative and innovative through constant interaction with R & D organizations and Industry.
- iv. To inculcate teamwork, imbibe leadership qualities, professional ethics and social responsibilities in students and faculty.

## **5. PEOs and POs**

### **Program Educational Objectives (PEOs):**

- I. To prepare students with excellent comprehension of basic sciences, mathematics and engineering subjects facilitating them to gain employment or pursue postgraduate studies with an appreciation for lifelong learning.
- II. To train students with problem solving capabilities such as analysis and design with adequate practical skills wherein they demonstrate creativity and innovation that would enable them to develop state of the art equipment and technologies of multidisciplinary nature for societal development.
- III. To inculcate positive attitude, professional ethics, effective communication and interpersonal skills which would facilitate them to succeed in the chosen profession exhibiting creativity and innovation through research and development both as team member and as well as leader.

### **Program Outcomes (POs):**

1. An ability to apply knowledge of Mathematics, Science, and Engineering to solve complex engineering problems of Electronics and Communication Engineering systems.
2. An ability to model, simulate and design Electronics and Communication Engineering systems, conduct experiments, as well as analyze and interpret data and prepare a report with conclusions.
3. An ability to design an Electronics and Communication Engineering system, component, or process to meet desired needs within the realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability and sustainability.
4. An ability to function on multidisciplinary teams involving interpersonal skills.
5. An ability to identify, formulate and solve engineering problems of multidisciplinary nature.
6. An understanding of professional and ethical responsibilities involved in the practice of Electronics and Communication Engineering profession.
7. An ability to communicate effectively with a range of audience on complex engineering problems of multidisciplinary nature both in oral and written form.
8. The broad education necessary to understand the impact of engineering solutions in a global, economic, environmental and societal context.
9. A recognition of the need for, and an ability to engage in life-long learning and acquire the capability for the same.

10. A knowledge of contemporary issues involved in the practice of Electronics and Communication Engineering profession
11. An ability to use the techniques, skills and modern engineering tools necessary for engineering practice.
12. An ability to use modern Electronic Design Automation (EDA) tools, software and electronic equipment to analyze, synthesize and evaluate Electronics and Communication Engineering systems for multidisciplinary tasks.
13. Apply engineering and project management principles to one's own work and also to manage projects of multidisciplinary nature.

## **6. Course Objectives and Outcomes**

### **Course Objectives ( as per JNTU-H)**

This course teaches:

- Designing digital circuits, behavioral and RTL modeling of digital circuits using Verilog HDL.
- Verifying these models and synthesizing RTL models to standard cell libraries and FPGAs.
- Students gain practical experience by designing, modeling, implementing and verifying several digital circuits.

This course aims to provide students with the understanding of the different technologies related to HDLs, construct, compile and execute Verilog HDL programs using provided software tools. Design digital components and circuits that are testable, reusable and synthesizable.

### **Course Outcomes**

After the completion of the course, the student would be able to

- CO 1: Describe Verilog hardware description languages (HDL).
- CO 2: Design Digital Circuits.
- CO 3: Write behavioral models of digital circuits.
- CO 4: Write Register Transfer Level (RTL) models of digital circuits.
- CO 5: Verify behavioral and RTL models.
- CO 6: Describe standard cell libraries and FPGAs.
- CO 7: Synthesize RTL models to standard cell libraries and FPGAs.
- CO 8: Implement RTL models on FPGAs and Testing & Verification.

## **7. Brief Importance of the Course and how it fits into the curriculum**

- a. This is the basic fundamental subject for the programming of the digital Electronics.
- b. This subject is required to understand the programming of the combinational and sequential circuit designs.
- b. By studying this subject, the students can design and understand digital systems and its importance.

c. The students logical thinking capability will be improved which will help in placements and in their future technical assignments.

## **8. Prerequisites if any**

1. Concepts of switching theory and logic design.

## **9. Instructional Learning Outcomes**

Learning outcomes are the key abilities and knowledge that will be assessed

### **Unit – I:**

#### **Introduction to Verilog HDL**

- Students understand the importance of HDL (Hardware Descriptive Language) and apply the knowledge of Boolean algebra to design and development digital Systems.
- Understand the difference between concurrent and sequential programming
- Issues related to simulation and synthesis models.

#### **Language Constructs and Conventions:**

- Knowledge of language constructs
- Pertaining to Semantic and syntactical errors in programming using HDL
- Limitation of HDL

### **Unit- II:**

#### **Gate Level Modeling**

- Student will learn conventional structural modeling of digital systems.
- Learn to model language defined primitive gates
- Understand importance component structure in Verilog.
- Learn Hierarchical digital system building

#### **Modeling at Dataflow Level**

- Continuous assignment operator based model construction will be learnt.

### **Unit – III:**

#### **Behavioral Modeling:**

- Students will be familiarized to high level abstraction of digital systems with behavioral modeling of systems.
- RTL modeling of digital systems



- Will be made familiar to behavioral constructs like ‘always’ ,’initial’, ‘if’, ‘if-else’, ‘case’..etc
- Register and array modeling

#### **Unit – IV:**

##### **Switch Level Modeling:**

- Students will learn low-level abstraction of digital systems.
- Switch level primitives will be learnt
- Made to familiarize to different strengths of logic values

##### **System Tasks, Functions and Compiler Directives:**

- Understand the importance of system tasks and functions
- Understand compiler directives.
- Understand user defined primitives and learn to model systems using UDP
- Learn the intricacies associated with usage of functions and tasks in packages
- Learn package declaration and package usage in project building

#### **Unit – V:**

##### **Sequential Circuit Description:**

- Learn to model Sequential circuits at higher level of abstraction using RTL modeling
- Will be able to design static and dynamic memories
- Will learn to model in behavioral style of binary encoding and one hot encoding

##### **Component Test and Verification:**

- Students understand Test bench generation
- Producing of test vectors to test the digital systems at higher level of abstraction.

## **10. Course mapping with PEOs and POs**

##### **Mapping of Course with Programme Educational Objectives:**

S.No	Course component	code	course	Semester	PEO 1	PEO 2	PEO 3
1	Digital Electronics		DDVH	2	√	√	

##### **Mapping of Course outcomes with Programme outcomes:**

\*When the course outcome weightage is < 40%, it will be given as moderately correlated (1).

\*When the course outcome weightage is >40%, it will be given as strongly correlated (2).

POs	1	2	3	4	5	6	7	8	9	10	11	12	13	Digital Systems
DDVH														
CO 1: Describe Verilog hardware description languages (HDL).	√	√	√					√		√	√	√		
CO 2: Design Digital Circuits.		√	√					√		√	√	√		
CO 3: Write behavioral models of digital circuits.	√	√	√					√		√	√	√		
CO 4: Write Register Transfer Level (RTL) models of digital circuits.	√	√	√					√		√	√	√		
CO 5: Verify behavioral and RTL models.	√	√	√					√		√	√	√		
CO 6: Describe standard cell libraries and FPGAs.	√	√	√					√		√	√	√		
CO 7: Synthesize RTL models to standard cell libraries and FPGAs.	√	√	√					√		√	√	√		
CO 8: Implement RTL models on FPGAs and Testing & Verification.	√	√	√					√		√	√	√		

### **11. Time table of concerned class**

### **12. Individual time table**

### 13. Lecture schedule with methodology being used / adopted

SL. NO	Unit No.	Total no. of periods	Date	Topics to be covered in one lecture	Regular/ Additional	Teaching aids used LCD/ OHP/ BB	Remarks
1	I	6		Verilog as HDL, Levels of Design Description, Concurrency, Verilog as HDL, Levels of Design Description	Regular	OHP, BB	
2				Concurrency Simulation and Synthesis, Functional Verification, System Tasks, Programming Language Interface (PLI)	Regular	OHP, BB	
3				Module, Simulation and Synthesis Tools  LANGUAGE CONSTRUCTS AND CONVENTIONS Introduction, Keywords	Regular	OHP, BB	
4				Identifiers, White Space Characters, Comments, Numbers, Strings, Logic Values, Strengths	Regular	BB	
5				Data Types, Scalars and Vectors, Parameters, Memory, Operators, System Tasks, Exercises.	Regular	OHP, BB	
6				Tutorial class-1		BB	
7	II	11		Introduction, AND Gate Primitive, Module Structure	Regular	OHP, BB	
8				Other Gate Primitives, Illustrative	Regular	OHP, BB	

				Examples, Tri-State Gates			
9				Array of Instances of Primitives, Additional Examples	Regular	OHP,BB	
10				Design of Flip-flops with Gate Primitives, Delays, Strengths and Contention Resolution	Regular	BB	
11				Net Types, Design of Basic Circuits, Exercises.	Regular	BB	
12				Verilog designs for various rounding methods	<b>Additional</b>	OHP,BB	
13				Introduction, Continuous Assignment Structures, Delays and Continuous Assignments	Regular	OHP,BB	
14				Assignment to Vectors, Operators.	Regular	OHP,BB	
15				Tutorial class-2		BB	
16				Solving University papers		BB	
17				Assignment test-1		BB	
18	<b>III</b>	6		Introduction, Operations and Assignments, Functional Bifurcation, <i>Initial</i> Construct, <i>Always</i> Construct, Examples	Regular	OHP,BB	
19				Assignments with Delays, <i>Wait</i> construct, Multiple Always Blocks, Designs at Behavioral Level	Regular	OHP,BB	
20				Blocking and Non blocking Assignments, The <i>case</i> statement, Simulation Flow. <i>if</i> and <i>if-else</i> constructs	Regular	OHP,BB	

21				<i>assign-deassign</i> construct, <i>repeat</i> construct, <i>for</i> loop, the <i>disable</i> construct	Regular	OHP,BB	
22				<i>while</i> loop, <i>forever</i> loop, parallel blocks, <i>force-release</i> construct, Event.	Regular	BB	
23				Tutorial class-3		BB	
24	<b>IV</b>	12		SWITCH LEVEL MODELING.  Introduction Basic Transistor Switches, CMOS Switch, Bi-directional Gates	Regular	OHP,BB	
25				Time Delays with Switch Primitives, Instantiations with Strengths and Delays	Regular	BB	
26				.Strength Contention with Trireg Nets, Exercises.	Regular	BB	
27				Combinational synthesis	<b>Additional</b>	OHP,BB	
28				Tutorial calss-4		BB	
29				Solving university papers		BB	
30				Assignment test-2			
31				Mid test-1			
32				Introduction, Parameters, Path Delays, Module Parameters, System Tasks and Functions	Regular	OHP,BB	
33				File-Based Tasks and Functions, Compiler Directives, Hierarchical Access,	Regular	OHP,BB	
34				User- Defined Primitives (UDP)	Regular	OHP,BB	
35				Tutorial class-5		BB	

36	V	11		Sequential Models – FeedBack Model, Capacitive Model, Implicit Model	Regular	OHP,BB	
37				Basic Memory Components, Functional Register	Regular	OHP,BB	
38				Static Machine Coding	Regular	OHP,BB	
39				Sequential Synthesis	Regular	OHP,BB	
40				Tutorial class – 6		BB	
41				Component Test and Verification: Test Bench – Combinational Circuit Testing	Regular	OHP,BB	
42				Test Bench – Sequential Circuit Testing	Regular	OHP,BB	
43				Test Bench Techniques	Regular	OHP,BB	
44				Design Verification	Regular	OHP,BB	
45				Assertion Verification	Regular	OHP,BB	
46				Tutorial Class – 7		BB	
47				Solving university papers		BB	
48				Assignment test-2			
49				Mid test-2			

## **14. Detailed Notes**

### **UNIT 1**

#### **INTRODUCTION TO VERILOG**

##### ***VERILOG AS HDL:***

Verilog HDL is a hardware description language used to design and document electronic systems. Verilog HDL allows designers to design at various levels of abstraction.

- Why use an HDL?

- Describe complex designs (millions of gates)
- Input to synthesis tools (synthesizable subset)
- Design exploration with simulation
- Why not use a general purpose language
- Support for structure and instantiation (objects?)
- Support for describing bit-level behavior
- Support for timing
- Support for concurrency
- Verilog vs. VHDL
- Verilog is relatively simple and close to C
- VHDL is complex and close to Ada
- Verilog has 60% of the world digital design market (larger share in US)
- Verilog modeling range
- From gates to processor level
- We'll focus on RTL (register transfer level)

### ***LEVELS OF DESIGN DESCRIPTION:***

For the design of a digital system using an automated design environment, the design flow begins with specification of the design at various levels of abstraction and ends with generating net list for an application specific integrated circuits (ASIC), layout for a custom IC, or a program for programmable logic devices (PLD). Figure 1.1 shows steps involved in this design flow.

In the design entry phase, a design is specified as a mixture of behavioral Verilog code, instantiation of Verilog modules, and bus and wire assignments. A design engineer is also responsible for generating test benches. for his or her design for verification of the design and later for verifying the synthesis output. Design verification can be done by simulation, assertion verification, formal verification, or a mix of all three. After performing this design validation phase (this is called the pre synthesis verification), this design is taken through the synthesis process to translate it into actual hardware of a target device. Here, target device refers to the specific field programmable logic device (FPLD) that is being programmed, the ASIC that is being manufactured by an outside source, or the custom IC that is being fabricated. After the

synthesis process and before the actual hardware is generated, another simulation, which is referred to as posts synthesis simulation, is done. This simulation can take advantage of the same test bench generated for the Verilog model of the system before it is synthesized. This way, the behavioral model of the design and its hardware model are tested with the same data. The difference between pre- and posts synthesis simulations is in the level of details obtained from each simulation.

### ***CONCURRENCY:***

It is desired that all the elements present in an electronic circuit must be active and function simultaneously as their voltages and current may vary at the same instant. Even, there is a possibility to change their logic state. Multiple activities that are distributed among various modules need to be run concurrently. Hence simulators are designed to carry out concurrent simulation. Simulation done at uniform time intervals obtains concurrency.

Like timing, concurrency is an essential feature of any language for description of hardware. When a software programmer develops code for performing a certain task, he or she thinks of this task in a sequential manner. The software developed this way will have a top down sequential flow. On the other hand, when a hardware designer or modeler is to describe a hardware system, he or she thinks of this hardware as interconnections of components. The functionality of the overall system is achieved by concurrently active components communicating through their input and output ports.

The functionality of each component may be described by concurrent subcomponents or described by a program in a sequential manner.

We refer to concurrency as the way the simulation of components or constructs appears to the user. Obviously, Verilog is a language for which simulators have been developed on single-processor platforms, and true concurrency in the execution of thousands of components cannot exist. Through the use of concurrent constructs, timing of interconnecting signals, and order of simulation of constructs or components, a Verilog simulator makes us (the users) think that such execution is being done concurrently.

### ***SIMULATION AND SYNTHESIS, tools:***

Simulation for design validation is done before a design is synthesized. This simulation pass is also referred to as behavioral, RT level, or pre synthesis simulation. At the RT level a design includes clock-level timing but no gate and wire delays are included. Simulation at this level is accurate to the clock level. Timing of RT-level simulation is at the clock level and does not usually consider hazards, glitches, race conditions, setup and hold violations, and other detailed timing issues. The advantage of this simulation is its speed compared with simulations at the gate or transistor levels.

Simulation of a design requires test data, and usually Verilog simulation environments provide various methods for application of these data to the design being tested. Test data can be generated graphically using waveform editors, or through a test bench. For simulating with a Verilog test bench, the test bench



instantiates the design under test, and as part of the code of the test bench it applies test data to the instantiated circuit.

Synthesis is the process of automatic hardware generation from a design description that has an unambiguous hardware correspondence. A Verilog description for synthesis cannot include signal and gate level timing specifications, file handling, and other language constructs that do not translate to sequential or combinational logic equations. Furthermore, Verilog descriptions for synthesis must follow certain styles of coding for combinational and sequential circuits. These styles and their corresponding Verilog constructs are defined under Verilog for RTL synthesis. In the design process, after a design is successfully entered and its Pre synthesis simulation results have been verified by the designer, it must be compiled to make it one step closer to an actual hardware on silicon. This design phase requires specification of the hardware that the design is to be realized in.

For example, we have to specify a specific ASIC, or a field programmable gate array (FPGA) part as our “target hardware.” When the target hardware is specified, technology files of that hardware (ASIC, FPGA, or custom IC) with detailed timing and functional specification become available to the compilation process. The compilation process, translates various parts of the design to an intermediate format (analysis phase), links all parts together, generates the corresponding logic (synthesis phase), places and routes components of the target hardware, and generates timing details.

### ***FUNCTIONAL VERIFICATION:***

Testing is done by two measurements namely:

1. Functional test
2. Timing test

It does both the measurements. Usually a test bench is provided for a design that is to be tested. Today’s functional verification flow mainly contains following steps:

1. Generate the stimulus vectors.
2. Send the Stimulus to the DUT.
3. Monitor the response generated by the DUT.
4. Verify the response generated.
5. Generate report about the DUT performance.
6. Some kind of feedback to show the quality of test bench

### ***SYSTEM TASKS:***

For test bench generation, data input and output, timing check, simulation flow control, data conversion, and memory initialization. Verilog provides a number of system tasks and functions categorized into ten groups. The names of system tasks and functions begin with a dollar sign (\$), followed by a task

specified. The name of the task or function usually contains characters and names that describe its functionality.

### **Display tasks**

Display tasks are used for outputting to the standard output device. The most basic display task is the \$display task, which writes its string argument to the display device. Other tasks include those for monitoring and outputting variable values as they change (the \$monitor group of tasks) and those for displaying variables at a selected time (the \$display tasks). Display tasks can display in binary, hexadecimal, or octal formats. The character b, h, or o at the end of the task name specifies the data type a task handles. For all display tasks, a generic task can be used to display data with specified formats and data types.

### **File I/O tasks**

File output tasks begin with a dollar sign followed by the letter f (for file) and then by the same task names as those of the display tasks. These tasks perform the same functionalities as their display task counterparts, except that their output is to a file instead of to the display terminal. The \$fopen function opens a file and assigns an integer file description. The file descriptor will be used as an argument for all file I/O tasks. In addition, there are string write tasks (\$swrite) that write their formatted outputs to a string. Verilog also provides tasks for inputting data from files or strings. Such tasks allow reading characters, formatted data, or complete memory data from external data files or declared strings. Examples of these tasks are \$fgetc, \$fscanf, and \$sscanf for getting a character from a file, reading formatted data from a file, and reading formatted data from a string, respectively. Other input tasks exist for reading memory data directly into a declared memory. Examples of such tasks are \$fread and \$readmemh. File positioning tasks, \$fseek and \$frewind are also available for positioning file pointer for read or write. Verilog I/O tasks are useful in developing complete hardware/software environments and developing test benches.

### **Timescale tasks**

Timescale tasks are \$printtimescale and \$timeformat. The \$printtimescale task displays the timescale and precision of the module whose hierarchical name is being passed to it as its argument. The \$timeformat task formats time for display by file IO and display tasks.

### **Simulation control tasks**

Simulation control tasks are \$finish and \$stop. The \$finish task ends the simulation and exits. Usually, simulation environments require a confirmation before the action of exiting the environment is taken. The \$stop task suspends the simulation and does not exit the simulation environment.

### **Timing check tasks**

Timing check tasks are used for checking timings, such as pulse width duration and setup and hold times. In general, timing check tasks check the timing on one signal or the relative timing of several signals for

certain conditions to hold. If a violation is detected, a message will be issued in the user simulation environment display area. For example, the statement shown below uses the \$nochange timing check task to report a violation if d\_input changes in the period of three time units before and five time units after the positive edge of the clock.

```
$nochange (posedge clock, d_input, 3, 5);
```

### **PROGRAMMING LANGUAGE INTERFACE :**

After finishing the compilation of a Verilog module, a dynamic interface is provided by the PLI that increases the scope of Verilog, so it can be linked with C program.

#### **MODULE:**

- The module is the basic building block in Verilog
- Modules can be interconnected to describe the structure of your digital system
- Modules start with keyword module and end with keyword end module.

#### **Module Ports**

- Similar to pins on a chip
- Provide a way to communicate with outside world
- Ports can be input, output or in out.

A module is the main structure for definition of hardware components and test benches. Modules begin with the module keyword and end with end module. Immediately following the module keyword, port list of the module appears enclosed in parenthesis. Declaration of mode, type, and size of ports can either appear in the port list or as separate declarations.

Example:

```
module FlipFlop (preset, reset, din, clk, qout);  
  
input preset, reset, din, clk;  
  
output qout;  
  
reg qout;
```

```

always @ (posedge clk) begin

if (reset) qout <= #7 0;

else if (preset) qout <= #7 1;

else qout <= #8 din;

end

end module

```

The body of a module consists of the specification of the operation of the hardware the module is representing. A test bench module has no ports. It instantiates the module under test (MUT) and through the use of concurrent statements or procedural blocks applies data to the ports of MUT. Multiple modules can be tested with the same test bench.

### ***TEST BENCHES:***

Values assigned to inputs of a circuit for examining its operation are either specified within a simulation environment using a waveform editor, or by a Verilog test bench. In this description, TriMux is instantiated

```

`timescale 1ns/100ps

module TriMuxTest;

reg i0=0, i1=0, s=0;

wire y;

TriMux MUT (i0, i1, s, y);

initial begin

#15 i1=1'b1;

#15 s=1'b1;

#15 s=1'b0;

#15 i0=1'b1;

#15 i0=1'b0;

#15 $finish;

```

end

end module

The initial statement is a procedural construct and uses delay control statements to delay the program flow in this procedural block. After each such delay, a value is assigned to i0, i1, or s. At the end of this block, after a 15-ns delay, the \$finish simulation control task finishes the simulation run. The delay before \$finish allows the last input change to have a chance to affect the circuit output. The delay values (15 ns) used in this example are chosen so that inputs remain stable while a change is propagating through the circuit.

## ***LANGUAGE CONSTRUCTS AND CONVENTIONS:***

### ***Keywords:***

Every language has some keywords reserved for certain use. They describe the language constructs. In Verilog there are many keywords. Few of them are:

1. **Module:** a module is defined starting with this keyword.
2. **End module:** a module is ended with this definition
3. **Begin:** a set of statements in a block start with this keyword.
4. **End:** a set of statements within the block are terminated with this word.
5. **If :** verifies conditional statements.

## ***IDENTIFIERS:***

Identifiers are names that are given to elements such as modules, registers, ports, wires, instances, and procedural blocks. An identifier is any sequence of letters, digits, and the underscore (\_) symbol except that:

the first character must not be a digit, and the identifier must be 1024 characters or less.

**Verilog is case sensitive**, ie Upper and lower case letters are considered to be different. System tasks and system functions are identifiers that always start with the dollar symbol. Escaped identifiers allow for any printable ASCII character to be included in the name. Escaped identifiers begin with white space. The backslash (“\”) character leads off the identifier, which is then terminated with white space. The leading backslash character is not considered part of the identifier.

Examples of escaped identifiers include:

\flip-flop

\a+b

Escaped identifiers are used for translators from other CAD systems. These systems may allow special characters in identifiers. Escaped identifiers should not be used under normal circumstances.

### **WHITE SPACE CHARACTERS & COMMENTS:**

White space is defined as any of the following characters: blanks, tabs, newlines, and form feeds. These are ignored except for when they are found in strings.

There are two forms of comments. The single line comment begins with the two characters

*// and ends with a new-line.*

A block comment begins with the two characters */\* and ends with the two characters \*/*. Block comments may span several lines. However, they may not be nested.

### **NUMBERS:**

Constants in Verilog are integer or real. Specification of integers can include X (or x) and Z (or z) in addition to the standard 0 and 1 logic values. Integer formats provide various ways for representing bit streams. Integers may be sized or un sized. A sized integer begins with the number of equivalent bits, followed by the single quote character ('), a base specifier, and the digits of the number in the specified base. The base specifier is a single lower or uppercase character, b, d, o, or h for binary, decimal, octal, and hexadecimal bases. The general format for integers is: number\_of\_bits 'base\_identifier digits

Digits in the decimal (d) system are 0 through 9. For hexadecimal, octal, and binary systems, in addition to their standard digits, X and Z (both upper and lowercase) characters are also allowed. Hexadecimal and octal X and Z digits expand to 4 or 3 bits of X and Z respectively. A number without the number\_of\_bits specification is regarded as an un sized number

```
`timescale 1ns/100ps
```

```
module NumberTest;
```

```
reg [11:0] a = 8'shA6; initial $displayb ("a=", a);
```

```
// a=111110100110
```

```
reg [11:0] b = 8'sh6A; initial $displayb ("b=", b);
```

```
// b=000001101010
```

```
reg [11:0] c = 'shA6; initial $displayb ("c=", c);
```

```
// c=000010100110
```

```
reg [11:0] d = 'sh6A; initial $displayb ("d=", d);
```

```
// d=000001101010
```

```
reg [11:0] e = -8'shA6; initial $displayb ("e=", e);
```

```
// e=000001011010
```

```
reg [11:0] f = -'shA6; initial $displayb ("f=", f);
```

```
// f=111101011010
```

```
reg [11:0] g = 9'shA6; initial $displayb ("g=", g);
```

```
// g=000010100110
```

```
reg [11:0] h = 9'sh6A; initial $displayb ("h=", h);
```

```
// h=000001101010
```

```
reg [11:0] i = -9'shA6; initial $displayb ("i=", i);
```

```
// i=111101011010
```

```
reg [11:0] j = -9'sh6A; initial $displayb ("j=", j);
```

```
// j=111110010110
```

```
reg [11:0] k = 596; initial $displayb ("k=", k);
```

```
// k=001001010100
```

```
reg [11:0] l = -596; initial $displayb ("l=", l);
```

```
//l=110110101100
```

```
Endmodule
```

### ***STRINGS:***

The strings in Verilog are sequences of 8-bit ASCII characters enclosed within quotation marks.

"This is an example"

As mentioned before white spaces are not ignored inside this string. There is no special data type available to store strings. reg should be used to store the strings. Above string example has 18 characters (including white spaces) so it needs following variable to store the complete string

```
reg[8*[18-1]:0] a;
```

Now 'a' can hold the above string

```
a = "This is an example";
```

If you want to include special characters like quotes(") you must use escape sequence.

```
text = "\"vlsi-world.com\"";
```

```
text1 = "vlsi-world.com";
```

text will produce "vlsi-world.com"

text1 will produce vlsi-world.com

Use \t to insert tabs

    \n to insert new lines

    \\ to insert \ character

    \" to insert " character

### ***LOGIC VALUES :***

0: zero, logic low, false, ground

- 1: one, logic high, power
- X: unknown
- Z: high impedance, unconnected, tri-state

Bit type, or bits of vectors or arrays, of Verilog wires and variables take the 4-value logic value system. Values in this system are 0, 1, Z, and X. The 0 value represents forcing 0 like a direct pull to the ground, or a resistive 0, or a capacitive 0. A resistive 0 is generated when there is a large resistance between a line and a forcing 0 value. A capacitive 0 is when a line is float; but has a capacitance that has a zero charge. The 1 value represents forcing 1, resistive 1, and a capacitive 1. These are defined similar to various modes of the 0 value. For example a forcing 1 is defined as the logic value driven by a supply voltage. The Z value represents an undriven, high-impedance value. This is the



electrical float which causes no current flow to either supply or ground voltage. Both Z and z are acceptable forms of this logic value. The X value represents a conflict in multiple driving values, an unknown, an uninitialized value, a short between two opposing values (0 and 1), or a bus contention. Driven wires and Verilog variables assume X for their initial values. Figure 3.5 shows several examples for the four values of Verilog's logic value system. Both X and x are acceptable forms of this logic value.

### **STRENGTHS:**

The strength declaration construct is used for modeling net type variables for a close correspondence with physical wires.

**(Strength1, Strength0)**

**(Strength0, Strength1)**

**Strength1:**

**supply1, strong1, pull1, large1, weak1, medium1, small1, highz1**

**Strength0:**

**supply0, strong0, pull0, large0, weak0, medium0, small0, highz0**

Strengths can be used to resolve which value should appear on a net or gate output.

There are two types of strengths: drive strengths (*Example 1*) and charge strengths (*Example 2*). The drive strengths can be used for nets (except **triereg** net), gates, and UDPs. The charge strengths can be used only for **triereg** nets. The drive strength types are **supply**, **strong**, **pull**, **weak**, and **highz** strengths. The charge strength types are **large**, **medium** and **small** strengths.

All strengths can be ordered by their value. The **supply** strength is the strongest and the **highz** strength is the weakest strength level. Strength value can be displayed by system tasks (**\$display**, **\$monitor** - by using of the %v characters - see Display tasks for more explanation).

<b>Strength</b>	<b>Value</b>	<b>Value displayed by display tasks</b>
supply	7	Su
strong	6	St
pull	5	Pu
large	4	La
weak	3	We
medium	2	Me
small	1	Sm
highz	0	HiZ

If two or more drivers drive a signal then it will have the value of the strongest driver (*Example 3*).

If two drivers of a net have the same strength and value, then the net result will have the same value and strength (*Example 4*).

If two drivers of a net have the same strength but different values then signal value will be unknown and it will have the same strength as both drivers (*Example 5*).

If one of the drivers of a net has an H or L value, then signal value will be **n1n2X**, where **n1** is the strength value of the driver that has the smaller strength, and **n2** is strength value of driver that has the larger strength (*Example 6*).

The combinations (**highz0, highz1**) and (**highz1, highz0**) are illegal.

### **DATA TYPES:**

Verilog has net and reg data types representing wires and variables,

respectively. The net type represents data carriers such as interconnecting wires, gate outputs, and busses. The reg data type represents variables that hold the value they are assigned until they are overwritten. Additionally, a net or a reg can be declared as signed, which determines how they interpret data assigned to them

- Nets

- Nets are physical connections between devices
- Nets always reflect the logic value of the driving device
- Many types of nets, but all we care about is wire

- Registers

- Implicit storage – unless variable of this type is modified it retains previously assigned value
- Does not necessarily imply a hardware register
- Register type is denoted by reg
- int is also used

### **SCALARS AND VECTORS :**

Vectors are multiple bit widths **net** or **reg** data type variables that can be declared by specifying their range.

Syntax:

**net\_type [msb:lsb] list\_of\_net\_identifiers;**

**reg [msb:lsb] list\_of\_register\_identifiers;**

Vector range specification contains two constant expressions: the msb (most significant bit) constant expression, which is the left-hand value of the range and the lsb (least significant bit) constant expression, which is the right-hand value of the range. The msb and lsb constant expressions should be separated by a colon.

Both the msb constant expression and the lsb constant expression can be any value - positive, negative, or zero. The lsb constant expression can be greater, equal or less than the msb constant expression.

Vectors can be declared for all types of **net** data types and for **reg** data types. Specifying vectors for **integer**, **real**, **realtime**, and **time** data types is illegal.

Vector nets and registers are treated as unsigned values (see: Arithmetic expressions with registers and integers for more explanations).

- Both the msb and the lsb expressions should be constant expressions.
- The msb and the lsb constant expressions may be positive, negative, or zero.
- The lsb constant expression may be greater, equal or less than the msb constant expression.
- Vectors can be declared only for **nets** and **reg** data types.
- Vector declaration for **integer**, **real**, **realtime**, and **time** data types are illegal.

#### **PARAMETERS:**

Parameters are constants typically used to specify the width of variables and time delays.

Syntax :

**parameter** identifier = constant\_expression ,

identifier = constant\_expression ;

**defparam** hierarchical\_path = constant\_expression ;

In Verilog HDL, parameters are constants and do not belong to any other data type such as net or register data types.

A constant expression refers to a constant number or a previously defined parameter .You are not allowed to modify parameter values at runtime, but you can modify a parameter value using the **defparam** statement. The defparam statement can modify parameters only at the time of compilation. Parameter values can also be modified using #delay specification with module instantiation.

In Verilog there are two ways to override a module parameter value during a module instantiation. The first method is by using the **defparam** keyword and the second method is called *module instance parameter value assignment*.

After the **defparam** keyword, the hierarchical path to the parameter is specified along with the new value of the parameter. In this case, the new value should be a constant expression . If the right-hand side expression references any parameters it should be declared within the module where defparam is invoked .

The *module instance parameter value assignment* method looks like an assignment of delay to gate instance .This method overrides parameters inside instantiated modules, in the order, that they appear in the module. Using this format, parameters cannot be skipped.

Constant expressions can contain previously declared parameters. When changes are detected on the previously declared parameters, all parameters that depend on this value are automatically updated.

Example :

```
module top;
reg Clk ;
reg [7:0] D ;
wire [7:0] Q ;
my_module inst_1(Clk, D, Q) ;
endmodule
```

```
module override ;
defparam top.inst_1.width = 7 ;
endmodule
```

- Parameters are constants.
- If you are using the defparam statement, remember that you have to specify a hierarchical path to your parameter.
- You cannot skip over a parameter in a *module instance parameter value assignment*. If you need do this, use the initial value for parameter that is not to be overwritten.
- When one parameter depends on the other, remember that if you change the first one, the second will automatically be updated.

## MEMORIES:

Memories are arrays of registers.

SYNTAX :

**reg** memory\_width memory\_identifier memory\_depth;

**integer** memory\_identifier memory\_length;

**time** memory\_identifier memory\_length;

Memories can be declared only for **reg**, **integer** and **time** data types. Depth of memory should be declared by specifying a range following the memory identifier . Registers and memories can be declared in the same line .Elements of memory type can be accessed by memory index . An assignment to a memory identifier without specified memory index is illegal. Bit-selects and part-selects on memory elements are not allowed. If access to individual bits of memory is needed, then a word containing that bit should be assigned to a register with the same width. All operations should then be done on this register and the result should be assigned back to the memory word . Memory words can be accessed individually, but bit-select and part-select operations cannot be done on memories or memory words directly.

Vector declaration and memory declaration are not the same. If a variable is declared as a vector, all bits can be assigned a value in one statement. If a variable is declared as memory then a value to each element should be assigned separately .

```
reg [7:0] vect;
reg array[7:0];
vect = 8'b11001010;
array[7] = 1'b1;
array[6] = 1'b1;
array[5] = 1'b0;
array[4] = 1'b0;
array[3] = 1'b1;
array[2] = 1'b0;
array[1] = 1'b1;
array[0] = 1'b0;
```

- Memories can be declared only for **reg**, **integer** and **time** registers types.
- Bit-selects and part-selects on memory elements are prohibited.

## OPERATORS :

Operators provide a means to construct expressions.

Syntax :

Arithmetic: + - \* /

Modulus: %

Relational: < <= > >=

Logical: ! && ||

Logical equality: == !=

Case equality: === !==

Bit-wise: ~ & | ^ ~^ ~^~

Reduction: & ~& | ~| ^ ~^ ~^~

Shift: << >>

Conditional: ?:

Event or: or

Concatenations: {} {{{}}

Verilog HDL operators can be divided into several groups.

Operator	Description
+ - ! ~	Unary
* / %	Arithmetic
+ - (binary)	Binary
<< >>	Shift
< <= > >=	Relational
== != === !==	Equality
& ~&	and nand
^ ~^ ~^~	xor xnor
~	or nor
&&	Logical and
	Logical or
?:	Conditional operator

**Table 13: Operator's priority**

### Arithmetic operators

The arithmetic operators can be used with all data types.

Operator	Description
a + b	a <b>plus</b> b
a - b	a <b>minus</b> b
a * b	a <b>multiply by</b> b
a / b	a <b>divide by</b> b
a % b	a <b>modulo</b> b

**Table 14: Arithmetic operators**

The modulus operator is not allowed for real data type variables. For the modulus operator, the result takes the sign of the first operand.

### Relational operators

The relational operators are used to compare expressions. The value returned by the relational operators is 0 if the expression evaluates to false and 1 if expression evaluates to true.

Operator	Description
a < b	a <b>less than</b> b
a > b	a <b>greater than</b> b
a <= b	a <b>less than or equal to</b> b
a >= b	a <b>greater than or equal to</b> b

### Equality operators

The equality operators are used to compare expressions. If a comparison fails, then the result will be 0, otherwise it will be 1.

If both operands of logical equality (==) or logical inequality (!=) contain unknown (x) or a high-impedance (z) value, then the result of comparison will be unknown (x). Otherwise it will be true or false.

If operands of case equality (===) or case inequality (!==) contain unknown (x) or a high-impedance (z) value, then the result will be calculated bit by bit.

Examples of using the equality operators are shown in .

### Logical operators

The logical operators are used to connect expressions.

Operator	Description
a && b	a <b>and</b> b
a    b	a <b>or</b> b

!a	not a
----	-------

The result for these operators is 0 (when false), 1 (when true), and unknown (x - when ambiguous). The negation operator (!) turns a nonzero or true value of the operand into 0, zero or false value into 1, and ambiguous value of operator results in x (unknown value).

### Bit-wise operators

The bit-wise operators calculate each bit of results by evaluating the logical expression on a pair of corresponding operand bits.

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

### Bit-wise and operator

	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

### Bit-wise or operator

^	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

### Bit-wise exclusive or operator

~^ ^~	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

### Bit-wise exclusive nor operator

~	Result
0	1
1	0
x	X
z	X

### Bit-wise negation operator

### Reduction operators

The reduction operator produces a 1-bit result. This result is calculated by recursively applying bit-wise operation on all bits of the operand. At each step of this recursive calculation the logical bit-wise operation is performed on the result of a previous operation and on the next bit of the operand. The operation is repeated for all bits of the operand.

### Shift operators

The shift operators perform left and right shifts on their left operand by the number of positions specified by their right operand. All vacated bits are filled with zeroes. If the expression that specifies the number of bits to shift (right operand) has unknown (x) or high-impedance (z) value, then result will be unknown.

### Conditional operator

The conditional operator is described in the Conditional operator section.

### Concatenations

Concatenations are described in the Concatenations section.

### Event or operator

The event or operator is described in the section on Procedural timing controls.

## UNIT II

### GATE LEVEL MODELLING

#### **INTRODUCTION:**

The previous section discussed the role of wires and basics of generating Verilog modules for simulation. Building upon that material, this section presents generation of Verilog modules using predefined gate



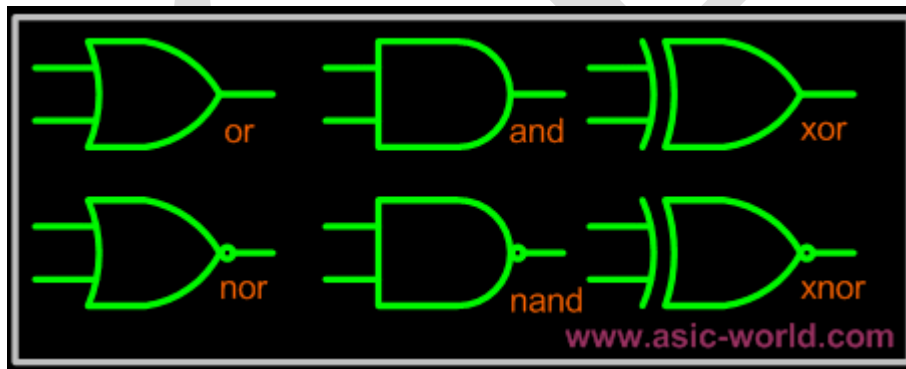
primitives of this language. We will also discuss delay issues related to these gates and ways of defining them and the way they affect timing of an entire system. Verilog has built in primitives like gates, transmission gates, and switches. These are rarely used in design (RTL Coding), but are used in post synthesis world for modeling the ASIC/FPGA cells; these cells are then used for gate level simulation, or what is called as SDF simulation. Also the output net list format from the synthesis tool, which is imported into the place and route tool, is also in Verilog gate level primitives.

### **AND& OTHERS GATE PRIMITIVES:**

Verilog gate level list includes standard n\_input, n\_output, and tri-state gates. Verilog instantiation of these gates are also shown in this figure. In addition, Verilog has switch level and transistor primitives that will be discussed in a later chapter. Gates categorized as n\_input gates are and, nand, or, nor, xor, and xnor. An n\_input gate has one output, which is its left-most argument, and can have any number of inputs that may be listed as its argument separated by commas. These gates can have up to two delay parameters that can appear after the name of the gate in a set of parenthesis followed by a sharp sign. An example instantiation of a 4-input nand

is shown here.

```
nand #(3, 5) gate1 (w, i1, i2, i3, i4);
```



### **EXAMPLE:**

#### **Example 1: Full Adder**

```
module FullAdder(X, Y, Cin, Cout, Sum);  
  
input X, Y, Cin; // input terminal definitions  
  
output Cout, Sum; // output terminal definitions  
  
wire w1, w2, w3, w4; // internal net declarations
```

```

xor #(10) (w1, X, Y); // delay time of 10 units

xor #(10) xor2(Sum, w1, Cin); // with instance name

and #(10) (w2, X, Y);

and #(10) (w3, X, Cin);

and #(10) (w4, Y, Cin); or #(10, 8)(Cout, w2, w3, w4); // 3 input or (rise time of 10, fall // time of
8)

Endmodule

```

### ***MODULE STRUCTURES:***

A module is comprised of the interface and the design behavior.

**module** | **macromodule** identifier (port\_list) ;

ports\_declaration ;

module\_body ;

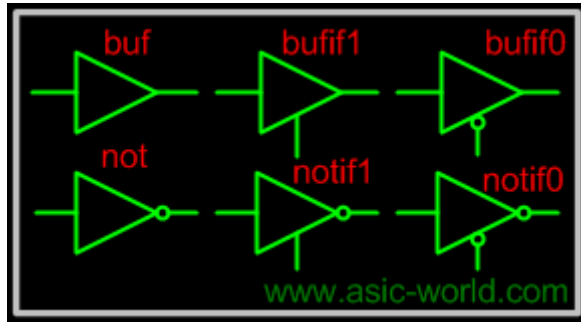
**endmodule**

All module declarations must begin with the **module** (or **macromodule**) keyword and end with the **endmodule** keyword. After the module declaration, an identifier is required. A ports list is an option. After that, ports declaration is given with declarations of the direction of ports and the optionally type. The body of module can be any of the following:

- Any declaration including parameter, function, task, event or any variable declaration.
- Continuous assignment.
- Gate, UDP or module instantiation.
- Specify block.
- Initial block
- Always block.

If there is no instantiation inside the module, it will be treated as a top-level module.

### ***TRI STATE GATES:***



Transmission gates tran and rtran are permanently on and do not have a control line. Tran can be used to interface two wires with separate drives, and rtran can be used to weaken signals.

```
module transmission_gates();
```

```
    reg data_enable_low, in;
```

```
    wire data_bus, out1, out2;
```

```
    bufif0 U1(data_bus,in, data_enable_low);
```

```
    buf U2(out1,in);
```

```
    not U3(out2,in);
```

```
initial begin
```

```
    $monitor(
```

```
        "@%g in=%b data_enable_low=%b out1=%b out2= b data_bus=%b",
```

```
        $time, in, data_enable_low, out1, out2, data_bus);
```

```
    data_enable_low = 0;
```

```
    in = 0;
```

```
    #4 data_enable_low = 1;
```

```
    #8 $finish;
```

```
end
```

```

always #2 in = ~in;

end

```

### **ARRAY OF INSTANCES OF PRIMITIVES :**

Verilog uses different constructs for describing a module with different levels of detail. Verilog basic logic gates are called primitives and for describing a component using these primitives, a construct called primitive instantiation is used. See for example the multiplexer that is made of AND and OR gates. This structure can be described in Verilog as

#### **Basic Gates**

```

module MultiplexerA (input a, b, s, output w);

wire a_sel, b_sel, s_bar;

not U1 (s_bar, s);

and U2 (a_sel, a, s_bar);

and U3 (b_sel, b, s);

or U4 (w, a_sel, b_sel);

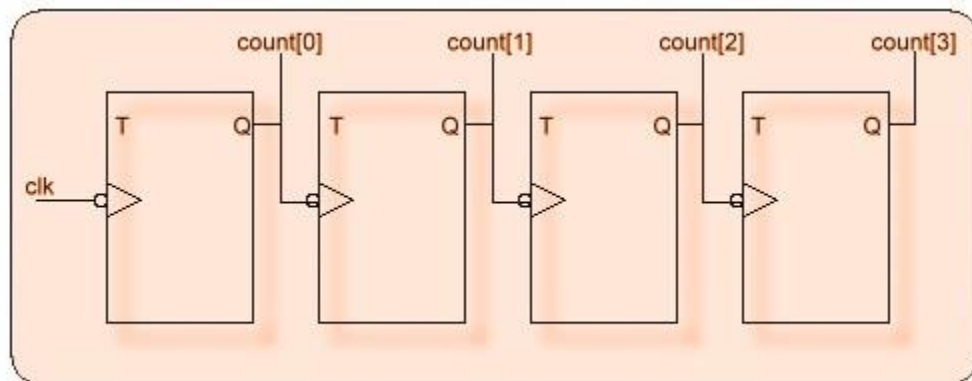
endmodule

module MultiplexerB (input a, b, s, output w);

```

### **GATE PRIMITIVES WITH FLIP FLOP:**

Consider a 4-bit asynchronous counter; block diagram using flip-flops is as follows. This is a simple counter without reset or load options.



```
module counter( clk, count );  
input clk;  
output[3:0] count;
```

```
reg[3:0] count;  
wire clk;
```

```
initial  
    count = 4'b0;
```

```
always @( negedge clk )  
    count[0] <= ~count[0];
```

```
always @( negedge count[0] )  
    count[1] <= ~count[1];
```

```
always @( negedge count[1] )  
    count[2] <= ~count[2];
```

```
always @( negedge count[2] )  
    count[3] <= ~count[3];
```

```
endmodule
```

### ***DELAYS:***

Delays specify a time in which assigned values propagate through nets or from inputs to outputs of gates.

Syntax:

#value

#(value)

#(value, value)

#(value, value, value)

Delays specify how values propagate through nets or gates.

The net delay declaration specifies a time needed to propagate values from drivers through the net. It can be used in continuous assignments (Example 1) and net declarations (Example 2).

The gate delay declaration specifies a time needed to propagate a signal change from the input of a gate input to its output. The gate delay declaration can be used in gate instantiations (Example 3).

The delays can be also used for delay control in procedural statements (Example 4 - see Procedural timing control for more explanations).

The delays declaration can contain up to three values: rise, fall, and turn-off delays. The default delay is zero. If only one delay value is specified then it is used for all signal changes. If two delays are specified then the first delay specifies the rise delay and the second delay specifies the fall delay. If the signal changes to high-impedance (z) or to unknown (x) then the smaller value will be used. This means that if delays are specified as follows: #(4,3) then the second value (3) will be used for signal changes to z or x value.

If three values are given, then the first value specifies the rise delay, the second specifies the fall delay, and the third specifies turn-off delay. If the signal changes to unknown (x) value, then the smallest of these three values will be used.

Value changes		Delay used for propagation if:		
From:	To:	1 delay specified	2 delays specified	3 delays specified
0	1	d1	d1	d1
0	x	d1	min(d1, d2)	min(d1, d2, d3)
0	z	d1	min(d1, d2)	d3
1	0	d1	d2	d2
1	x	d1	min(d1, d2)	min(d1, d2, d3)
1	z	d1	min(d1, d2)	d3
x	0	d1	d2	d2
x	1	d1	d1	d1
x	z	d1	min(d1, d2)	d3
z	0	d1	d2	d2
z	1	d1	d1	d1
z	x	D1	min(d1, d2)	min(d1, d2, d3)

### ***STRENGTHS AND CONTENT RESOLUTION:***

Strengths can be used to resolve which value should appear on a net or gate output.

There are two types of strengths: drive strengths (*Example 1*) and charge strengths (*Example 2*). The drive strengths can be used for nets (except **triereg** net), gates, and UDPs. The charge strengths can be

used only for **tri**reg nets. The drive strength types are **supply**, **strong**, **pull**, **weak**, and **highz** strengths. The charge strength types are **large**, **medium** and **small** strengths.

All strengths can be ordered by their value. The **supply** strength is the strongest and the **highz** strength is the weakest strength level. Strength value can be displayed by system tasks (**\$display**, **\$monitor** - by using of the %v characters - see Display tasks for more explanation).

Strength	Value	Value displayed by display tasks
supply	7	Su
strong	6	St
pull	5	Pu
large	4	La
weak	3	We
medium	2	Me
small	1	Sm
highz	0	HiZ

### **NET TYPES:**

Nets are data types that can be used to model physical connections.

#### **Net declaration:**

**wire** range delays list\_of\_identifiers;

**wand** range delays list\_of\_identifiers;

**wor** range delays list\_of\_identifiers;

**tri** range delays list\_of\_identifiers;

**triand** range delays list\_of\_identifiers;

**trior** range delays list\_of\_identifiers;

**tri0** range delays list\_of\_identifiers;

**tri1** range delays list\_of\_identifiers;

**supply0** range delays list\_of\_identifiers;

**supply1** range delays list\_of\_identifiers;

**tri**reg strength range delays list\_of\_identifiers;

### Net declaration assignment:

**wire** strength range delays list\_of\_identifiers = expression;  
**wand** strength range delays list\_of\_identifiers = expression;  
**wor** strength range delays list\_of\_identifiers = expression;  
**tri** strength range delays list\_of\_identifiers = expression;  
**triand** strength range delays list\_of\_identifiers = expression;  
**trior** strength range delays list\_of\_identifiers = expression;  
**tri0** strength range delays list\_of\_identifiers = expression;  
**tri1** strength range delays list\_of\_identifiers = expression;  
**supply0** strength range delays list\_of\_identifiers = expression;  
**supply1** strength range delays list\_of\_identifiers = expression;  
**triereg** strength range delays list\_of\_identifiers = expression;

Net data types are used to model physical connections. They do not store values (there is only one exception - **triereg**, which stores a previously assigned value). The net data types have the value of their drivers. If a net variable has no driver, then it has a high-impedance value (z).

Nets can be declared in a net declaration statement or in a net declaration assignment

Net declarations can contain strength declarations, which specifies the strength of the logic values driven by the net (see Strengths for more details). The range declaration is used to specify multi-bit nets (vectors). The delays are used to specify propagation delays through the nets. The strength, delay and range declarations are optional

```
wire [7:0] a;  
tri tristate_buffer;  
wand #5 sig_1;  
triereg (small) t;
```

The 'a' variable is a 8-bit **wire** net.

The 'tristate\_buffer' is 1-bit **tri** net type variable.

The 'sig\_1' variable is 1-bit **wand** net type variable, which propagates driven value to its output in 5 time units.

The 't' variable is **triereg** net variable with small charge strength.

### EXAMPLE OF BASIC DESIGN

```
module example();
```



```
reg [3:0] a, b;
```

```
reg [7:0] c, d;
```

```
initial
```

```
begin
```

```
    a = 4'b1110; //14
```

```
    b = 4'b0110; //5
```

```
    $display( "%b", a < b );// false - 0
```

```
    $display( "%b", a > 8 );// true - 1
```

```
    $display( "%b", a <= b );// false - 0
```

```
    $display( "%b", a >= 10 );// true - 1
```

```
    $display( "%b", a < 4'b1zzz );// unknown - x
```

```
    $display( "%b", b < 4'b1x01 );// unknown - x
```

```
    a = 4'b1100;
```

```
    b = 4'b101x;
```

```
    $display( "%b", a == 4'b1100 );// true - 1
```

```
    $display( "%b", a != 4'b1100 );// false - 0
```

```
    $display( "%b", a == 4'b1z10 );// false - 0
```

```
    $display( "%b", a != 4'b100x );// true ? 1
```

```
    $display( "%b", b == 4'b101x );// unknown - x
```

```
    $display( "%b", b != 4'b101x );// unknown - x
```

```
    $display( "%b", b === 4'b101x );// true - 1
```

```
    $display( "%b", b !== 4'b101x );// false - 0
```

```
    a = 4'b1100;
```

```

b = 4'b0000;

$display( "%b", !a );// 0 - false

$display( "%b", !b );// 1 - true

$display( "%b", a && b ); // 0 - false

$display( "%b", a || b );// 1 ? true


c = 8'b1010xzxz;
d = 8'b10010011;

$display( "%b", c & d ); //= 8'b100000xx;

$display( "%b", c | d ); //= 8'b1011xx11;

$display( "%b", c ^ d ); //= 8'b0011xxxx;

$display( "%b", c ~^ d ); //= 8'b1100xxxx;

$display( "%b", ~ c ); //= 8'b0101xxxx;


a = 4'b1111;

$display( "%b", a << 3 ); //= 4'b1000

$display( "%b", a >> 3 ); //= 4'b0001

$display( "%b", a << 1'bz ); //= 4'bxxxx

$display( "%b", a >> 1'bx ); //= 4'bxxxx

end

endmodule

```

## **MODELLING AT DATA FLOW LEVEL**

### ***INTRODUCTION:***

Dataflow modeling provides a powerful way to implement a design. Verilog allows a design processes data rather than instantiation of individual gates. Dataflow modeling has become a popular design

approach as logic synthesis tools have become sophisticated. This approach allows the designer to concentrate on optimizing the circuit in terms of data flow.

### ***CONTINUOUS ASSIGNMENT STRUCTURE:***

A continuous assignment is the most basic statement in data flow modeling, used to drive a value onto a net. A continuous assignment replaces gates in the description of the circuit and describes the circuit at a higher level of abstraction. A continuous assignment statement starts with the keyword assign.

//Syntax of assign statement in the simplest form

< continuous\_assign >

: : = assign < drive\_strength > ? < delay > ? < list\_of\_assignments > ;

### ***DELAYS***

Delay value control the time between the change in a right-hand-side operand and when the new value is assigned to the left-hand-side.

The first method is to assign a delay value in a continuous assignment statement. The delay value is specified after the keyword assign.

```
module stimulus;
```

```
wire OUT;
```

```
reg IN1, IN2;
```

```
initial
```

```
begin
```

```
    IN1 = 0; IN2= 0;
```

```
    #20 IN1=1; IN2= 1;
```

```
    #40 IN1 = 0;
```

```
    #40 IN1 = 1;
```

```
    #5 IN1 = 0;
```

```
    #150 $stop;
```

```
end
```

initial

```
$monitor("out", OUT, "in1", IN1, "in2",
```

```
IN2);
```

```
regular_delay rd1(OUT, IN1, IN2);
```

```
endmodule
```

### ***ASSIGNMENT TO VECTORS:***

Vector range specification contains two constant expressions: the msb (most significant bit) constant expression, which is the left-hand value of the range and the lsb (least significant bit) constant expression, which is the right-hand value of the range. The msb and lsb constant expressions should be separated by a colon.

Both the msb constant expression and the lsb constant expression can be any value - positive, negative, or zero. The lsb constant expression can be greater, equal or less than the msb constant expression.

Vectors can be declared for all types of **net** data types and for **reg** data types. Specifying vectors for **integer**, **real**, **real time**, and **time** data types is illegal.

Vector nets and registers are treated as unsigned values (see: Arithmetic expressions with registers and integers for more explanations).

### ***OPERATORS:***

Here is a small selection of the Verilog Operators which look similar but have different effects. Logical Operators evaluate to TRUE or FALSE. Bitwise operators act on each bit of the operands to produce a multi-bit result. Unary Reduction operators perform the operation on all bits of the operand to produce a single bit result.

Operator	Name	Examples
!	logical negation	
~	bitwise negation	
&&	logical and	
&	bitwise and	abus = bbus&cbus;
&	reduction and	abit = &bbus;
~&	reduction nand	
	logical or	
	bitwise or	
	reduction or	
~	reduction nor	

<code>^</code>	bitwise xor	
<code>^</code>	reduction xor	
<code>~^ ~</code>	bitwise xnor	
<code>~^ ~</code>	reduction xnor	
<code>==</code>	logical equality, result may be unknown if x or z in the input	if (a == b)
<code>===</code>	logical equality including x and z	
<code>!=</code>	logical inequality, result may be unknown if x or z in the input	
<code>!==</code>	logical inequality including x and z	
<code>&gt;</code>	relational greater than	
<code>&gt;&gt;</code>	shift right by a number of positions	a = shiftvalue >> 2;
<code>&gt;=</code>	relational greater than or equal	
<code>&lt;</code>	relational less than	
<code>&lt;&lt;</code>	shift left by a number of positions	
<code>&lt;=</code>	relational less than or equal	if (a <= b)
<code>&lt;=</code>	non blocking assignment statement, schedules assignment for future and allows next statement to execute	#5 b <= b + 2;
<code>=</code>	blocking assignment statement, waits until assignment time before allowing next statement to execute	#5 a = a + 2;

Verilog also supports arithmetic, replication, and concatenation operators

### **UNIT III** **BEHAVIORAL MODELLING**

#### ***INTRODUCTION :***

procedural statements provide a mechanism for describing hardware at still a higher level of abstraction than any of the formats discussed so far. This level of abstraction is often referred to as the behavioral level. Verilog's procedural blocks are bodies within which statements are executed sequentially. This form of hardware description is, generally, easier for designers to describe their complex hardware. Procedural bodies do provide mechanisms for specification of timing, but, in general, a hardware described with procedural structures contains less timing details than a hardware described with assign statements or primitives. In this section we first present basics of procedural blocks, we will then discuss timing and flow control in procedural blocks. Various types of statements and their simulation semantics and hardware implications.

#### ***FUNCTIONAL BIFURCATION:***

Functions provide a means of splitting code into small parts that are frequently used in a model.

**function** type\_or\_range identifier;

parameter\_declaration;

input\_declaration;

register\_declaration;

event\_declaration;

statement;

**endfunction**

Functions can only be declared inside a module declaration.

Function definition begins with the **function** keyword and ends with the **endfunction** keyword. The returned type or range declaration followed by a function identifier and semicolon should appear after the **function** keyword. Function can contain declarations of range, returned type, parameters, input arguments, registers and events (these declarations are similar to module items declaration). Net declarations are illegal. Declaration of parameters, registers, events and returned type or range are not required. A function without a range or return type declaration, returns a one-bit value. Functions should have at least one input declaration and a statement that assigns a value to the register with the same name as the function.

Any expression can be used as a function call argument. Functions cannot contain any time-controlled statements, and they cannot enable tasks. Functions can return only one value

**function** [15:0] negation;

**input** [15:0] a;

negation = ~a;

**endfunction**

### ***INITIAL CONSTRUCT :***

Verilog supports constructs.

module synthesis\_initial(

clk,q,d);

input clk,d;

output q;

reg q;

initial begin

q <= 0;

end

always @ (posedge clk)

begin

q <= d;

end

endmodule

Construct Type	Keyword or Description	Notes
ports	input, inout, output	Use inout only at IO level.
parameters	parameter	This makes design more generic
module definition	module	
signals and variables	wire, reg, tri	Vectors are allowed
instantiation	module instances / primitive gate instances	E.g.- nand (out,a,b), bad idea to code RTL this way.
function and tasks	function , task	Timing constructs ignored
procedural	always, if, else, case, casex, casez	initial is not supported
procedural blocks	begin, end, named blocks, disable	Disabling of named blocks allowed
data flow	assign	Delay information is ignored
named Blocks	disable	Disabling of named block supported.
loops	for, while, forever	While and forever loops must contain @(posedge clk) or @(negedge clk)

#### ***ASSIGNMENT WITH DELAYS:***

Another form of delay specification in procedural statements is intra assignment delay. Unlike the delay control statement that is by itself a separate procedural statement, an intra-assignment delay (or event) is considered as part of an assignment.

```
timescale 1ns/100ps
```

```
module maj3 (input a, b, c, output reg y);
```

```
always @(a, b, c) y = #5 (a & b) | (b & c) | (a & c);
```

```
endmodule
```

```
and #1 and_gate (o, i1, i2);
```

```
or #(5,1) or_gate (o, i1, i2);
```

```
bufif1 #(3,4,5) buffer (o, i, c);
```

#### **WAIT CONSTRUCT :**

The wait statement is used as a level-sensitive control. The syntax is:

```
wait (expression) statement
```

The processor waits when the expression is FALSE. When the expression is TRUE, the statement is executed.

The expression is treated as a Boolean value, therefore wait responds to TRUE and FALSE only. Values '0', 'x' and 'z' are FALSE. Logic '1' is TRUE.

#### Comparison Between Event and Level Sensitive Processes

An example of an event-driven control is given below as a comparison to the level-sensitive control which will be described later.

```
always @(start) #10 go = ~go;
```

This process uses the @(expression) to trigger the process. The statement will be executed whenever there is an event on the start signal.

In comparison the following example illustrates a level-sensitive control:

```
forever wait(start) #10 go = ~go;
```

The process waits until start is '1'. When the start expression is TRUE, the go signal toggles after 10 time units. If start continues to stay '1' then go will continue to toggle after every 10 time units due to the forever definition. The toggling statement will only stop when start returns to '0'.

An event sensitive process is triggered by the edge on a control signal, while a level sensitive process is triggered by the value on the control signal.

#### Applications Of The wait Statement



The wait statement can be used to:

Synchronise concurrent processes

Hand shake between concurrent processes

The above example of a wait statement is also an example of synchronising. If more than one process is controlled by the start signal in a similar manner to that above, then when **start** goes high, several processes will start to run together. Thus they have been synchronised using the **start** signal.

```
always begin
```

```
    read = 1;
```

```
    forever begin
```

```
        wait (write)
```

```
        // manipulate data
```

```
        storeddata = datain;
```

```
    #10;
```

```
    read = 0;
```

```
    wait (!write)
```

```
        read = 1;
```

```
    end // forever begin
```

```
end // always begin
```

### ***MULTIPLE ALWAYS BLOCKS:***

The block statements provide a means of grouping two or more statements in the block.

**begin** : name

```
    statement;
```

```
    ...
```

**end**

**fork** : name

statement;

...

## Join

### ***DESIGNS AT BEHAVIORAL LEVEL MODELLING:***

The behavior level is used to describe a system intuitively. Therefore, this level is frequently used to verify algorithms or system behavior like a high-level language. This level is not focused on synthesizability or structural realization of the design.

This behavior-level description includes an initial statement and thus it is not synthesizable.

```
module adder4 (in1, in2, sum, zero);
```

```
input [3:0] in1, in2;
```

```
output [4:0] sum;
```

```
output zero;
```

```
reg [4:0] sum;
```

```
reg zero;
```

```
initial
```

```
begin
```

```
sum = 0;
```

```
zero = 1;
```

```
end
```

```
always @ (in1 or in2)
```

```
begin
```

```
sum = in1 + in2;
```

```
if (sum == 0) zero = 1;
```

```
else zero = 0;
```

```
end
```

endmodule

### ***BLOCKING AND NON BLOCKING ASSIGNMENTS :***

Procedural assignments discussed so far in this chapter are all of the blocking type. This means that while the assignment is taking place, the flow of the program into the procedural block is halted (or blocked). This is especially noticeable when using intra-assignment delays as discussed above. A different type of procedural assignment is a non blocking assignment that uses <= instead of =. This type of assignment schedules its right hand side into the left-hand side reg and continues on to the next statement.

### ***CASE STATEMENT :***

The case statement is a decision instruction that chooses one statement for execution. The statement chosen is one with a value that matches that of the case statement.

#### **case (expression)**

expression : statement

expression {, expression} : statement

**default** : statement

#### **endcase**

#### **casez (expression)**

expression : statement

expression {, expression} : statement

**default** : statement

#### **endcase**

#### **casex (expression)**

expression : statement

expression {, expression} : statement

**default** : statement

## endcase

The case statement starts with a **case** or **casex** or **casez** keyword followed by the case expression (in parenthesis) and case items or **default** statement. It ends with the **endcase** keyword. The default statement is optional and should be used only once. A case item contains a list of one or more case item expressions, separated by comma, and the case item statement. The case item expression and the case item statement should be separated by a colon.

During the evaluation of the case statement, all case item expressions are evaluated and compared in the order in which they are given. If the first case item expression matches the case expression, then the statement which is associated with that expression is executed and the execution of the case statement is terminated. If comparison fails, then the next case item expression is evaluated and compared with the case expression. If all comparisons fail and the **default** section is given, then its statements are executed. Otherwise none of the case items will be executed.

Both case expression and case item expressions should have the same bit length. None of the expressions are required to be a constant expression.

The case expression comparison is effective when all compared bits are identical. Therefore, special types of case statement are provided, which can contain don't-care values in the case expression and in the case item expression. These statements can be used in the same way as the case statement, but they begin with the keywords **casex** and **casez**.

The **casez** statement treats high-impedance (z) values as don't-care values and the **casex** statement treats high-impedance and unknown (x) values as don't care values. If any of the bits in the case expression or case item expression is a don't-care value then that bit position will be ignored.

The don't-care value can be also specified by the question mark (?), which is equal to z value.

```
reg a;
case (a)
  1'b0 : statement1;
  1'b1 : statement2;
  1'bx : statement3;
  1'bz : statement4;
endcase
```

### ***SIMULATION FLOW:***

When the \$time system function is called, it returns the current time as a 64-bit integer value. However, this value is scaled to the `timescale unit. (See Timescale chapter)

The \$stime system function returns current time as a 32-bit unsigned integer value. If the current simulation time is too large and the value does not fit in 32 bits, the function only returns the 32 low order bits of the value. The returned value is also scaled to the `timescale.

The \$realtime system function returns a value as a real number. As with the other time tasks, the returned value is scaled to the `timescale

```
Integer cur_time ;
cur_time = $time ;
```

### Example 2

```
integer cur_time ;
cur_time = $stime ;
```

### Example 3

```
real cur_time ;  
cur_time = $realtime ;
```

Example 4

```
$display($time, "is current time.");
```

### ***IF AND IF ELSE CONSTRUCTS :***

The if statement is used to choose which statement should be executed depending on the conditional expression.

```
if (conditional expression)
```

```
    statement1;
```

```
else
```

```
    statement2;
```

```
if (conditional expression)
```

```
    statement1;
```

```
else if (conditional expression)
```

```
    statement2;
```

```
else
```

```
    statement3;
```

The 'if' statement can be used in two ways: as a single 'if-else' statement (Example 1) or as a multiple 'if-else-if' statement (nested if statement - Example 2).

In the first case when the *conditional* expression is evaluated to true (or non-zero), then *statement1* is executed and if condition is false (or zero), then *statement2* is executed.

In the second case, if the first conditional expression is evaluated to be true, then *statement1* is executed. Otherwise, the second conditional expression is evaluated and depending on its values, *statement2* or *statement3* is executed.

Every statement can be a group of statements (enclosed in a begin-end block - Example 3) or a null statement (; - Example 4). The conditional expression can be any valid expression.

```

if (a == 5)
    b = 15;
else
    b = 25;

```

### ***ASSIGN AND DEASSIGN CONSTRUCTS :***

The assign and deassign procedural assignment statements allow continuous assignments to be placed onto registers for controlled periods of time. The assign procedural statement overrides procedural assignments to a **register**. The deassign procedural statement ends a continuous assignment to a register.

```

module assign_deassign ();

    reg clk,rst,d,preset;

    wire q;

    initial begin

        $monitor("@%g clk %b rst %b preset %b d %b q %b",

            $time, clk, rst, preset, d, q);

        clk = 0;

        rst = 0;

        d = 0;

        preset = 0;

        #10 rst = 1;

        #10 rst = 0;

        repeat (10) begin

            @ (posedge clk);

            d <= $random;

            @ (negedge clk) ;

            preset <= ~preset;

        end

```

```

    #1 $finish;

end

// Clock generator

always #1 clk = ~clk;


// assign and deassign q of flip flop module

always @(preset)

if (preset) begin

    assign U.q = 1; // assign procedural statement

end else begin

    deassign U.q; // deassign procedural statement

end

d_ff U (clk,rst,d,q);


endmodule


// D Flip-Flop model

module d_ff (clk,rst,d,q);

input clk,rst,d;

output q;

reg q;


always @ (posedge clk)

if (rst) begin

    q <= 0;

end else begin

```

```
q <= d;  
end
```

```
endmodule
```

### ***REPEAT CONSTRUCTS:***

The repeat loop executes < statement > a fixed < number > of times.

```
module repeat_example();  
  
    reg [3:0] opcode;  
  
    reg [15:0] data;  
  
    reg      temp;  
  
  
    always @ (opcode or data)  
    begin  
  
        if (opcode == 10) begin  
            // Perform rotate  
  
            repeat (8) begin  
  
                #1 temp = data[15];  
  
                data = data << 1;  
  
                data[0] = temp;  
  
            end  
  
        end  
  
    end  
  
    // Simple test code  
  
    initial begin  
  
        $display (" TEMP DATA");
```



```

$monitor (" %b    %b ",temp, data);

#1 data = 18'hF0;

#1 opcode = 10;

#10 opcode = 0;

#1 $finish;

end

```

### ***FOR LOOP:***

Loop statements provide a means of modeling blocks of procedural statements.

**for** (assignment; expression; assignment) statement;

The **for** instruction executes a given statement until the expression is true. At the initial step, the first assignment is executed. In the second step, the expression will be evaluated. If the expression evaluates to an unknown, high-impedance, or false, the **for** statement will be terminated. Otherwise, the statement and second assignment will be executed. After

#### **initial begin**

```

for (index=0; index < 10; index = index + 2)
  mem[index] = index;
end

```

### ***DISABLE CONSTRUCT :***

The disable statement provides means of terminating active procedures

**disable** task\_identifier;

**disable** block\_identifier

The disable statement can be used to terminate tasks (Example 1), named blocks (Example 2) and loop statements (Example 3) or for skipping statements in loop iteration statements (Example 4). Using the **disable** keyword followed by a task or block identifier will only disable tasks and named blocks. It cannot disable functions. If the task that is being disabled enables other tasks, all enabled tasks will be terminated.

If a task is enabled more than once, then disabling that task terminates all its instances.

```

task t;
output o;
integer o;

```

```
#100 o = 15;
endtask
disable t; // Disabling task t.
```

### **WHILE LOOP:**

Looping statements appear inside procedural blocks only; Verilog has four looping statements like any other programming language.

The **while** instruction executes a given statement until the expression is true. If a **while** statement starts with a false value, then no statement will be executed.

```
module test;
parameter MSB = 8;
reg [MSB-1:0] Vector;
integer t;
initial

begin
  t = 0;
  while (t < MSB)
    begin
      //Initializes vector elements
      Vector[t] = 1'b0;
      t = t + 1;
    end
  end
endmodule
```

### **FOREVER LOOP:**

The **forever** instruction continuously repeats the statement that follows it. Therefore, it should be used with procedural timing controls (otherwise it hangs the simulation).

```
forever statement;

always begin
  counter = 0;
  forever #10 counter = counter + 1;
end
```

### **PARALLEL BLOCKS:**

**Syntax (for parallel block):**

*fork*

**[parallel execution statements]**

*join*

A Block is a section of Verilog code within a module which can be contained within ***begin...end*** statements. In many simulations different blocks run in parallel: operations from different blocks are executed in one time slice. Within a block it is more usual for commands to be executed in a serial manner, particularly when learning the language as this approach mirrors programming languages more closely. The following two examples illustrate serial execution:

```
module serial_no_delay;
  reg    a, b;
  reg [1:0] x, y, z;

  initial begin
    a = 1'b0;
    b = 1'b1;
    x = {a, b};
    y = {b, a};
    z = y;
  end
endmodule
```

EXAMPLE 1

```
module serial_with_delays;
  reg    a, b;
  reg [1:0] x, y, z;

  initial begin
    a = 1'b0;
    #20 b = 1'b1;
    #10 x = {a, b};
    #30 y = {b, a};
    #30 z = y;
  end
endmodule
```

EXAMPLE 2

## **EVENTS :**

In Verilog, named events are static objects that can be triggered via the **->** operator, and processes can block until an event is triggered via the **@** operator. System Verilog events support the same basic operations, but enhance Verilog events in several ways. The most salient semantic difference is that Verilog named events do not have a value or duration, whereas SystemVerilog events can have a persistency that lasts throughout the time-step on which they are triggered. Also, System Verilog events act as handles to synchronization queues, thus, they can be passed as arguments to tasks, and they can be

dynamically allocated and reclaimed. System Verilog provides for two different types of events: persistent events and non-persistent events.

## **Unit-IV**

### ***SWTICH LEVEL MODELLING:***

Switch level models are used to allow detailed construction of logical gates and functions and also to allow complex delay modeling to be used. Usually, transistor level modeling is referred to modeling hardware structures using transistor models with analog input and output signal values. On the other hand, gate level modeling refers to modeling hardware structures using gate models with digital input and output signal values. Between these two modeling schemes is what is referred to as switch level modeling. At this level, a hardware component is described at the transistor level, but transistors only exhibit digital behavior and their input, and output signal values are only limited to digital values.

At the switch level, transistors behave as on-off switches. Verilog uses a 4-value logic value system, so Verilog switch input and output signals can take any of the four 0, 1, Z, and X logic values. Switch constructs, their simulation behavior and simulation of hardware constructs based on such switches will be discussed here.

### ***BASIC TRANSISTOR& cmos SWITCHES:***

A 2-input NAND gate using NMOS and PMOS transistors is shown. The inputs of the pmos primitives are tied to Vdd to supply logic 1 to the output, and the input of the lower nmos primitive is tied to Gnd to supply logic 0 to the output. For unidirectional switches, the switch input is the transistor Source, its output is the Drain, and the switch control input is the transistor Gate input. The input-output arrangement of switches are such that the input sides of the nmos switches are on the Gnd side and the input sides of the pmos switches are on the Vdd side. This arrangement is justified by an actual transistor level circuit because the Source of an NMOS transistor feeds logic 0 to the output (discharging the output through Gnd), and the Source of a PMOS transistor feeds logic 1 to the output (charging it through Vdd) of a CMOS gate

```
cmos c1(out , data , ncontrol , pcontrol ) ;
```

CMOS switches are declared with the keyword `cmos`. A CMOS device can be modeled with a NMOS and PMOS devices. The symbol for a

```
tran t1( inout1, inout2 ) ;
```

```
tranif0 t2 (inout1, inout2 , control ) ;
```

```
tranif1 t3 (inout1, inout2 , control ) ;
```

The `tran` switch acts as a buffer between the two signals `inout1` and `inout2`. Either `inout1`, or `inout2` can be driver signal. The `tranif0` switch connects the two signals `inout1` and `inout2` only if the control signal is logic 0. If the control signal is a logic 1, the nondriver signal gets a high impedance value `z`. The driver signal retains value from its driver. The `tranif1` switch conducts if the control signal is a logic 1.

EXAMPLE:

```
module my_nor(out, A, B);
```

```
output out;
```

```
input A, B;
```

```
wire c;
```

```
supply1 pwr; //pwr is connected to Vdd
```

```
supply0 gnd; //gnd is connected to Vss(ground)
```

```
pmos (c, pwr, B);
```

```
pmos (out, c, A);
```

```
nmos (out, gnd, A);
```

```
nmos (out, gnd, B);
```

```
endmodule
```

### **BI DIRECTIONAL GATES:**

Verilog Provides in-built primitives for basic gate and switch level modeling. Any circuit can be modeled by using continuous assignment of gate and switch level primitives.

```
and (strong1, weak0)#(1,2) gate1(out, in1, in2);
```

This is an and gate with output 'out' and two inputs in1 and in2. Strong1 and weak0 are optional driving strengths and gate1 is optional instance name that can be used while debugging. First parameter in the bracket is output and you can have any number of inputs after that. This is how you use a 3 input and gate without instance name, delay and driving strengths:

```
and (out, in1, in2, in3);
```

and, nand, not, nor, or, xor, xnor, buf, bufif0, bufif1, rtranif1, nmos, pmos, rpmos, tran, rtran, pullup, pulldown, cmos, rnmos, tranif1, tranif0, notif0, notif1, rtranif0, rcmos are the built-in primitives.

Transmission gates are bi-directional and can be resistive or non-resistive.

**Syntax:** *keyword unique\_name (inout1, inout2, control);*

e.g.      tranif0 trans\_gate1 (net5, net8, cnt);  
            rtranif1 rtrans\_gate2 (net5, net12, cnt);

example 2

Transmission gates *tran* and *rtran* are permanently on and do not have a control line. *Tran* can be used to interface two wires with separate drives, and *rtran* can be used to weaken signal

### **TIMING DELAYS:**

Verilog defines some basic logic gates as part of the language. In Figure 1, module *some\_logic\_component* instantiates two gate primitives: the not gate and the and gate. The output of the gate is the first parameter, and the inputs are the rest of the parameters. These primitives are scalable so you can get multiple input gates just by adding inputs into the parameter list. For example:

```
nand a1(out1, in1, in2);           //2-input NAND gate
nand a2(out1, in1, in2, in3, in4, in5); //5-input NAND gate
```

By default the timing delay for the gate primitives is zero time. You can define the rising delay, falling delay using the #(rise, fall) delay operator. And for tri-state gates you can also define the turn-off delay (transition to high impedance state Z) by using the #(rise, fall, off) delay operator. For example

```
notif0 #(10,11,27) inv2(c,d,control) //rise=10, fall=11, off=27(not if control=0)
nor  #(10,11)  nor1(c,a,b); //rise=10, fall=11  (nor gate)
xnor #(10)    xnor1(i,g,h); //rise=10, fall=10  (xnor gate)
```

Also each of the 3 delays can be defined to have minimum, typical, and a maximum value using the a colon to separate the values like 8:10:12 instead of 10 in the above examples. At run time, the Verilog simulator looks for to see if the **+mindelay**, **+typdelay**, or **+maxdelay** option has been defined so that it will know which of the 3 time values to use. In VeriLogger these options are set using the **Project > Project Preferences** menu. If none of the options are specified then the typical value is used.

```
// min:typ:max values defined for the (rise, fall) delays
or  #(8:10:12, 10:11:13) or1(c,a,b);
```

The delay operator has one subtle side effect: it swallows narrow input pulses. Normally, the delay operator causes the output response of a gate to be delayed a certain amount of time. However if the input pulse width is shorter then the overall delay of the gate then the change will not be shown on the output.

Here is a list of logic primitives defined for Verilog:

Gate	Parameter List	Examples
nand nor and or xor xnor	scalable, requires at least 2 inputs(output, input1, input2, , inputx)	and a1(C,A,B);nand na1(out1,in1,in2,in3,in4);nor #(5) n1(D,A,B);//delay = 5 time unitsxor #(3,4,5) x1(E,A,B);//rise,fall,off delaysnor #(3:4:5) n2(F,A,B);//min:typ:max of delays
not buf	(output, input)	not inv1(c,a);
notif0bufif0	control signal active low(output, input, control)	notif0 inv2(c,a, control);
notif1bufif1	control signal active high(output, input, control)	not inv1(c,a, control);

## INSTANTIATIONS WITH STRENGTHS:

The four-value logic in Verilog provides an adequate precision for most logic level simulations. The previous section showed that more precise simulation data can be obtained by using switch level models for the basic logic level constructs. Another feature of Verilog for a more precise simulation data is signal

strength. This section discusses logic strengths and application of this language facility in modeling gate and switch level circuits.

Verilog allows specification of drive strength for primitive gate output and nets. Gate output or net signal strength values are specified in a set of parenthesis that include a strength value for logic 0 and one for logic 1. Allowable drive strengths for logic 0 (i.e., strength0) are supply0, strong0, pull0, weak0, and highz0. Similarly, allowable strengths for logic 1 (strength1) are supply1, strong1, pull1, weak1, and highz1. Strength values can appear in any order in the set of parenthesis that follows a primitive name, a net declaration, or the assign keyword. The default strengths for a gate output or a net are strong0 and strong1 for logic 0 and logic 1, respectively. Charge strengths, representing the strength of a capacitive net, are also supported in Verilog. Charge strength values are large, medium, and small.

Gate output drive strengths are specified after the primitive name when the primitive is instantiated. The example below shows a nand primitive with pull0 and pull1 output strength values.

```
nand (pull0, pull1) # (3, 5) n1 (w, a, b, c);
```

#### ***STRENGTH CONTENTION WITH TRIREG NETS :***

The ability to model varying signal levels as produced by digital hardware is fundamentally important for the simulation of switch level circuits. This is accomplished by assigning various signal strengths as defined in Table 1. The signal strengths vary from supply level own to high impedance level, covering six distinct intermediate levels. Each level can represent a logic 1 (supply1, strong 1 ..... highz1) or a logic 0 (supply0, strong0 ..... highz0).

Strength Name	Strength Level	Element Modelled	Declaration Abbreviation
Supply Drive	7	Power supply connections.	supply
Strong Drive	6	Default gate & assign output strength.	strong
Pull Drive	5	Gate & assign output strength.	pull
Large Capacitor	4	Size of trireg net capacitor.	large
Weak Capacitor	3	Gate & assign output strength.	weak
Medium Capacitor	2	Size of trireg net capacitor.	medium
Small Capacitor	1	Size of trireg net capacitor.	small
High Impedence	0	Not Applicable.	highz

Table 1 : Signal strength definitions.

#### Modelling of weak (resistive) transistors :

Weak or resistive transistors have many uses within the domain of digital design. A property associated with resistive transistors is that any signal passed through one is degraded. This is modelled within Verilog using signal strengths which are reduced when they encounter resistive transistors. Signals are degraded as indicated in Table 2.

Input Strength	Output Strength
supply	pull
strong	pull
pull	weak
weak	medium
large	medium
medium	small
small	small
highz	highz

Table 2: Signal degradation through a resistive transistor as modelled by Verilog

One such occasion where this technique is needed is for the digital latch (Figure 1), where a weak transistor is used to reinforce the poor logic 1 passed by the transmission gate. The logic 1 imposed by this weak transistor can however be overcome by the strong logic 0 passed by the transmission gate.



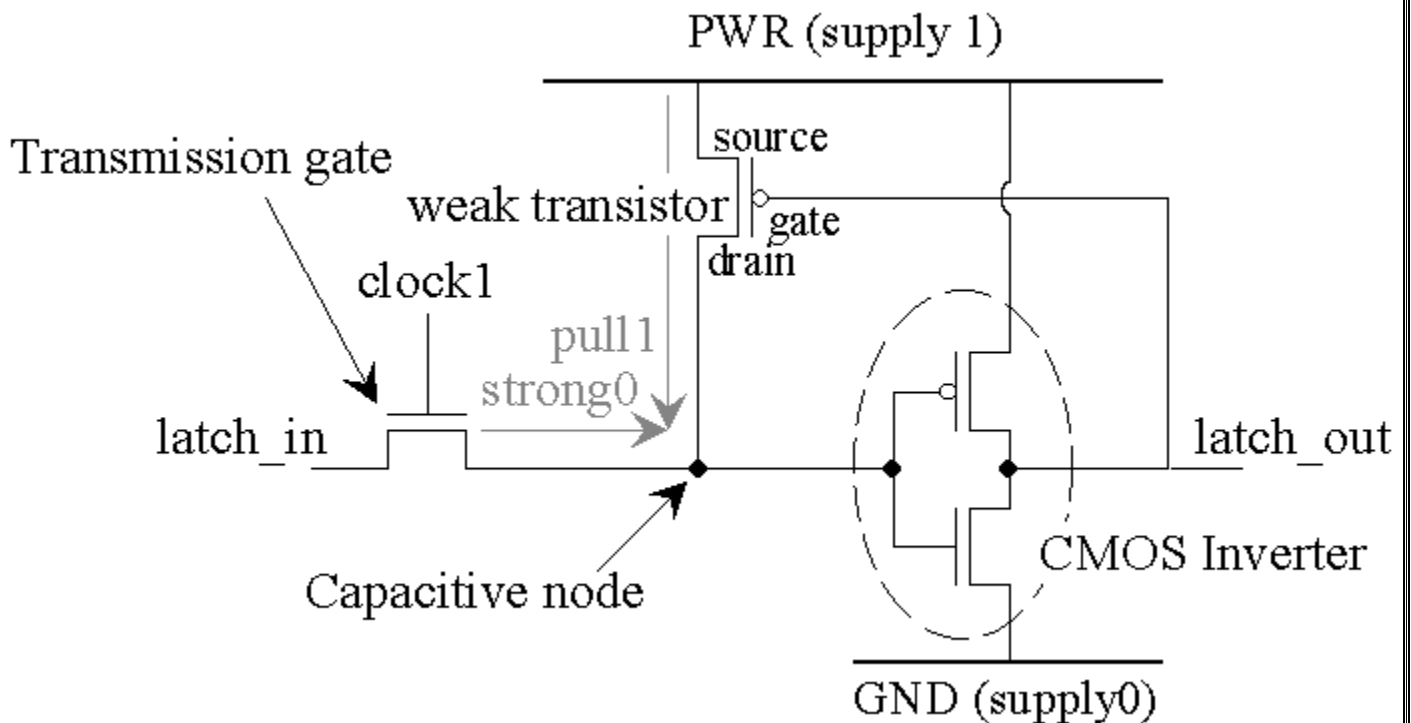


Figure 1 : simple latch circuit exhibiting signal contention.

Resolving signal contention :

The situation often arises whereby two signals drive one node in a circuit resulting in signal contention (Figure 2). The strength and logic value of the two signals are used to resolve the signal conflict to produce a single logic value and strength on the node.

Three situations can arise.

The signals are logically the same but of different strength (Figure 2a) resulting in the output node having the same logic value and a strength equal to the strongest input signal.

The signals are of the same strength but logically opposite (Figure 2b) resulting in an unknown signal of the same strength.

The signals are logically opposite and of different strength (Figure 2c) resulting in an ambiguous signal

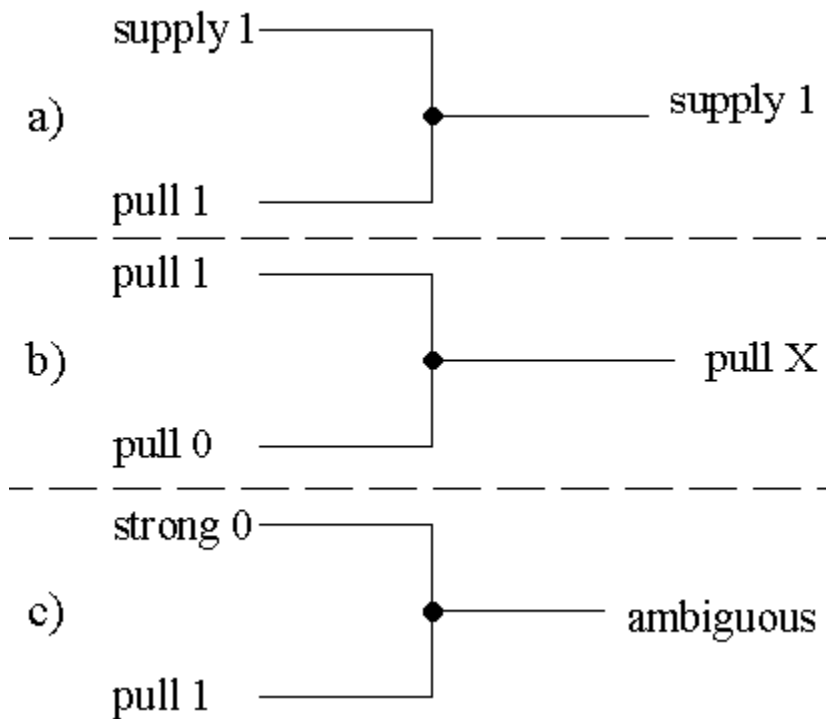


Figure 2 : a) Similar logic, strength contention.

b) Similar strength, logic contention.

c) Logic and strength contention.

## **SYSTEM TASKS, FUNCTIONS, AND COMPILER DIRECTIVES**

### ***INTRODUCTION:***

A number of facilities in Verilog relate to the management of simulation; starting and stopping of the simulation, selectively monitoring the activities, testing the design for timing constraints, etc are among them.

### ***PARAMETERS:***

The flexibility of modules calls for two main reasons:

1. They should be adaptable designs conforming to different technologies.
2. They should have scalable features including timing parameters.

Two types of parameters are of use in modules:

1. Parameters related to timings, time delays, rise times are technology specific and used during simulation.: “specparam” precedes the assignments.
2. Parameters related to design, bus width and register size are of different category and assigned with keyword called “defparam”.

### Timing related parameters:

The constructs associated are discussed here through the specific designs:

```
// half adder
```

```
Module ha_1(s,ca,a,b);
```

```
Input a,b;
```

```
Output s,ca;
```

```
Xor#(1,2)(s,a,b);
```

```
And#(3,4)(cs,a,b);
```

```
End module
```

```
Module tsth_1();
```

```
Reg a,b;
```

```
Wire s,ca;
```

```
Ha_1hh(s,ca,a,b);
```

```
Initial
```

```
Begin
```

```
a=0;
```

```
b=0;
```

```
end
```

```
always begin #5 a=1;b=0;
```

```
#5 a=0;b=1;
```

```
#5 a=1;b=1;
```

```
#5 a=0;b=0;
```

```
End
```

```
Initial monitor($time."a=%b,b=%b, ca=%b,s=%b",a,b,ca,s);
```

```
Initial #30 $stop;
```

```
End module
```

### Module parameters:

It is explained by scaling the ALU size

```
Module alu(d,d,a,b,f,ci);
```

```
Parameter msb=3;
```

```
Output[msb:0];
```

```
Output c;
```

```
Wire[msb:0]d;
```

```
Input ci;
```

```
Input [msb:0]a,b;
```

```
Input [1:0]f;
```

```
Specify
```

```
(a,b=>d)=(1,2);
```

```
(a,b,ci*>c)=1;
```

```
End specify
```

```
Assign {c,d}=(f==2'b00)? a+b+c): ((f==2'b01)? (a-b): ((f==2'b10)? {1'bZ,a^b}: {1'bZ,~a}));
```

```
End module
```

```
Module tst_alu();
```

```
Defparamaa.msb=7; parameter n1=7;
```

```
Reg[n1:0]a,b;
```

```
Reg ci;
```

```
Wire[n1:0]d;
```

```
Wire co;
```

```
Alu_aa(d,d,a,b,f,ci);
```

```
Initial
```

```
Begin
```

```

Ci=1'b0;
F=2'b=00;
A=8'h00;
#30 $stop;
End
Always
Begin
#3 Ci=1'b0;F=2'b=00;A=8'h00;
End
Initial $monitor( $time,"c=%b,a=%b,b=%b,f=%b,d=%b,c=%b",cci,a,b,f,d,c);
End module;

```

### ***PATH DELAYS:***

The time delays are all delays associated with individual operations or activities in a module. They refer to the basic element in the design. Such path and delays are at the chip or system level are referred to as “module path delays”

Module paths are specified and values assigned to their delays through specify blocks.

Specify

```
Specparam rise_time =5, fall_time =6;
```

```
(a=>b)=(rise_time, fall_time);
```

```
(c=>d)=(6,7);
```

Endspecify

The pin to pin path of a signal may change depending on the value of another signal. Conditional selection and assignment of path delays facilitates such simulations. Behavior level modules can have signal paths activated following an edge in a different signal. They can be specified in two ways. It may be specified for positive or negative edge. It may also be specified for rise or fall times. We can assign as edge sensitive state dependent path”.

All transitions on an input pin with less than a specified module path delay are termed as pulses.

They are defined as specparam pathpulse\$(x,y)=(a,b);

### ***MODULE PARAMETERS:***

Module parameters are associated with the size of bus, register, memory, ALU and so on. They can be specified with the concerned module but their values can be altered during instantiation. The alterations can be brought up through the assignments made up with the defparam. It can appear anywhere in the module.

### ***SYSTEM TASKS and FUNCTIONS:***

The Verilog which has number of system tasks and functions are used for taking the output from simulation, control simulation, debugging design modules etc.

\$monitor, \$display are the main output tasks used for displaying and monitoring.

\$strobe task: when a variable or set of variables is sampled and its value displayed it is used. It senses the value of specified variable and displays them. It is executed as the last activity in the concerned time step.

\$stop task suspends the simulation, whereas \$finish stops simulation and closes the simulation environment.

\$random function is used to generate random number of sequences for testing

### ***FILE BASED TASKS and FUNCTIONS:***

/LRM has provision to accommodate and integrate design and test modules kept in different files. It makes room for structuring the design in an elegant manner and developing it with a cross functional approach. Different facilities are specified in LRM.

To carry out any file based task, the file has to be opened, reading, writing etc. completed and the file closed. The keywords for all the file based tasks start with f to differ from other tasks. Eg : \$fdisplay, \$fopen

### ***COMPILER DIRECTIVES:***

A large number of compiler directives are available in Verilog. They allow for macros, file inclusion, time scale related parameters for simulation. They are preceded by the ‘#’ character.

**Define** directive: it is used for macro substitution. It substitutes macro by text.

**Time scale** compiler directives allow the time scale to be specified for the design.

**Timescale a  $\mu$ s/ b ns**

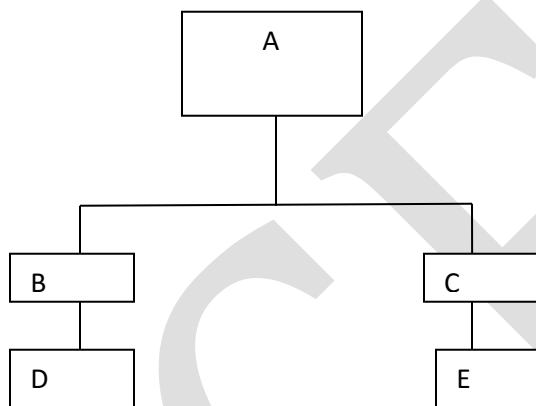
**\$time format** the time scale and display format can be changed during the simulation.

### ***HIERARCHICAL ACCESS***

A Verilog design will have a module or two at apex level. A number of modules and UDP will be instantiated within it. They can have other instantiations within them. They can also have tasks and functions defined within them and can be invoked repeatedly. Verilog has facility to access each such item uniquely and hierarchically.

Every entity in a design has a unique and hierarchical name. but the automatic tasks or functions cannot be accessed hierarchically.

Example:



### ***USER-DEFINED PRIMITIVES***

The primitives available in the Verilog are all of gate or switch types. Verilog has the provision to define primitives called user defined primitives and use them. It can be defined anywhere in the source text and can be instantiated anyway here in the module. Their definition is in the form of a table in a specified format. They are basically of two types – combinational, sequential.

They respectively are used to define combinational and sequential functions.

#### **Combinational UDP:**

A combinational UDP accepts as a set of scalar inputs and gives a scalar output. An inout declaration is not supported by UDP. The first statement starts with a key word primitive. Input and output are declared in the body of UDP where as inout is not.

Simple UDP for AND operation:

Primitive udp\_and(out,in1,in2);

Output out;

Input in1,in2;

Table

//	in1	in2	out
	0	0:	0;
	0	1:	0;
	1	0:	0;
	1	1:	1;

End table

End primitive

### Sequential UDP:

An sequential circuit has set of steps. A positive or negative going edge can trigger the transition from one state to the other state of the circuit. A sequential UDP can accommodate all these. It differs from the combinational UDP in two aspects:\

1. The output has to be defined as a reg. if any change in any of the inputs so demands, the output can change.
2. Values of all input variables as well as present state of the output can affect the next state of the output.

The output of UDPs also can take on values with time delays. The delays can be specified separately for rising and falling transitions on the output.

Example : udp\_and\_b#(1,2)g1(out.in1,in2);

UDPs are scalar in nature. They can be used with vector with proper declarations; however it may not be supported by some simulators.

## Unit-V

### STATE MACHINE CHARTS:



Just as flowcharts are useful in software design, flowcharts are useful in the hardware design of digital systems. We introduce the SM chart, which is also called an ASM (algorithmic state machine) chart. We will see that the SM chart offers several advantages over state graphs.

The state of the system is represented by a state box. The state box contains a state name, followed by a slash (/) and an optional output list. After a state assignment has been made, a state code may be placed outside the box at the top. A decision box is represented by a diamond-shaped symbol with true and false branches. The condition placed in the box is a Boolean expression that is evaluated to determine which branch to take. The conditional output box, which has curved ends, contains a conditional output list. The conditional outputs depend on both the state of the system and the inputs. SM chart constructed from SM blocks.

### DERIVATION OF SM CHARTS:

The method used to derive an SM chart for a sequential control network is similar to that used to derive the state graph. We should draw a block diagram of the system we are controlling. Next we should define the required input and output signals to the control network. Then we can construct an SM chart that tests the input signals and generates the proper sequence of output signals.

### REALIZATION OF SM CHARTS:

Methods used to realize SM charts are similar to the methods used to realize state graphs. As with any sequential network, the realization will consist of a combinational subnetwork, together with flip-flops for storing the state of the network. If the number of states in an SM chart can be reduced, it is not always desirable to do so, since combining states may make the SM chart more difficult to interpret.

The logic equations for the multiplier control are

$$A+ = A'BM'K + A'BM + AB'K = A'B(M+K) + AB'K$$

$$B+ = A'B'ST + A'BM'(K'+K) + AB'(K'+K) = A'B'St + A'BM' + AB'$$

$$\text{Load} = A'B'St$$

$$St = A'BM(K'+K) + AB'(K'+K) = A'BM' + AB'$$

$$Ad = A'BM'$$

$$\text{Done} = AB$$

PLA Table for Multiplier Control:

	A	B	St	M	K	A+	B+	Load	St	Ad	Done
S0	0	0	0	-	-	0	0	0	0	0	0
	0	0	1	-	-	0	1	1	0	0	0
S1	0	1	-	0	0	0	1	0	1	0	0

	0	1	-	0	1	1	1	0	1	0	0
	0	1	-	1	-	1	0	0	0	1	0
S2	1	0	-	-	0	0	1	0	1	0	0
	1	0	-	-	1	1	1	0	1	0	0
S3	1	1	-	-	-	0	0	0	0	0	1
	0	1	0	1	0	1	0	0	0	1	0
	0	1	0	1	1	1	0	0	0	1	0
	0	1	1	1	0	1	0	0	0	1	0
	0	1	1	1	1	1	0	0	0	1	0

### IMPLEMENTATION OF THE DICE GAME:

We realize the SM chart for the dice game using a PLA and three D flip-flops. We use a straight binary state assignment. The PLA has 9 inputs and 7 outputs.

### ALTERNATIVE REALIZATIONS FOR SM CHARTS USING MICROPROGRAMMING:

In applications where the number of inputs to the control network is large, the number of inputs to the PLA, ROM, or PAL will be large. Several methods can be used to reduce the number of inputs required. These methods generally require more states in the SM chart and more output functions to be realized.

### LINKED STATE MACHINES:

When a sequential machine becomes large and complex, it is desirable to divide the machine up into several smaller machines that are linked together. Each of the smaller machines is easier to design and implement. Also, one of the submachines may be “called” in several different places by the main machine. This is analogous to dividing a large software program into procedures that are called by the main program.

1. Explain Dice game with block diagram.
1. Explain Dice game using flow chart.
3. Explain SM chart for Dice game.
1. Design state graph for Dice game controller.
5. Explain about XC4000 implementation of multiplier control.
6. Write differences between FPGA and CPLD.
7. Explain PLA realization of SM charts.
8. Explain PLA table for multiplier control.

## DESIGNING WITH PROGRAMMABLE GATE ARRAYS AND COMPLEX PROGRAMMABLE LOGIC DEVICES

### **Xilinx 3000 series FPGAs:**

As an example of a FPGA, we will describe the Xilinx XC3020 Logic Cell Array (LCA). Which consists of an interior array of 64 configurable logic blocks (CLBs) surrounded by a ring of 64 input-output interface blocks. The interconnections between these blocks can be programmed by storing data in internal configuration memory cells. Each configurable logic blocks contains some combinational logic and two D flip-flops and can be programmed to perform a variety of logic functions.

### **INPUT-OUTPUT BLOCKS:**

The I/O pad connects to one of the pins on the IC package so that external signals can be input to or output from the array of logic cells.

### **PROGRAMMABLE INTERCONNECTS:**

The programmable interconnections between the configurable logic blocks and I/O blocks can be made in several ways –general purpose interconnections, direct interconnects, and long lines.

### **DESIGNING WITH FPGAS:**

Sophisticated CAD tools are available to assist with the design of systems using programmable gate arrays. When the final system is built, the bit pattern for programming the FPGA is normally stored in an EPROM and automatically loaded into the FPGA when the power is turned on.

### **USING A ONE-HOT STATE ASSIGNMENT:**

When designing with PGAs, We should keep in mind that each logic cell contains two flip-flops. This means that it may not be important to minimize the number of flip-flops used in the design. We should try to reduce the total number of logic cells used and try to reduce the interconnections between cells. Using a one-hot state assignment will often help to accomplish this. The one-hot assignment uses one flip-flop for each state, so a state machine with N states requires N flip-flops.

### **ALTERA COMPLEX PROGRAMMABLE LOGIC DEVICES (CPLDS):**

CPLDs are an extension of the PAL concept. A CPLD consists of a number of PAL-like logic blocks together with a programmable interconnect matrix. Each PAL-like logic block has a programmable AND array that feeds macrocells, and the outputs of these macrocells can be routed to the inputs of other logic blocks within the same IC. Many CPLDs are electrically erasable and reprogrammable and, as such are sometimes referred to as EPLDs (erasable PLDs). The Altera MAX 7000 series is a family of high-performance CMOS CPLDs. In contrast to the Xilinx FPGAs, the Altera 7000 series uses EEPROM based configuration memory cells, so that once the configuration is programmed, it is retained until it is erased.

### **ALTERA FLEX 10K SERIES CPLDS:**

The Altera FLEX 10K embedded programmable logic family provides high-density logic along with RAM memory in each device. The logic and interconnections are programmed using configuration

RAM cells in a manner similar to the Xilinx FPGAs. Each row of the logic array contains several logic array blocks (LABs) and an embedded array block (EAB). Each LAB contains eight logic elements and local interconnect channel. The EAB consists 2048 bits of RAM memory.

#### **STATIC RAM MEMORY:**

Static RAM means that once the data is stored in the RAM, it remains there until the power is turned off. Static RAMs are used to store several million bytes of data.

#### **A SIMPLIFIED 486 BUS MODEL:**

A 486 bus model is very complex and supports many different types of bus cycles. A quick data transfer is also supported by it.

#### **INTERFACING MEMORY TO A MICROPROCESSOR BUS:**

In order to design the interface, timing specifications of memory and microprocessor has to be satisfied. The set-up and hold time specifications must also be satisfied. If the memory is slow, it may be necessary to insert wait states in the bus cycle.

#### **SERIAL COMMUNICATION INTERFACE DESIGN:**

The serial communication interface, which receives and transmits serial data is called UART (universal Asynchronous Receiver-Transmitter). It is used to communicate with devices such as mouse, keyboard etc.,

### **15. ADDITIONAL TOPICS**

Shall be provided later, as this has a revised syllabus and the course content is to be studied in details.

## 16. University previous Question papers

Code No: 114AF

**R13**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B.Tech II Year II Semester Examinations, May-2015

**DIGITAL DESIGN USING VERILOG HDL**

(Electronics and Communication Engineering)

Time: 3 Hours

Max. Marks: 75

**Note:** This question paper contains two parts A and B.

Part A is compulsory which carries 25 marks. Answer all questions in Part A.

Part B consists of 5 Units. Answer any one full question from each unit.

Each question carries 10 marks and may have a, b, c as sub questions.

### PART-A

(25 Marks)

- 1.a) Write difference between tasks and functions. [2M]
- b) Illustrate with an example Array of Instances of Primitives. [3M]
- c) What are Tristate gates? [2M]
- d) Mention data types used in Verilog HDL. [3M]
- e) Write any two sequential models can be used. [2M]
- f) Write about bidirectional gates. [3M]
- g) What are parallel blocks? [2M]
- h) What are time delays with switch primitives? [3M]
- i) Draw the diagram of NAND gate using CMOS switches. [2M]
- j) Write Verilog code using Case statement. [3M]

### PART-B

(50 Marks)

2. Explain the following "lexical conventions" with examples.  
a) White space      b) strengths      c) Operators [3+3+4]
- OR**
3. write a short notes on concurrency and functional verification.  
b) Explain port Declaration with an example using Verilog code. [5+5]
- 4.a) Classify and explain strengths and contention resolution.  
b) Write Verilog code for 1 to 4 demultiplexer module by using 2 to 4 decoder? [5+5]
- OR**
- 5.a) Write Verilog module for a positive edge triggered flip flop with test bench.  
b) Explain how the ALWAYS statements are used in Verilog. [5+5]
- 6.a) Design Verilog module Event construct for a serial data receive and test bench for the same.  
b) Design a counter module and test bench to illustrate the use of WAIT. [5+5]
- OR**
- 7.a) Describe procedural continuous assignment statements assign, de assign, force and release.  
b) Explain the compiles directives in detail. [5+5]
- 8.a) Design CMOS switch of parallel combination.  
b) Explain and specify blocks of Path Delay Modeling. [5+5]
- OR**
- 9.a) Write the code for CMOS switch of parallel combination.  
b) Briefly explain combinational and sequential UDPs in Verilog. [5+5]
- 10.a) Write the verilog code for basic functional unit of a dynamic shift register.  
b) Write a short note on Design verification. [5+5]
- OR**
- 11.a) Briefly explain any one method used for sequential circuit testing.  
b) Write Verilog module for 8-bit comparator with test bench. [5+5]

## **17. Question Bank**

### **UNIT1:**

- 1) Explain programming language interface
- 2) Explain levels of design description
- 3) Explain simulation and synthesis with differences
- 4) Write about system tasks with examples?
- 5) Mention keywords and their significance?
- 6) Explain data types of Verilog
- 7) Explain the following (a) scalars and vectors (b) parameters (c) white space
- 8) Explain operator in Verilog.
- 9) Explain system tasks.

### **UNIT II:**

- 1) Explain gate level modeling with example?
- 2) Design half adder using gate level modeling?
- 3) Design full adder using gate level modeling?
- 4) Design full adder using half adder using gate level modeling?
- 5) Explain delays with an example
- 6) Explain net types
- 7) Design d flip flop with gate primitives
- 8) Explain tri state gates
- 9) Write about module structure
- 10) Write the Verilog program for 2 bit comparator in gate model?
- 11) Write about continuous assignment structures
- 12) Explain assignment to vectors
- 13) Explain delays with a program
- 14) Design half adder using data flow modeling?
- 15) Design full adder using data flow modeling?
- 16) Design full adder using half adder using data flow modeling?
- 17) Write the Verilog code for cmos NOR in data flow model
- 18) Write the Verilog code for nmos NOR in data flow model
- 19) Write the Verilog code for cmos NAND in data flow model

### **UNIT III**

- 1) Explain operators in data flow?
- 2) Explain wait construct with an example
- 3) Explain force release construct with an example

- 4) Explain forever loop
- 5) Explain the difference between blocking and non blocking assignments
- 6) Explain repeat construct
- 7) Explain design at behavioral levels
- 8) Explain if and else if constructs
- 9) Explain case statement with a program
- 10) Write about simulation flow?

#### **UNIT IV**

- 1) Explain basic transistor switches.
- 2) Explain CMOS switches
- 3) Write the Verilog code for cmos NOR in switch level model
- 4) Write the Verilog code for nmos NOR in switch level model
- 5) Write the Verilog code for cmos NAND in switch level model
- 6) Explain parameters
- 7) Explain path delays
- 8) Explain file based tasks with an examples
- 9) Explain hierarchical access with a program
- 10) Explain system based tasks and functions
- 11) Explain sequence detector with fsm program
- 12) What are user defined primitives
- 13) What are compiler directives
- 14) Explain module parameters

#### **Unit- V**

1. Explain Dice game with block diagram.
2. Explain Dice game using flow chart.
3. Explain SM chart for Dice game.
2. Design state graph for Dice game controller.
5. Explain about XC4000 implementation of multiplier control.
6. Write differences between FPGA and CPLD.
7. Explain PLA realization of SM charts.
8. Explain PLA table for multiplier control.

## **18. Assignment topics**

### **UNIT 1:**

- 1) Explain programming language interface
- 2) Explain levels of design description
- 3) Explain simulation and synthesis with differences
- 4) Explain data types of Verilog
- 5) Explain the following (a) scalars and vectors (b) parameters (c) white space
- 6) Explain operator in Verilog.

### **UNIT 2:**

- 1) Design full adder using gate level modeling
- 2) Explain net types
- 3) Design d flip flop with gate primitives
- 4) Explain tri state gates
- 5) Design full adder using data flow modeling?
- 6) Design full adder using half adder using data flow modeling?

### **UNIT 3**

- 1) Explain operators in data flow?
- 2) Explain wait construct with an example
- 3) Explain the difference between blocking and non blocking assignments
- 4) Explain repeat construct
- 5) Explain design at behavioral levels
- 6) Write about simulation flow?

### **UNIT 4**

- 1) Explain basic transistor and CMOS switches.
- 2) Write the Verilog code for cmos NOR, nmos NOR, cmos NAND in switch level model
- 3) Explain parameters and path delays
- 4) Explain hierarchical access with a program
- 5) Explain system based tasks and functions
- 6) What are user defined primitives
- 7) What are compiler directives

### **UNIT 5**

1. Explain Dice game with block diagram.  
Explain Dice game using flow chart.



2.Explain SM chart for Dice game.

Design state graph for Dice game controller.

3.Explain about XC4000 implementation of multiplier control.

Write differences between FPGA and CPLD.

4.Explain PLA realization of SM charts.

Explain PLA table for multiplier control.

## **19. Unit-wise quiz questions and long answer questions**

### **Feb 2015 Mid-1 Quiz Paper**

- 1 In Verilog constants defined in a module by the keyword [ C ]  
A) Constant B) Parameter C) Const D) None
- 2 How many logic values defined in Verilog with their strength's [ B ]  
A) One B) Two C) Three D) Four
- 3 Trireg nets can have \_\_\_\_\_ values. [ C ]  
A) 0, 1, x, z B) 0, 1, z only C) 0, 1, x only D) 0, 1 only
- 4 An escaped identifier should start with a \_\_\_\_\_. [ C ]  
A)\* B) \$ C) \ D) white space
- 5 \_\_\_\_\_ is an array of reg variables. [ D ]  
A) Data type B) String C) Vector D) Memory
- 6 Special characters can be displayed in strings only when they are preceded by [ B ]  
A) Null characters B) Escape characters C) Variable D)None
- 7 \$stop is used for [ A ]  
A) Break point B) Start point C) Initial point D) Terminate the program
- 8 String can be stored in [ B ]  
A) Wire B) Data C) Reg D) None
- 9 Which of the following is not a white space character [ D ]  
A) \t B) \n C) \b D) \s
- 10 'time' is an example of \_\_\_\_\_ data type. [ B ]  
A) Fixed B) Variable C) Net D) Dataflow
- 11 Process of converting a high – level description of design into an optimized gate level representation is called logic synthesis.
- 12 Event based timing control is possible with behavioral modeling
- 13 Implicit continuous assignment of delay can be used in data flow modeling.
- 14  $\gg$  is the symbol of logical right shift operator.
- 15 What first character identifies a system task or a system function \$.
- 16 To represent physical connection between structural elements net data type can be used.
- 17 Delay associated with a gate output transition to a '0' from another value is called fall time delay.
- 18 Write the syntax for repeat construct? \_\_\_\_\_
- 19 Give an example for scalar-net data type representation wire x;.
- 20 Verilog HDL was first developed by Gateway Automation.

## **UNIT 1:**

1. ASIC stands for **Application specific integrated circuits**
2. Verilog has **same code** for test bench and design
3. Simulation at uniform levels obtains **concurrency**
4. Testing is done in **Functional Test** and **Timing Test**.
5. The names of system tasks and functions begin with a **dollar Sign (\$)**  
Timescale tasks are **\$sprinttimescale and \$timeformat**
6. Declaration of mode, type, and size of ports can either appear in the **portlist**
7. **IF** verifies conditional statements.
8. A block comment begins with **the two characters**
9. Constants in Verilog are **integer or real**
1. Verilog has \_\_\_\_\_ for test bench and design
  - a) **Different code**
  - b) Same code
  - c) No code
  - d) None of the above
2. A block comment begins with
  - a) /
  - b) //
  - c) !!
  - d) --:
3. simulation at uniform levels obtains \_\_\_\_\_
  - a) non persistence
  - b) loops
  - c) **concurrency**
  - d) synthesis

## **UNIT 2**

1. cells used for gate level simulation, or what is called as **SDF simulation**
2. Verilog gate level list includes **standard n input, n output, and tri-state gates**
3. A module is comprised of the **interface and the design behavior**
4. All module declarations must begin with the **module**
5. If there is no instantiation inside the module, it will be treated as **a top-level module**.
6. Transmission gates tran and rtran are permanently on and do not have a **control line**
7. Verilog basic logic gates are called **primitives**
8. **Delays** specify a time in which assigned values propagate through nets or from inputs to outputs of gates
9. The delays declaration can contain up to three values: **rise, fall, and turn-off delays**
10. **Strengths** can be used to resolve which value should appear on a net or gate output.
11. **All strengths** can be ordered by their value
12. Nets are data types that can be used to **model physical connections**

13. The 't' variable is **trireg** net variable with small charge strength
14. Nets can be declared in **a net declaration statement or in a net declaration assignment**
15. If a net variable has no driver, then it has a **high-impedance value**
16. **Dataflow modeling** provides a powerful way to implement a design
17. **Delay value control** the time between the change in a right-hand-side operand and when the new value is assigned to the left-hand-side
18. Vectors can be declared for all types **of net data types and for reg data types**
19. Vector nets and registers are treated **as unsigned values**
20. Verilog uses a **4-value logic value**
21. **The delay operator** has one subtle side effect: it swallows narrow input pulses
22. The signals are logically opposite and of different strength resulting in an **ambiguous signal**
23. Parameters related to timings, time delays, rise times are technology specific and used during simulation. **“specparam” precedes the assignments**
24. Allowable drive strengths for **logic 0 (i.e., strength0)** are **supply0, strong0, pull0, weak0, and highz0**
25. **delay operator** causes the output response of a gate to be delayed a certain amount of time

### **UNIT 3:**

1. level of abstraction is often referred to as the **behavioral level**
2. **Procedural bodies** do provide mechanisms for specification of timing
3. Functions provide a means of **splitting code into small parts** that are frequently used in a model.
4. **Functions** can only be declared inside a module declaration.
5. Any expression can be used as a **function call argument**
6. Another form of delay specification in procedural statements is **intra assignment delay**
7. The **wait statement** is used as a level-sensitive control.
8. **The processor waits** when the expression is FALSE.
9. When the expression is **TRUE**, the statement is executed.
10. When the start expression is **TRUE**, the go signal toggles after 10 time units
11. **An event sensitive process** is triggered by the edge on a control signal, while a level sensitive process is triggered by the value on the control signal
12. The wait statement can be used to: **Synchronise concurrent processes**
13. The **block statements** provide a means of grouping two or more statements in the block.
14. The **behavior level** is used to describe a system intuitively.
15. The behavior-level description includes an initial statement and thus it is not **synthesizable**.

### **UNIT 4:**

1. The **tran switch** acts as a buffer between the two signals inout1 and in-out2
2. Verilog Provides **in-built primitives** for basic gate and switch level modeling
3. Transmission gates are bi-directional and can **be resistive or non-resistive**
4. **The delay operator** has one subtle side effect: it swallows narrow input pulses
5. The signals are logically opposite and of different strength resulting in an **ambiguous signal**

6. Parameters related to timings, time delays, rise times are technology specific and used during simulation.: **“specparam” precedes the assignments**
7. The logic 1 imposed by this weak transistor can however be overcome by the strong logic 0 passed by **the transmission gate**.
8. The ability to model varying signal levels as produced by digital hardware is fundamentally important for the simulation **of switch level circuits**
9. Allowable drive strengths for **logic 0 (i.e., strength0) are supply0, strong0, pull0, weak0, and highz0**
10. **delay operator** causes the output response of a gate to be delayed a certain amount of time
11. path and delays at the chip or system level are referred to as **“module path delays”**
12. Module paths are specified and values assigned to their delays **through specify blocks**
13. **The pin to pin path of a** signal may change depending on the value of another signal
14. All transitions on an input pin with less than a specified module path delay are termed as **pulses 1**
15. **Module parameters** are associated with the size of bus, register, memory
16. The alterations can be brought up through the assignments made up with the **defparam**
17. **\$monitor, \$display** are the main output tasks used for displaying and monitoring
18. A Verilog design will have **a module or two at apex level**
19. Every entity in a design has a **unique and hierarchical name**
20. **Bigger designs** are better arranged in small functional blocks
21. UDP is **a user defined primitive**
22. A function is like **a subroutine or a procedure** in the program
23. a function has **only input argument**
24. The primitives available in the Verilog are **all of gate or switch types**
25. **A combinational UDP** accepts as a set of scalar inputs and gives a scalar output

## **Unit 5**

- 1) In digital circuits, storage of data is done either by **feedback** or by **gate capacitances** that are refreshed frequently.
- 2) Feedback models and capacitive models are **technology dependent**.
- 3) Verilog provides **timing check constructs** for ensuring correct operation of implicit modeling.
- 4) A sequential UDP has the format of the combinational UDP except that its inputs, outputs and **present state** is also specified.
- 5)  $Q = d$ ;  $q\_b = \sim d$ ; are **blocking assignments**.
- 6)  $Q <= d$ ;  $q\_b <= \sim d$ ; are **non-blocking assignments**.
- 7) With each clock edge, the entire procedural block is executed once from **begin to end**.
- 8) A **fork-join bracketing** instead of begin-end causes all sequential statements to be executed in parallel.
- 9) A **sequential assign statement** forces a value into reg type variable, and a **sequential deassign** removes it.
- 10) **Setup time** is the minimum necessary time that a data input requires to setup before it is clocked into a flip-flop.
- 11) **Hold time** is the minimum necessary time that a flip-flop data must stay stable after it is clocked.
- 12) **\$readmemh** and **\$readmemb** tasks are for reading external data files and using them for initialization of memory blocks.
- 13) An **inout** bus is only considered as net and cannot be declared as a reg.

- 14) Verilog PLA modeling tasks use a personality memory whose contents determine PLA fusing.
- 15) A register is a group of flip-flops with a common clock.
- 16) A moore machine is a state machine in which all outputs are fully synchronized with the circuit clock.
- 17) In a mealy machine, its output depends on its current state and inputs.
- 18) Verilog simulation environment provide tools for graphical or textual display of simulation results.
- 19) A Verilog testbench is a Verilog module that instantiates an MUT applies data to it and monitors its output.
- 20) Testing sequential circuits involves synchronization of circuit clock with other data inputs.
- 21) \$stop and \$finish are simulation control tasks.
- 22) Formal verification is a way of automating design verification by eliminating testbenches and problems associated with their data generation and response observation.
- 23) The assert one hot assertion monitor checks that while the monitor is active only one bit of its n-bit expression is 1.
- 24) The assert cycle sequence is a very useful assertion for verifying state machines.
- 25) A useful assertion for checking expected events or events implied by other events is the assert implication assertion.
- 26) Assert next assertion verifies that starting and an ending events occur with a specified number of clocks in between.
- 27) In assertion verification, in-code monitors take the responsibility of issuing a message if something happens that is not expected.

## **20. Tutorial Problems:**

Shall be provided later, as this has a revised syllabus and the course content is to be studied in details.

## **21. Known Curriculum Gaps and inclusion of the same in the lecture schedule:**

Shall be provided later, as this has a revised syllabus and the course content is to be studied in details.

## **22. Group discussion topics**

Shall be provided later.

## **23. References, Journals, websites and E-links**

### **REFERENCES:**

1. Fundamentals of Logic Design with Verilog Design– Stephen. Brown and Zvonko Vranesic, TMH, 2<sup>nd</sup> Edition 2010.
2. Advanced Digital Logic Design using Verilog, State Machine & Synthesis for FPGA – Sunggu Lee, Cengage Learning , 2012.
3. Verilog HDL – Samir Palnitkar, 2<sup>nd</sup> Edition, Pearson Education, 2009.
4. Advanced Digital Design with Verilog HDL – Michael D. Ciletti, PHI, 2009.

## WEBSITES

1. <http://onlinelibrary.wiley.com/doi/10.1002/9780470823798.app1/pdf>
2. <http://verilog.renerta.com/source/vrg00047.htm>
3. <http://electrosofts.com/verilog/beginend.html>
4. <http://www.ee.iitb.ac.in/student/~vivektiru/img/Verilog.pdf>

## JOURNALS

1. www.mcjournal.com (web journal on VLSI)
2. Microprocessors and Microcomputer System
3. VLSI Hardware Desig

## **24. Quality Control Sheets**

### A. Course End Survey:

Course end survey will be collected at the end of the semester.

### B. Teaching Evaluation

Quality control department conducts online feedback, two times in the semester.

## **25. Students list**

ECE – 2-1A		
S.No	Roll Number	Name of the Candidate
1	14R11A0401	ADITYA B
2	14R11A0402	ADULLA JANARDHAN REDDY
3	14R11A0403	ANDE HEMANTH REDDY
4	14R11A0404	ANKATI NAVYA
5	14R11A0405	ASHFAQ AZIZ AHMED
6	14R11A0406	BANDI SANDHYA
7	14R11A0407	BASWARAJ SHASHANK YADAV
8	14R11A0408	BITLA SRIKANTH REDDY
9	14R11A0409	BUDDANA DHARANI KUMAR
10	14R11A0410	CHEBARTHI RAMYA GAYATHRI
11	14R11A0411	CHETLAPALLI NAGA SAI SUSHMITHA

12	14R11A0412	DASARI DHAMODHAR REDDY
13	14R11A0413	G AYESHA SULTANA
14	14R11A0414	G MADHURI
15	14R11A0415	G RISHI RAJ
16	14R11A0416	G VAMSHI KRISHNA
17	14R11A0417	G VENKATESH YADAV
18	14R11A0418	GONDA RISHIKA
19	14R11A0419	GUDE GOPI
20	14R11A0420	JAGGANNAGARI MANOJKUMAR REDDY
21	14R11A0421	JAGGARI SRINIJA REDDY
22	14R11A0422	JALAGAM NANDITHA
23	14R11A0423	JAMMIKUNTALA SHIVA CHARAN
24	14R11A0424	JATAPROLU LAKSHMI SOWMIKA
25	14R11A0425	JEKSANI SHREYA
26	14R11A0426	K VIJAY KUMAR
27	14R11A0427	KAALISSETTY KRISHNA CHAITANYA
28	14R11A0428	KAKARLA MOUNICA
29	14R11A0429	KARRE PRIYANKA
30	14R11A0430	KL N SATYANARAYANA MURTHY
31	14R11A0431	KONDA KRITISH KUMAR
32	14R11A0432	KOPPULA RAHUL
33	14R11A0433	KURUGANTI RUNI TANISHKA SHARMA
34	14R11A0434	L THRILOK
35	14R11A0435	MANDULA SANTOSHINI
36	14R11A0436	MATLA PRINCE TITUS
37	14R11A0437	NARSETTI SAIPRAVALIKA
38	14R11A0438	NIKITHA RAGI
39	14R11A0439	P VIJAYA ADITYA VARMA
40	14R11A0440	PASHAM VIKRAM REDDY
41	14R11A0441	PELLURI KARAN KUMAR
42	14R11A0442	PERURI CHANDANA
43	14R11A0443	PODUGU SRUJANA DEVI
44	14R11A0444	RAJNISH KUMAR
45	14R11A0445	RAJU PAVANA KUMARI
46	14R11A0446	RAMIDI NITHYA
47	14R11A0447	RAMOJI RAJESH
48	14R11A0448	S ALEKHYA
49	14R11A0449	SARANGA SAI KIRAN
50	14R11A0450	SHAIK SAMEER ALI
51	14R11A0451	SOUMYA MISHRA
52	14R11A0452	SRIRAMOJU MANASA
53	14R11A0453	T ARUN KUMAR
54	14R11A0454	T S SANTHOSH KUMAR

55	14R11A0455	V BAL RAJ
56	14R11A0456	V POOJA
57	14R11A0457	V SRIVATS VISHWAMBER
58	14R11A0458	VEMI REDDY VISHNU VARDHAN REDDY
59	14R11A0459	VENNAMANENI VAMSI KRISHNA
60	14R11A0460	YERASI TEJASRI
61	15R15A0401	RAMIDI SANDEEP REDDY
62	15R15A0402	ODDARAPU HARISHBABU
63	15R15A0403	KOLUKURI BHARGAVI
64	15R15A0404	ADEPU MOUNIKA
65	15R15A0405	AVANCHA PRAVALIKA
66	15R15A0406	NELLUTLA VISHAL CHAITANYA
67	15R15A0407	VEMUNA JAMEENA
<b>ECE -2-1B</b>		
<b>S.No</b>	<b>Roll Number</b>	<b>Name of the Candidate</b>
1	14R11A0461	ADDAKULA SURESH
2	14R11A0462	AGARTI MADHU VIVEKA
3	14R11A0463	AKULA SAI KIRAN
4	14R11A0464	ANUMULA SNIGDHA
5	14R11A0465	B DIVYA
6	14R11A0466	B MANOHAR
7	14R11A0467	BANDARI MAMATHA
8	14R11A0468	BINGI DIVYA SUDHA RANI
9	14R11A0469	BIRE BHAVYA
10	14R11A0470	CH SAI BHARGAVI
11	14R11A0471	CHAVALI SUMA SIREESHA
12	14R11A0472	CHELLABOINA SHIVA KUMAR
13	14R11A0473	CHETTY AKHIL CHAND
14	14R11A0474	CHINTAPALLI MADHAV REDDY
15	14R11A0475	CHIVUKULA VENKATA SUBRAMANYA PRASANTH
16	14R11A0476	D NAGA SUMANVITHA
17	14R11A0477	D VAMSI
18	14R11A0478	DHARMENDER KEERTHI
19	14R11A0479	EADARA NAGA SIRISHA
20	14R11A0480	ERANKI SAI UDAYASRI ALAKANANDA
21	14R11A0481	GANGA STEPHEN RAVI KUMAR
22	14R11A0482	GUNDAM SHRUTHI
23	14R11A0483	GUNDREVULA SAMEERA
24	14R11A0484	K NAGA REKHA
25	14R11A0485	KANDADI VARSHA
26	14R11A0486	KURELLI SAI VINEETH KUMAR GOUD
27	14R11A0487	MADDIKUNTA SOMA SHEKAR REDDY
28	14R11A0488	MAMILLA SAI NISHMA



29	14R11A0489	MARELLA NAGA LASYA PRIYA
30	14R11A0490	MARKU VENKATESH
31	14R11A0491	MOHAMED KHALEEL
32	14R11A0492	MOHAMMED WASEEM AKRAM
33	14R11A0493	MOTURI DIVYA
34	14R11A0494	MUDIUM KOUSHIKA
35	14R11A0495	MYLAPALLI RAMBABU
36	14R11A0496	NAGU MOUNIKA
37	14R11A0497	NEELAM SNEHANJALI
38	14R11A0498	NIDAMANURI VENKATA VAMSI KRISHNA
39	14R11A0499	NIKHIL KUMAR N
40	14R11A04A0	ORUGANTI HARSHINI
41	14R11A04A1	PARAMKUSAM NIHARIKA
42	14R11A04A2	PASAM ABHIGNA
43	14R11A04A3	PATI VANDANA
44	14R11A04A4	PODISHETTY MANOGNA
45	14R11A04A5	PONAKA SREEVARDHAN REDDY
46	14R11A04A6	R NAVSHETHA
47	14R11A04A7	R PRANAY KUMAR
48	14R11A04A8	RAMIDI ROJA
49	14R11A04A9	RUDRA VAMSHI
50	14R11A04B0	S SHARAD KUMAR
51	14R11A04B1	SAGGU SOWMYA
52	14R11A04B2	TADELA SARWANI
53	14R11A04B3	THOTA SAI BHUVAN
54	14R11A04B4	VALLAPU HARIKRISHNA
55	14R11A04B5	VECHA PAVAN KUMAR
56	14R11A04B6	Y SAI VISHWANATH
57	15R15A0408	ERUKALA NIKITHA
58	15R15A0409	PUNGANUR JAYACHANDRA BHARATHWAJ
59	15R15A0410	GALIPALLY BHARGAVA
60	15R15A0411	PADMA ARUNRAJ
61	15R15A0412	JAMALAPURAM NAVEEN
62	15R15A0413	MACHANNI BALAKRISHNA YADAV
63	15R15A0414	ANABOINA MAHENDER
64	15R15A0415	ANABOINA SHIVA SAI
65	15R15A0416	VEMULA VINITHA
66	15R15A0417	CHEVU NAGESH
<b>ECE – 2-1C</b>		
<b>S.No</b>	<b>Roll Number</b>	<b>Name of the Candidate</b>
1	14R11A04B9	ANAMALI REETHIKA
2	14R11A04C0	ARUMILLI LEKYA
3	14R11A04C1	ARUMUGAM ASHWINI

4	14R11A04C2	BASAVARAJU MEGHANA
5	14R11A04C3	BEERAM TEJASRI REDDY
6	14R11A04C4	BHARAT SAKETH
7	14R11A04C5	BOMMANA HARIKADEVI
8	14R11A04C6	BYRAGONI ROJA
9	14R11A04C7	CANDHI SHASHI REKHA
10	14R11A04C8	CH RENUKA
11	14R11A04C9	CHAGANTI MOUNICA
12	14R11A04D0	CHITTARLA LOKESH GOUD
13	14R11A04D1	D LAVANYA
14	14R11A04D2	D MANIKANTA
15	14R11A04D3	DASARI VENKATA NAGA SAISH
16	14R11A04D4	DODDA MANOJ
17	14R11A04D5	E RAHUL CHOWDHARY
18	14R11A04D6	GOWRISHETTY VINEETHA
19	14R11A04D7	GUNTUPALLI RAVI TEJA
20	14R11A04D8	K L ANUSHA
21	14R11A04D9	K SASIDHAR
22	14R11A04E0	KANAKA RAMYA PRATHIMA
23	14R11A04E1	KASTURI SHIVA SHANKER REDDY
24	14R11A04E2	KODHIRIPAKA DHENUSRI
25	14R11A04E3	KOLA AISHWARYA
26	14R11A04E4	KONDOJU AKSHITHA
27	14R11A04E5	KOUDAGANI ALEKHYA REDDY
28	14R11A04E6	KUMMARIKUNTA PRASHANTH
29	14R11A04E7	KURVA SAI KUMAR
30	14R11A04E8	M AJAY KRISHNA
31	14R11A04E9	M MRIDULA GAYATRI
32	14R11A04F0	MANGALAPALLI SRAVANTHI
33	14R11A04F1	MERUGU PALLAVI
34	14R11A04F2	MITHIN VARGHESE
35	14R11A04F3	MOHD EESA SOHAIL
36	14R11A04F4	MUCHUMARI HARSHA VARDHAN REDDY
37	14R11A04F5	MUNUGANTI PRADHYUMNA
38	14R11A04F6	N DURGA RAJU
39	14R11A04F7	N SAKETH
40	14R11A04F8	N SANDHYA
41	14R11A04F9	NALLAGONI SRAVANTHI
42	14R11A04G0	P MANMOHAN SHASHANK VARMA
43	14R11A04G1	PRABHALA SRUTHI
44	14R11A04G2	PRAYAGA VENKATA SATHYA KAMESWARA PAV
45	14R11A04G3	R SAILESH
46	14R11A04G4	SAMBANGI POOJA

47	14R11A04G5	SAMEENA
48	14R11A04G6	SANGOJI SAI CHANDU
49	14R11A04G7	SURANENI NAMRATHA
50	14R11A04G8	TADAKAPALLY VIVEK REDDY
51	14R11A04G9	THUMUKUMTA VAMSHI TEJA
52	14R11A04H0	TIRUNAGARI SRAVAN KUMAR
53	14R11A04H1	TRIPURARI SOWGANDHIKA
54	14R11A04H2	TUNIKI MADHULIKA REDDY
55	14R11A04H3	U SAI MANASWINI
56	14R11A04H4	VAIDYA KEERTHI MALINI
57	14R11A04H5	VANGETI PRAVALLIKA
58	14R11A04H6	VASIREDDY VENKATA SAI
59	14R11A04H7	VELDURTHY SAI KEERTHI
60	14R11A04H8	WILSON DAVIES
61	15R11A0418	KOTA RAJESH
62	15R11A0419	N MOUNIKA
63	15R11A0420	ARTHI SHARMA
64	15R11A0421	RAJPET SHIRISHA
65	15R11A0422	MALOTH RAMESH NAIK
66	15R11A0423	PAILLA PREM RAJ REDDY
<b>ECE – 2-1D</b>		
<b>S.No</b>	<b>Roll Number</b>	<b>Name of the Candidate</b>
1	14R11A04H9	A SHIRISHA
2	14R11A04J0	ABHIJEET KUMAR
3	14R11A04J1	ADULLA PRANAV REDDY
4	14R11A04J2	AINAPARTHI SAIVIJAYALAKSHMI SANDHYA
5	14R11A04J3	AMBATI SHIVA SAI
6	14R11A04J4	ANU PRASAD
7	14R11A04J5	B SAI APOORVA
8	14R11A04J6	B SRI KRISHNA SAI KIREETI
9	14R11A04J7	CHITTOJU LAKSHMI NARAYANAMMA
10	14R11A04J8	CHOWDARAPALLY SANTOSH KUMAR
11	14R11A04J9	D SAHITHI
12	14R11A04K0	DEVULAPALLI SAI CHAITANYA SANDEEP
13	14R11A04K1	DUSARI ANUSHA
14	14R11A04K2	GOLLAPUDI SRIKETH
15	14R11A04K3	GOLLIPALLY TEJASREE
16	14R11A04K4	GOUTE SHRAVAN KUMAR
17	14R11A04K5	GUDA PRATHYUSHA REDDY
18	14R11A04K6	JUNNU RAVALI
19	14R11A04K7	K DEVI PRIYANKA
20	14R11A04K8	KANDULA MANI
21	14R11A04K9	KARRA AVINASH

22	14R11A04L0	KASULA PRADEEP GOUD
23	14R11A04L1	KOMARAKUNTA SHASHANK
24	14R11A04L2	KOTHAKOTA PHANI RISHITHA
25	14R11A04L3	MADHADI NIKHIL KUMAR REDDY
26	14R11A04L4	MANDUMULA RAGHAVENDRA
27	14R11A04L5	MOHD HAMEED
28	14R11A04L6	MOHD SHAMS TABREZ
29	14R11A04L7	MORSU GANESH REDDY
30	14R11A04L8	MUKKERA VARUN
31	14R11A04L9	NAGULAPALLY MANOHAR REDDY
32	14R11A04M0	NAMBURI LAKSHMI MANJUSHA
33	14R11A04M1	NIROGI SURYA PRIYANKA
34	14R11A04M2	NUNE SAI CHAND
35	14R11A04M3	PALLETI SUSHMITHA
36	14R11A04M4	PANCHAYAT SHAMILI
37	14R11A04M5	POOSA JAI SAI NISHANTH
38	14R11A04M6	PRANAV RAJU A
39	14R11A04M7	RAYCHETTI CHANDRASENA
40	14R11A04M8	REBBA BHAVANI
41	14R11A04M9	S BHARATH SAGAR
42	14R11A04N0	S V M SURYA TEJASWINI
43	14R11A04N1	SAMA MANVITHA REDDY
44	14R11A04N2	SHAMALA MEGHANA
45	14R11A04N3	SMITHA KUMARI PATRO
46	14R11A04N4	T L SARADA RAMYA KAPARDHINI
47	14R11A04N5	T VINAY KUMAR
48	14R11A04N6	TABELA OMKAR
49	14R11A04N7	TADACHINA SAINATH REDDY
50	14R11A04N8	VANGA MOUNIKA
51	14R11A04N9	VARRI PRASHANTHI
52	14R11A04P0	VASARLA SAI TEJA
53	14R11A04P1	VISHWANATHAM ANUSHA
54	14R11A04P2	Y SRI SAI ADITYA
55	14R11A04P3	YAKKALA ASHIKA
56	14R11A04P4	YALAVARTHY MAHIMA
57	14R11A04P5	YALLAPRAGADA SAI TEJASRI
58	14R11A04P6	YARASI SAI RAMYA REDDY
59	14R11A04P7	S TARUN
60	15R15A0424	ARURI REJENDER
61	15R15A0425	KALALI BHAVANI
62	15R15A0426	JANUGANI SAI KRISHNA
63	15R15A0427	SATHENDER KUMAR YADAV
64	15R15A0428	KADEM PRAVEEN

65	15R15A0429	ARROJU AKHIL
66	15R15A0430	CH POOJA
67	15R18A0401	G SHREEHARSHA REDDY

## **26. Group-wise students list for discussion topic:**

ECE – 2-1A			
S.No	Roll Number	Name of the Candidate	Groups No.
1	14R11A0401	ADITYA B	G-1
2	14R11A0402	ADULLA JANARDHAN REDDY	
3	14R11A0403	ANDE HEMANTH REDDY	
4	14R11A0404	ANKATI NAVYA	
5	14R11A0405	ASHFAQ AZIZ AHMED	
6	14R11A0406	BANDI SANDHYA	G-2
7	14R11A0407	BASWARAJ SHASHANK YADAV	
8	14R11A0408	BITLA SRIKANTH REDDY	
9	14R11A0409	BUDDANA DHARANI KUMAR	
10	14R11A0410	CHEBARTHI RAMYA GAYATHRI	
11	14R11A0411	CHETLAPALLI NAGA SAI SUSHMITHA	G-3
12	14R11A0412	DASARI DHAMODHAR REDDY	
13	14R11A0413	G AYESHA SULTANA	
14	14R11A0414	G MADHURI	
15	14R11A0415	G RISHI RAJ	
16	14R11A0416	G VAMSHI KRISHNA	G-4
17	14R11A0417	G VENKATESH YADAV	
18	14R11A0418	GONDA RISHIKA	
19	14R11A0419	GUDE GOPI	
20	14R11A0420	JAGGANNAGARI MANOJKUMAR REDDY	
21	14R11A0421	JAGGARI SRINIJA REDDY	G-5
22	14R11A0422	JALAGAM NANDITHA	
23	14R11A0423	JAMMIKUNTALA SHIVA CHARAN	
24	14R11A0424	JATAPROLU LAKSHMI SOWMIKA	
25	14R11A0425	JEKSANI SHREYA	
26	14R11A0426	K VIJAY KUMAR	G-6
27	14R11A0427	KAALISSETTY KRISHNA CHAITANYA	
28	14R11A0428	KAKARLA MOUNICA	
29	14R11A0429	KARRE PRIYANKA	
30	14R11A0430	KL N SATYANARAYANA MURTHY	
31	14R11A0431	KONDA KRITISH KUMAR	G-7
32	14R11A0432	KOPPULA RAHUL	
33	14R11A0433	KURUGANTI RUNI TANISHKA SHARMA	
34	14R11A0434	L THRILOK	
35	14R11A0435	MANDULA SANTOSHINI	

36	14R11A0436	MATLA PRINCE TITUS	G-8
37	14R11A0437	NARSETTI SAIPRAVALIKA	
38	14R11A0438	NIKITHA RAGI	
39	14R11A0439	P VIJAYA ADITYA VARMA	
40	14R11A0440	PASHAM VIKRAM REDDY	
41	14R11A0441	PELLURI KARAN KUMAR	G-9
42	14R11A0442	PERURI CHANDANA	
43	14R11A0443	PODUGU SRUJANA DEVI	
44	14R11A0444	RAJNISH KUMAR	
45	14R11A0445	RAJU PAVANA KUMARI	
46	14R11A0446	RAMIDI NITHYA	G-10
47	14R11A0447	RAMOJI RAJESH	
48	14R11A0448	S ALEKHYA	
49	14R11A0449	SARANGA SAI KIRAN	
50	14R11A0450	SHAIK SAMEER ALI	
51	14R11A0451	SOUMYA MISHRA	G-11
52	14R11A0452	SRIRAMOJU MANASA	
53	14R11A0453	T ARUN KUMAR	
54	14R11A0454	T S SANTHOSH KUMAR	
55	14R11A0455	V BAL RAJ	
56	14R11A0456	V POOJA	G-12
57	14R11A0457	V SRIVATS VISHWAMBER	
58	14R11A0458	VEMI REDDY VISHNU VARDHAN REDDY	
59	14R11A0459	VENNAMANENI VAMSI KRISHNA	
60	14R11A0460	YERASI TEJASRI	
61	15R15A0401	RAMIDI SANDEEP REDDY	G-13
62	15R15A0402	ODDARAPU HARISHBABU	
63	15R15A0403	KOLUKURI BHARGAVI	
64	15R15A0404	ADEPU MOUNIKA	
65	15R15A0405	AVANCHA PRAVALIKA	G-14
66	15R15A0406	NELLUTLA VISHAL CHAITANYA	
67	15R15A0407	VEMUNA JAMEENA	
ECE -2-1B			
S.No	Roll Number	Name of the Candidate	Groups No.
1	14R11A0461	ADDAKULA SURESH	G-1
2	14R11A0462	AGARTI MADHU VIVEKA	
3	14R11A0463	AKULA SAI KIRAN	
4	14R11A0464	ANUMULA SNIGDHA	
5	14R11A0465	B DIVYA	
6	14R11A0466	B MANOHAR	G-2
7	14R11A0467	BANDARI MAMATHA	
8	14R11A0468	BINGI DIVYA SUDHA RANI	
9	14R11A0469	BIRE BHAVYA	

10	14R11A0470	CH SAI BHARGAVI	
11	14R11A0471	CHAVALI SUMA SIREESHA	
12	14R11A0472	CHELLABOINA SHIVA KUMAR	
13	14R11A0473	CHETTY AKHIL CHAND	
14	14R11A0474	CHINTAPALLI MADHAV REDDY	
15	14R11A0475	CHIVUKULA VENKATA SUBRAMANYA PRASANTH	G-3
16	14R11A0476	D NAGA SUMANVITHA	
17	14R11A0477	D VAMSI	
18	14R11A0478	DHARMENDER KEERTHI	G-4
19	14R11A0479	EADARA NAGA SIRISHA	
20	14R11A0480	ERANKI SAI UDAYASRI ALAKANANDA	
21	14R11A0481	GANGA STEPHEN RAVI KUMAR	
22	14R11A0482	GUNDAM SHRUTHI	
23	14R11A0483	GUNDREVULA SAMEERA	G-5
24	14R11A0484	K NAGA REKHA	
25	14R11A0485	KANDADI VARSHA	
26	14R11A0486	KURELLI SAI VINEETH KUMAR GOUD	
27	14R11A0487	MADDIKUNTA SOMA SHEKAR REDDY	
28	14R11A0488	MAMILLA SAI NISHMA	G-6
29	14R11A0489	MARELLA NAGA LASYA PRIYA	
30	14R11A0490	MARKU VENKATESH	
31	14R11A0491	MOHAMED KHALEEL	
32	14R11A0492	MOHAMMED WASEEM AKRAM	
33	14R11A0493	MOTURI DIVYA	G-7
34	14R11A0494	MUDIUM KOUSHIKA	
35	14R11A0495	MYLAPALLI RAMBABU	
36	14R11A0496	NAGU MOUNIKA	
37	14R11A0497	NEELAM SNEHANJALI	
38	14R11A0498	NIDAMANURI VENKATA VAMSI KRISHNA	G-8
39	14R11A0499	NIKHIL KUMAR N	
40	14R11A04A0	ORUGANTI HARSHINI	
41	14R11A04A1	PARAMKUSAM NIHARIKA	
42	14R11A04A2	PASAM ABHIGNA	
43	14R11A04A3	PATI VANDANA	G-9
44	14R11A04A4	PODISHETTY MANOGNA	
45	14R11A04A5	PONAKA SREEVARDHAN REDDY	
46	14R11A04A6	R NAVSHETHA	
47	14R11A04A7	R PRANAY KUMAR	
48	14R11A04A8	RAMIDI ROJA	G-10
49	14R11A04A9	RUDRA VAMSHI	
50	14R11A04B0	S SHARAD KUMAR	
51	14R11A04B1	SAGGU SOWMYA	
52	14R11A04B2	TADELA SARWANI	G-11

53	14R11A04B3	THOTA SAI BHUVAN	
54	14R11A04B4	VALLAPU HARIKRISHNA	
55	14R11A04B5	VECHA PAVAN KUMAR	
56	14R11A04B6	Y SAI VISHWANATH	
57	15R15A0408	ERUKALA NIKITHA	G-12
58	15R15A0409	PUNGANUR JAYACHANDRA BHARATHWAJ	
59	15R15A0410	GALIPALLY BHARGAVA	
60	15R15A0411	PADMA ARUNRAJ	
61	15R15A0412	JAMALAPURAM NAVEEN	G-13
62	15R15A0413	MACHANNI BALAKRISHNA YADAV	
63	15R15A0414	ANABOINA MAHENDER	
64	15R15A0415	ANABOINA SHIVA SAI	
65	15R15A0416	VEMULA VINITHA	
66	15R15A0417	CHEVU NAGESH	
ECE – 2-1C			
S.No	Roll Number	Name of the Candidate	Groups No.
1	14R11A04B9	ANAMALI REETHIKA	G-1
2	14R11A04C0	ARUMILLI LEKYA	
3	14R11A04C1	ARUMUGAM ASHWINI	
4	14R11A04C2	BASAVARAJU MEGHANA	
5	14R11A04C3	BEERAM TEJASRI REDDY	
6	14R11A04C4	BHARAT SAKETH	G-2
7	14R11A04C5	BOMMANA HARIKADEVI	
8	14R11A04C6	BYRAGONI ROJA	
9	14R11A04C7	CANDHI SHASHI REKHA	
10	14R11A04C8	CH RENUKA	
11	14R11A04C9	CHAGANTI MOUNICA	G-3
12	14R11A04D0	CHITTARLA LOKESH GOUD	
13	14R11A04D1	D LAVANYA	
14	14R11A04D2	D MANIKANTA	
15	14R11A04D3	DASARI VENKATA NAGA SAISH	
16	14R11A04D4	DODDA MANOJ	G-4
17	14R11A04D5	E RAHUL CHOWDHARY	
18	14R11A04D6	GOWRISHETTY VINEETHA	
19	14R11A04D7	GUNTUPALLI RAVI TEJA	
20	14R11A04D8	K L ANUSHA	
21	14R11A04D9	K SASIDHAR	G-5
22	14R11A04E0	KANAKA RAMYA PRATHIMA	
23	14R11A04E1	KASTURI SHIVA SHANKER REDDY	
24	14R11A04E2	KODHIRIPAKA DHENUSRI	
25	14R11A04E3	KOLA AISHWARYA	
26	14R11A04E4	KONDOJU AKSHITHA	G-6
27	14R11A04E5	KOUDAGANI ALEKHYA REDDY	



28	14R11A04E6	KUMMARIKUNTA PRASHANTH	
29	14R11A04E7	KURVA SAI KUMAR	
30	14R11A04E8	M AJAY KRISHNA	
31	14R11A04E9	M MRIDULA GAYATRI	
32	14R11A04F0	MANGALAPALLI SRAVANTHI	G-7
33	14R11A04F1	MERUGU PALLAVI	
34	14R11A04F2	MITHIN VARGHESE	
35	14R11A04F3	MOHD EESA SOHAIL	
36	14R11A04F4	MUCHUMARI HARSHA VARDHAN REDDY	G-8
37	14R11A04F5	MUNUGANTI PRADHYUMNA	
38	14R11A04F6	N DURGA RAJU	
39	14R11A04F7	N SAKETH	
40	14R11A04F8	N SANDHYA	
41	14R11A04F9	NALLAGONI SRAVANTHI	
42	14R11A04G0	P MANMOHAN SHASHANK VARMA	
43	14R11A04G1	PRABHALA SRUTHI	
44	14R11A04G2	PRAYAGA VENKATA SATHYA KAMESWARA PAV	G-9
45	14R11A04G3	R SAILESH	
46	14R11A04G4	SAMBANGI POOJA	
47	14R11A04G5	SAMEENA	
48	14R11A04G6	SANGOJI SAI CHANDU	G-10
49	14R11A04G7	SURANENI NAMRATHA	
50	14R11A04G8	TADAKAPALLY VIVEK REDDY	
51	14R11A04G9	THUMUKUMTA VAMSHI TEJA	
52	14R11A04H0	TIRUNAGARI SRAVAN KUMAR	
53	14R11A04H1	TRIPURARI SOWGANDHIKA	
54	14R11A04H2	TUNIKI MADHULIKA REDDY	
55	14R11A04H3	U SAI MANASWINI	
56	14R11A04H4	VAIDYA KEERTHI MALINI	G-12
57	14R11A04H5	VANGETI PRAVALLIKA	
58	14R11A04H6	VASIREDDY VENKATA SAI	
59	14R11A04H7	VELDURTHY SAI KEERTHI	
60	14R11A04H8	WILSON DAVIES	
61	15R11A0418	KOTA RAJESH	
62	15R11A0419	N MOUNIKA	
63	15R11A0420	ARTHI SHARMA	
64	15R11A0421	RAJPET SHIRISHA	G-13
65	15R11A0422	MALOTH RAMESH NAIK	
66	15R11A0423	PAILLA PREM RAJ REDDY	
ECE – 2-1D			
S.No	Roll Number	Name of the Candidate	Groups No.
1	14R11A04H9	A SHIRISHA	G-1
2	14R11A04J0	ABHIJEET KUMAR	

3	14R11A04J1	ADULLA PRANAV REDDY	
4	14R11A04J2	AINAPARTHI SAI VIJAYALAKSHMI SANDHYA	
5	14R11A04J3	AMBATI SHIVA SAI	
6	14R11A04J4	ANU PRASAD	G-2
7	14R11A04J5	B SAI APOORVA	
8	14R11A04J6	B SRI KRISHNA SAI KIREETI	
9	14R11A04J7	CHITTOJU LAKSHMI NARAYANAMMA	
10	14R11A04J8	CHOWDARAPALLY SANTOSH KUMAR	
11	14R11A04J9	D SAHITHI	G-3
12	14R11A04K0	DEVULAPALLI SAI CHAITANYA SANDEEP	
13	14R11A04K1	DUSARI ANUSHA	
14	14R11A04K2	GOLLAPUDI SRIKETH	
15	14R11A04K3	GOLLIPALLY TEJASREE	
16	14R11A04K4	GOUTE SHRAVAN KUMAR	G-4
17	14R11A04K5	GUDA PRATHYUSHA REDDY	
18	14R11A04K6	JUNNU RAVALI	
19	14R11A04K7	K DEVI PRIYANKA	
20	14R11A04K8	KANDULA MANI	
21	14R11A04K9	KARRA AVINASH	G-5
22	14R11A04L0	KASULA PRADEEP GOUD	
23	14R11A04L1	KOMARAKUNTA SHASHANK	
24	14R11A04L2	KOTHAKOTA PHANI RISHITHA	
25	14R11A04L3	MADHADI NIKHIL KUMAR REDDY	
26	14R11A04L4	MANDUMULA RAGHAVENDRA	G-6
27	14R11A04L5	MOHD HAMEED	
28	14R11A04L6	MOHD SHAMS TABREZ	
29	14R11A04L7	MORSU GANESH REDDY	
30	14R11A04L8	MUKKERA VARUN	
31	14R11A04L9	NAGULAPALLY MANOHAR REDDY	G-7
32	14R11A04M0	NAMBURI LAKSHMI MANJUSHA	
33	14R11A04M1	NIROGI SURYA PRIYANKA	
34	14R11A04M2	NUNE SAI CHAND	
35	14R11A04M3	PALLETI SUSHMITHA	
36	14R11A04M4	PANCHAYAT SHAMILI	G-8
37	14R11A04M5	POOSA JAI SAI NISHANTH	
38	14R11A04M6	PRANAV RAJU A	
39	14R11A04M7	RAYCHETTI CHANDRASENA	
40	14R11A04M8	REBBA BHAVANI	
41	14R11A04M9	S BHARATH SAGAR	G-9
42	14R11A04N0	S V M SURYA TEJASWINI	
43	14R11A04N1	SAMA MANVITHA REDDY	
44	14R11A04N2	SHAMALA MEGHANA	
45	14R11A04N3	SMITHA KUMARI PATRO	

46	14R11A04N4	T L SARADA RAMYA KAPARDHINI	G-10
47	14R11A04N5	T VINAY KUMAR	
48	14R11A04N6	TABELA OMKAR	
49	14R11A04N7	TADACHINA SAINATH REDDY	
50	14R11A04N8	VANGA MOUNIKA	
51	14R11A04N9	VARRI PRASHANTHI	G-11
52	14R11A04P0	VASARLA SAI TEJA	
53	14R11A04P1	VISHWANATHAM ANUSHA	
54	14R11A04P2	Y SRI SAI ADITYA	
55	14R11A04P3	YAKKALA ASHIKA	
56	14R11A04P4	YALAVARTHY MAHIMA	G-12
57	14R11A04P5	YALLAPRAGADA SAI TEJASRI	
58	14R11A04P6	YARASI SAI RAMYA REDDY	
59	14R11A04P7	S TARUN	
60	15R15A0424	ARURI REJENDER	
61	15R15A0425	KALALI BHAVANI	G-13
62	15R15A0426	JANUGANI SAI KRISHNA	
63	15R15A0427	SATHENDER KUMAR YADAV	
64	15R15A0428	KADEM PRAVEEN	
65	15R15A0429	ARROJU AKHIL	
66	15R15A0430	CH POOJA	G-14
67	15R18A0401	G SHREEHARSHA REDDY	