

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

## Lecture 01: INTRODUCTION

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Main Objectives of the Course

### Hardware Modeling Using Verilog

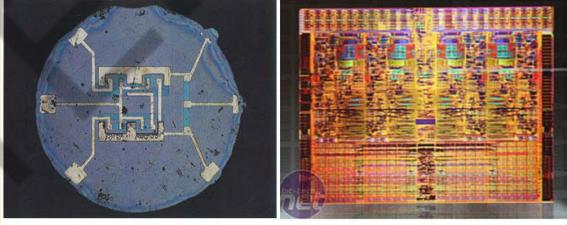
1. Learn about the Verilog hardware description language.
2. Understand the difference between behavioral and structural design styles.
3. Learn to write test benches and analyze simulation results.
4. Learn to model combinational and sequential circuits.
5. Distinguish between good and bad coding practices.
6. Case studies with some complex designs.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 2

## VLSI Design Process

- Design complexity increasing rapidly
  - Increased size and complexity
  - Fabrication technology improving
  - CAD tools are essential
  - Conflicting requirements like area, speed, and energy consumption
- The present trend
  - Standardize the design flow
  - Emphasis on low-power design, and increased performance

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 3

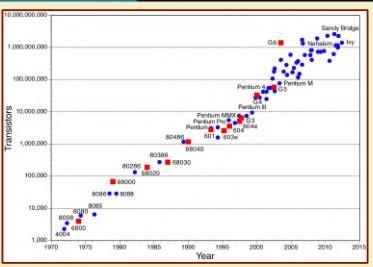


First Planar IC (1961) and Intel Nehalem Quad Core Die

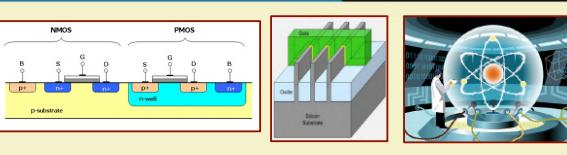
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 4

## Moore's Law

- Exponential growth
- Design complexity increases rapidly
- Automated tools are essential
- Must follow well-defined design flow



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 5



NMOS                          PMOS

CMOS (up to 22nm)

FinFET (14nm)

QUANTUM?

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 6

## VLSI Design Flow

- Standardized design procedure
  - Starting from the design idea down to the actual implementation.
- Encompasses many steps:
  - Specification
  - Synthesis
  - Simulation
  - Layout
  - Testability analysis
  - and many more .....

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 7

- Need to use Computer Aided Design (CAD) tools.
  - Based on Hardware Description Language (HDL).
  - HDLs provide formats for representing the outputs of various design steps.
  - A CAD tool transforms its HDL input into a HDL output that contains more detailed information about the hardware.
    - Behavioral level to register transfer level
    - Register transfer level to gate level
    - Gate level to transistor level
    - Transistor level to the layout level

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 8

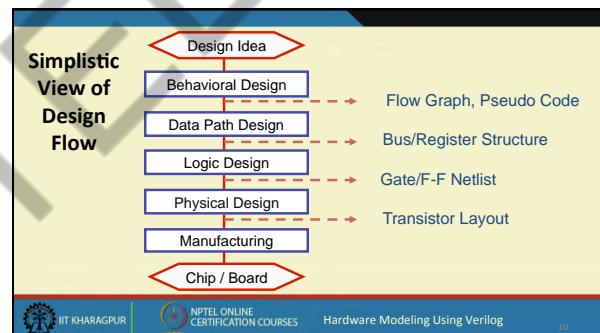
## Two Competing HDLs

- Verilog
- VHDL

*Designs are created typically using HDLs, which get transformed from one level of abstraction to the next as the design flow progresses.*

There are other HDLs like SystemC, SystemVerilog, and many more ...

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 9



## Steps in the Design Flow

- Behavioral design
  - Specify the functionality of the design in terms of its *behavior*.
  - Various ways of specifying:
    - Boolean expression or truth table.
    - Finite-state machine behavior (e.g. state transition diagram or table).
    - In the form of a high-level algorithm.
  - Needs to be synthesized into more detailed specifications for hardware realization.

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 11

- Data path design
  - Generate a netlist of register transfer level components, like registers, adders, multiplexers, decoders, etc.
  - A **netlist** is a directed graph, where the vertices indicate components, and the edges indicate interconnections.
  - A netlist specification is also referred to as **structural design**.
    - Netlist may be specified at various levels, where the components may be functional modules, gates or transistors.
    - Systematically transformed from one level to the next.

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 12

- Logic design
  - Generate a netlist of gates/flip-flops or standard cells.
  - A standard cell is a pre-designed circuit module (like gates, flip-flops, multiplexer, etc.) at the layout level.
  - Various logic optimization techniques are used to obtain a cost effective design.
  - There may be conflicting requirements during optimization:
    - Minimize number of gates.
    - Minimize number of gate levels (i.e. delay).
    - Minimize signal transition activities (i.e. dynamic power).

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 13

- Physical design and Manufacturing
  - Generate the final layout that can be sent for fabrication.
  - The layout contains a large number of regular geometric shapes corresponding to the different fabrication layers.
  - Alternatively, the final target may be Field Programmable Gate Array (FPGA), where technology mapping from the gate level netlist is used.
    - Can be programmed in-field.
    - Much greater flexibility, but less speed.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 14

## Other Steps in the Design Flow

- Simulation for verification
  - At various levels: logic level, switch level, circuit level
- Formal verification
  - Used to verify the designs through formal techniques
- Testability analysis and Test pattern generation
  - Required for testing the manufactured devices

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 15

## END OF LECTURE 01

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 16

## Lecture 02: DESIGN REPRESENTATION

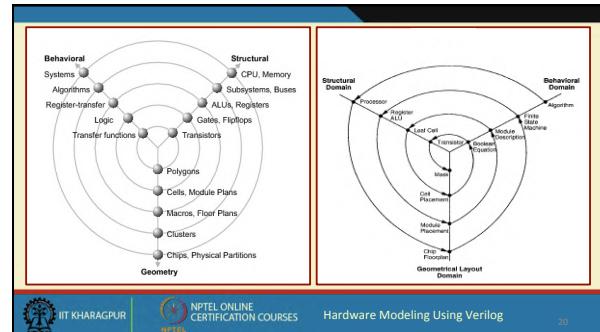
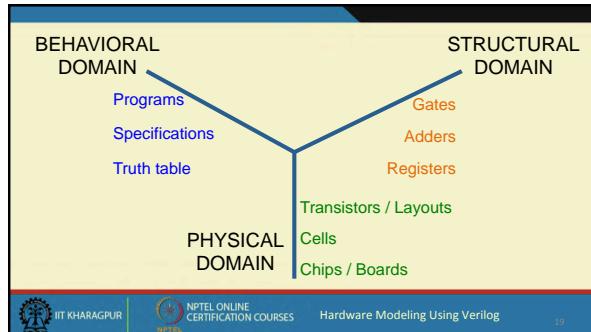
PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

## Design Representation

- A design can be represented at various levels from three different points of view:
  1. Behavioral
  2. Structural
  3. Physical
- Can be conveniently expressed by [Y-diagram](#).

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 18



## Behavioral Representation

- Specifies how a particular design should respond to a given set of inputs.
- May be specified by:
  - Boolean equations
  - Tables of input and output values
  - Algorithms written in standard HLL like C
  - Algorithms written in special HDL like Verilog or VHDL

## Behavioral Representation :: Example

**Full Adder:**

- two operand inputs A and B
- a carry input C
- a carry output Cy
- a sum output S

Express in terms of Boolean expressions:

$$S = A \cdot B' \cdot C' + A' \cdot B' \cdot C + A' \cdot B \cdot C' + A \cdot B \cdot C = A \oplus B \oplus C$$

$$Cy = A \cdot B + A \cdot C + B \cdot C$$

- Express in Verilog in terms of Boolean expressions

```
module carry (S, Cy, A, B, C);
  input A, B, C;
  output S, Cy;
  assign S = A ^ B ^ C;
  assign Cy = (A & B) | (B & C) | (C & A);
endmodule
```

- Express in Verilog in terms of truth table (only Cy is shown)

```
primitive carry (Cy, A, B, C);
  input A, B, C;
  output Cy;
  table
    // A   B   C   Cy
    1   1   ?   :   1 ;
    1   ?   1   :   1 ;
    ?   1   1   :   1 ;
    0   0   ?   :   0 ;
    0   ?   0   :   0 ;
    ?   0   0   :   0 ;
  endtable
endprimitive
```

## Structural Representation

- Specifies how components are interconnected.
- In general, the description is a list of modules and their interconnection.
  - Called *netlist*.
  - Can be specified at various levels.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

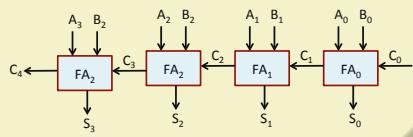
- At the structural level, the levels of abstraction are:
  - The module (functional) level
  - The gate level
  - The transistor level
  - Any combination of above
- In each successive level more detail is revealed about the implementation.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

26

## Example: A 4-bit Ripple Carry Adder

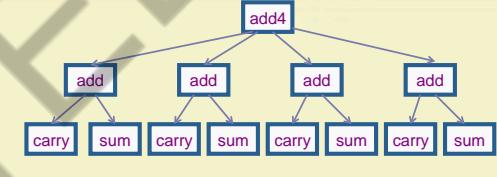


- Consists of four full adders.
- Each full adder consists of a sum circuit and a carry circuit.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27



$$\text{carry} = A \cdot B + B \cdot C + C \cdot A$$

$$\text{sum} = A \oplus B \oplus C$$

- We instantiate carry and sum circuits to create a full adder.
- We instantiate four full adders to create the 4-bit adder.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

```
module add4 (s, cy_out, sum, a, b, cy_in);
  input a, b, cy_in;
  output sum, cy_out;
  output [3:0] s;
  output cy4;
  wire [2:0] cy_out;
  add B0 (cy_out[0], s[0], x[0], y[0], ci);
  add B1 (cy_out[1], s[1], x[1], y[1], cy_out[0]);
  add B2 (cy_out[2], s[2], x[2], y[2], cy_out[1]);
  add B3 (cy4, s[3], x[3], y[3], cy_out[2]);
endmodule
```

```
module add (s, cy_out, sum, a, b, cy_in);
  input a, b, cy_in;
  output sum, cy_out;
  output s1 (sum, a, b, cy_in);
  carry c1 (cy_out, a, b, cy_in);
endmodule
```

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

29

```
module sum (sum, a, b, cy_in);
  input a, b, cy_in;
  output sum;
  wire t;
  xor x1 (t, a, b);
  xor x2 (sum, t, cy_in);
endmodule
```

```
module carry (cy_out, a, b, cy_in);
  input a, b, cy_in;
  output cy_out;
  wire t1, t2, t3;
  and g1 (t1, a, b);
  and g2 (t2, a, c);
  and g3 (t3, b, c);
  or g4 (cy_out, t1, t2, t3);
endmodule
```

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

30

## Physical Representation

- The lowest level of physical specification.
  - Photo-mask information required by the various processing steps in the fabrication process.
- At the module level, the physical layout for the 4-bit adder may be defined by a rectangle or polygon, and a collection of ports.
- At the layout level, there can be a large number of rectangles or polygons.



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

31

- Partial physical description for 4-bit adder in Verilog

```
module add4;
  input x[3:0], y[3:0], cy_in;
  output s[3:0], cy4;
  boundary [0, 0, 130, 500];
  port x[0] aluminum width = 1 origin = [0, 35];
  port y[0] aluminum width = 1 origin = [0, 85];
  port cy_in polysilicon width = 2 origin = [70, 0];
  port s[0] aluminum width = 1 origin = [120, 65];

  add a0 origin = [0, 0];
  add a1 origin = [0, 120];
endmodule
```



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

32

## Digital IC Design Flow: A quick look



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

33

## END OF LECTURE 02



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

34

## Lecture 03: GETTING STARTED WITH VERILOG

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

## Why do we use Verilog?

- To describe a digital system as a set of *modules*.
  - Each of the modules will have an interface to other modules, in addition to its description.
  - Two ways to specify a module:
    - By specifying its internal logical structure (called *structural representation*).
    - By describing its behavior in a program-like manner (called *behavioral representation*).
  - The modules are interconnected using *nets*, which allow them to work with each other.



IIT Kharagpur  
NPTEL

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

## What next?

- After specifying the system in Verilog, we can do two things:
  - Simulate the system and verify the operation.
    - Just like running a program written in some high-level language.
    - Requires a *test bench* or *test harness*, that specifies the inputs that are to be applied and the way the outputs are to be displayed.
  - Use a synthesis tool to map it to hardware.
    - Converts it to a netlist of low-level primitives.
    - The hardware can be *Application Specific Integrated Circuit* (ASIC).
    - Or else, it can be *Field Programmable Gate Array* (FPGA).

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

37

- When the design is mapped to hardware, we do not need test bench for simulation any more.
- Signals can be actually applied from some source (e.g. signal generator), and response evaluated by some equipment (e.g. oscilloscope or logic analyzer).

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

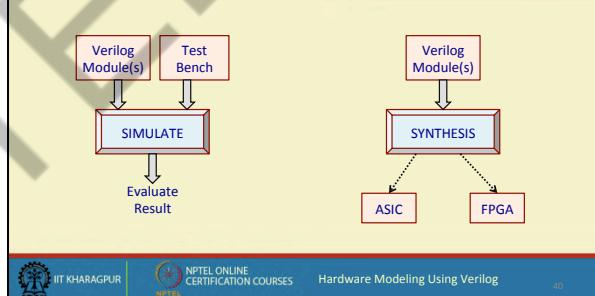
38

- Using ASIC as hardware target?
  - When high performance and high packing density is required.
  - When the manufactured hardware is expected to be used in large numbers (e.g. processor chips).
- Using FPGA as hardware target?
  - When fast turnaround time is required to validate the design.
  - The mapping can be done in the laboratory itself with a FPGA kit and associated software.
  - There is a tradeoff in performance.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

39

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

40

## How to Simulate?

- Free software available:
  - Icarus Verilog (<http://verilog.icarus.com>)
  - GTKWave (<http://gtkwave.sourceforge.net>)
- Commercial software (with free versions available):
  - From Xilinx (ISE, Vivado)
  - Many more.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41

## How to Synthesize?

- For FPGA as target, specific software is required:
  - Xilinx ISE or Vivado for Xilinx FPGA kits.
  - Similar software available from other FPGA vendors.
- For ASIC as target, commercial CAD tools exist:
  - Tool suite from Cadence.
  - Tool suite from Synopsys.
  - Several others ...

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

## Scope of this Course

- We shall be discussing features of the Verilog language, and verifying the design through simulation.
  - How to design combinational and sequential digital circuits?
  - How to verify the functionality through simulation?
- We shall not be discussing the synthesis tools; however:
  - Shall discuss various tricks that help in synthesis.
  - Shall discuss the common design flow – first code the modules in behavioral design style, and then translate selected subset of modules to structural design style.



IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

43

## How to Simulate Verilog Module(s)

- Using a test bench to verify the functionality of a design coded in Verilog (called Design-under-Test or DUT), comprising of:
  - A set of stimulus for the DUT.
  - A monitor, which captures or analyzes the outputs of the DUT.
- Requirement:
  - The inputs of the DUT need to be connected to the test bench.
  - The outputs of the DUT needs also to be connected to the test bench.



IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

44

## TEST BENCH

Stimulus → Design Under Test (DUT) → Monitor



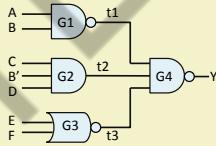
IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

45

## An Example



```
module example (A,B,C,D,E,F,Y);
  input A,B,C,D,E,F;
  output Y;
  wire t1, t2, t3, Y;
  nand #1 G1 (t1,A,B);
  and #2 G2 (t2,C,-B,D);
  nor #1 G3 (t3,E,F);
  nand #1 G4 (Y,t1,t2,t3);
endmodule
```

We can combine declarations of same type of gate together:

```
nand #1 G1 (t1,A,B),
      G4 (Y,t1,t2,t3);
```



IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

46

```
module testbench;
  reg A,B,C,D,E,F; wire Y;
  example DUT(A,B,C,D,E,F,Y);
begin
  $monitor ($time, " A=%b, B=%b, C=%b,
            D=%b, E=%b, F=%b, Y=%b",
            A,B,C,D,E,F,Y);
  #5 A=1; B=0; C=0; D=1; E=0; F=0;
  #5 A=0; B=0; C=1; D=1; E=0; F=0;
  #5 A=1; C=0;
  #5 F=1;
  #5 $finish;
end
endmodule
```

```
example.v
module example
  (A,B,C,D,E,F,Y);
  wire t1, t2, t3, Y;
  nand #1 G1 (t1,A,B);
  and #2 G2 (t2,C,-B,D);
  nor #1 G3 (t3,E,F);
  nand #1 G4 (Y,t1,t2,t3);
endmodule
```

example-test.v



IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

47

### Simulation results:

```
0 A=x, B=x, C=x, D=x, E=x, F=x, Y=x
5 A=1, B=0, C=0, D=1, E=0, F=0, Y=x
8 A=1, B=0, C=0, D=1, E=0, F=0, Y=1
10 A=0, B=0, C=1, D=1, E=0, F=0, Y=1
13 A=0, B=0, C=1, D=1, E=0, F=0, Y=0
15 A=1, B=0, C=0, D=1, E=0, F=0, Y=0
18 A=1, B=0, C=0, D=1, E=0, F=0, Y=1
20 A=1, B=0, C=0, D=1, E=0, F=1, Y=1
```

### Command in Verilog:

a) iverilog -o mysim example.v example-test.v

b) vvp mysim



IIT Kharagpur  
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

48

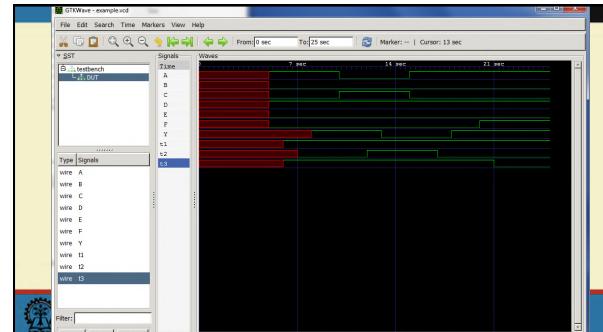
To display the waveforms

Run the command:  
gtkwave example.vcd

```
module testbench;
reg A,B,C,D,E,F; wire Y;
example DUT(A,B,C,D,E,F,Y);

initial
begin
$dumpfile ("example.vcd");
$dumpvars(0,testbench);
$monitor ($time," A=%b, B=%b, C=%b,
D=%b, E=%b, F=%b, Y=%b",
A,B,C,D,E,F,Y);
#5 A=1; B=0; C=0; D=1; E=0; F=0;
#5 A=0; B=0; C=1; D=1; E=0; F=0;
#5 A=1; C=0;
#5 F=1;
#5 $finish;
end
endmodule
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



## END OF LECTURE 03

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

## Lecture 04: VLSI DESIGN STYLES (PART 1)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

### VLSI Design Cycle

- Large number of devices
- Optimization requirements for high performance
- Time-to-market competition
- Cost

System Specifications

Manual Automation

Chip

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

### VLSI Design Cycle (contd.)

1. System specification
2. Functional design
3. Logic design
4. Circuit design
5. Physical design
6. Design verification
7. Fabrication
8. Packaging, testing, and debugging

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

## Physical Design

- Converts a circuit description into a geometric description.
  - This description is used for fabrication of the chip.
- Basic steps in the physical design cycle:
  - Partitioning, floorplanning and placement
  - Routing
  - Static timing analysis
  - Signal integrity and crosstalk analysis
  - Physical verification and signoff

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

55

## Various Design Styles

- Programmable Logic Devices
  - Field Programmable Gate Array (FPGA)
  - Gate Array
- Standard Cell (Semi-Custom Design)
- Full-Custom Design

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

56

## Which Design Style to Use?

- Basically a tradeoff among several design parameters.
  - Hardware cost
  - Circuit delay
  - Time required
- Optimizing on these parameters is often conflicting.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

57

## Field Programmable Gate Array (FPGA)

NPTEL ONLINE  
CERTIFICATION COURSES

## What does FPGA offer?

- User / Field Programmability.
  - Array of logic cells connected via routing channels.
  - Different types of cells:
    - Special I/O cells.
    - Logic cells (Mainly lookup tables (LUT) with associated registers).
  - Interconnection between cells:
    - Using SRAM based switches.
    - Using anti-fuse elements.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

59

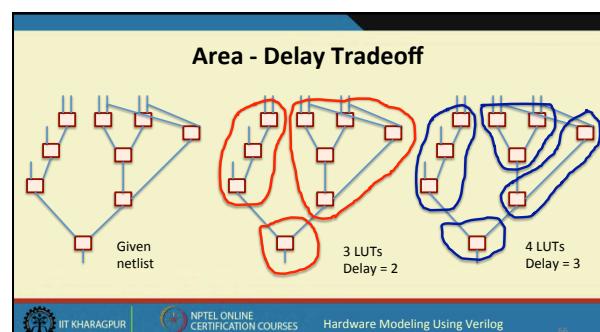
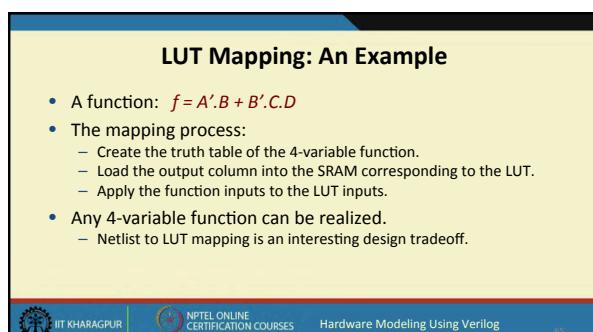
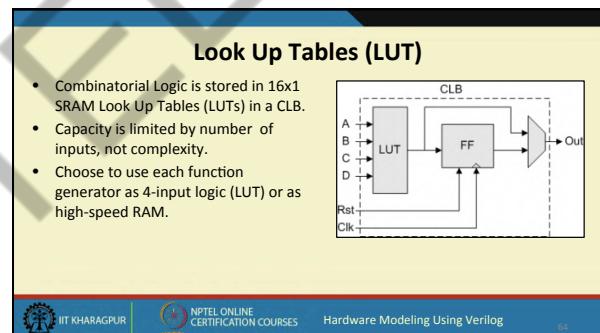
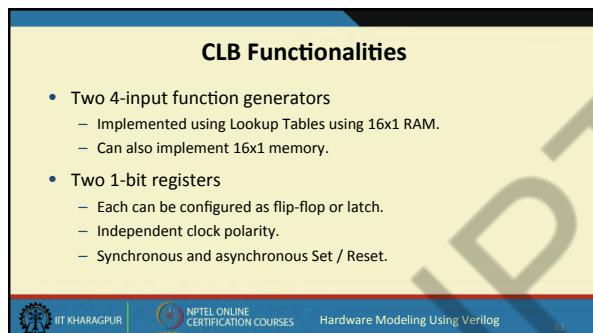
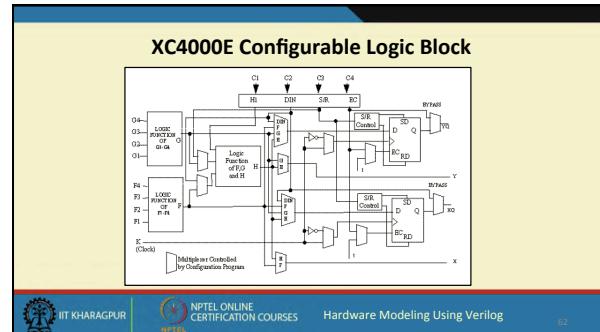
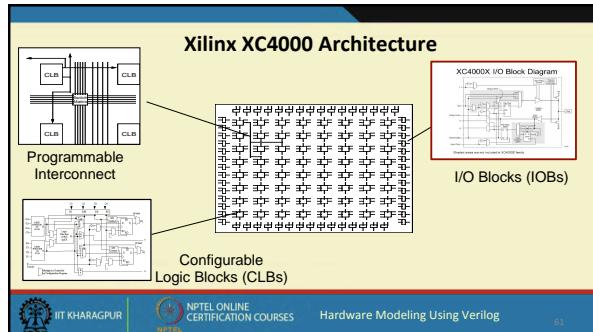
## Ease of Use

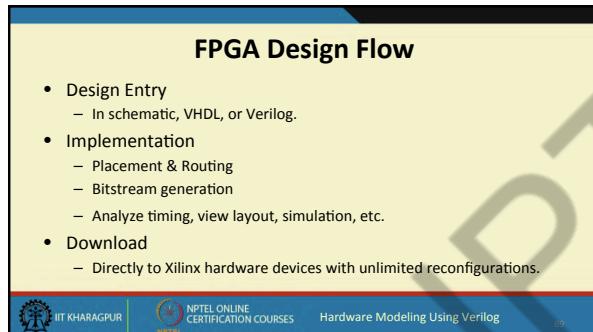
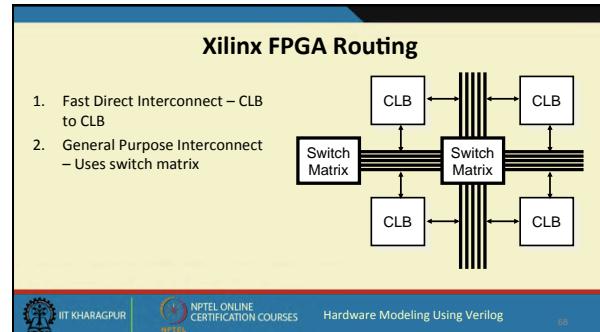
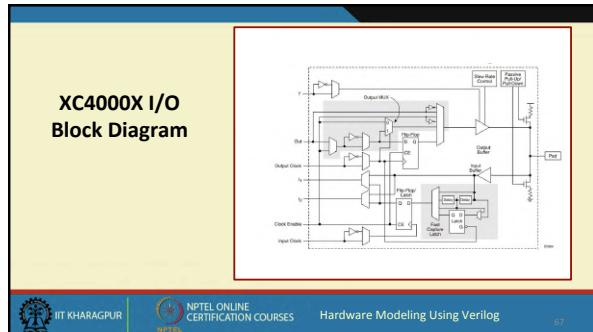
- FPGA chips are manufactured by a number of vendors:
  - Xilinx, Altera, Actel, etc.
  - Products vary widely in capability.
- FPGA development boards and CAD software available from many sellers.
  - Allows rapid prototyping in laboratory.

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

60





**END OF LECTURE 04**

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 70

**Lecture 05: VLSI DESIGN STYLES (PART 2)**

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

**Gate Array**

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

## Introduction

- In view of the speed of prototyping capability, the gate array (GA) comes after the FPGA.
- Design implementation of
  - FPGA chip is done with user programming,
  - Gate array is done with metal mask design and processing.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

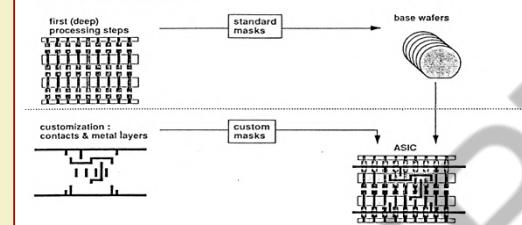
- Gate array implementation requires a two-step manufacturing process:
  - a) The first phase, which is based on generic (standard) masks, results in an array of uncommitted transistors on each GA chip.
  - b) These uncommitted chips can be customized later, which is completed by defining the metal interconnects between the transistors of the array.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74

### two-step manufacture :

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

75

- The GA chip utilization factor is higher than that of FPGA.
  - The used chip area divided by the total chip area.
- Chip speed is also higher.
  - More customized design can be achieved with metal mask designs.
- Typical gate array chips can implement millions of logic gates.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

76

## Standard Cell Based Design

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

77

## Introduction

- One of the most prevalent design styles.
  - Also called semi-custom design style.
  - Requires developing full custom mask set.
- Basic idea:
  - Commonly used logic cells are developed, and stored in a standard cell library.
  - Typical library may contain a few hundred cells (*Inverters, NAND gates, NOR gates, AOI gates, OAI gates, 2-to-1 MUX, D-latches, flip-flops, etc.*).

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

78

## Characteristic of the Cells

- Each cell is designed with a fixed height.
  - To enable automated placement of the cells, and routing of inter-cell connections.
  - A number of cells can be abutted side-by-side to form rows.
- The power and ground rails typically run parallel to upper and lower boundaries of cell.
  - Neighboring cells share a common power and ground bus.
- The input and output pins are located on the upper and lower boundaries of the cell.



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

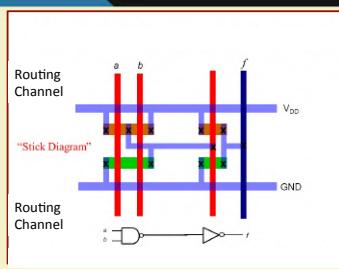
Hardware Modeling Using Verilog

79

## Standard Cell Example

Made to stack side by side

- Fixed height
- Width can vary
- Can abut at  $V_{DD}$  and GND



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

80

## Floorplan for Standard Cell Design

- Inside the I/O frame which is reserved for I/O cells, the chip area contains rows or columns of standard cells.
  - Between cell rows are channels for routing.
  - Over-the-cell routing is also possible.
- The physical design and layout of logic cells ensure that
  - When placed into rows, their heights match.
  - Neighboring cells can abut side-by-side, which provides natural connections for power and ground lines in each row.

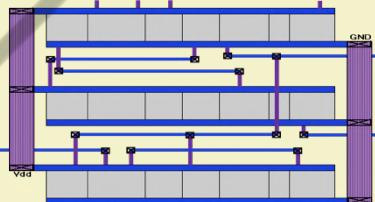


NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

81

## Standard Cell Layout

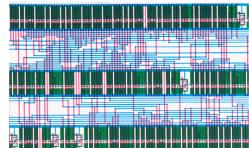


NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

82

## Standard Cell Layout



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

83

## Full Custom Design



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Introduction

- Standard-cells based design is often called semi custom design.
  - The cells are pre-designed for general use.
- In the full custom design, the entire mask design is done anew without use of any library.
  - The development cost of such a design style is prohibitively high.
  - The concept of design reuse is becoming popular to reduce design cycle time and cost.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

85

- The most rigorous full custom design can be the design of a memory cell.
  - Static or dynamic.
  - Since the same layout design is replicated, there would not be any alternative to high density memory chip design.
- For logic chip design, a good compromise can be achieved by combining different design styles on the same chip.
  - Standard cells, data-path cells and PLAs.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

86

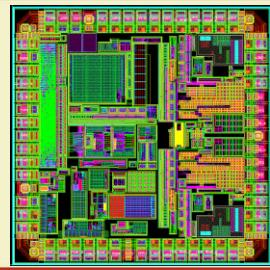
- In real full-custom layout in which the geometry, orientation and placement of every transistor is done individually by the designer.
  - Design productivity is usually very low (typically 10 to 20 transistors per day, per designer).
- In digital CMOS VLSI, full-custom design is rarely used due to the high labor cost.
  - Exceptions to this include the design of high-volume products such as memory chips, high-performance microprocessors and FPGA masters.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

87

A full custom layout

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

88

## Comparison Among Various Design Styles

	Design Style			
	FPGA	Gate array	Standard cell	Full custom
Cell size	Fixed	Fixed	Fixed height	Variable
Cell type	Programmable	Fixed	Variable	Variable
Cell placement	Fixed	Fixed	In row	Variable
Interconnect	Programmable	Variable	Variable	Variable
Design time	Very fast	Fast	Medium	Slow

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

89

**END OF LECTURE 05**

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

90

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

## Lecture 06: VERILOG LANGUAGE FEATURES (PART 1)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Concept of Verilog “Module”

- In Verilog, the basic unit of hardware is called a **module**.
  - A module cannot contain definition of other modules.
  - A module can, however, be **instantiated** within another module.
  - Instantiation allows the creation of a **hierarchy** in Verilog description.

```
module module_name (list_of_ports);
  input/output declarations
  local net declarations
  Parallel statements
endmodule
```

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 2

```
// A simple AND function
module simpleand (f, x, y);
  input x, y;
  output f;
  assign f = x & y;
endmodule
```

This is a behavioral description. The synthesis tool will decide how the realize f:

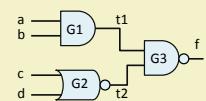
- Using a single AND gate
- Using a NAND gate followed by a NOT gate.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 1

This is also behavioral description.

- One possible gate level realization is shown.
- t1 and t2 are intermediate lines; termed as wire data type.

```
/* A 2-level combinational circuit */
module two_level (a, b, c, d, f);
  input a, b, c, d;
  output f;
  wire t1, t2; // Intermediate lines
  assign t1 = a & b;
  assign t2 = ~(c | d);
  assign f = ~(t1 & t2);
endmodule
```



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 4

- Point to note:**
  - The “assign” statement represents continuous assignment, whereby the variable on the LHS gets updated whenever the expression on the RHS changes.
  - assign variable = expression;
  - The LHS must be a “net” type variable, typically a “wire”.
  - The RHS can contain both “register” and “net” type variables.
  - A Verilog module can contain any number of “assign” statements; they are typically placed in the beginning after the port declarations.
  - The “assign” statement models behavioral design style, and is typically used to model combinational circuits.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 5

### Data Types in Verilog

- A variable in Verilog belongs to one of two data types:
  - Net
    - Must be continuously driven.
    - Cannot be used to store a value.
    - Used to model connections between continuous assignments and instantiations.
  - Register
    - Retains the last value assigned to it.
    - Often used to represent storage elements, but sometimes it can translate to combinational circuits also.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 6

### (a) Net data type

- Nets represents connection between hardware elements.
  - Nets are continuously driven by the outputs of the devices they are connected to.
- 
- Net "a" is continuously driven by the output of the AND gate.
- Nets are 1-bit values by default unless they are declared explicitly as vectors.
  - Default value of a net is "z".



- Various "Net" data types are supported for synthesis in Verilog:
  - wire, wor, wand, tri, supply0, supply1, etc.
- "wire" and "tri" are equivalent; when there are multiple drivers driving them, the driver outputs are shorted together.
- "wor" and "wand" inserts an OR and AND gate respectively at the connection.
- "supply0" and "supply1" model power supply connections.
- The Net data type "wire" is most common.



```
module use_wire (A, B, C, D, f);
  input A, B, C, D;
  output f;
  wire f;
  // net f declared as 'wire'

  assign f = A & B;
  assign f = C | D;
endmodule
```

For  $A = B = 1$ , and  $C = D = 0$ ,  
f will be indeterminate.

```
module use_wand (A, B, C, D, f);
  input A, B, C, D;
  output f;
  wand f;
  // net f declared as 'wire'

  assign f = A & B;
  assign f = C | D;
endmodule
```

Here, function realized will be  
 $f = (A \& B) \& (C | D)$



```
module using_supply_wire (A, B, C, f);
  input A, B, C;
  output f;
  supply0 gnd;
  supply1 vdd;
  nand G1 (t1, vdd, A, B);
  xor G2 (t2, C, gnd);
  and G3 (f, t1, t2);
endmodule
```

supply0 and supply1 have the greatest signal strength.



### Data Values and Signal Strengths

- Verilog supports 4 value levels and 8 strength levels to model the functionality of real hardware.
  - Strength levels are typically used to resolve conflicts between signal drivers of different strengths in real circuits.

Value Level	Represents
0	Logic 0 state
1	Logic 1 state
x	Unknown logic state
z	High impedance state

- Initialization:
- All unconnected nets are set to "z".
  - All register variables set to "x".



Strength	Type
supply	Driving
strong	Driving
pull	Driving
large	Storage
weak	Driving
medium	Storage
small	Storage
highz	High impedance

Strength increases ↑

- If two signals of unequal strengths get driven on a wire, the stronger signal will prevail.
- These are particularly useful for MOS level circuits, e.g. dynamic MOS.



## END OF LECTURE 06

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 13

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

## Lecture 07: VERILOG LANGUAGE FEATURES (PART 2)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### (b) Register Data Type

- In Verilog, a “*register*” is a variable that can *hold* a value.
  - Unlike a “*net*” that is continuously driven and cannot hold any value.
  - Does not necessarily mean that it will map to a hardware register during synthesis.
  - Combinational circuit specifications can also use register type variables.
- Register data types supported by Verilog:
  - reg* : Most widely used
  - integer* : Used for loop counting (typical use)
  - real* : Used to store floating-point numbers
  - time* : Keeps track of simulation time (not used in synthesis)

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 15

- “*reg*” data type:
  - Default value of a “*reg*” data type is “x”.
  - It can be assigned a value in synchronism with a clock or even otherwise.
  - The declaration explicitly specifies the size (default is 1-bit):
 

```
reg x, y; // Single-bit register variables
reg [15:0] bus; // A 16-bit bus
```
  - Treated as an unsigned number in arithmetic expressions.
  - Must be used when we model actual sequential hardware elements like counters, shift registers, etc.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 16

```
module simple_counter (clk, rst, count);
  input clk, rst;
  output [31:0] count;
  reg [31:0] count;

  always @(posedge clk)
  begin
    if (rst)
      count = 32'b0;
    else
      count = count + 1;
  end
endmodule
```

32-bit counter with synchronous reset.

- Count value increases at the positive edge of the clock.
- If “rst” is high, the counter is reset at the positive edge of the next clock.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 17

```
module simple_counter (clk, rst, count);
  input clk, rst;
  output [31:0] count;
  reg [31:0] count;

  always @(posedge clk or posedge rst)
  begin
    if (rst)
      count = 32'b0;
    else
      count = count + 1;
  end
endmodule
```

32-bit counter with asynchronous reset.

- Here reset occurs whenever “rst” goes high.
- Does not synchronize with clock.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 18

- “**integer**” data type:
  - It is a general-purpose register data type used for manipulating quantities.
  - More convenient to use in situations like loop counting than “**reg**”.
  - It is treated as a 2’s complement signed integer in arithmetic expressions.
  - Default size is 32 bits; however, the synthesis tool tries to determine the size using data flow analysis.
  - Example:
 

```
wire [15:0] X, Y;
integer C;
Z = X + Y;
```

 Size of Z can be deduced to be 17 (16 bits plus a carry).



NPTEL  
NPTEL

Hardware Modeling Using Verilog

19

- “**real**” data type:
  - Used to store floating-point numbers.
  - When a real value is assigned to an integer, the real number is rounded off to the nearest integer.

```
real e, pi;
initial integer x;
begin
  e = 2.718;
  pi = 314.159e-2;
end
```



NPTEL  
NPTEL

Hardware Modeling Using Verilog

20

- “**time**” data type:
  - In Verilog, simulation is carried out with respect to a logical clock called simulation time.
  - The “**time**” data type can be used to store simulation time.
  - The system function “**\$time**” gives the current simulation time.
  - Example:
 

```
time curr_time;
initial
...
curr_time = $time;
```



NPTEL  
NPTEL

Hardware Modeling Using Verilog

21

- ## Vectors
- Nets or “**reg**” type variable can be declared as vectors, of multiple bit widths.
    - If bit width is not specified, default size is 1-bit.
  - Vectors are declared by specifying a range [**range1:range2**], where **range1** is always the most significant bit and **range2** is the least significant bit.
  - Examples:

```
wire x, y, z;           // Single bit variables
wire [7:0] sum;         // MSB is sum[7], LSB is sum[0]
reg [31:0] MDR;
reg [1:10] data;        // MSB is data[1], LSB is data[10]
reg clock;
```



NPTEL  
NPTEL

Hardware Modeling Using Verilog

22

- Parts of a vector can be addressed and used in an expression.
- Example:
  - A 32-bit instruction register, that contains a 6-bit opcode, three register operands of 5 bits each, and an 11-bit offset.

```
reg [31:0] IR;           opcode = IR[31:26];
reg [5:0] opcode;         reg1 = IR[25:21];
reg [4:0] reg1, reg2, reg3; reg2 = IR[20:16];
reg [10:0] offset;       reg3 = IR[15:11];
                        offset = IR[10:0];
```



NPTEL  
NPTEL

Hardware Modeling Using Verilog

23

## Multi-dimensional Arrays and Memories

- Multi-dimensional arrays of any dimension can be declared in Verilog.
- Example:
 

```
reg [31:0] register_bank[15:0]; // 16 32-bit registers
integer matrix[7:0][15:0];
```
- Memories can be modeled in Verilog as a 1-D array of registers.
  - Each element of the array is addressed by a single array index.
  - Examples:
 

```
reg mem_bit[0:2047];           // 2K 1-bit words
reg [15:0] mem_word[0:1023];    // 1K 16-bit words
```



NPTEL  
NPTEL

Hardware Modeling Using Verilog

24

### Specifying Constant Values

- A constant value may be specified in either the **sized** or the **unsized** form.
  - Syntax of sized form:  
 $<\text{size}><\text{base}><\text{number}>$
  - Variables of type integer and real are typically expressed in unsized form.
- Examples:
 

```
4'b0101          // 4-bit binary number 0101
1'b0              // Logic 0 (1-bit)
12'hB3C           // 12-bit number 1011 0011 1100
12'h8xF           // 12-bit number 1000 xxxx 1111
25                // signed number, in 32 bits (size not specified)
```

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 25

### Parameters

- A parameter is a constant with a given name.
  - We cannot specify the size of a parameter.
  - The size gets decided from the constant value itself; if size is not specified, it is taken to be 32 bits.
- Examples:
 

```
parameter HI = 25, LO = 5;
parameter up = 2b'00, down = 2b'01, steady = 2b'10;
parameter RED = 3b'100, YELLOW = 3b'010, GREEN = 3b'001;
```

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 26

```
// Parameterized design:: an N-bit counter
module counter (clear, clock, count);
  parameter N = 7;
  input clear, clock;
  output [0:N] count;  reg [0:N] count;

  always @ (negedge clock)
    if (clear)
      count <= 0;
    else
      count <= count + 1;
endmodule
```

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 27

## END OF LECTURE 07

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 28

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | NPTEL

### Lecture 08: VERILOG LANGUAGE FEATURES (PART 3)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

### Predefined Logic Gates in Verilog

- Verilog provides a set of predefined logic gates.
  - Can be instantiated within a module to create a structured design.
  - The gates respond to logic values (0, 1, x or z) in a logical way.

2-input AND	2-input OR	2-input EXOR
0 & 0 = 0	0   0 = 0	0 ^ 0 = 0
0 & 1 = 0	0   1 = 1	0 ^ 1 = 1
1 & 1 = 1	1   1 = 1	1 ^ 1 = 0
1 & x = x	1   x = 1	1 ^ x = x
0 & x = 0	0   x = x	0 ^ x = x
1 & z = x	1   z = x	1 ^ z = x
z & x = x	z   x = x	z ^ x = x

 IIT Kharagpur |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 30

### List of Primitive Gates

```

and   G (out, in1, in2);      bufif1  G (out, in, ctrl);
nand  G (out, in1, in2);      bufif0  G (out, in, ctrl);
or    G (out, in1, in2);      notif0  G (out, in, ctrl);
nor   G (out, in1, in2);      notif1  G (out, in, ctrl);
xor   G (out, in1, in2);      There are gates with tristate
xnor  G (out, in1, in2);      controls
not   G (out, in);
buf   G (out, in);

```

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

31

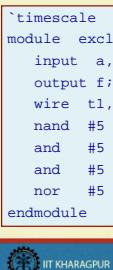
- Some restriction when instantiating primitive gates:

- The output port must be connected to a net (e.g. a wire).
  - An "**output**" signal is a wire by default, unless explicitly declared as a register.
- The input ports may be connected to nets or register type variables.
- They have a single output but can have any number of inputs (except NOT and BUF).
- When instantiating a gate, an optional delay may be specified.
  - Used for simulation.
  - Logic synthesis tools ignore the time delays.

NPTEL  
ONLINE  
CERTIFICATION COURSES

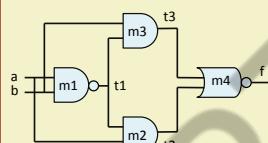
Hardware Modeling Using Verilog

32

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

33



```

`timescale 10ns/1ns
module exclusive_or (f, a, b);
  input a, b;
  output f;
  wire t1, t2, t3;
  nand #5 m1 (t1, a, b);
  and #5 m2 (t2, a, t1);
  and #5 m3 (t3, t1, b);
  nor #5 m4 (f, t2, t3);
endmodule

```

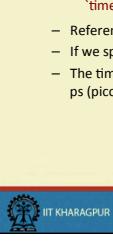
### The `timescale Directive

- Often in a single simulation, delay values in one module need to be specified in terms of some time unit, while those in some other module need to be specified in terms of some other time unit.
- The `timescale compiler directive can be used:
  - `'timescale <reference_time_units> / <time_precision>`
  - The `<reference_time_unit>` specifies the unit of measurement for time.
  - The `<time_precision>` specifies the precision to which the delays are rounded off during simulation.
    - Valid values for specifying time unit and time precision are 1, 10 and 100.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

34

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

- Example:

- `'timescale 10ns/1ns`
- Reference time unit is 10ns, and simulation precision is 1ns.
- If we specify #5 as delay, it will mean 50ns.
- The time units can be specified in s (second), ms (millisecond), us (microsecond), ps (picosecond), and fs (femtosecond).

### Specifying Connectivity during Instantiation

- When a module is instantiated within another module, there are two ways to specify the connectivity of the signal lines between the two modules.
  - Positional association
    - The parameters of the module being instantiated are listed in the same order as in the original module description.
  - Explicit association
    - The parameters of the module being instantiated are listed in arbitrary order.
    - Chance of errors is less.

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36

**Positional Association**

```

module testbench;
reg X1,X2,X3,X4,X5,X6; wire OUT;
example DUT(X1,X2,X3,X4,X5,X6,OUT);

initial
begin
$monitor ($time," X1=%b, X2=%b,
X3=%b, X4=%b, X5=%b, X6=%b,
OUT=%b", X1,X2,X3,X4,X5,X6,OUT);
#5 X1=1;X2=0; X3=0; X4=1; X5=0; X6=0;
#5 X1=0; X3=1;
#5 X1=1; X3=0;
#5 X6=1;
#5 $finish;
end
endmodule

```

**Hardware Modeling Using Verilog**

**Explicit Association**

```

module testbench;
reg X1,X2,X3,X4,X5,X6; wire OUT;
example DUT(.OUT(Y),.X1(A),.X2(B),.X3(C),
.X4(D),.X5(E),.X6(F));

initial
begin
$monitor ($time," X1=%b, X2=%b,
X3=%b, X4=%b, X5=%b, X6=%b,
OUT=%b", X1,X2,X3,X4,X5,X6,OUT);
#5 X1=1;X2=0; X3=0; X4=1; X5=0; X6=0;
#5 X1=0; X3=1;
#5 X1=1; X3=0;
#5 X6=1;
#5 $finish;
end
endmodule

```

**Hardware Modeling Using Verilog**

**Hardware Modeling Issues**

- In terms of the hardware realization, the value computed can be assigned to:
  - A "wire"
  - A "flip-flop" (edge-triggered storage cell)
  - A "latch" (level-triggered storage cell)
- A variable in Verilog can be either "*net*" or "*register*".  
  - A "*net*" data type always map to a "wire" during synthesis.
  - A "*register*" data type maps either to a "wire" or a "storage cell" depending upon the context under which a value is assigned.

**Hardware Modeling Using Verilog**

```

module reg_maps_to_wire (A, B, C, f1, f2);
input A, B, C;
output f1, f2;
wire A, B, C;
reg f1, f2;
always @(A or B or C)
begin
f1 = ~(A & B);
f2 = f1 ^ C;
end
endmodule

```

The synthesis system will generate a wire for f1.

**Hardware Modeling Using Verilog**

```

module a_problem_case (A, B, C, f1, f2);
input A, B, C;
output f1, f2;
wire A, B, C;
reg f1, f2;
always @(A or B or C)
begin
f2 = f1 ^ f2;
f1 = ~(A & B);
end
endmodule

```

The synthesis system will generate a wire for f1, and a storage cell for f2.

**Hardware Modeling Using Verilog**

```

// A latch gets inferred here
module simple_latch (data, load, d_out);
input data, load;
output d_out;
wire t;
always @(load or data)
begin
if (!load)
t = data;
d_out = !t;
end
endmodule

```

The "else" part is missing. So a latch will be generated for "t".

**Hardware Modeling Using Verilog**

**END OF LECTURE 08**

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 43

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

**Lecture 09: VERILOG OPERATORS**

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Verilog Operators**

**Arithmetic Operators:**

- + unary (sign) plus
- unary (sign) minus
- + binary plus (add)
- binary minus (subtract)
- \* multiply
- / divide
- % modulus
- \*\* exponentiation

**Examples:**

- (b + c)
- (a - b) + (c \* d)
- (a + b) / (a - b)
- a % b
- a \*\* 3

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 45

**Logical Operators:**

- ! logical negation
- && logical AND
- || logical OR

**Examples:**

- (done && ack)
- (a || b)
- !(a && b)
- ((a > b) || (c == 0))
- ((a > b) && !(b > c))

- The value 0 is treated as logical FALSE while any non-zero value is treated as TRUE.
- Logical operators return either 0 (FALSE) or 1 (TRUE).

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 46

**Relational Operators:**

- != not equal
- == equal
- >= greater or equal
- <= less or equal
- > greater
- < less

Relational operators operate on numbers, and return a Boolean value (true or false).

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 47

**Bitwise Operators:**

- ~ bitwise NOT
- & bitwise AND
- | bitwise OR
- ^ bitwise exclusive-OR
- ~^ bitwise exclusive-NOR

**Examples:**

```
wire a, b, c, d, f1, f2, f3, f4;
assign f1 = ~a | b;
assign f2 = (a & b) | (b & c) | (c & a);
assign f3 = a ^ b ^ c;
assign f4 = (a & ~b) | (b & c & ~d);
```

Bitwise operators operate on bits, and return a value that is also a bit.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 48

Reduction operators accept a single word operand and produce a single bit as output.

- Operates on all the bits within the word.

**Reduction Operators:**

&	bitwise AND
	bitwise OR
~&	bitwise NAND
~	bitwise NOR
^	bitwise exclusive-OR
~^	bitwise exclusive-NOR

**Examples:**

```
wire [3:0] a, b, c; wire f1, f2, f3;
assign a = 4'b0111;
assign b = 4'b1100;
assign c = 4'b0100;
assign f1 = ^a;           // gives a 1
assign f2 = & (a ^ b); // gives a 0
assign f3 = ~a & ~^b; // gives a 1
```



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

49

**Shift Operators:**

>>	shift right
<<	shift left
>>>	arithmetic shift right

**Examples:**

```
wire [15:0] data, target;
assign target = data >> 3;
assign target = data >>> 2;
```

**Conditional Operator:**

```
cond_expr ? true_expr : false_expr;
```

**Examples:**

```
wire a, b, c;
wire [7:0] x, y, z;
assign a = (b > c) ? b : c;
assign z = (x == y) ? x+2 : x-2;
```



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

50

**Concatenation Operator:**  
{..., ..., ...}

**Replication Operator:**  
{n{m}}

Joins together bits from two or more comma-separated expressions.

Joins together n copies of an expression m, where n is a constant.

**Examples:**

```
assign f = {a, b};
assign f = {a, 3'b101, b};
assign f = {x[2], y[0], a};
assign f = {2'b10, 3{b'01}, x};
```



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

51

```
module operator_example (x, y, f1, f2);
  input x, y;
  output f1, f2;
  wire [9:0] x, y; wire [4:0] f1; wire f2;
  assign f1 = x[4:0] & y[4:0];
  assign f2 = x[2] | ~f1[3];
  assign f2 = ~& x;
  assign f1 = f2 ? x[9:5] : x[4:0];
endmodule
```



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

52

```
// An 8-bit adder description
module parallel_adder (sum, cout, in1, in2, cin);
  input [7:0] in1, in2;
  input cin;
  output [7:0] sum;
  output cout;
  assign #20 {cout,sum} = in1 + in2 + cin;
endmodule
```



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

53

## Operator Precedence

- Operators on same line have the same precedence.
- All operators associate left to right in an expression, except ?:.
- Parentheses can be used to change the precedence.

+ - ! ~ (unary)
**
<< >> >>>
< <= > >=
== != === !==
& ~&
^ ~^
~
&&
? :

↑  
Precedence increases



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

54

### Some Points

- The presence of a 'z' or 'x' in a **reg** or **wire** being used in an arithmetic expression results in the whole expression being unknown ('x').
- The logical operators (!, &&, | |) all evaluate to a 1-bit result (0, 1 or x).
- The relational operators (>, <, <=, >=, ~=, ==) also evaluate to a 1-bit result (0 or 1).
- Boolean *false* is equivalent to 1'b0.  
Boolean *true* is equivalent to 1'b1.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 55

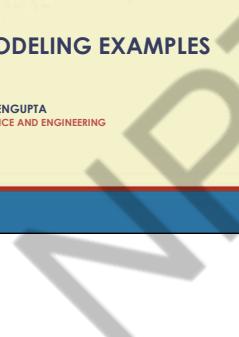
### END OF LECTURE 09

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 56

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | NPTEL

### Lecture 10: VERILOG MODELING EXAMPLES

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 57

### Example 1

- The structural hierarchical description of a 16-to-1 multiplexer.
  - Using pure behavioral modeling.
  - Structural modeling using 4-to-1 multiplexer specified using behavioral model.
  - Make structural modeling of 4-to-1 multiplexer, using behavioral modeling of 2-to-1 multiplexer.
  - Make structural gate-level modeling of 2-to-1 multiplexer, to have a complete structural hierarchical description.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 58

#### Version 1: Using pure behavioral modeling

```
module mux16to1 (in, sel, out);
  input [15:0] in;
  input [3:0] sel;
  output out;

  assign out = in[sel];
endmodule
```

Selects one of the input bits depending upon the value of "sel".

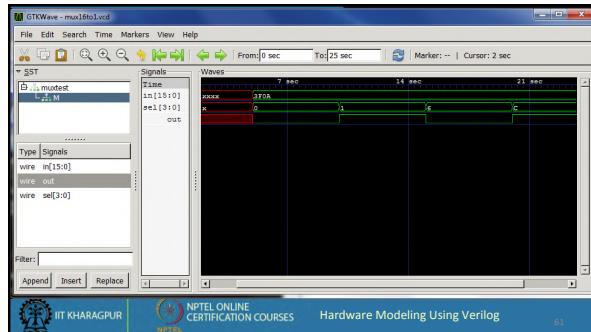
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 59

```
module muxtest;
  reg [15:0] A;      reg [3:0] S;      wire F;
  mux16to1 M (.in(A), .sel(S), .out(F));

  initial
    begin
      $dumpfile ("mux16to1.vcd");
      $dumpvars (0,muxtest);
      $monitor ($time," A=%h, S=%h, F=%b", A,S,F);
      #5 A=16'h3f0a; S=4'h0;
      #5 S=4'h1;
      #5 S=4'h6;
      #5 S=4'hc;
      #5 $finish;
    end
endmodule
```

0 A=xxxx, S=x, F=x
5 A=3f0a, S=0, F=0
10 A=3f0a, S=1, F=1
15 A=3f0a, S=6, F=0
20 A=3f0a, S=c, F=1

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 60



**Version 2: Behavioral modeling of 4-to-1 MUX  
Structural modeling of 16-to-1 MUX**

```
module mux16to1 (in, sel, out);
    input [15:0] in;
    input [3:0] sel;
    output out;
    wire [3:0] t;

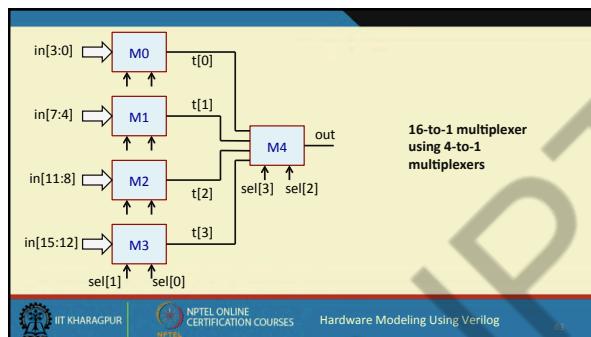
    assign out = in[sel];
endmodule
```

```
module mux4to1 (in, sel, out);
    input [3:0] in;
    input [1:0] sel;
    output out;
    wire [3:0] t;

    assign out = in[sel];
endmodule
```

```
module mux16to1 (in, sel, out);
    input [15:0] in;
    input [3:0] sel;
    output out;
    wire [3:0] t;

    mux4to1 M0 (in[3:0],sel[1:0],t[0]);
    mux4to1 M1 (in[7:4],sel[1:0],t[1]);
    mux4to1 M2 (in[11:8],sel[1:0],t[2]);
    mux4to1 M3 (in[15:12],sel[1:0],t[3]);
    mux4to1 M4 (t,sel[3:2],out);
endmodule
```



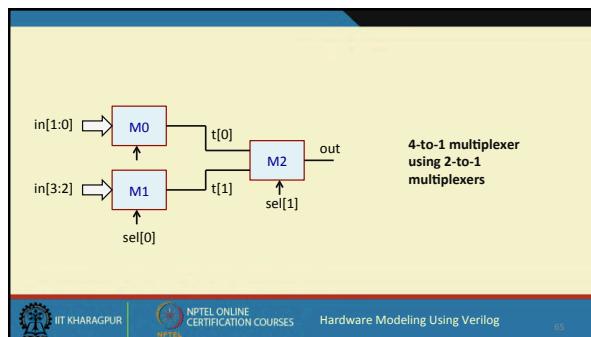
**Version 3: Behavioral modeling of 2-to-1 MUX  
Structural modeling of 4-to-1 MUX**

```
module mux4to1 (in, sel, out);
    input [3:0] in;
    input [1:0] sel;
    output out;
    wire [3:0] t;

    assign out = in[sel];
endmodule
```

```
module mux2to1 (in, sel, out);
    input [3:0] in;
    input sel;
    output out;
    wire [1:0] t;

    mux2to1 M0 (in[1:0],sel[0],t[0]);
    mux2to1 M1 (in[3:2],sel[0],t[1]);
    mux2to1 M2 (t,sel[1],out);
endmodule
```



**Version 4: Structural modeling of 2-to-1 MUX**

```
module mux2to1 (in, sel, out);
    input [1:0] in;
    input sel;
    output out;
    wire t1, t2, t3;

    NOT G1 (t1,sel);
    AND G2 (t2,in[0],t1);
    AND G3 (t3,in[1],sel);
    OR G4 (out,t2,t3);
endmodule
```

Point to note:

- Same test bench can be used for all the versions.
- The versions illustrate hierarchical refinement of design.

## END OF LECTURE 10

67

 IIT KHARAGPUR | 
  NPTEL ONLINE CERTIFICATION COURSES  
**Lecture 11: VERILOG MODELING EXAMPLES (contd.)**  
 PROF. INDRANIL SENGUPTA  
 DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Example 2

**Version 1: Behavioral description of a 16-bit adder.**

- Generation of status flags:
  - Sign : whether the sum is negative or positive
  - Zero : whether the sum is zero
  - Carry : whether there is a carry out of the last stage
  - Parity : whether the number of 1's in the sum is even or odd
  - Overflow : whether the sum cannot fit in 16 bits

69

```

module ALU (X, Y, Z, Sign, Zero, Carry, Parity, Overflow);
    input [15:0] X, Y;
    output [15:0] Z;
    output Sign, Zero, Carry, Parity, Overflow;

    assign {Carry, Z} = X + Y; // 16-bit addition
    assign Sign = Z[15];
    assign Zero = ~Z;
    assign Parity = ~^Z;
    assign Overflow = (X[15] & Y[15] & ~Z[15]) |
                      (~X[15] & ~Y[15] & Z[15]);
endmodule

```

70

```

module alutest;
    reg [15:0] X, Y;
    wire [15:0] Z;           wire S, ZR, CY, P, V;
    ALU DUT (X, Y, Z, S, ZR, CY, P, V);
    initial
        begin
            $dumpfile ("alu.vcd");
            $dumpvars (0,alutest);
            $monitor ($time," X=%h, Y=%h, Z=%h, S=%b, Z=%b, CY=%b, P=%b,
                        V=%b", X, Y, Z, S, ZR, CY, P, V);
            #5 X = 16'hffff; Y = 16'h8000;
            #5 X = 16'hffff; Y = 16'h0002;
            #5 X = 16'hAAAAAA; Y = 16'h5555;
            #5 $finish;
        end
    endmodule

```

71

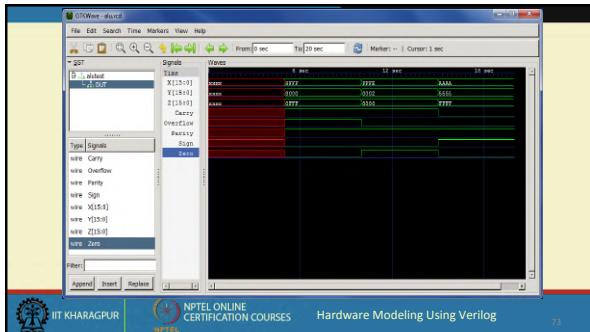
### Simulation Output

```

0 X=xxxx, Y=xxxx, Z=xxxx, S=x, Z=x, CY=x, P=x, V=x
5 X=8fff, Y=8000, Z=0fff, S=0, Z=0, CY=1, P=1, V=1
10 X=ffff, Y=0002, Z=0000, S=0, Z=1, CY=1, P=1, V=0
15 X=aaaa, Y=5555, Z=ffff, S=1, Z=0, CY=0, P=1, V=0

```

72



### Version 2: Structural description of 16-bit adder using 4-bit adder blocks (with ripple carry between blocks).

```
module ALU (X, Y, Z, Sign, Zero, Carry, Parity, Overflow);
    input [15:0] X, Y;
    output [15:0] Z;
    output Sign, Zero, Carry, Parity, Overflow;
    wire c[3:1];

    assign Sign = Z[15];
    assign Zero = ~Z;
    assign Parity = ~Z;
    assign Overflow = (X[15] & Y[15] & ~Z[15]) |
                      (~X[15] & ~Y[15] & Z[15]);

```

... Contd...

`adder4 A0 (Z[3:0], c[1], X[3:0], Y[3:0], 1'b0);  
adder4 A1 (Z[7:4], c[2], X[7:4], Y[7:4], c[1]);  
adder4 A2 (Z[11:8], c[3], X[11:8], Y[11:8], c[2]);  
adder4 A3 (Z[15:12], Carry, X[15:12], Y[15:12], c[3]);  
endmodule`

**Behavioral description of a 4-bit adder**

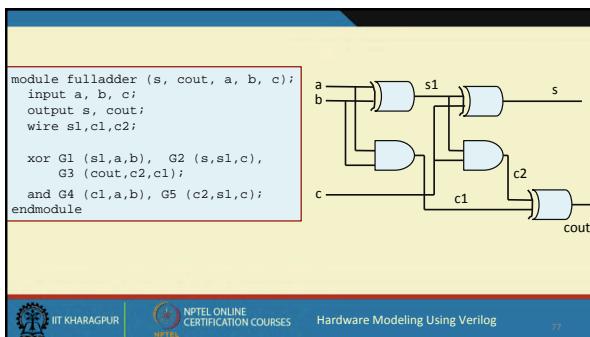
```
module adder4 (S, cout, A, B, cin);
    input [3:0] A, B;      input cin;
    output [3:0] S;        output cout;
    assign {cout,S} = A + B + cin;
endmodule
```

### Version 3: Structural Modeling of Ripple Carry Adder

```
module adder4 (S, cout, A, B, cin);
    input [3:0] A, B;      input cin;
    output [3:0] S;        output cout;
    wire c1,c2,c3;

    fulladder FA0 (S[0],c1,A[0],B[0],cin);
    fulladder FA1 (S[1],c2,A[1],B[1],c1);
    fulladder FA2 (S[2],c3,A[2],B[2],c2);
    fulladder FA3 (S[3],cout,A[3],B[3],c3);

endmodule
```



### Version 4: Structural Modeling of Carry Lookahead Adder

```
module adder4 (S, cout, A, B, cin);
    input [3:0] A, B;      input cin;
    output [3:0] S;        output cout;
    wire p0, g0, p1, g1, p2, g2, p3, g3;
    wire c1, c2, c3;

    assign p0 = A[0] ^ B[0],   p1 = A[1] ^ B[1],
           p2 = A[2] ^ B[2],   p3 = A[3] ^ B[3];

    assign g0 = A[0] & B[0],  g1 = A[1] & B[1],
           g2 = A[2] & B[2],  g3 = A[3] & B[3];


```

Contd...



```

assign cl = g0 | (p0 & cin),
      c2 = g1 | (p1 & g0) | (p1 & p0 & cin),
      c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & cin),
      cout = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) | (p3 & p2 & p1 & p0 & cin);

assign S[0] = p0 ^ cin,
      S[1] = p1 ^ cl,
      S[2] = p2 ^ c2,
      S[3] = p3 ^ c3;

endmodule

```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 79

### How does a Carry Look-ahead Adder work?

- The propagation delay of an n-bit ripple carry order is proportional to n.
  - Due to the rippling effect of carry sequentially from one stage to the next.
- One possible way to speedup the addition.
  - Generate the carry signals for the various stages in parallel.
  - Time complexity reduces from  $O(n)$  to  $O(1)$ .
  - Hardware complexity increases rapidly with n.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 80

- Consider the i-th stage in the addition process.
- We define the *carry generate* and *carry propagate* functions as:
$$g_i = A_i \cdot B_i$$

$$p_i = A_i \oplus B_i$$
- $g_i = 1$  represents the condition when a carry is generated in stage-i independent of the other stages.
- $p_i = 1$  represents the condition when an input carry  $c_i$  will be propagated to the output carry  $c_{i+1}$ .

$c_{i+1} = g_i + p_i \cdot c_i$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 81

### Unrolling the Recurrence

$$\begin{aligned}
 c_{i+1} &= g_i + p_i \cdot c_i = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1}) = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1} \\
 &= g_i + p_i g_{i-1} + p_i p_{i-1} (g_{i-2} + p_{i-2} c_{i-2}) \\
 &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + p_i p_{i-1} p_{i-2} c_{i-2} = \dots
 \end{aligned}$$

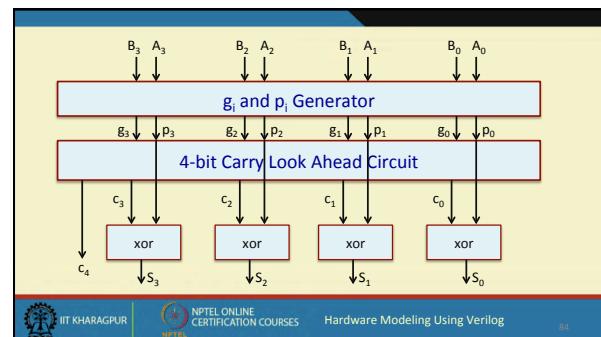
$$c_{i+1} = g_i + \sum_{k=0}^{i-1} g_k \prod_{j=k+1}^i p_j + c_0 \prod_{j=0}^i p_j$$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 82

### Generation of the Carry and Sum bits

$C_4 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3$ $C_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$ $C_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$ $C_1 = g_0 + c_0 p_0$ $S_0 = A_0 \oplus B_0 \oplus c_0 = p_0 \oplus c_0$ $S_1 = p_1 \oplus c_1$ $S_2 = p_2 \oplus c_2$ $S_3 = p_3 \oplus c_3$	4 AND2 gates 3 AND3 gates 2 AND4 gates 1 AND5 gate 1 OR2, 1 OR3, 1 OR4 and 1 OR5 gate 4 XOR2 gates
--	---

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 83



**END OF LECTURE 11**



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

85

NPTEL



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 12: VERILOG DESCRIPTION STYLES

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Description Styles in Verilog

- Two different styles of description:
  1. Data Flow
    - Continuous assignment

*Using assignment statements.*
  2. Behavioral
    - Procedural assignment
      - Blocking
      - Non-blocking

*Using procedural statements similar to a program in high-level language.*



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

## Data Flow Style: Continuous Assignment

- Identified by the keyword “assign”.
- Forms a static binding between:
  - The “net” being assigned on the left-hand side (LHS).
  - The expression on the right-hand side (RHS), which may consist of both “net” and “register” type variables.
- The assignment is continuously active:
  - Almost exclusively used to model combinational circuits.
  - We shall also see some examples of modeling sequential circuit elements.

```
assign a = b + c;
assign sign = Z[15];
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

- Some points to note:
  - A Verilog module can contain any number of “assign” statements.
  - Typically, the “assign” statements are followed by procedural descriptions.
  - The “assign” statements are used to model behavioral descriptions.
- We shall illustrate various usages of “assign” statements for modeling combinational and also some sequential logic blocks.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

```
module generate_MUX (data, select, out);
    input [15:0] data;
    input [3:0] select;
    output out;
    assign out = data[select];
endmodule
```

Non-constant index in expression on RHS generates a MUX



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

5

- Point to note:
  - Whenever there is an array reference on the RHS with a variable index, a MUX is generated by the synthesis tool.
  - If the index is a constant, just a wire will be generated.  
Example: `assign out = data[2];`



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

6

```
module generate_set_of_MUX (a, b, f, sel);
    input [0:3] a, b;
    input sel;
    output [0:3] f;
    assign f = sel ? a : b;
endmodule
```

Conditional operator generates a MUX



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

- Point to note:
  - Whenever a conditional is encountered in the RHS of an expression, a 2-to-1 MUX is generated.
  - In the previous example, since the variables “a”, “b” and “f” are vectors, an array of 2-to-1 MUX-es are generated.
  - What hardware will be generated by the following?

```
assign f = (a==0) ? (c+d) : (c-d);
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

```
module generate_decoder (out, in, select);
    input in;
    input [0:1] select;
    output [0:3] out;
    assign out[select] = in;
endmodule
```

Non-constant index in expression on LHS generates a decoder



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

9

- Point to note:
  - A constant index in the expression on the LHS will not generate a decoder.
  - Example: `assign out[5] = in;`  
This will simply generate a wire connection.
  - As a rule of thumb, whenever the synthesis tool detects a variable index in the LHS, a decoder is generated.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

10

```
module level_sensitive_latch (D, Q, En);
    input D, En;
    output Q;
    assign Q = En ? D : Q;
endmodule
```

En	D	Q <sub>n</sub>
0	x	Q <sub>n-1</sub>
1	0	0
1	1	1

Generates a D-type latch

Here is an example to describe a sequential logic element using “assign” statement.



IIT KHARAGPUR

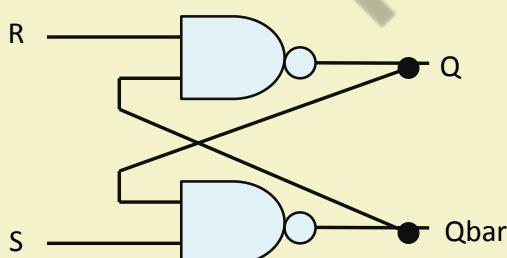


NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

11

- Modeling a simple S-R latch:



S	R	Q <sub>n</sub>
1	1	Q <sub>n-1</sub>
0	1	0
1	0	1
0	0	?



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

12

```

module sr_latch (Q, Qbar);
    input S, R;
    output Q, Qbar;
    assign Q = ~(R & Qbar);
    assign Qbar = ~(S & Q);
endmodule

module latchtest;
    reg S, R;    wire Q, Qbar;
    sr_latch LAT (Q, Qbar, S, R);
    initial
        begin
            $monitor ($time, "S=%b R=%b, Q=%b, Qbar=%b",
                      S, R, Q, Qbar);
            S = 1'b0;  R = 1'b1;
            #5 S = 1'b1;  R = 1'b1;
            #5 S = 1'b1;  R = 1'b0;
            #5 S = 1'b1;  R = 1'b1;
            #5 S = 1'b0;  R = 1'b0;
            #5 S = 1'b1;  R = 1'b1;
        end
    endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

13

### Simulation Output

```

0 S=0, R=1, Q=0, Qbar=1
5 S=1, R=1, Q=0, Qbar=1
10 S=1, R=0, Q=1, Qbar=0
15 S=1, R=1, Q=1, Qbar=0
20 S=0, R=0, Q=1, Qbar=1
and then the simulator hangs

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14

## END OF LECTURE 12



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 13: PROCEDURAL ASSIGNMENT

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Behavioral Style: Procedural Assignment

- Two kinds of procedural blocks are supported in Verilog:
  - The “initial” block
    - Executed once at the beginning of simulation.
    - Used only in test benches; cannot be used in synthesis.
  - The “always” block
    - A continuous loop that never terminates
- The procedural block defines:
  - A region of code containing *sequential* statements.
  - The statements execute in the order they are written.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

## The “initial” Block

- All statements inside an “initial” statement constitute an “initial block”.
  - Grouped inside a “begin ... end” structure for multiple statements.
  - The statements starts at time 0, and execute only once.
  - If there are multiple “initial” blocks, all the blocks will start to execute concurrently at time 0.
- The “initial” block is typically used to write test benches for simulation:
  - Specifies the stimulus to be applied to the design-under-test (DUT).
  - Specifies how the DUT outputs are to be displayed / handled.
  - Specifies the file where the waveform information is to be dumped.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18

```

module testbench_example;
    reg a, b, cin, sum, cout;

    initial
        cin = 1'b0;

    initial
        begin
            #5 a = 1'b1; b=1'b1;
            #5 b = 1'b0;
        end

    initial
        #25 $finish;

endmodule

```

- The three “initial” blocks execute concurrently.
- The first block executes at time 0.
- The third block terminates simulation at time 25 units.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

19

## Some Short Cuts in Declarations

- “output” and “reg” can be declared together in the same statement.

`output reg [7:0] data;`  
 instead of `output [7:0] data; reg [7:0] data;`

- A variable can be initialized when it is declared:

`reg clock = 0;`  
 instead of `reg clock; initial clock = 0;`



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

20

## The “always” Block

- All behavioral statements inside an “always” statement constitute an “always block”.
  - Multiple statements are grouped using “begin ... end”.
- An “always” statement starts at time 0 and executes the statements inside the block repeatedly, and never stops.
  - Used to model a block of activity that is repeated indefinitely in a digital circuit.
  - For example, a clock signal that is generated continuously.
  - We can specify delays for simulation; however, for real circuits, the clock generator will be active as long as there is power supply.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

21

```
module generating_clock;
  output reg clk;

  initial
    clk = 1'b0; // initialized to 0 at time 0

  always
    #5 clk = ~clk; // Toggle after time 5 units

  initial
    #500 $finish;
endmodule
```

- “initial” and “always” blocks can coexist within the same Verilog module.
- They all execute concurrently; “initial” only once and “always” repeatedly.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

22

- A module can contain any number of “always” blocks, all of which execute concurrently.
- The @(event\_expression) part is required for both combinational and sequential circuit descriptions.

***Basic syntax of “always” block:***

```
always @(event_expression)
begin
    sequential_statement_1;
    sequential_statement_2;
    ...
    sequential_statement_n;
end
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

23

- Only “reg” type variable can be assigned within an “initial” or ‘always’ block.
- Basic reason:
  - The sequential “always” block executes only when the event expression triggers.
  - At other times the block is doing nothing.
  - An object being assigned to must therefore remember the last value assigned (not continuously driven).
  - So, only “reg” type variables can be assigned within the “always” block.
  - Of course, any kind of variable may appear in the event expression (reg, wire, etc.).



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

24

## Sequential Statements in Verilog

- In Verilog, one or more sequential statements can be present inside an “initial” or “always” block.
  - The statements are executed sequentially.
  - Multiple assignment statements inside a “begin ... end” block may either execute sequentially or concurrently depending upon on the type of assignment.
    - Two types of assignment statements: blocking (`a = b + c;`) or non-blocking (`a <= b + c;`).
- The sequential statements are explained next.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

### (a) begin ... end

```
begin
  sequential_statement_1;
  sequential_statement_2;
  ...
  sequential_statement_n;
end
```

- A number of sequential statements can be grouped together using “begin .. end”.
- If n=1, “begin ... end” is not required.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

26

## (b) if ... else

```
if (<expression>)
    sequential_statement;
```

```
if (<expression>)
    sequential_statement;
else
    sequential_statement;
```

```
if (<expression1>)
    sequential_statement;
else if (<expression2>)
    sequential_statement;
else if (<expression3>)
    sequential_statement;
else default_statement;
```

- Each sequential\_statement can be a single statement or a group of statements within “begin ... end”.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27

## (c) case

```
case (<expression>)
    expr1: sequential_statement;
    expr2: sequential_statement;
    ...
    exprn: sequential_statement;
    default: default_statement;
endcase
```

- Each sequential\_statement can be a single statement or a group of statements within “begin ... end”.
- Can replace a complex “if ... else” statement for multiway branching.
- The expression is compared to the alternatives (expr1, expr2, etc.) in the order they are written.
- If none of the alternatives matches, the default statement is executed.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

- Two variations: “casez” and “casex”.
  - The “casez” statement treats all “z” values in the case alternatives or the case expression as don’t cares.
  - The “casex” statement treats all “x” and “z” values in the case item as don’t cares.

If state is “4'b01zx”, the second expression will give match, and next\_state will be 1.

```
reg [3:0] state; integer next_state;
casex (state)
  4'b1xxx : next_state = 0;
  4'bx1xx : next_state = 1;
  4'bxx1x : next_state = 2;
  4'bxxx1 : next_state = 3;
  default : next_state = 0;
endcase
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

29

## END OF LECTURE 13



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

30



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 14: PROCEDURAL ASSIGNMENT (CONTD.)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### (d) “while” loop

```
while (<expression>)
    sequential_statement;
```

- The “while” loop executes until the expression is *not true*.
- The sequential\_statement can be a single statement or a group of statements within “begin ... end”.

#### Example:

```
integer mycount;
initial
begin
    while (mycount <= 255)
        begin
            $display ("My count:%d", mycount);
            mycount = mycount + 1;
        end
    end
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

32

## (e) “for” loop

```
for (expr1; expr2; expr3)
    sequential_statement;
```

- The “for” loop executes as long as the expression expr2 is true.
- The sequential\_statement can be a single statement or a group of statements within “begin ... end”.

- The “for” loop consists of three parts:
  - a) An initial condition (expr1).
  - b) A check to see if the terminating condition is true (expr2).
  - c) A procedural assignment to change the value of the control variable (expr3).
- The “for” loop can be conveniently used to initialize an array or memory.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

33

### Example:

```
integer mycount;
reg [100:1] data;
integer i;

initial
    for (mycount=0; mycount<=255; mycount=mycount+1)
        $display ("My count:%d", mycount);

initial
    for (i=1; i<=100; i=i+1)
        data[i] = 1'b0;
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

34

## (f) “repeat” loop

```
repeat (<expression>)
    sequential_statement;
```

- The “repeat” construct executes the loop a fixed number of times.
- It cannot be used to loop on a general logical expression like “while”.

- The expression in the “repeat” construct can be a constant, a variable or a signal value.
  - If it is a variable or a signal value, it is evaluated only when the loop starts and not during execution of the loop.
- The sequential\_statement can be a single statement or a group of statements within “begin ... end”.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

### Example:

```
reg clock;
initial
begin
    clock = 1'b0;
    repeat (100)
        #5 clock = ~clock;
end
```

Exactly 100 clock pulses  
are generated.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36

## (g) “forever” loop

```
forever
    sequential_statement;
```

- The “forever” loop is typically used along with timing specifier.
  - If delay is not specified, the simulator would execute this statement indefinitely without advancing \$time.
  - Rest of design will never be executed.

- The “forever” construct does not use any expression and executes forever until \$finish is encountered in the test bench.
  - Equivalent to a “while” loop for which the expression is always true.
- The sequential\_statement can be a single statement or a group of statements within “begin ... end”.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

37

```
// Clock generation using “forever” construct
reg clk;

initial
  begin
    clk = 1'b0;
    forever #5 clk = ~clk; // Clock period of 10 units
  end
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

38

## Other Constructs Available

`# (time_value)`

- Makes a block suspend for “time\_value” units of time.
- The time unit can be specified using the `timescale command.

`@ (event_expression)`

- Makes a block suspend until “event\_expression” triggers.
- Various keywords associated with “event\_expression” shall be discussed with examples..



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

39

### @ (event\_expression)

- The event expression specifies the event that is required to resume execution of the procedural block.
- The event can be any one of the following:
  - Change of a signal value.
  - Positive or negative edge occurring on signal (*posedge* or *negedge*).
  - List of above-mentioned events, separated by “or” or comma.
- A “posedge” is any transition from {0, x, z} to 1, and from 0 to {z, x}.
- A “negedge” is any transition from {1, x, z} to 0, and from 1 to {z, x}.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

40

- Examples:

- @ (in) // “in” changes
- @ (a or b or c) // any of “a”, “b”, “c” changes
- @ (a, b, c) // -- do --
- @ (posedge clk) // positive edge of “clk”
- @ (posedge clk or negedge reset) // positive edge of “clk” or negative edge of “reset”
- @ (\*) // any variable changes



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41

```
// D flip-flop with synchronous set and reset
module dff (q, qbar, d, set, reset, clk);
    input d, set, reset, clk;
    output reg q;      output qbar;
    assign qbar = ~q;

    always @ (posedge clk)
        begin
            if (reset == 0) q <= 0;
            else if (set == 0) q <= 1;
            else q <= d;
        end
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

```
// D flip-flop with asynchronous set and reset
module dff (q, qbar, d, set, reset, clk);
    input d, set, reset, clk;
    output reg q;      output qbar;
    assign qbar = ~q;

    always @ (posedge clk or negedge set or negedge reset)
        begin
            if (reset == 0) q <= 0;
            else if (set == 0) q <= 1;
            else q <= d;
        end
    endmodule
```



```
// Transparent latch with enable
module latch (q, qbar, din, enable);
    input din, enable;
    output reg q;      output qbar;
    assign qbar = ~q;

    always @ (din or enable)
        begin
            if (enable) q = din;
        end
    endmodule
```



## END OF LECTURE 14



The image shows a blue header bar with white text. On the left is the IIT Kharagpur logo. Next to it is the NPTEL logo. Below the logos, the lecture title 'Lecture 15: PROCEDURAL ASSIGNMENT (EXAMPLES)' is displayed in a large, bold, dark blue font. A red horizontal bar is at the bottom of the slide.

**PROF. INDRANIL SENGUPTA**  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

```
// A combinational logic example
module mux21 (in1, in0, s, f);
    input in1, in0, s;
    output reg f;

    always @(in1 or in0 or s)
        if (s)
            f = in1;
        else
            f = in0;
endmodule
```

- The event expression in the “always” block triggers whenever at least one of “in1”, “in0” or “s” changes.
- The “or” keyword specifies the condition.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

47

```
// A combinational logic example
module mux21 (in1, in0, s, f);
    input in1, in0, s;
    output reg f;

    always @(*)
        if (s)
            f = in1;
        else
            f = in0;
endmodule
```

- An alternate way to specify the event condition by using comma instead of “or”.
- Supported in later versions of Verilog.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

48

```
// A combinational logic example
module mux21 (in1, in0, s, f);
    input in1, in0, s;
    output reg f;

    always @(*)
        if (s)
            f = in1;
        else
            f = in0;
endmodule
```

- An alternate way to specify the event condition by using a "\*" instead of naming the variables.
- "\*" is activated whenever *any* of the variables change.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

49

```
// A sequential logic example
module dff_nededge (D, clock, Q, Qbar);
    input D, clock;
    output reg Q, Qbar;

    always @(negedge clock)
        begin
            Q = D;
            Qbar = ~D;
        end
endmodule
```

- The keyword "negedge" means at the negative going edge of the specified signal.
- Similarly, we can use "posedge".
- We can combine various triggering conditions by separating them by commas or "or".



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

50

```
// 4-bit counter with asynchronous
reset

module counter (clk, rst, count);

    input clk, rst;
    output reg [3:0] count;

    always @(posedge clk or posedge rst)
    begin
        if (rst)
            count <= 0;
        else
            count <= count + 1;
    end
endmodule
```

The event condition triggers when either a positive edge of "clk" comes, or a positive edge of "rst".



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

51

```
// Another sequential logic example

module incomp_state_spec (curr_state, flag);
    input [0:1] curr_state;
    output reg [0:1] flag;

    always @(curr_state)
        case (curr_state)
            0,1 : flag = 2;
            3    : flag = 0;
        endcase
endmodule
```

The variable "flag" is not assigned a value in all the branches of the "case" statement.

- A latch (2-bit) will be generated for "flag".



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

52

```
// A small modification

module incomp_state_spec (curr_state, flag);
    input [0:1] curr_state;
    output reg [0:1] flag;

    always @(curr_state)
    begin
        flag = 0;
        case (curr_state)
            0,1 : flag = 2;
            3    : flag = 0;
        endcase
    end
endmodule
```

Here the variable “flag” is defined for all the possible values of “curr\_state”.

- A pure combinational circuit will be generated.
- The latch is avoided.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

53

- When a “case” statement is incompletely decoded, the synthesis tool will infer the need for a latch to hold the residual output when the select bits take the unspecified values.
  - It is up to the designer to code the design in such a way that latch can be avoided where possible.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

54

```
// A simple 4-function ALU
module ALU_4bit (f, a, b, op);
    input [1:0] op;      input [7:0] a, b;
    output reg [7:0] f;
    parameter ADD=2'b00, SUB=2'b01, MUL=2'b10, DIV=2'b11;
    always @(*)
        case (op)
            ADD : f = a + b;
            SUB : f = a - b;
            MUL : f = a * b;
            DIV : f = a / b;
        endcase
    endmodule
```



```
module priority_encoder (in, code);
    input [7:0] in;
    output reg [2:0] code;
    always @(in)
        begin
            if (in[0]) code = 3'b000;
            else if (in[1]) code = 3'b001;
            else if (in[2]) code = 3'b010;
            else if (in[3]) code = 3'b011;
            else if (in[4]) code = 3'b100;
            else if (in[5]) code = 3'b101;
            else if (in[6]) code = 3'b110;
            else if (in[7]) code = 3'b111;
            else           code = 3'bxxx;
        end
    endmodule
```

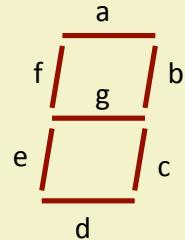
- The inputs bits are checked sequentially one by one (in order of priority).
  - “in[0]” has the highest priority.
  - For simultaneously active inputs, the first active input encountered will be encoded.



```

module bcd_to_7seg (bcd, seg);
    input [3:0] bcd;
    output reg [6:0] seg;
    always @(bcd)
        case
            0: seg = 6'b0000001;
            1: seg = 6'b1001111;
            2: seg = 6'b0010010;
            3: seg = 6'b0000110;
            4: seg = 6'b1001100;
            5: seg = 6'b0100100;
            6: seg = 6'b0100000;
            7: seg = 6'b0001111;
            8: seg = 6'b0000000;
            9: seg = 6'b0000100;
        default : seg = 6'b1111111;
    endcase
endmodule

```



Segment bit assignment:  
(a, b, c, d, e, f, g)

*A segment glows when the corresponding bit of seg is 0.*



NPTEL ONLINE  
CERTIFICATION COURSES

```

// An n-bit comparator
module compare (A, B, lt, gt, eq);
    parameter word_size = 16;
    input [word_size-1:0] A, B;
    output reg lt, gt, eq;

    always @ (*)
        begin
            gt = 0; lt = 0; eq = 0;
            if (A > B) gt = 1;
            else if (A < B) lt = 1;
            else eq = 1
        end
    endmodule

```

For actual synthesis, it is common to have a structured design representation of the comparator.



NPTEL ONLINE  
CERTIFICATION COURSES

```
// A 2-bit comparator
module compare (A1, A0, B1, B0, lt, gt, eq);
    input A1, A0, B1, B0;
    output reg lt, gt, eq;

    always @ (A1, A0, B1, B0)
        begin
            lt = ({A1,A0} < {B1,B0});
            gt = ({A1,A0} > {B1,B0});
            eq = ({A1,A0} == {B1,B0});
        end
endmodule
```



```
module alu_example (alu_out, A, B, operation, en);
    input [2:0] operation;    input [7:0] A, B;
    input en;
    output [7:0] alu_out;    reg [7:0] alu_reg;

    assign alu_out = (en == 1) ? alu_reg : 4'bzz;
    always @ (*)
        case (operation)
            3'b000 : alu_reg = A + B;
            3'b001 : alu_reg = A - B;
            3'b011 : alu_reg = ~ A;
            default : alu_reg = 4'b0;
        endcase
endmodule
```



## END OF LECTURE 15



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

NPTEL



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 16: BLOCKING / NON-BLOCKING ASSIGNMENTS (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Procedural Assignment

- Procedural assignment statements can be used to update variables of types “reg”, “integer”, “real” or “time”.
- The value assigned to a variable remains unchanged until another procedural assignment statement assigns a new value to the variable.
  - This is different from continuous assignment (using “assign”) that results in the expression on the RHS to continuously drive the “net” type variable on the left.
- Two types of procedural assignment statements:
  - a) **Blocking** (denoted by “=“)
  - b) **Non-blocking** (denoted by “<=“)



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

- The left-hand side of a procedural assignment statement can be one of:
  - A register type variable (“reg”, “integer”, “real”, or “time”)
  - A bit select of these variables (e.g. sum[15])
  - A part select of these variables (e.g. IR[31:26])
  - A concatenation of any of the above
- The right-hand side can be any expression consisting of “net” and “register” type variables that evaluates to a value.
- Procedural assignment statements can only appear within procedural blocks (“initial” or “always”).



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

## (i) Blocking Assignment

- General syntax:
 

```
variable_name = [delay_or_event_control] expression;
```
- The “=” operator is used to specify blocking assignment.
- Blocking assignment statements are executed in the order they are specified in a procedural block.
  - The target of an assignments gets updated before the next sequential statement in the procedural block is executed.
  - They do not block execution of statements in other procedural blocks.
- This is the recommended style for modeling combinational logic.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

- Blocking assignments can also generate sequential circuit elements during synthesis (e.g. incomplete specification in multi-way branching with “case”).
- An example of blocking assignment:

```
integer a, b, c;
initial
begin
    a = 10; b = 20; c = 15;
    a = b + c;
    b = a + 5;
    c = a - b;
end
```

- Initially, a=10, b=20, c=15
- a becomes 35
- b becomes 40
- c becomes -5



```
Module blocking_example;
reg X, Y, Z;
reg [31:0] A, B;      integer sum;

initial
begin
    X = 1'b0;  Y = 1'b0;  Z = 1'b1;      // At time = 0
    sum = 1;                // At time = 0
    A = 31'b0;  B = 31'habababab;        // At time = 0
    #5 A[5] = 1'b1;            // At time = 5
    #10 B[31:29] = {X, Y, Z};        // At time = 15
    sum = sum + 5;            // At time = 15
end
endmodule
```



## Simulation of an Example

```
module blocking_assgn;
integer a, b, c, d;
always @ (*)
repeat (4)
begin
#5 a = b + c;
#5 d = a - 3;
#5 b = d + 10;
#5 c = c + 1;
end
```

```
initial
begin
$monitor ($time, "a=%4d, b=%4d,
c=%4d, d=%4d", a, b, c, d);
a = 30; b = 20; c = 15; d = 5;
#100 $finish;
end
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

```
0 a= 30, b= 20, c= 15, d= 5
5 a= 35, b= 20, c= 15, d= 5
10 a= 35, b= 20, c= 15, d= 32
15 a= 35, b= 42, c= 15, d= 32
20 a= 35, b= 42, c= 16, d= 32
25 a= 58, b= 42, c= 16, d= 32
30 a= 58, b= 42, c= 16, d= 55
35 a= 58, b= 65, c= 16, d= 55
40 a= 58, b= 65, c= 17, d= 55
45 a= 82, b= 65, c= 17, d= 55
50 a= 82, b= 65, c= 17, d= 79
55 a= 82, b= 89, c= 17, d= 79
60 a= 82, b= 89, c= 18, d= 79
65 a= 107, b= 89, c= 18, d= 79
70 a= 107, b= 89, c= 18, d= 104
75 a= 107, b= 114, c= 18, d= 104
80 a= 107, b= 114, c= 19, d= 104
```

### Simulation Results

Initially:  
 $a=30, b=20, c=15, d=5$

```
always @ (*)
repeat (4)
begin
#5 a = b + c;
#5 d = a - 3;
#5 b = d + 10;
#5 c = c + 1;
end
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

## (ii) Non-Blocking Assignment

- General syntax:

```
variable_name <= [delay_or_event_control] expression;
```

- The “`<=`” operator is used to specify non-blocking assignment.
- Non-blocking assignment statements allow scheduling of assignments without blocking execution of statements that follow within the procedural block.
  - The assignment to the target gets scheduled for the end of the simulation cycle (at the end of the procedural block).
  - Statements subsequent to the instruction under consideration are not blocked by the assignment.
  - Allows concurrent procedural assignment, suitable for sequential logic.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

9

- This is the recommended style for modeling sequential logic.
  - Several “`reg`” type variables can be assigned synchronously, under the control of a common clock.

```
integer a, b, c;
initial
begin
  a = 10; b = 20; c = 15;
end
initial
begin
  a <= #5 b + c;
  b <= #5 a + 5;
  c <= #5 a - b;
end
```

- Initially,  $a=10$ ,  $b=20$ ,  $c=15$
- $a$  becomes 35 at time = 5
- $b$  becomes 15 at time = 5
- $c$  becomes -10 at time = 5

All the right hand side expressions are evaluated together based on the previous values of “ $a$ ”, “ $b$ ” and “ $c$ ”. They are assigned together at time 5.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

10

```
always @(posedge clk)
begin
    a <= b & c;
    b <= a ^ d;
    c <= a | b;
end
```

Recommended style for modeling synchronous circuits, where assignments take place in synchronism with clock.

All assignments take place synchronously at the rising edge of the clock.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

11

## Swapping values of two variables “a” and “b”

```
always @(posedge clk)
    a = b;
always @(posedge clk)
    b = a;
```

- Either  $a=b$  will execute before  $b=a$ , or vice versa, depending on simulator implementation.
- Both registers will get the same value (either “a” or “b”).
  - Race condition.*

```
always @(posedge clk)
    a <= b;
always @(posedge clk)
    b <= a;
```

- Here the variables are correctly swapped.
- All RHS variables are read first, and assigned to LHS variables at the positive clock edge.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

12

## Trying to swap using blocking assignment

```
always @(posedge clk)
begin
    a = b;
    b = a;
end
```

- Both "a" and "b" will be getting the value previously stored in "b".

```
always @(posedge clk)
begin
    ta = a;
    tb = b;
    a = tb;
    b = ta;
end
```

- Correct swapping will occur, but we need two temporary variables "ta" and "tb"



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

13

## Simulation of an Example

```
module nonblocking_assgn;
integer a, b, c, d;
reg clock;
always @ (posedge clock)
begin
    a <= b + c;
    d <= a - 3;
    b <= d + 10;
    c <= c + 1;
end
```

```
initial
begin
    $monitor ($time, "a=%4d, b=%4d,
c=%4d, d=%4d", a, b, c, d);
    a = 30; b = 20; c = 15; d = 5;
    clock = 0;
    forever #5 clock = ~clock;
end
initial
#100 $finish;
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14

### Simulation Results

```

0 a= 30, b= 20, c= 15, d= 5
5 a= 35, b= 15, c= 16, d= 27
15 a= 31, b= 37, c= 17, d= 32
25 a= 54, b= 42, c= 18, d= 28
35 a= 60, b= 38, c= 19, d= 51
45 a= 57, b= 61, c= 20, d= 57
55 a= 81, b= 67, c= 21, d= 54
65 a= 88, b= 64, c= 22, d= 78
75 a= 86, b= 88, c= 23, d= 85
85 a= 111, b= 95, c= 24, d= 83
95 a= 119, b= 93, c= 25, d= 108

```

```

Initially:
a=30, b=20, c=15, d=5
always @ (posedge clock)
begin
    a <= b + c;
    d <= a - 3;
    b <= d + 10;
    c <= c + 1;
end

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15

## Some Rules to be Followed

- It is recommended that blocking and non-blocking assignments *are not mixed* in the same “always” block.
  - Simulator may allow, but this is not good design practice.
- Verilog synthesizer ignores the delays specified in a procedural assignment statement (blocking or non-blocking).
  - May lead to functional mismatch between the design model and the synthesized netlist.
- A variable cannot appear as the target of both a blocking and a non-blocking assignment.
  - This is not permissible → 

```
x = x + 5;
x <= y;
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

## END OF LECTURE 16



## Lecture 17: BLOCKING / NON-BLOCKING ASSIGNMENTS (PART 2)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Introduction

- We shall be looking at some examples of modeling using blocking and non-blocking assignments.
- Objective is to get a feel of the type of assignment statement to use for some particular scenario.
- Avoid some of the “not-so-good” design practices in modeling.



```
// 8-to-1 multiplexer: behavioral description
module mux_8to1 (in, sel, out);
    input [7:0] in;    input [2:0] sel;
    output reg out;
    always @(*)
        begin
            case (sel)
                3'b000: out = in[0];
                3'b001: out = in[1];
                3'b010: out = in[2];
                3'b011: out = in[3];
                3'b100: out = in[4];
                3'b101: out = in[5];
                3'b110: out = in[6];
                3'b111: out = in[7];
                default: out = 1'bx;
            endcase
        end
endmodule
```



```
// Up-down counter (synchronous clear)
module counter (mode, clr, ld, d_in, clk, count);
    input mode, clr, ld, clk;
    input [0:7] d_in;
    output reg [0:7] count;

    always @ (posedge clk)
        if (ld)         count <= d_in;
        else if (clr)  count <= 0;
        else if (mode) count <= count + 1;
        else           count <= count - 1;
endmodule
```



Make a design general for any number of bits.

Using the keyword “*parameter*”.

- Parameter values are substituted before simulation or synthesis.

```
// Parameterized design:: an N-bit counter
module counter (clear, clock, count);
    parameter N = 7;
    input clear, clock;
    output reg [0:N] count;

    always @(negedge clock)
        if (clear)
            count <= 0;
        else
            count <= count + 1;
endmodule
```



```
// Using more than one clocks in a module

module multiple_clk (clk1, clk2, a, b, c, f1, f2);
    input clk1, clk2, a, b, c;
    output reg f1, f2;

    always @(posedge clk1)
        f1 <= a & b;
    always @(negedge clk2)
        f2 <= b ^ c;
endmodule
```



```
// Using multiple edges of the same clock

module multi_edge_clk (a, b, f, clk);
    input a, b, clk;
    output reg f; reg t;

    always @(posedge clk)
        f <= t & b;
    always @(negedge clk)
        t <= a | b;
endmodule
```



```
// Another example

module multi_edge_clk (a, b, c, d, f, clk);
    input a, b, clk;
    output reg f;  reg t;
    always @(posedge clk)
        c <= a + b;
    always @(negedge clk)
        f <= c - d;
endmodule
```

- Two operations are carried out every clock cycle.
  - “c” is assigned at the rising edge.
  - “f” is assigned at the falling edge.
- It is assumed that addition or subtraction can be completed in half a clock cycle.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

```
// A ring counter

module ring_counter (clk, init, count);
    input clk, init;
    output reg [7:0] count;
    always @ (posedge clk)
        begin
            if (init) count = 8'b10000000;
            else begin
                count = count << 1;
                count[0] = count[7];
            end
        end
    end
endmodule
```

- This solution is wrong.
- count[7] will get overwritten in the first statement.
  - Rotation of the bits will not happen.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

26

```
// A ring counter (Modified version 1)
module ring_counter (clk, init, count);
    input clk, init;
    output reg [7:0] count;
    always @ (posedge clk)
    begin
        if (init) count = 8'b10000000;
        else begin
            count <= count << 1;
            count[0] <= count[7];
        end
    end
endmodule
```

- This is the correct version.
- Since non-blocking assignments are used, rotation will take place correctly.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27

```
// A ring counter (Modified version 2)
module ring_counter (clk, init, count);
    input clk, init;
    output reg [7:0] count;
    always @ (posedge clk)
    begin
        if (init) count = 8'b10000000;
        else
            count = {count[6:0], count[7]};
    end
endmodule
```

- This is a correct way of modeling using blocking assignment.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

## END OF LECTURE 17



## Lecture 18: BLOCKING / NON-BLOCKING ASSIGNMENTS (PART 3)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Blocking vrs. Non-blocking Assignments

- We shall now illustrate some examples that show how the modeling style influences the simulator or synthesizer to capture the behavior of the modeled circuit.
  - Very important concept required to be clearly understood by the designer.
  - Even a slight error in modeling can result in a drastically different circuit.
- Highly recommended:
  - For any confusion, write a Verilog code, simulate it and analyze the output(s).



### Example 1

```
begin
  a = #5 b;
  c = #5 a;
end
```

```
begin
  a <= #5 b;
  c <= #5 a;
end
```

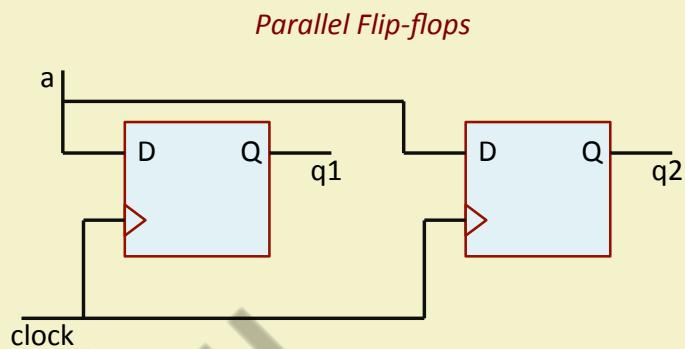
- The value of “b” will be assigned to “c” 10 time units after the “begin ... end” block starts.
- “a” is scheduled to get the value of “b” 5 time units into the future.
- “c” is also scheduled to get the value of “a” 5 time units into the future.

*Value of “c” will be different for the two cases*



## Example 2

```
always @(posedge clock)
begin
    q1 = a;
    q2 = q1;
end
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

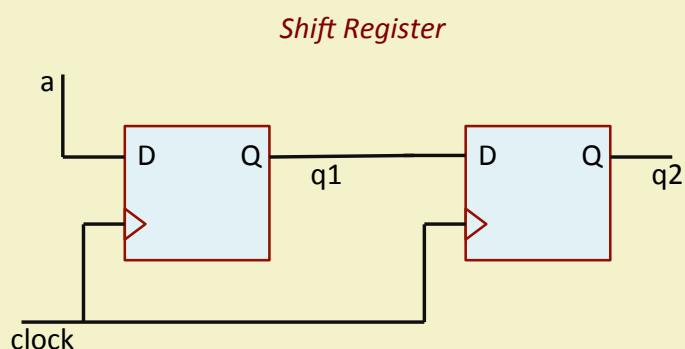
Hardware Modeling Using Verilog

33

## Example 3

```
always @(posedge clock)
begin
    q2 = q1;
    q1 = a;
end
```

```
always @(posedge clock)
begin
    q1 <= a;
    q2 <= q1;
end
```



IIT KHARAGPUR

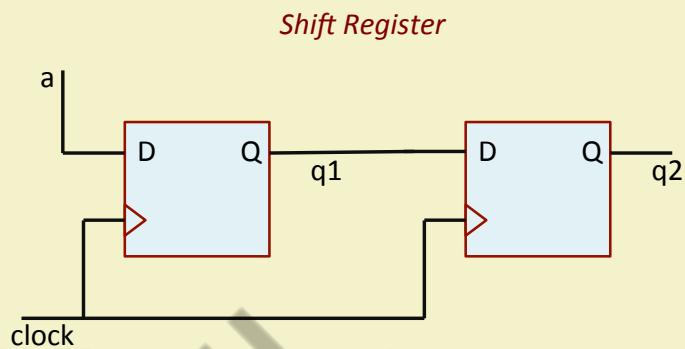
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

34

### Example 4

```
always @(posedge clock)
  q2 <= q1;
always @(posedge clock)
  q1 <= a;
```



IIT KHARAGPUR

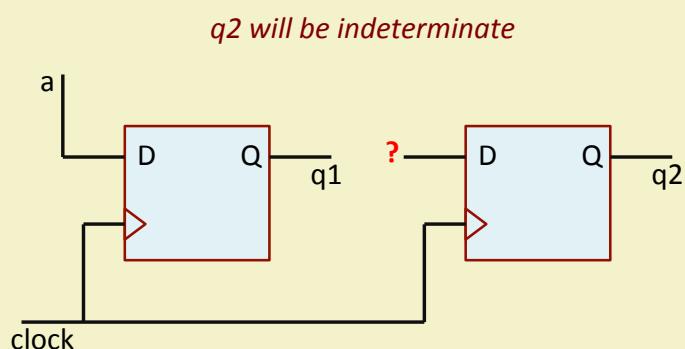
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

### Example 5

```
always @(posedge clock)
  q1 = a;
always @(posedge clock)
  q2 = q1;
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36

## Example 6

What circuit will the synthesis tool generate?

```
module shiftreg_4bit (clock, clear, A, E);
    input clock, clear, A;
    output reg E;
    reg B, C, D;

    always @(posedge clock or negedge clear)
        begin
            if (!clear) begin B=0; C=0; D=0; E=0; end
            else begin
                E = D;
                D = C;
                C = B;
                B = A;
            end
        end
    endmodule
```

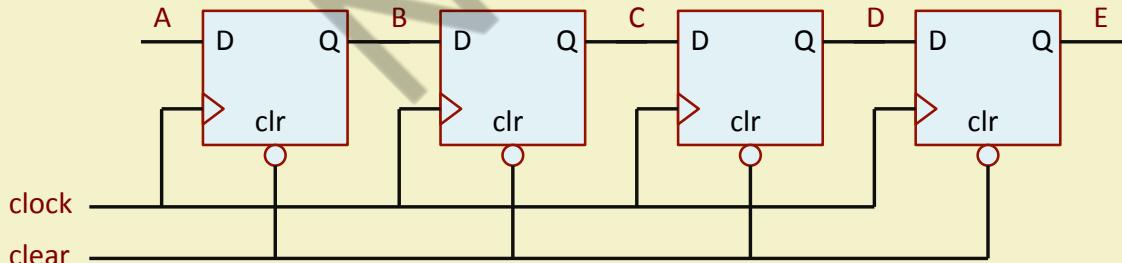


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

37



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

38

### Example 6a

We just reverse the order of the procedural assignments.

What circuit will the synthesis tool generate?

```
module shiftreg_4bit (clock, clear, A, E);
    input clock, clear, A;
    output reg E;
    reg B, C, D;

    always @(posedge clock or negedge clear)
        begin
            if (!clear) begin B=0; C=0; D=0; E=0; end
            else begin
                B = A;
                C = B;
                D = C;
                E = D;
            end
        end
    endmodule
```



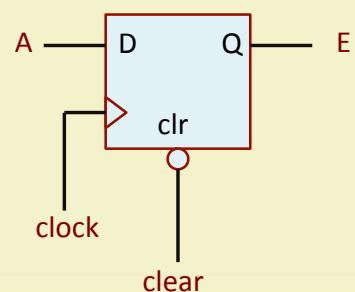
IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

39

- The effect of the assignment made by the first statement ( $B = A$ ) is immediate.
- Thus,  $B$  changes, and the updated value is used in the second statement ( $C = B$ ).
- The updated value of  $C$  is used in the third statement ( $D = C$ ).
- The updated value of  $D$  is used in the fourth statement ( $E = D$ ).
- The statements execute sequentially.
  - But at the same time step of the simulator.
  - The four statements are equivalent to a single statement that assigns  $A$  to  $E$ .



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

40

### Example 6b

Recommended style for modeling sequential circuits.

- Statements can appear in any order.
- Chances of errors are less.

```
module shiftreg_4bit (clock, clear, A, E);
    input clock, clear, A;
    output reg E;
    reg B, C, D;

    always @(posedge clock or negedge clear)
        begin
            if (!clear) begin B<=0; C<=0; D<=0; E<=0; end
            else begin
                E <= D;
                D <= C;
                C <= B;
                B <= A;
            end
        end
    endmodule
```

The right-hand side expressions are evaluated in parallel, so that order of the statements is not important.

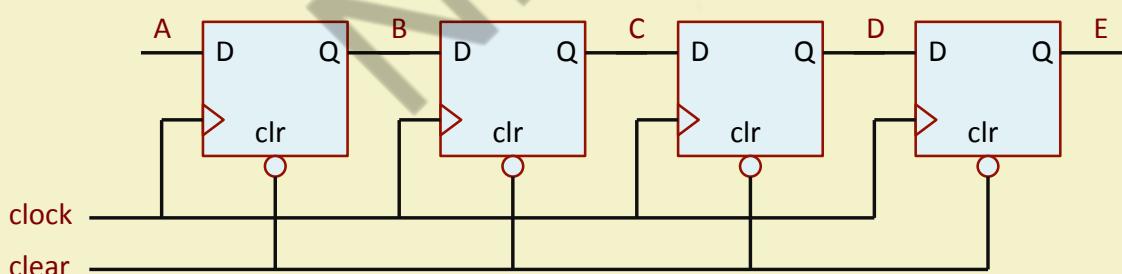


IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41



IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

## END OF LECTURE 18



## Lecture 19: BLOCKING / NON-BLOCKING ASSIGNMENTS (PART 4)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## MIXING BLOCKING AND NON-BLOCKING ASSIGNMENTS IN A PROCEDURAL BLOCK

***NOT RECOMMENDED***



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

45

### Basic Idea

- It is possible to combine both blocking and non-blocking assignments in the same procedural block (viz. “always”).
  - Simulator or synthesis tool supports this type of usage.
- However, interpretation of the circuit behavior under such mixed usage is not very straightforward.
  - Not recommended for designers.
  - We shall explain the semantics using two simple examples.
- Such mixing should be avoided in a design.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

46

## Example 1

```
always @(*)
begin
    x = 10;
    x = 20;
    y = x;
    #10;
end
```

```
always @(*)
begin
    x = 10;
    x = 20;
    y <= x;
    #10;
end
```

- Same result will be shown for both the versions:
  - “x” will be assigned 10 and then 20, both at time 0.
  - The value of “x” at time 0 will be assigned to “y”.

$x = 20, y = 20$



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

47

## Example 2

```
always @(*)
begin
    x = 10;
    y = x;
    x = 20;
    #10;
end
```

```
always @(*)
begin
    x = 10;
    y <= x;
    x = 20;
    #10;
end
```

- Same result will be shown for both the versions:
  - “y” will be assigned 10 at time 0.
  - The final value of “x” will be 20.

$x = 20, y = 10$



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

48

## Generate Blocks

- “*generate*” statements allow Verilog code to be generated dynamically before the simulation or synthesis begins.
  - Very convenient to create parameterized module descriptions.
  - Example: N-bit ripple carry adder for arbitrary value of N.
- Requires the keywords “*generate*” and “*endgenerate*”.
- Generate instantiations can be carried out for various Verilog blocks:
  - Modules, user-defined primitives, gates, continuous assignments, “initial” and “always” blocks, etc.
  - Generated instances have unique identifier names and can be referenced hierarchically.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

49

- Special “*genvar*” variables:
  - The keyword “*genvar*” can be used to declare variables that are used only in the evaluation of generate block.
  - These variables do not exist during simulation or synthesis.
  - The value of a “*genvar*” can be defined only in a generate loop.
  - Every generate loop is assigned a name, so that variables inside the generate loop can be referenced hierarchically.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

50

## Example 1

```
module xor_bitwise (f, a, b);
    parameter N = 16;
    input [N-1:0] a, b;
    output [N-1:0] f;
    genvar p;

    generate for (p=0; p<N; p=p+1)
        begin xorlp
            xor XG (f[p], a[p], b[p]);
        end
    endgenerate
endmodule
```

```
module generate_test;

reg [15:0] x, y;
wire [15:0] out;

xor_bitwise G (.f(out),.a(x),.b(y));

initial
begin
    $monitor ("x: %b, y: %b,
              Out: %b", x, y, out);
    x = 16'haaaa; y = 16'h00ff;
    #10 x = 16'h0f0f; y = 16'h3333;
    #20 $finish;
end
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

51

## Simulation Results

```
x: 1010101010101010, y: 0000000011111111, Out: 1010101001010101
x: 0000111100001111, y: 0011001100110011, Out: 0011110000111100
```

- In the bitwise xor example, the name “xorlp” was given to the generate loop.
- The relative hierarchical names of the xor gates will be:

`xorlp[0].XG, xorlp[1].XG, ..., xorlp[15].XG`



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

52

## Example 2: Design of N-bit Ripple Carry Adder

```
// Structural gate-level description of a full adder
module full_adder (a, b, c, sum, cout);
    input a, b, c;
    output sum, cout;
    wire t1, t2, t3;
    xor G1 (t1, a, b), G2 (sum, t1, c);
    and G3 (t2, a, b), G4 (t3, t1, c);
    or G5 (cout, t2, t3);
endmodule
```

*How to use “generate” to dynamically create N copies of full adder, and connect them to make a N-bit ripple-carry adder?*



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

53

```
module RCA (carry_out, sum, a, b, carry_in);
parameter N = 8;
input [N-1:0] a, b;    input carry_in;
output [N-1:0] sum,    output carry_out;
wire [N:0] carry; // carry[N] is carry out
assign carry[0] = carry_in;
assign carry_out = carry[N];
genvar i;
generate for (i=0; i<N; i++)
begin fa_loop
    wire t1, t2, t3;
    xor G1 (t1, a[i], b[i]), G2 (sum[i], t1, carry[i]);
    and G3 (t2, a[i], b[i]), G4 (t3, t1, carry[i]);
    or G5 (carry[i+1], t2, t3);
end
endgenerate
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

54

- Some of the relative hierarchical instance names that are generated are:
  - fa\_loop[0].G1, fa\_loop[1].G1, fa\_loop[7].G1, etc.
- Some of the nets (“wires”) that are generated are:
  - fa\_loop[0].t1, fa\_loop[1].T2, fa\_loop[0].t3, etc.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

55

## END OF LECTURE 19



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

56



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 20: USER-DEFINED PRIMITIVES

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### User Defined Primitives (UDP)

- They are used to define custom Verilog primitives by the use of lookup tables.
- They can specify:
  - Truth table for combinational functions.
  - State table for sequential functions.
  - Don't care, rising and falling edges, etc. can be specified.
- For combinational functions, truth table entries are specified as:  

$$<\text{input1}> <\text{input2}> \dots <\text{inputN}> : <\text{output}>;$$
- For sequential functions, state table entries are specified as:  

$$<\text{input1}> <\text{input2}> \dots <\text{inputN}> : <\text{present\_state}> : <\text{next\_state}>$$



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

58

## Some Rules for using UDPs

- The input terminals to a UDP can only be scalar variables.
  - Multiple input terminals can be used.
  - The input terminals are declared as “*input*”.
  - Input entries in the table must be in the same order as the “*input*” terminal list.
- Only one scalar output terminal must be used.
  - The output terminal must appear in the beginning of the terminal list.
  - For combinational UDPs, the output terminal is declared as “*output*”.
  - For sequential UDPs, the output terminal is declared as “*reg*”.
- For sequential UDPs, the state can be initialized with an “*initial*” statement.
  - This is optional.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

59

## Some Guidelines

- User defined primitives (UDPs) model functionality only.
  - They do not model timing or process technology.
- A functional block can be modeled as a UDP only if it has exactly one output.
  - If a block has more than one outputs, it has to be modeled as a module.
  - As an alternative, multiple UDPs can be used, one per output.
- Inside the simulator, a UDP is typically implemented as a lookup table in memory.
- The UDP state tables should be specified as completely as possible.
  - For unspecified cases, the output is set to “x”.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

60

# MODELING COMBINATIONAL CIRCUITS



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

```
// Full adder sum generation using UDP
primitive udp_sum (sum, a, b, c);
    input a, b, c;
    output sum;
    table
        // a  b  c      sum
        // 0  0  0      :  0;
        0  0  1      :  1;
        0  1  0      :  1;
        0  1  1      :  0;
        1  0  0      :  1;
        1  0  1      :  0;
        1  1  0      :  0;
        1  1  1      :  1;
    endtable
endprimitive
```

The truth table is specified for all input combinations.

We can also specify don't care input combinations as "?".



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

62

```
// Full adder carry generation
primitive udp_cy (cout, a, b, c);
  input a, b, c;
  output cout;
  table
    // a   b   c   cout
    0   0   0   : 0;
    0   0   1   : 0;
    0   1   0   : 0;
    0   1   1   : 1;
    1   0   0   : 0;
    1   0   1   : 1;
    1   1   0   : 1;
    1   1   1   : 1;
  endtable
endprimitive
```

```
// Full adder carry generation
// Using don't care ("?")
primitive udp_cy (cout, a, b, c);
  input a, b, c;
  output cout;
  table
    // a   b   c   cout
    0   0   ?   : 0;
    0   ?   0   : 0;
    ?   0   0   : 0;
    1   1   ?   : 1;
    1   ?   1   : 1;
    ?   1   1   : 1;
  endtable
endprimitive
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

63

```
// Instantiating UDP's
// A full adder description

module full_adder (sum, cout, a, b, c);
  input a, b, c;
  output sum, cout;

  udp_sum  SUM  (sum, a, b, c);
  udp_cy   CARRY (cout, a, b, c);
endmodule
```

UDPs can be instantiated just like any other Verilog module.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

64

```
// A 4-input AND function
primitive udp_and4 (f, a, b, c, d);
  input a, b, c, d;
  output f;
  table
    // a b c d      f
    0 ? ? ? : 0;
    ? 0 ? ? : 0;
    ? ? 0 ? : 0;
    ? ? ? 0 : 0;
    1 1 1 1 : 1;
  endtable
endprimitive

// A 4-input OR function
primitive udp_or4 (f, a, b, c, d);
  input a, b, c, d;
  output f;
  table
    // a b c d      f
    1 ? ? ? : 1;
    ? 1 ? ? : 1;
    ? ? 1 ? : 1;
    ? ? ? 1 : 1;
    0 0 0 0 : 0;
  endtable
endprimitive
```



```
// A 4-to-1 multiplexer
primitive udp_mux41 (f, s0, s1, i0, i1, i2, i3);
  input s0, s1, i0, i1, i2, i3;
  output f;
  table
    // s0 s1      i0 i1 i2 i3 : f
    0 0      0 ? ? ? : 0;
    0 0      1 ? ? ? : 1;
    1 0      ? 0 ? ? : 0;
    1 0      ? 1 ? ? : 1;
    0 1      ? ? 0 ? : 0;
    0 1      ? ? 1 ? : 1;
    1 1      ? ? ? 0 : 0;
    1 1      ? ? ? 1 : 1;
  endtable
endprimitive
```



# MODELING SEQUENTIAL CIRCUITS



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

67

```
// A level-sensitive D type latch

primitive Dlatch (q, d, clk, clr);
    input d, clk, clr;
    output reg q;

    initial
        q = 0;      // This is optional

    table
        //  d  clk  clr      q   q_new
        ?  ?  1  :  ?  :  0;    // latch is cleared
        0  1  0  :  ?  :  0;    // latch is reset
        1  1  1  :  ?  :  1;    // latch is set
        ?  0  0  :  ?  :  -;    // retains previous state
    endtable
endprimitive
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

68

```
// A T flip-flop

primitive TFF (q, clk, clr);
    input clk, clr;
    output reg q;

    table
        // clk  clr      q   q_new
        ?    1       : ?  : 0;      // FF is cleared
        ?  (10)  : ?  : -;      // ignore -ve edge of "clr"
        (10) 0     : 1  : 0;      // FF toggles at -ve edge of "clk"
        (10) 0     : 0  : 1;      // - do -
        (0?) 0    : ?  : -;      // ignore +ve edge of "clk"
    endtable
endprimitive
```



```
// Constructing a 6-bit ripple counter using T flip-flops

module ripple_counter (count, clk, clr);
    input clk, clr;
    output [5:0] count;

    TFF F0 (count[0], clk, clr);
    TFF F1 (count[1], count[0], clr);
    TFF F2 (count[2], count[1], clr);
    TFF F3 (count[3], count[2], clr);
    TFF F4 (count[4], count[3], clr);
    TFF F5 (count[5], count[4], clr);
endmodule
```



```
// A negative edge sensitive JK flip-flop
primitive JKFF (q, j, k, clk, clr);
    input j, k, clk, clr;
    output reg q;

    table
        // j   k   clk  clr      q   q_new
        ?   ?   ?   1   : ?   : 0;   // clear
        ?   ?   ?   (10) : ?   : -;  // ignore .. no change
        0   0   (10) 0   : ?   : -;  // no change
        0   1   (10) 0   : ?   : 0;  // reset condition
        1   0   (10) 0   : ?   : 1;  // set condition
        1   1   (10) 0   : 0   : 1;  // toggle condition
        1   1   (10) 0   : 1   : 0;  // toggle condition
        ?   ?   (01) 0   : ?   : -;  // no change
    endtable
endprimitive
```



```
// A positive edge sensitive SR flip-flop
Primitive SRFF (q, s, r, clk, clr);
    input s, r, clk, clr;
    output reg q;

    table
        // s   r   clk  clr      q   q_new
        ?   ?   ?   1   : ?   : 0;   // clear
        ?   ?   ?   (10) : ?   : -;  // ignore .. no change
        0   0   (01) 0   : ?   : -;  // no change
        0   1   (01) 0   : ?   : 0;  // reset condition
        1   0   (01) 0   : ?   : 1;  // set condition
        1   1   (01) 0   : ?   : x; // invalid condition
        ?   ?   (10) 0   : ?   : -;  // ignore .. no change
    endtable
endprimitive
```



## Some Rules to Follow

- The “?” symbol cannot be specified in an output field.
- The “-” symbol, indicating no change in the state value, can be specified only in an output field.
- The shortcut “r”, indicating rising edge, can be used instead of (01).
- The shortcut “f”, indicating falling edge, can be used instead of (10).
- The shortcut “\*” indicates any value change in the signal.



IIT KHARAGPUR

NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

## END OF LECTURE 20



IIT KHARAGPUR

NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 21: VERILOG TEST BENCH

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Verilog Test Bench

- What is test bench?
  - A Verilog procedural block that executes only once.
  - Used for simulation.
  - Test bench generates clock, reset, and the required test vectors for a given *design-under-test* (DUT).
  - The test bench can monitor the DUT outputs and present them in a way as specified by the creator.
    - Print the values of the signal lines.
    - Dump the values in a file from where waveforms can be viewed.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

- Basic requirements:
  - The inputs of the DUT need to be connected to the test bench.
  - The outputs of the DUT needs also to be connected to the test bench.
- Points to note:
  - Test benches use the “*initial*” procedural block that executes only once.
  - Can also use “*always*” for generating some test inputs, like a clock signal.

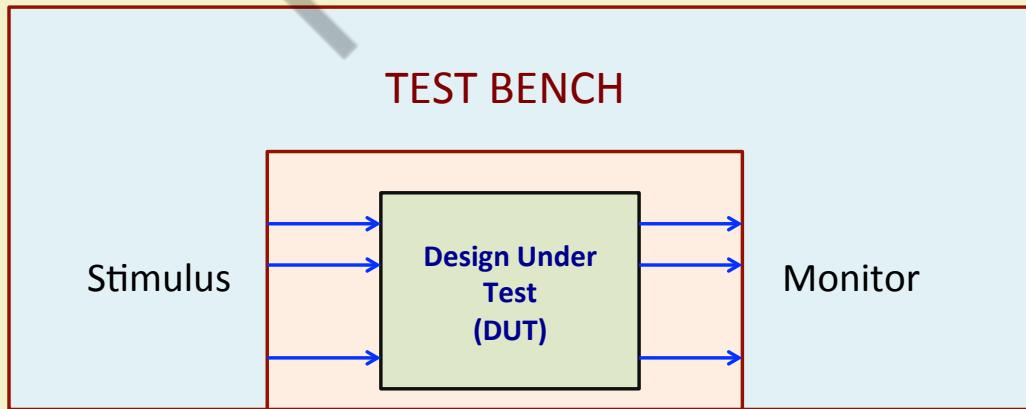


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

## A Simple Example

```
module example (A,B,C,D,E,F,Y);
    input A,B,C,D,E,F;
    output Y;
    wire t1, t2, t3, Y;
    nand #1 G1 (t1,A,B);
    and #2 G2 (t2,C,~B,D);
    nor #1 G3 (t3,E,F);
    nand #1 G4 (Y,t1,t2,t3);
endmodule
```

```
module testbench;
    reg A,B,C,D,E,F;    wire Y;
    example DUT(A,B,C,D,E,F,Y);

    initial
        begin
            $monitor ($time," A=%b, B=%b, C=%b,
                      D=%b, E=%b, F=%b, Y=%b",
                      A,B,C,D,E,F,Y);
            #5 A=1; B=0; C=0; D=1; E=0; F=0;
            #5 A=0; B=0; C=1; D=1; E=0; F=0;
            #5 A=1; C=0;
            #5 F=1;
            #5 $finish;
        end
    endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

5

## How to write test benches?

- Create a dummy template
  - Declare inputs to the design-under-test (DUT) as “*reg*”, and the outputs as “*wire*”.
    - Because we have to initialize the DUT inputs inside procedural block(s), typically “*initial*”, where only “*reg*” type variables can be assigned.
  - Instantiate the DUT.
- Initialization and Monitoring
  - Assign some known values to the DUT inputs.
  - Monitor the DUT outputs for functional verification.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

6

- For synchronous sequential circuits:
  - We need some clock generation logic.
  - Various ways to specify clock signal.
- Test bench can include various simulator directives:
  - *\$display, \$monitor, \$dumpfile, \$dumpvars, \$finish*, etc.
- Important point:
  - We do not need test bench when we are synthesizing a design.
  - Required only during simulation.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

## The Simulator Directives

- **`$display ("<format>", expr1, expr2, ...);`**
  - Used to print the immediate values of text or variables to stdout.
  - Syntax is very similar to “printf” in C.
  - Additional format specifiers are supported, like “b” (binary), “h” (hexadecimal), etc.
- **`$monitor ("<format>", var1, var2, ...);`**
  - Similar in syntax to *\$display*, but does not print immediately.
  - It will print the value(s) whenever the value of some variable(s) in the given list changes.
  - Has the functionality of *event-driven* print.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

- **\$finish;**
  - Terminates the simulation process.
- **\$dumpfile (<filename>);**
  - Specifies the file that will be used for storing the values of the selected variables so that they can be graphically visualized later.
  - The file typically has an extension **.vcd (Value Change Dump)**, and contains information about any value changes on the selected variables.
- **\$dumpoff;**
  - This directive stops the dumping of variables. All variables are dumped with “x” values and the next change of variables will not be dumped.
- **\$dumpon;**
  - This directive starts previously stopped dumping of variables.



- **\$dumpvars (level, list\_of\_variables\_or\_modules);**
  - Specifies which variables should be dumped to the **.vcd** file.
  - Both the parameters are optional; if both are omitted, all variables are dumped.
  - If **level=0**, then all variables within the modules from the list will be dumped. If any module from the list contains module instances, then all variables from these modules will also be dumped.
  - If **level=1**, then only listed variables and variables of listed modules will be dumped.
- **\$dumpall;**
  - The current values of all variables will be written to the file, irrespective of whether there has been any change in their values or not.
- **\$dumplimit (filesize);**
  - Used to set the maximum size of the **.vcd** file.



## A Complete Example :: 2-bit equality checker

```

`timescale 1ns / 100ps
module comparator (x, y, z);
    input [1:0] x, y;  output z;
    assign z = (x[0]&y[0]&x[1]&y[1])
              (~x[0]&~y[0]&x[1]&y[1])
              (~x[0]&~y[0]&~x[1]&~y[1])
              (x[0]&y[0]&~x[1]&~y[1]);
endmodule

```



```

`timescale 1ns / 100ps
module testbench;
    reg [1:0] x, y;  wire z;
    comparator C2 (.x(x), .y(y), .z(z));
    initial
    begin
        $dumpfile ("comp.vcd");
        $dumpvars (0, testbench);
        x = 2'b01; y = 2'b00;
        #10 x = 2'b10; y = 2'b10;
        #10 x = 2'b01; y = 2'b11;
    end
    initial
    begin
        $monitor ("t=%d x=%2b y=%2b z=%d", $time, x, y, z);
    end
endmodule

```



## END OF LECTURE 21



## Lecture 22: WRITING VERILOG TEST BENCHES

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Writing Test Benches

- We shall be illustrating the process of writing test benches through a number of examples.
- We shall be looking at how to:
  - Write test benches for combinational designs.
  - Write test benches for sequential designs.
  - Generate clock and synchronize the applied inputs.
  - Automatically verifying the outputs generated by the design under test.
  - Generating random test vectors.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15

## Example 1: Full Adder

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



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

```

module testbench;
reg a, b, c; wire sum, cout;
full_adder FA (sum, cout, a, b, c);

initial
begin
$monitor ($time," a=%b, b=%b, c=%b, sum=%b, cout=%b",
          a, b, c, sum, cout);
#5 a=0; b=0; c=1;
#5 b=1;
#5 a=1;
#5 a=0; b=0; c=0;
#5 $finish;
end
endmodule

```

```

0 a=x, b=x, c=x, sum=x, cout=x
5 a=0, b=0, c=1, sum=1, cout=0
10 a=0, b=1, c=1, sum=0, cout=1
15 a=1, b=1, c=1, sum=1, cout=1
20 a=0, b=0, c=0, sum=0, cout=0

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

```

module testbench;
reg a, b, c; wire sum, cout;
full_adder FA (sum, cout, a, b, c);

initial
begin
a=0; b=0; c=1; #5;
$display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",$time,a,b,c,sum,cout);
b=1; #5;
$display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",$time,a,b,c,sum,cout);
a=1; #5;
$display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",$time,a,b,c,sum,cout);
a=0; b=0; c=0; #5;
$display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",$time,a,b,c,sum,cout);
#5 $finish;
end
endmodule

```

```

T= 5, a=0, b=0, c=1, sum=1, cout=0
T=10, a=0, b=1, c=1, sum=0, cout=1
T=15, a=1, b=1, c=1, sum=1, cout=1
T=20, a=0, b=0, c=0, sum=0, cout=0

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18

```

module testbench;
  reg a, b, c; wire sum, cout;
  integer i;
  full_adder FA (sum, cout, a, b, c);

  initial
    begin
      for (i=0; i<8; i=i+1)
        begin
          {a,b,c} = i; #5;
          $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",
                    $time, a, b, c, sum, cout);
        end
      #5 $finish;
    end
endmodule

```

```

T= 5, a=0, b=0, c=0, sum=0, cout=0
T=10, a=0, b=0, c=1, sum=1, cout=0
T=15, a=0, b=1, c=0, sum=1, cout=0
T=20, a=0, b=1, c=1, sum=0, cout=1
T=25, a=1, b=0, c=0, sum=1, cout=0
T=30, a=1, b=0, c=1, sum=0, cout=1
T=35, a=1, b=1, c=0, sum=0, cout=1
T=40, a=1, b=1, c=1, sum=1, cout=1

```



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

19

```

module testbench;
  reg a, b, c; wire sum, cout;
  integer i;
  full_adder FA (sum, cout, a, b, c);

  initial
    begin
      $dumpfile ("fulladder.vcd");
      $dumpvars (0, testbench);
      for (i=0; i<8; i=i+1)
        begin
          {a,b,c} = i; #5;
          $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",
                    $time, a, b, c, sum, cout);
        end
      #5 $finish;
    end
endmodule

```

```

T= 5, a=0, b=0, c=0, sum=0, cout=0
T=10, a=0, b=0, c=1, sum=1, cout=0
T=15, a=0, b=1, c=0, sum=1, cout=0
T=20, a=0, b=1, c=1, sum=0, cout=1
T=25, a=1, b=0, c=0, sum=1, cout=0
T=30, a=1, b=0, c=1, sum=0, cout=1
T=35, a=1, b=1, c=0, sum=0, cout=1
T=40, a=1, b=1, c=1, sum=1, cout=1

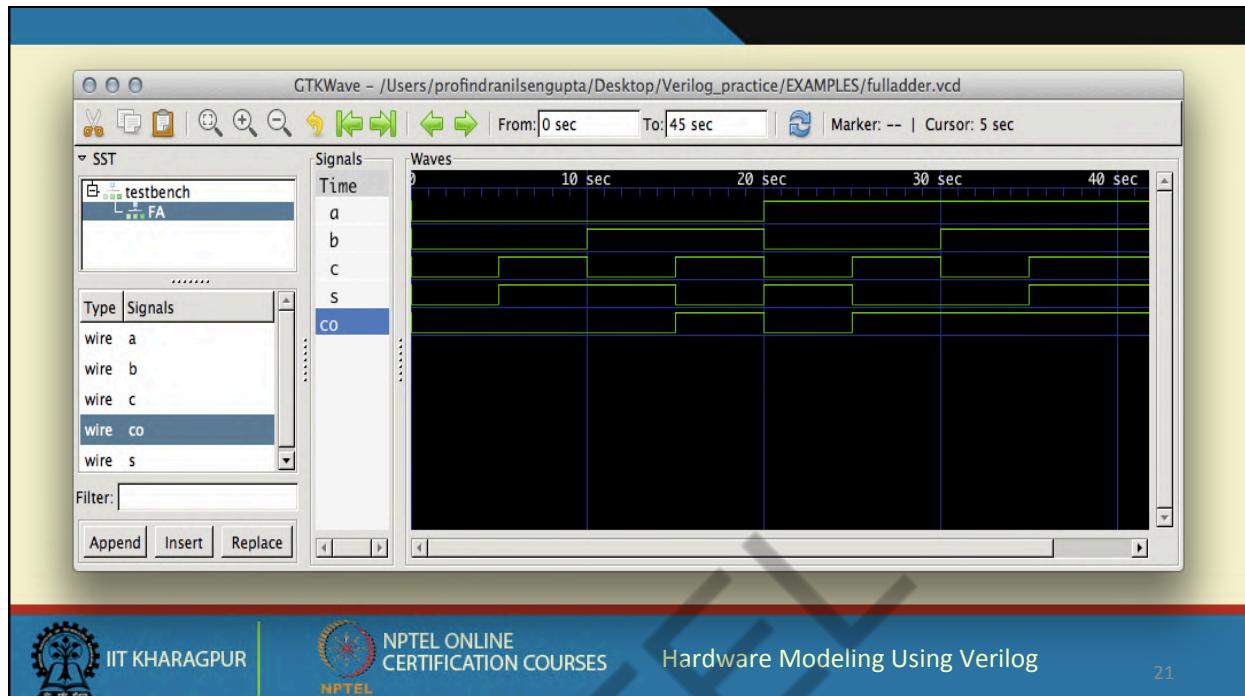
```



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

20



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

21

## Example 2: 4-bit shift register

```
module shiftreg_4bit (clock, clear, A, E);
    input clock, clear, A;
    output reg E;
    reg B, C, D;
    always @(posedge clock or negedge clear)
        begin
            if (!clear) begin B<=0; C<=0; D<=0; E<=0; end
            else begin
                E <= D;
                D <= C;
                C <= B;
                B <= A;
            end
        end
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

22

```

module shift_test;
  reg clk, clr, in;    wire out;    integer i;
  shiftreg_4bit SR (clk, clr, in, out);

  initial
    begin clk = 1'b0; #2 clr = 0; #5 clr = 1; end

  always #5 clk = ~clk;

  initial begin #2;
    repeat (2)
      begin #10 in=0; #10 in=0; #10 in=1; #10 in=1; end
  end

  initial
  begin
    $dumpfile ("shifter.vcd");
    $dumpvars (0, shift_test);
    #200 $finish;
  end
endmodule

```

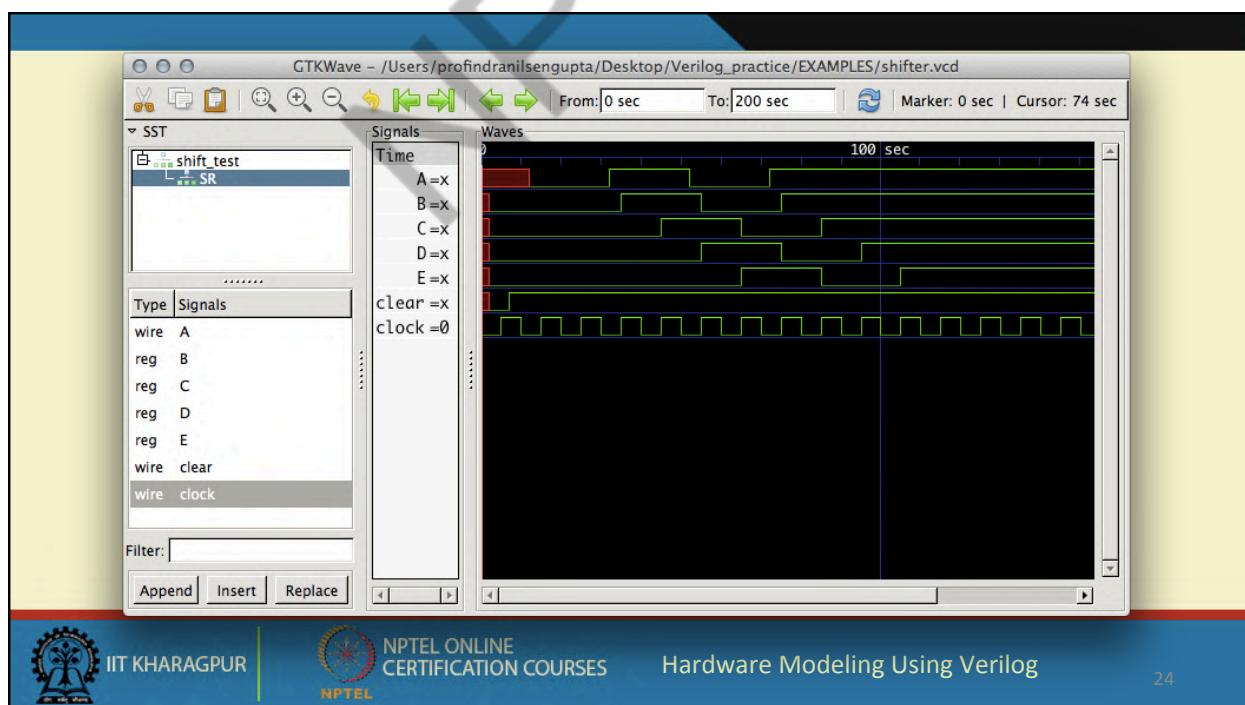


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

23



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

24

### Example 3: 7-bit binary counter

```
module counter (clear, clock, count);
parameter N = 7;
input clear, clock;
output reg [0:N] count;

always @(negedge clock)
if (clear)
count <= 0;
else
count <= count + 1;
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

```
module test_counter;
reg clk, clr;
wire [7:0] out;

counter CNT (clr, clk, out);

initial clk = 1'b0;

always #5 clk = ~clk;

initial
begin
clr = 1'b1;
#15 clr = 1'b0;
#200 clr = 1'b1;
#10 $finish;
end

initial
begin
$dumpfile ("counter.vcd");
$dumpvars (0, test_counter);
$monitor ($time, " Count: %d", out);
end
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

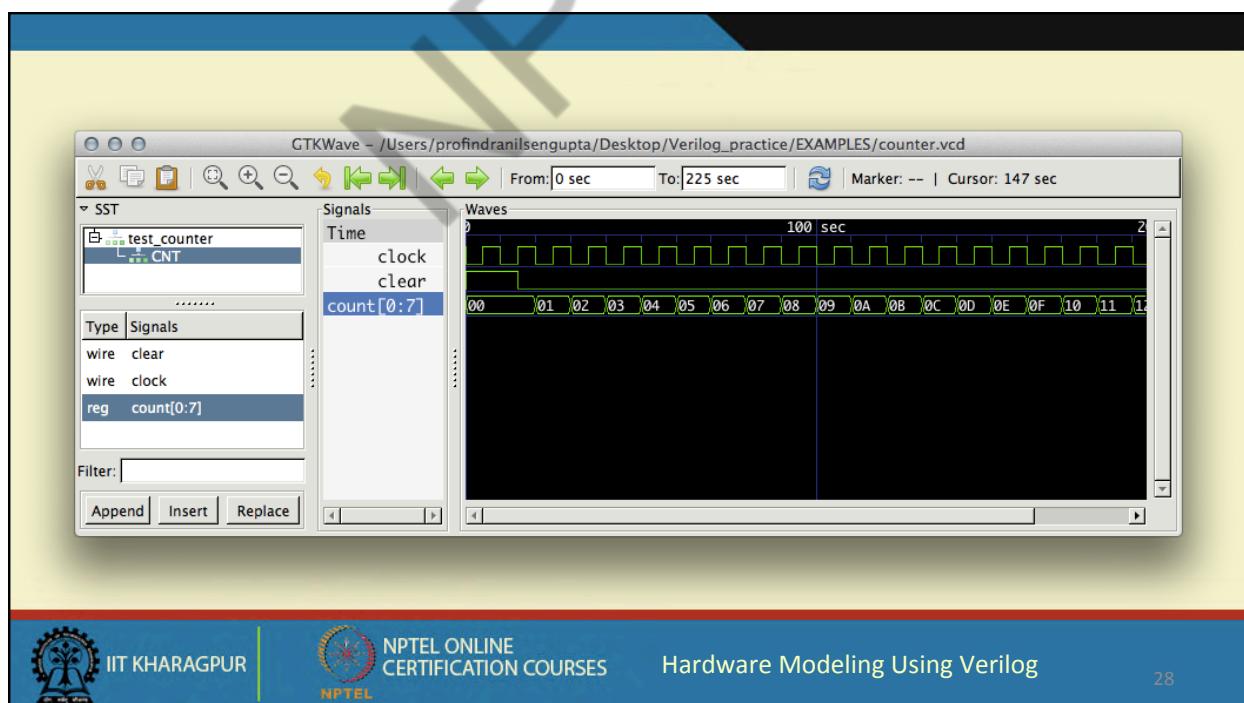
26

```

0 Count: 0
20 Count: 1
30 Count: 2
40 Count: 3
50 Count: 4
60 Count: 5
70 Count: 6
80 Count: 7
90 Count: 8
100 Count: 9
110 Count: 10
120 Count: 11
130 Count: 12
140 Count: 13
150 Count: 14
160 Count: 15
170 Count: 16
180 Count: 17
190 Count: 18
200 Count: 19
210 Count: 20
220 Count: 0

```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog 27



## Example 4: Automatic verification of output

```
module fulladder (a, b, c, s, cout);
    input a, b, c;
    output s, cout;

    assign s = a ^ b ^ c;
    assign cout = (a&b) | (b&c) | (c&a);

endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

29

```
module fulladder_test;
reg a,b,c;
wire s, cout;
integer correct;

fulladder FA (a,b,c,s,cout);

initial
begin
    correct = 1;

#5 a=1; b=1; c=0; #5;
    if ((s != 1) || (cout != 1))
        correct = 0;

```

```
#5 a=1; b=1; c=1; #5;
    if ((s != 1) || (cout != 1))
        correct = 0;

#5 a=0; b=1; c=0; #5;
    if ((s != 0) || (cout != 0))
        correct = 0;

#5 $display ("%d", correct);
end

endmodule
```

*Shall display 1 if outputs are correct; and display 0 otherwise.*



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

30

## Example 5: Generating random test vectors

```
module adder (out, cout, a, b);
    input [7:0] a, b;
    output [7:0] out;
    output cout;

    assign #5 {cout,out} = a + b;
endmodule
```

- The system task `$random` can be used to generate a random number.
- It is called as : `$random (<seed>)`
  - The value of `<seed>` is optional and is used to ensure that the same sequence of random numbers are generated each time the test is run.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

31

```
module test_adder;
    reg [7:0] a, b;
    wire [7:0] sum;    wire cout;
    integer myseed;
    adder ADD (sum, cout, a, b);
    initial myseed = 15;

    initial
        begin
            repeat (5)
                begin
                    a = $random(myseed);
                    b = $random(myseed); #10;
                    $display ("T: %d, a: %h, b: %h, sum: %h", $time, a, b, sum);
                end
        end
endmodule
```

T: 10, a: 00, b: 52, sum: 52
T: 20, a: ca, b: 08, sum: d2
T: 30, a: 0c, b: 6a, sum: 76
T: 40, a: b1, b: 71, sum: 22
T: 50, a: 23, b: df, sum: 02



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

32

## END OF LECTURE 22



## Lecture 23: MODELING FINITE STATE MACHINES

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Introduction

- Combinational and Sequential Circuits
  - In a combinational circuit, the outputs depend only on the applied input values and not on the past history.
  - In a sequential circuit, the outputs depend not only on the applied input values but also on the internal state.
    - The internal states also change with time.
    - The number of states is finite, and hence a sequential circuit is also referred to as a *Finite State Machine (FSM)*.
- Most of the practical circuits are sequential in nature.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

# Finite State Machine (FSM)

- A FSM can be represented either in the form of a *state table* or in the form of a *state transition diagram*.
  - Variations exist, e.g. *Algorithmic State Machine (ASM) chart*.
- Example:
  - A circuit to detect 3 or more 1's in a serial bit stream.
  - The bits are applied serially in synchronism with a clock.
  - The output will become 1 whenever it detects 3 or more consecutive 1's in the stream.

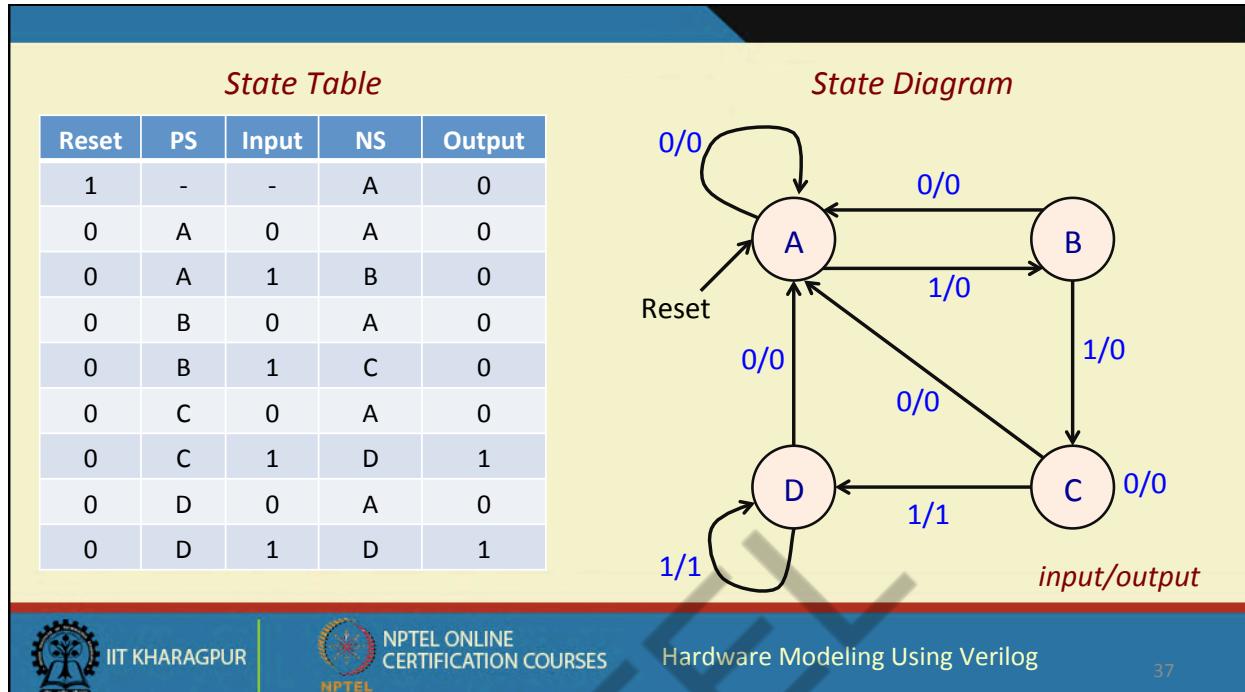


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36

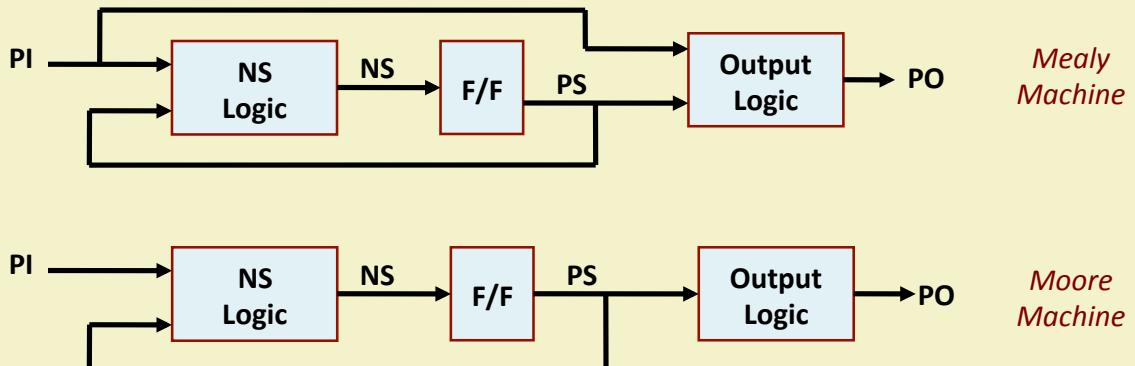


## Mealy and Moore FSM Types

- A deterministic FSM can be mathematically defined as a 5-tuple  $(\Sigma, \Gamma, S, s_0, \delta, \omega)$  where  $\Sigma$  is the set of input combinations,  $\Gamma$  is the set of output combinations,  $S$  is a finite set of states,  $s_0 \in S$  is the initial state,  $\delta$  is the state-transition function, and  $\omega$  is the output function.
- Here,  $\delta : S \times \Sigma \rightarrow S$ 
  - Present state (PS) and present input determines the next state (NS).
- For Mealy machine,  $\omega : S \times \Sigma \rightarrow \Gamma$  (output depends on state + inputs)
- For Moore machine,  $\omega : S \rightarrow \Gamma$  (output depends only on the state)

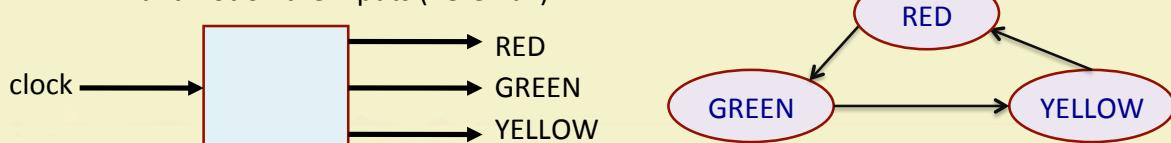
IIT Kharagpur
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog
38

## Pictorial Depiction



## Example 1

- There are three lamps, **RED**, **GREEN** and **YELLOW**, that should glow cyclically with a fixed time interval (say, 1 second).
- Some observations:
  - The FSM will have three states, corresponding to the glowing state of the lamps.
  - The input set is null; state transition will occur whenever clock signal comes.
  - This is a **Moore Machine**, since the lamp that will glow only depends on the state and not on the inputs (here null).



```

module cyclic_lamp (clock, light);
    input clk;
    output reg [0:2] light;
    parameter S0=0, S1=1, S2=2;
    parameter RED=3'b100, GREEN=3'b010, YELLOW=3'b001;
    reg [0:1] state;
    always @(posedge clock)
        case (state)
            S0: begin          // S0 means RED
                light <= GREEN; state <= S1;
            end
            S1: begin          // S1 means GREEN
                light <= YELLOW; state <= S2;
            end
            S2: begin          // S2 means YELLOW
                light <= RED; state <= S0;
            end
        default: begin
                light <= RED;
                state <= S0;
            end
        endcase
    endmodule

```



```

module test_cyclic_lamp;
    reg clk;
    wire [0:2] light;
    cyclic_lamp LAMP (clk, light);
    always #5 clk = ~clk;
    initial
        begin
            clk = 1'b0;
            #100 $finish;
        end
    initial
        begin
            $dumpfile ("cyclic.vcd"); $dumpvars (0, test_cyclic_lamp);
            $monitor ($time, " RGY: %b", light);
        end
    endmodule

```

0 RGY: xxx
5 RGY: 100
15 RGY: 010
25 RGY: 001
35 RGY: 100
45 RGY: 010
55 RGY: 001
65 RGY: 100
75 RGY: 010
85 RGY: 001
95 RGY: 100



- Some comments on the solution:
  - The synthesis tool will generate five flip-flops – 2 for *state*, and 3 for *light*.
  - The three output lines are also getting stored in flip-flops.
    - We have used non-blocking assignment triggered by clock edge.
  - But actually we do not need separate flip-flops for the outputs, as the outputs can be directly generated from the *state*.
  - How to achieve this?
    - Modify the Verilog code such that all assignments to *light* is made in a separate “*always*” block.
    - Use blocking assignment triggered by state change, and not by clock.



```

module cyclic_lamp (clock, light);
  input clk;
  output reg [0:2] light;
  parameter S0=0, S1=1, S2=2;
  parameter RED=3'b100, GREEN=3'b010, YELLOW=3'b001;
  reg [0:1] state;

  always @(posedge clk)
    case (state)
      S0: state <= S1;
      S1: state <= S2;
      S2: state <= S0;
      default: state <= S0;
    endcase
endmodule

always @(*)
  case (state)
    S0: light = RED;
    S1: light = GREEN;
    S2: light = YELLOW;
    default: light = RED;
  endcase
endmodule

```



- Comment on the solution:
  - The synthesis tool will be generating only 2 flip-flops corresponding to the first clock-triggered “always” block.
  - The second “always” block will be generating a combinational circuit that takes *state* as input and produces *light* as outputs.

state ( $s_1 s_0$ )	Light (RGY)
S0: 00	1 0 0
S1: 01	0 1 0
S2: 10	0 0 1
11	x x x

Logic expressions after minimization:

$$R = s_0' s_1'$$

$$G = s_0$$

$$Y = s_1$$



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

45

## END OF LECTURE 23



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

46



## Example 2

- Design of a serial parity detector.
  - A continuous stream of bits is fed to a circuit in synchronism with a clock. The circuit will be generating a bit stream as output, where a 0 will indicate “*even number of 1’s seen so far*” and a 1 will indicate “*odd number of 1’s seen so far*”.
  - Also a *Moore Machine*.

A block diagram showing a rectangular block with two inputs:  $x$  (data) and  $clk$  (clock), and one output  $z$ .

A state transition diagram with two states: 'EVEN' and 'ODD'. The initial state is 'EVEN'. Transitions are labeled with input pairs and outputs:

- From 'EVEN' to 'EVEN' on input  $0/0$  (output  $0/0$ ).
- From 'EVEN' to 'ODD' on input  $1/1$  (output  $1/1$ ).
- From 'ODD' to 'EVEN' on input  $1/0$  (output  $0/1$ ).
- From 'ODD' to 'ODD' on input  $0/1$  (output  $0/1$ ).

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 48

24

```

module parity_gen (x, clk, z);
    input x, clk;
    output reg z;
    reg even_odd;      // The machine state
    parameter EVEN=0, ODD=1;
    always @(posedge clk)
        case (even_odd)
            EVEN: begin
                z <= x ? 1 : 0;
                even_odd <= x ? ODD : EVEN;
            end
            ODD: begin
                z <= x ? 0 : 1;
                even_odd <= x ? EVEN : ODD;
            end
            default: even_odd <= EVEN;
        endcase
    endmodule

```

This design will cause the synthesis tool to generate a latch for the output "even\_odd".

```

module test_parity;
    reg clk, x;    wire z;
    parity_gen PAR (x, clk, z);

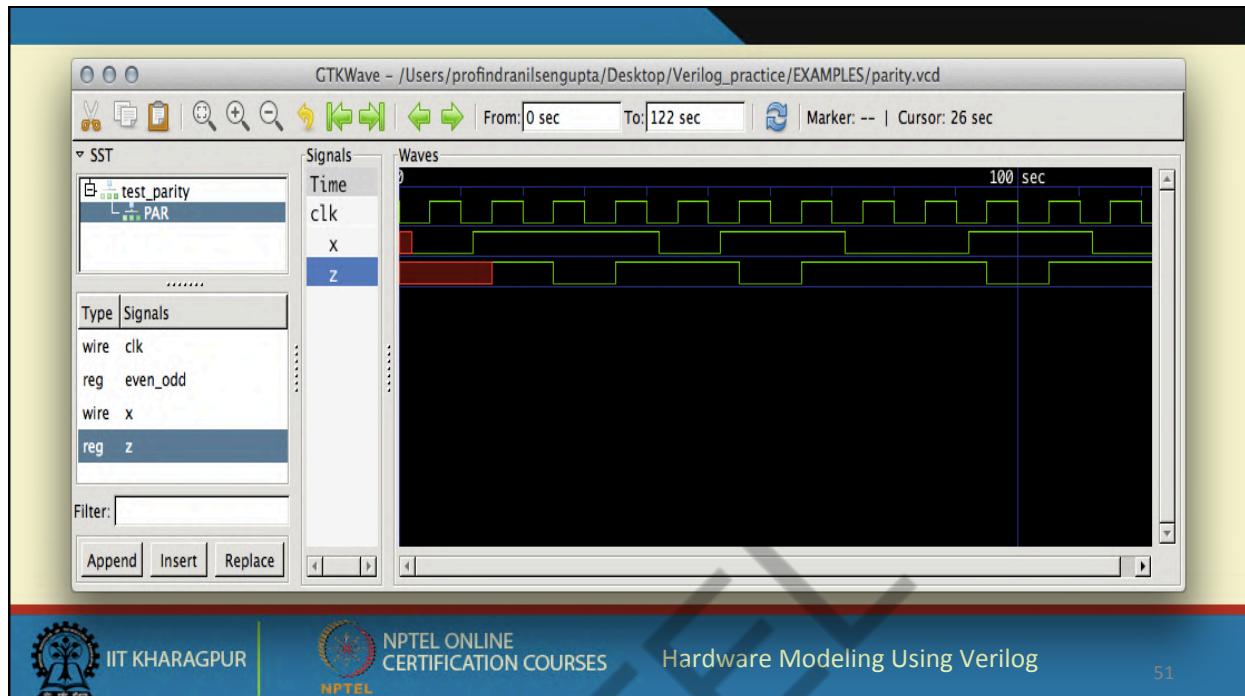
    initial
        begin
            $dumpfile ("parity.vcd");  $dumpvars (0, test_parity);
            clk = 1'b0;
        end

    always #5 clk = ~clk;

    initial
        begin
            #2 x = 0; #10 x = 1; #10 x = 1; #10 x = 1;
            #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
            #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
            #10 $finish;
        end
    endmodule

```





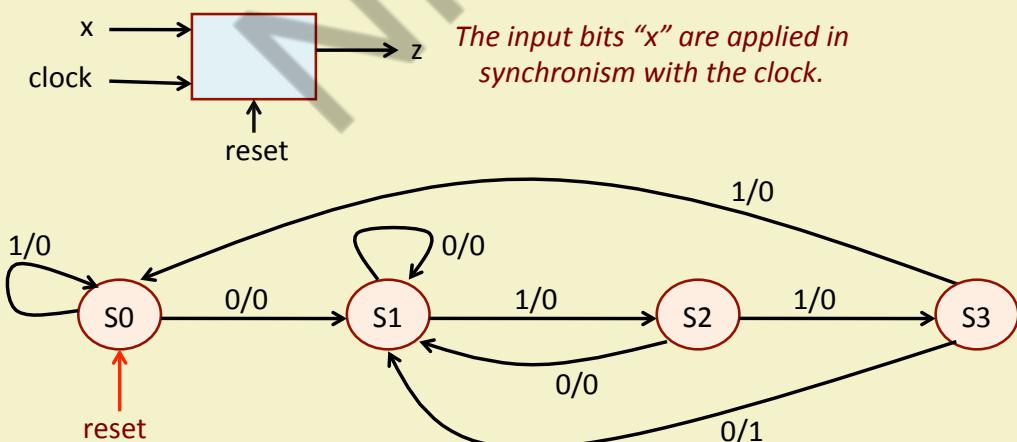
```
module parity_gen (x, clk, z);
    input x, clk;    output reg z;
    reg even_odd;    // The machine state
    parameter EVEN=0, ODD=1;
    always @(posedge clk)
        case (even_odd)
            EVEN: even_odd <= x ? ODD : EVEN;
            ODD: even_odd <= x ? EVEN : ODD;
            default : even_odd <= EVEN;
        endcase
    always @(even_odd)
        case (even_odd)
            EVEN: z = 0;
            ODD: z = 1;
        endcase
    endmodule
```

This design will not cause the synthesis tool to generate a latch for the output "z".

IIT KARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 52

## Example 3

- Design of a sequence detector.
  - A circuit accepts a serial bit stream “x” as input and produces a serial bit stream “z” as output.
  - Whenever the bit pattern “0110” appears in the input stream, it outputs  $z = 1$ ; at all other times,  $z = 0$ .
  - Overlapping occurrences of the pattern are also detected.
  - This is a *Mealy Machine*.
  - Example:  $x := 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0$   
 $z := 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0$



```
// Sequence detector for pattern "0110"
module seq_detector (x, clk, reset, z);
    input x, clk, reset;
    output reg z;
    parameter S0=0, S1=1, S2=2, S3=3;
    reg [0:1] PS, NS;

    always @(posedge clk or posedge reset)
        if (reset) PS <= S0;
        else         PS <= NS;

    always @(PS,x)
        case (PS)
            S0: begin
                z = x ? 0 : 0;
                NS = x ? S0 : S1;
            end

```

```
S1: begin
    z = x ? 0 : 0;
    NS = x ? S2 : S1;
end
S2: begin
    z = x ? 0 : 0;
    NS = x ? S3 : S1;
end
S3: begin
    z = x ? 0 : 1;
    NS = x ? S0 : S1;
end
endcase
endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

55

```
module test_sequence;
    reg clk, x, reset;    wire z;
    seq_detector SEQ (x, clk, reset, z);

    initial
        begin
            $dumpfile ("sequence.vcd");  $dumpvars (0, test_sequence);
            clk = 1'b0;  reset = 1'b1;
            #15 reset = 1'b0;
        end

    always #5 clk = ~clk;

    initial
        begin
            #12 x = 0; #10 x = 0; #10 x = 1; #10 x = 1;
            #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
            #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
            #10 $finish;
        end
endmodule
```

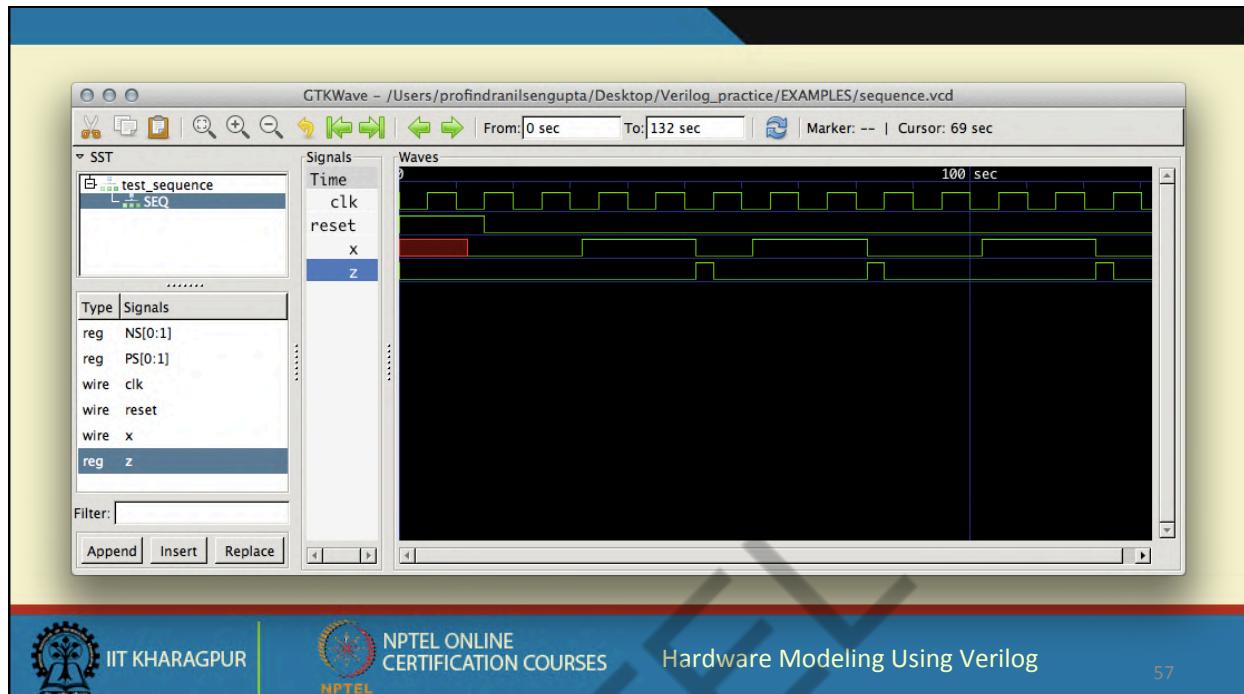


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

56



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

57

## Example 4

- Design a sequence detector for the bit pattern “101010”.
  - Work out the state diagram in a similar way.
  - Then code the state diagram in Verilog.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

58

## END OF LECTURE 24



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

59

NPTEL



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 25: DATAPATH AND CONTROLLER DESIGN (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Introduction

- In a complex digital system, the hardware is typically partitioned into two parts:
  - a) *Data Path*, which consists of the functional units where all computations are carried out.
    - Typically consists of registers, multiplexers, bus, adders, multipliers, counters, and other functional blocks.
  - b) *Control Path*, which implements a finite-state machine and provides control signals to the data path in proper sequence.
    - In response to the control signals, various operations are carried out by the data path.
    - Also takes inputs from the data path regarding various status information.



IIT KHARAGPUR

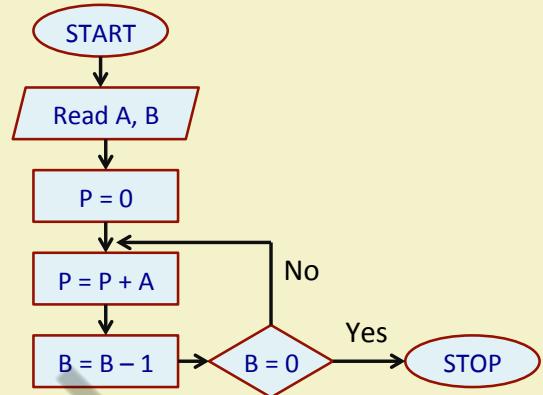
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

## Example 1: Multiplication by Repeated Addition

- We consider a simple algorithm using repeated addition.
  - Assume B is non-zero.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the multiplication algorithm using the data path.

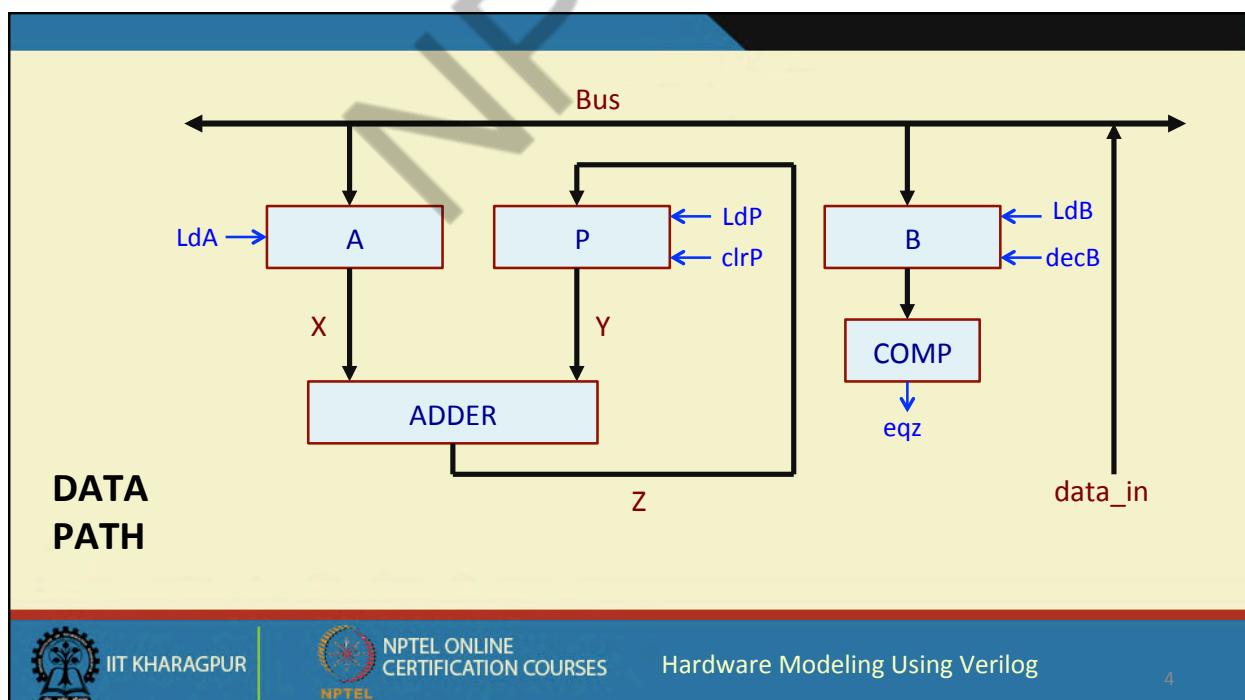


IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

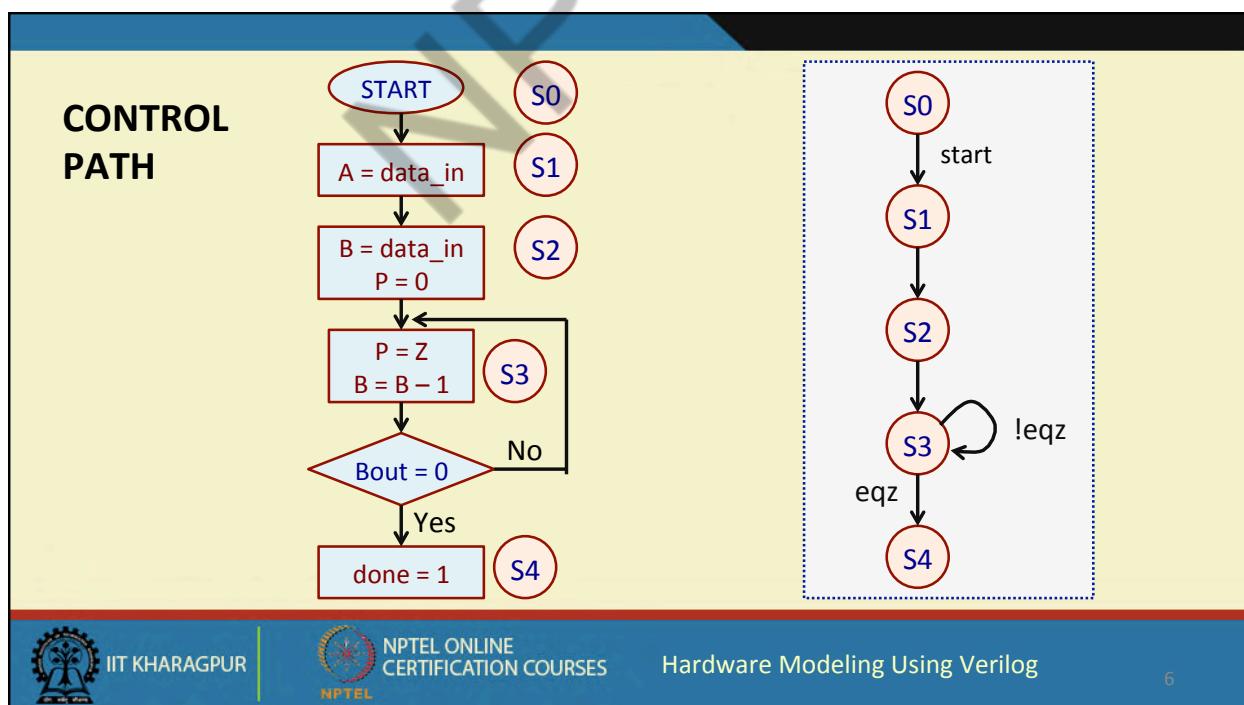
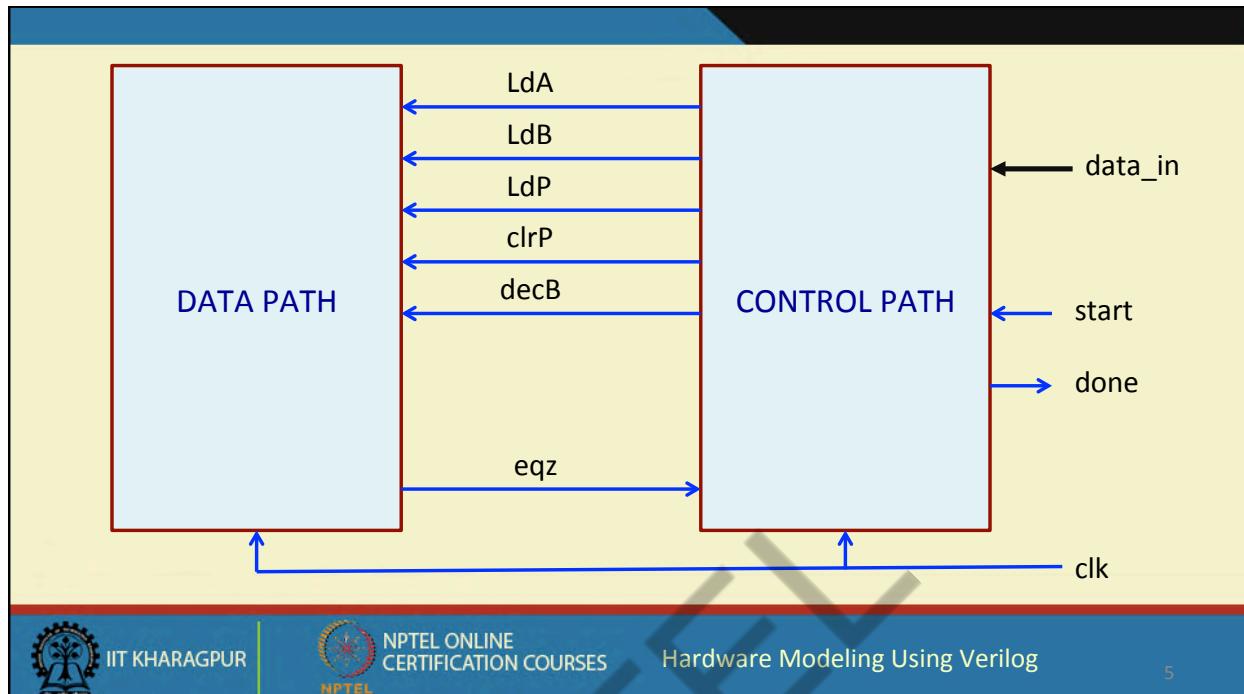


IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4



```

module MUL_datapath (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
    input LdA, LdB, LdP, clrP, decB, clk;
    input [15:0] data_in;
    output eqz;
    wire [15:0] X, Y, Z, Bout, Bus;

    PIPO1 A (X, Bus, LdA, clk);
    PIPO2 P (Y, Z, LdB, decB, clk);
    CNTR B (Bout, Bus, LdB, decB, clk);
    ADD AD (Z, X, Y);
    EQZ COMP (eqz, Bout);
endmodule

```

## THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

```

module PIPO1 (dout, din, ld, clk);
    input [15:0] din;
    input ld, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if (ld) dout <= din;
endmodule

module ADD (out, in1, in2);
    input [15:0] in1, in2;
    output reg [15:0] out;
    always @(*)
        out = in1 + in2;
endmodule

```

```

module PIPO2 (dout, din, ld,
              clr, clk);
    input [15:0] din;
    input ld, clr, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if (clr) dout <= 16'b0;
        else if (ld) dout <= din;
endmodule

module EQZ (eqz, data);
    input [15:0] data;
    output eqz;
    assign eqz = (data == 0);
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

```
module CNTR (dout, din, ld, dec, clk);
    input [15:0] din;
    input ld, dec, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if (ld) dout <= din;
        else if (dec) dout <= dout - 1;
endmodule
```



```
module controller (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);
    input clk, eqz, start;
    output reg LdA, LdB, LdP, clrP, decB, done;
    reg [2:0] state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
    always @(posedge clk)
        begin
            case (state)
                S0:   if (start) state <= S1;
                S1:   state <= S2;
                S2:   state <= S3;
                S3:   #2 if (eqz) state <= S4;
                S4:   state <= S0;
            endcase
        end
end
```

### THE CONTROL PATH



```

always @(state)
begin
  case (state)
    S0: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
    S1: begin #1 LdA = 1; end
    S2: begin #1 LdA = 0; LdB = 1; clrP = 1; end
    S3: begin #1 LdB = 0; LdP = 1; clrP = 0; decB = 1; end
    S4: begin #1 done = 1; LdB = 0; LdP = 0; decB = 0; end
  default: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
  endcase
end
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

11

```

module MUL_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  MUL_datapath DP (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  controller CON (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #500 $finish;
    end

  always #5 clk = ~clk;

```

**THE TEST BENCH**

```

initial
begin
  #17 data_in = 17;
  #10 data_in = 5;
end

initial
begin
  $monitor ($time, " %d %b", DP.Y, done);
  $dumpfile ("mul.vcd"); $dumpvars (0, MUL_test);
end

```

```

endmodule

```

0	x x
6	x 0
35	0 0
45	17 0
55	34 0
65	51 0
75	68 0
85	85 0
88	85 1

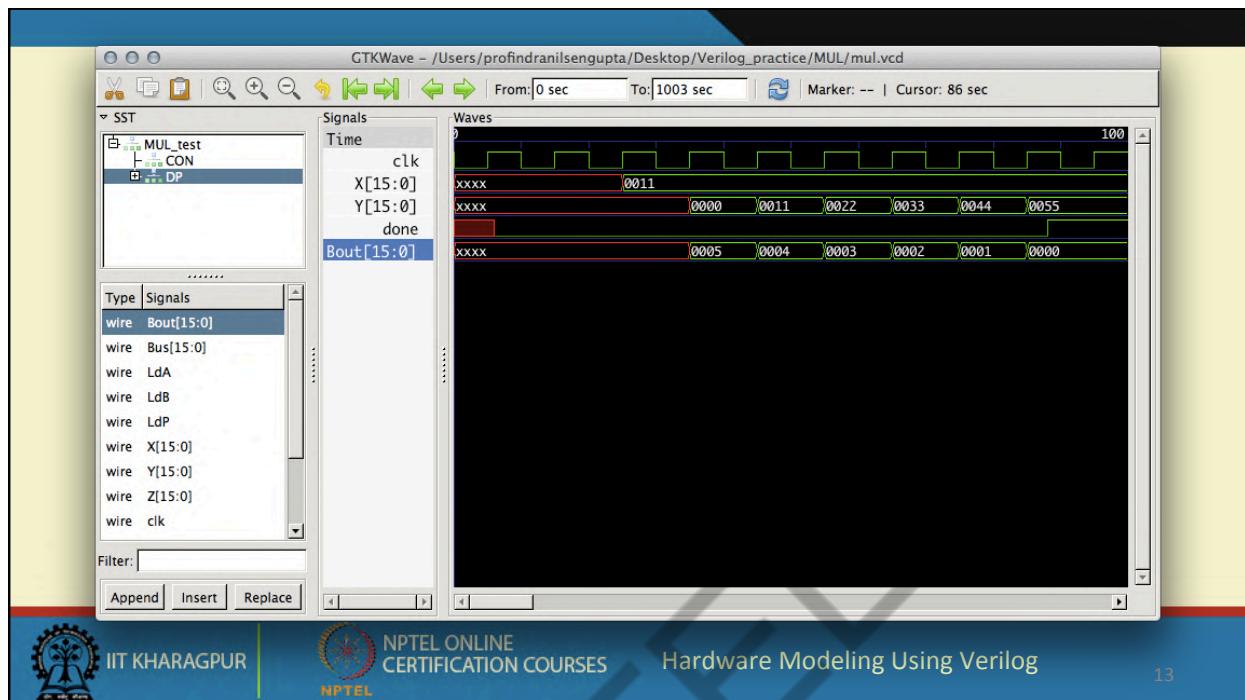


IIT KHARAGPUR

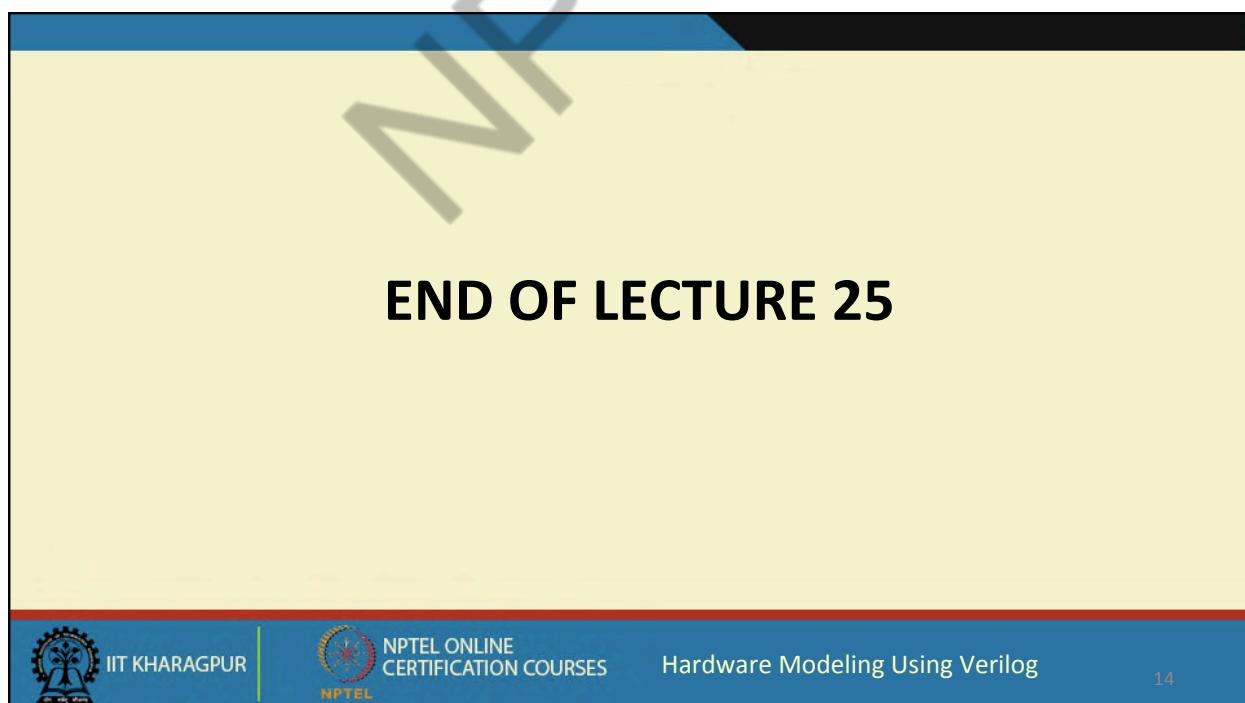
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

12



END OF LECTURE 25





IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 26: DATAPATH AND CONTROLLER DESIGN (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### A Better Style of Modeling Data / Control Path

- In the previous example, in the “*always*” block activated by clock edge, both state change as well as computation of the next state is performed.
- A better and recommended approach:
  - Only trigger the state change in the clock activated “*always*” block.
  - In a separate “*always*” block using blocking assignments, compute the next state.
  - As in the previous example, in a separate “*always*” block, generate the control signals for the data path.



IIT KHARAGPUR

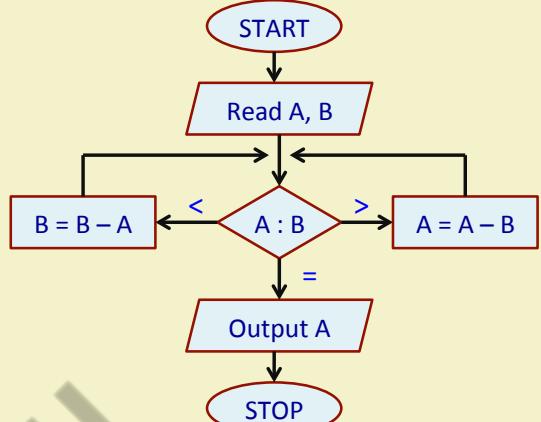
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

## Example 2: GCD Computation

- We consider a simple algorithm using repeated subtraction.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the GCD computation algorithm using the data path.



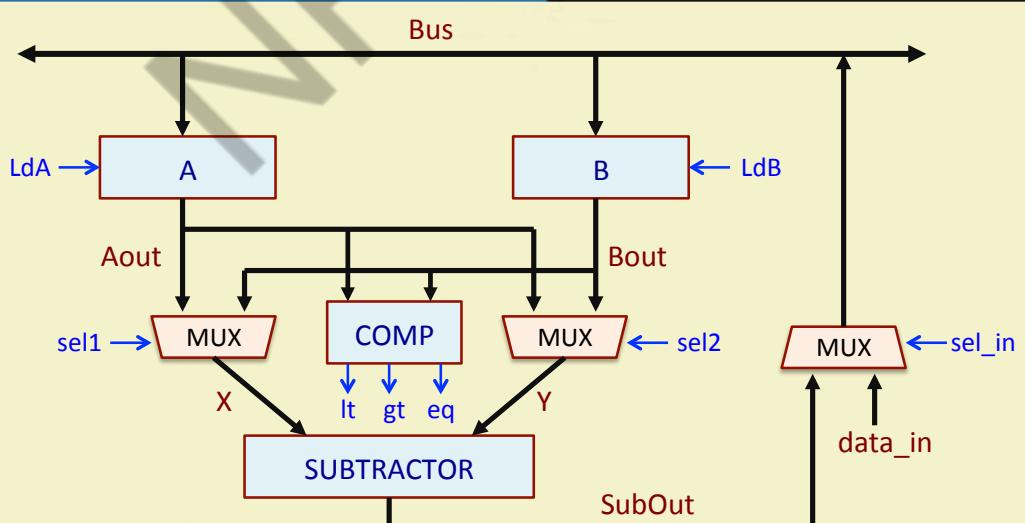
IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

**DATA  
PATH**

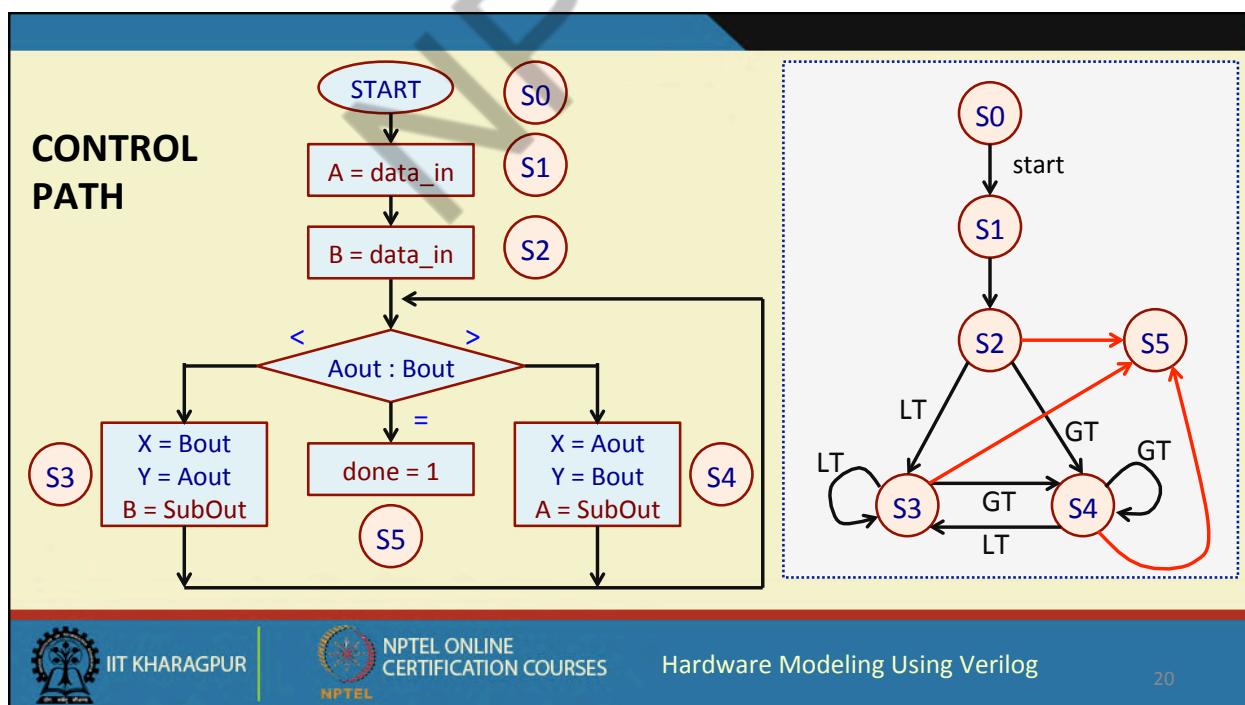
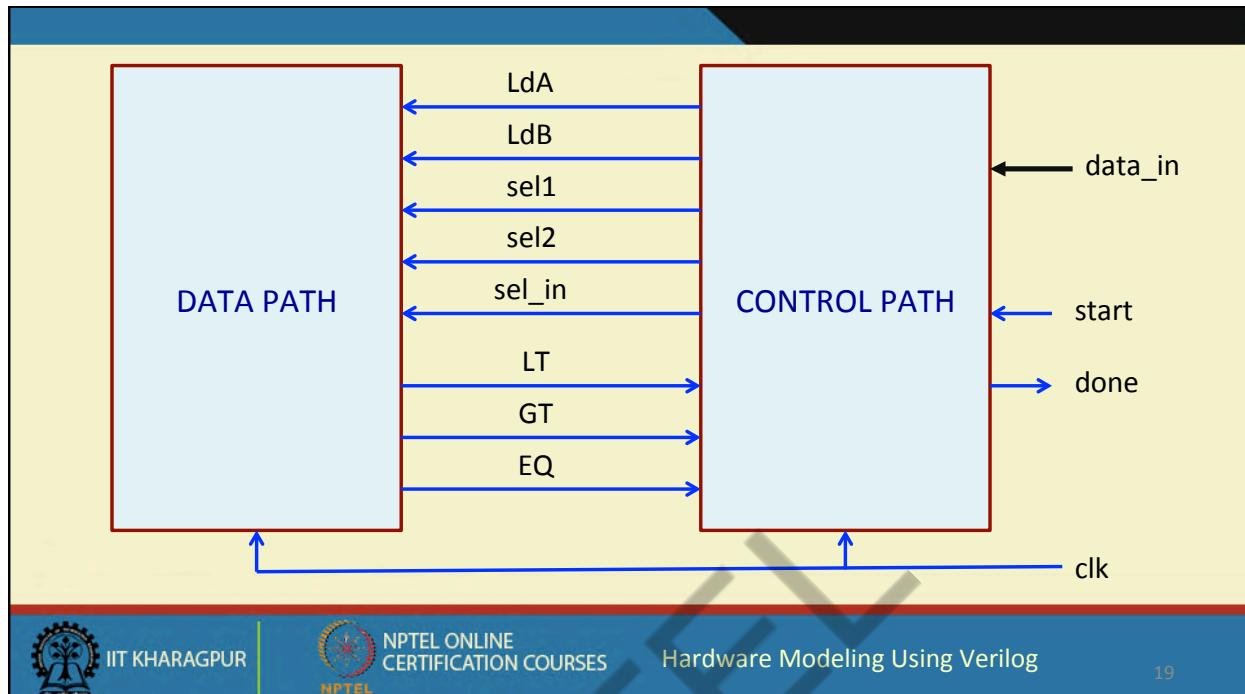


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18



```

module GCD_datapath (gt, lt, eq, ldA, ldB, sel1, sel2, sel_in,
data_in, clk);
    input ldA, ldB, sel1, sel2, sel_in, clk;
    input [15:0] data_in;
    output gt, lt, eq;
    wire [15:0] Aout, Bout, X, Y, Bus, SubOut;

    PIPO A (Aout, Bus, ldA, clk);
    PIPO B (Bout, Bus, ldB, clk);
    MUX MUX_in1 (X, Aout, Bout, sel1);
    MUX MUX_in2 (Y, Aout, Bout, sel2);
    MUX MUX_load (Bus, SubOut, data_in, sel_in);
    SUB SB (SubOut, X, Y);
    COMPARE COMP (lt, gt, eq, Aout, Bout);
endmodule

```

## THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

21

```

module PIPO (data_out, data_in,
             load, clk);
    input [15:0] data_in;
    input load, clk;
    output reg [15:0] data_out;
    always @(posedge clk)
        if (load) data_out <= data_in;
endmodule

module SUB (out, in1, in2);
    input [15:0] in1, in2;
    output reg [15:0] out;
    always @(*)
        out = in1 - in2;
endmodule

```

```

module COMPARE (lt, gt, eq, data1,
                 data2);
    input [15:0] data1, data2;
    output lt, gt, eq;
    assign lt = data1 < data2;
    assign gt = data1 > data2;
    assign eq = data1 == data2;
endmodule

module MUX (out, in0, in1, sel);
    input [15:0] in0, in1;
    input sel;
    output [15:0] out;
    assign out = sel ? in1 : in0;
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

22

```

module controller (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);
  input clk, lt, gt, eq, start;
  output reg ldA, ldB, sel1, sel2, sel_in, done;
  reg [2:0] state;
  parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101;
  always @(posedge clk)
    begin
      case (state)
        S0: if (start) state <= S1;
        S1: state <= S2;
        S2: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S3: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S4: #2 if (eq) state <= S5;
              else if (lt) state <= S3;
              else if (gt) state <= S4;
        S5: state <= S5;
        default: state <= S0;
      endcase
    end

```

**THE CONTROL  
PATH**

```

always @(state)
begin
  case (state)
    S0: begin sel_in = 1; ldA = 1; ldB = 0; done = 0; end
    S1: begin sel_in = 1; ldA = 0; ldB = 1; end
    S2: if (eq) done = 1;
          else if (lt) begin
            sel1 = 1; sel2 = 0; sel_in = 0;
            #1 ldA = 0; ldB = 1;
          end
          else if (gt) begin
            sel1 = 0; sel2 = 1; sel_in = 0;
            #1 ldA = 1; ldB = 0;
          end
    S3: if (eq) done = 1;
          else if (lt) begin
            sel1 = 1; sel2 = 0; sel_in = 0;
            #1 ldA = 0; ldB = 1;
          end
          else if (gt) begin
            sel1 = 0; sel2 = 1; sel_in = 0;
            #1 ldA = 1; ldB = 0;
          end
  end
end

```

```

S4:      if (eq) done = 1;
          else if (lt) begin
              sel1 = 1; sel2 = 0; sel_in = 0;
              #1 ldA = 0; ldB = 1;
              end
          else if (gt) begin
              sel1 = 0; sel2 = 1; sel_in = 0;
              #1 ldA = 1; ldB = 0;
              end
      S5:      begin
              done = 1; sel1 = 0; sel2 = 0; ldA = 0;
              ldB = 0;
              end
          default: begin ldA = 0; ldB = 0; end
      endcase
  end

endmodule

```



**THE TEST  
BENCH**

```

module GCD_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  reg [15:0] A, B;

  GCD_datapath DP (gt, lt, eq, ldA, ldB, sel1, sel2, sel_in, data_in, clk);
  controller CON (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #1000 $finish;
    end

  always #5 clk = ~clk;

```



```

initial
begin
    #12 data_in = 143;
    #10 data_in = 78;
end

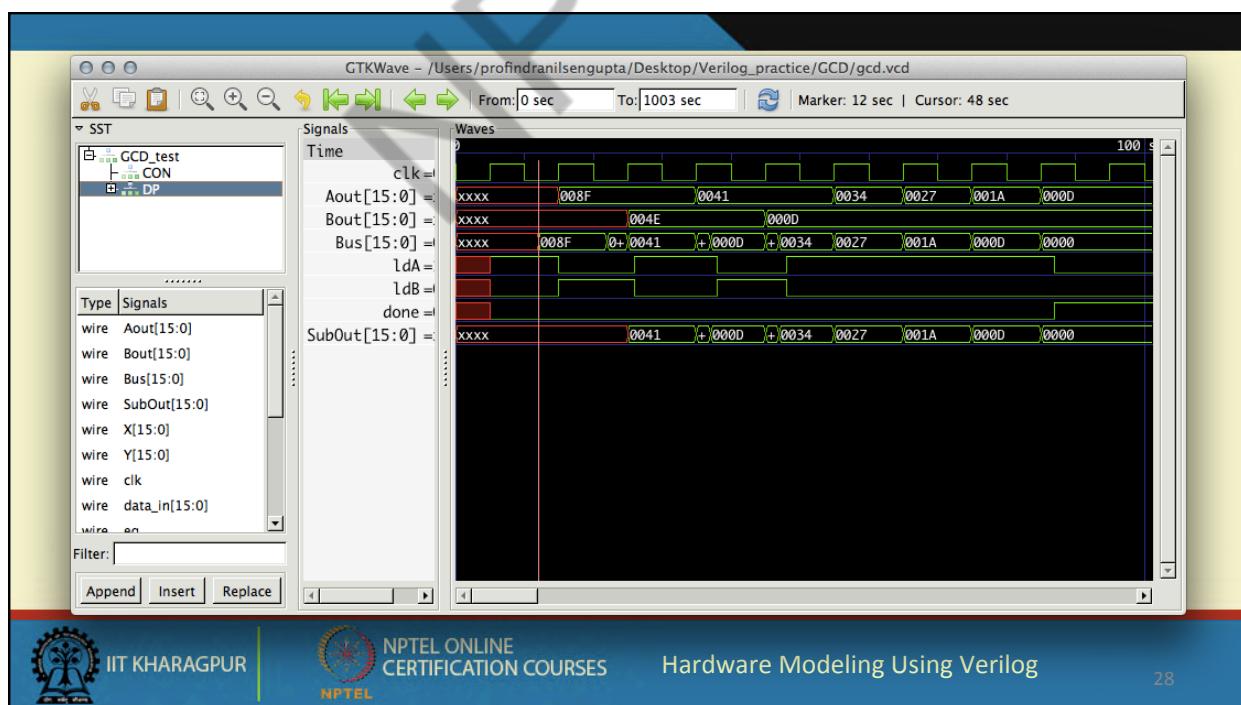
initial
begin
    $monitor ($time, " %d %b", DP.Aout, done);
    $dumpfile ("gcd.vcd"); $dumpvars (0, GCD_test);
end

endmodule

```

0	x x
5	x 0
15	143 0
35	65 0
55	52 0
65	39 0
75	26 0
85	13 0
87	13 1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 27



# MODELING THE CONTROL PATH USING THE ALTERNATE APPROACH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

29

```
module controller (ldA, ldB, sel1, sel2, sel_in, done, clk, lt, gt, eq, start);
    input clk, lt, gt, eq, start;
    output reg ldA, ldB, sel1, sel2, sel_in, done;
    reg [2:0] state, next_state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101;
    always @(posedge clk)
        begin
            state <= next_state;
        end

```

**THE CONTROL  
PATH**



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

30

```

always @(state)
begin
  case (state)
    S0: begin sel_in = 1; ldA = 1; ldB = 0; done = 0; end
    S1: begin sel_in = 1; ldA = 0; ldB = 1; end
    S2: if (eq) begin done = 1; next_state = S5; end
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
           #1 ldA = 1; ldB = 0;
         end
    S3: if (eq) begin done = 1; next_state = S5; end
         else if (lt) begin
           sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
           #1 ldA = 0; ldB = 1;
         end
         else if (gt) begin
           sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
           #1 ldA = 1; ldB = 0;
         end
end

```

```

S4: if (eq) begin done = 1; next_state = S5; end
     else if (lt) begin
       sel1 = 1; sel2 = 0; sel_in = 0; next_state = S3;
       #1 ldA = 0; ldB = 1;
     end
     else if (gt) begin
       sel1 = 0; sel2 = 1; sel_in = 0; next_state = S4;
       #1 ldA = 1; ldB = 0;
     end
S5: begin
      done = 1; sel1 = 0; sel2 = 0; ldA = 0;
      ldB = 0; next_state = S5;
    end
  default: begin ldA = 0; ldB = 0; next_state = S0; end
endcase
end

endmodule

```



## END OF LECTURE 26



## Lecture 27: DATAPATH AND CONTROLLER DESIGN (PART 3)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Example 3: Booth's Multiplication

- In the conventional shift-and-add multiplication, for  $n$ -bit multiplication, we iterate  $n$  times.
  - Add either 0 or the multiplicand to the  $2n$ -bit partial product (depending on the next bit of the multiplier).
  - Shift the  $2n$ -bit partial product to the right.
- Essentially we need  $n$  additions and  $n$  shift operations.
- Booth's algorithm is an improvement whereby we can avoid the additions whenever consecutive 0's or 1's are detected in the multiplier.
  - Makes the process faster.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

35

## Basic Idea Behind Booth's Algorithm

- We inspect two bits of the multiplier ( $Q_i, Q_{i-1}$ ) at a time.
  - If the bits are same (00 or 11), we only shift the partial product.
  - If the bits are 01, we do an addition and then shift.
  - If the bits are 10, we do a subtraction and then shift.
- $Q_{-1}$  is assumed to be equal to 0.
- Significantly reduces the number of additions / subtractions.

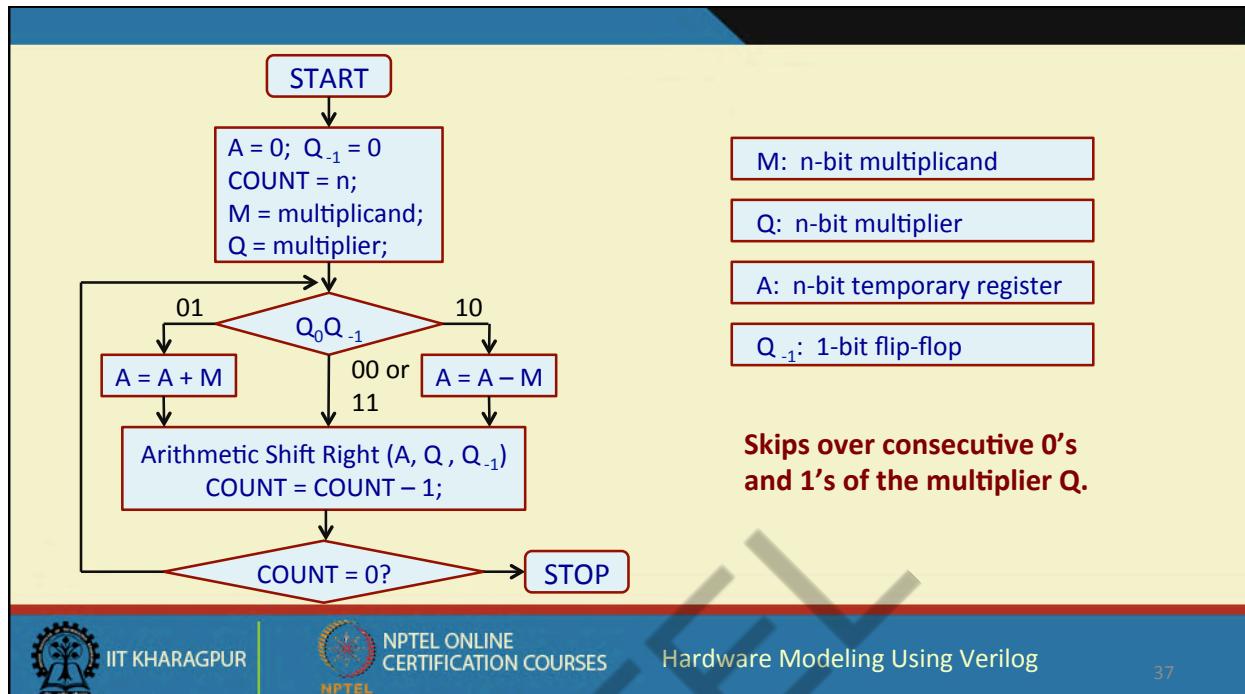


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

36

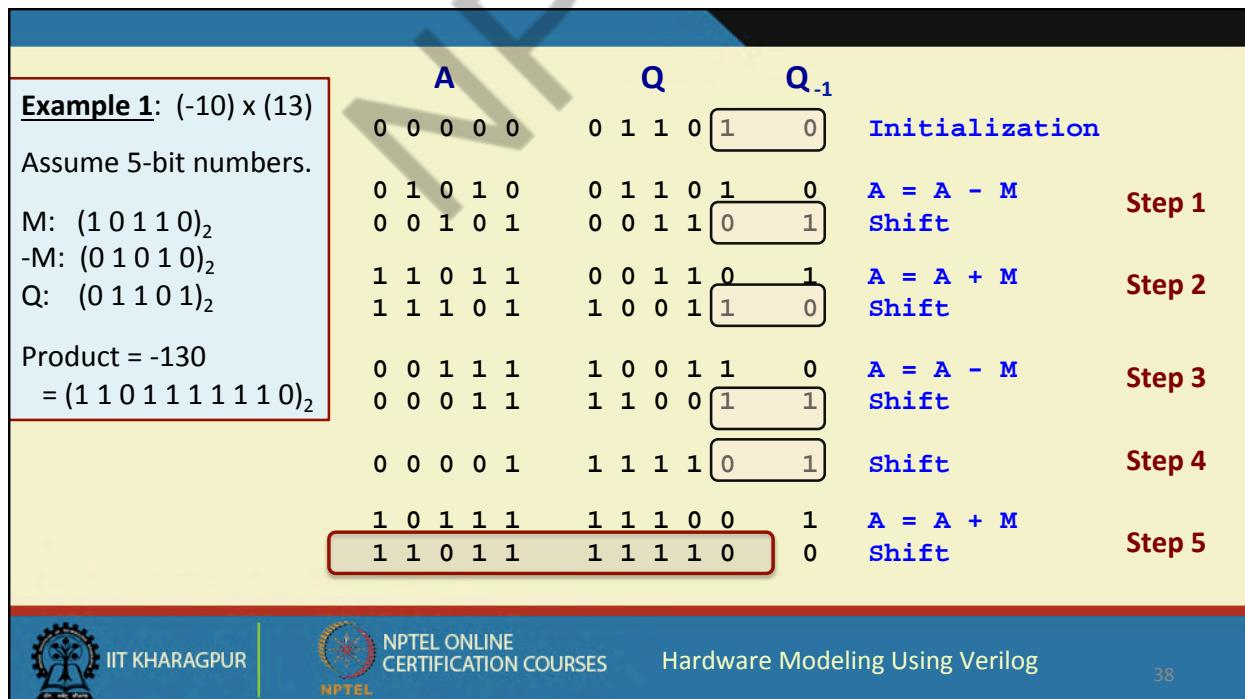


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

37



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

38

A	Q	$Q_{-1}$	
0 0 0 0 0 0	0 1 1 1 0	0 0	Initialization
0 0 0 0 0 0	0 0 1 1 1	0 0	Shift Step 1
0 0 0 0 0 0	0 0 0 1 1	1 0	Shift Step 2
0 1 1 1 1 1	0 0 0 1 1	1 0	$A = A - M$ Step 3
0 0 1 1 1 1	1 0 0 0 1	1 1	Shift Step 4
0 0 0 1 1 1	1 1 0 0 0	1 1	Shift Step 5
0 0 0 0 1 1	1 1 1 0 0	0 1	$A = A + M$ Step 6
1 0 0 1 0 0	1 1 1 0 0 0	1	
1 1 0 0 1 0	0 1 1 1 0 0	0	

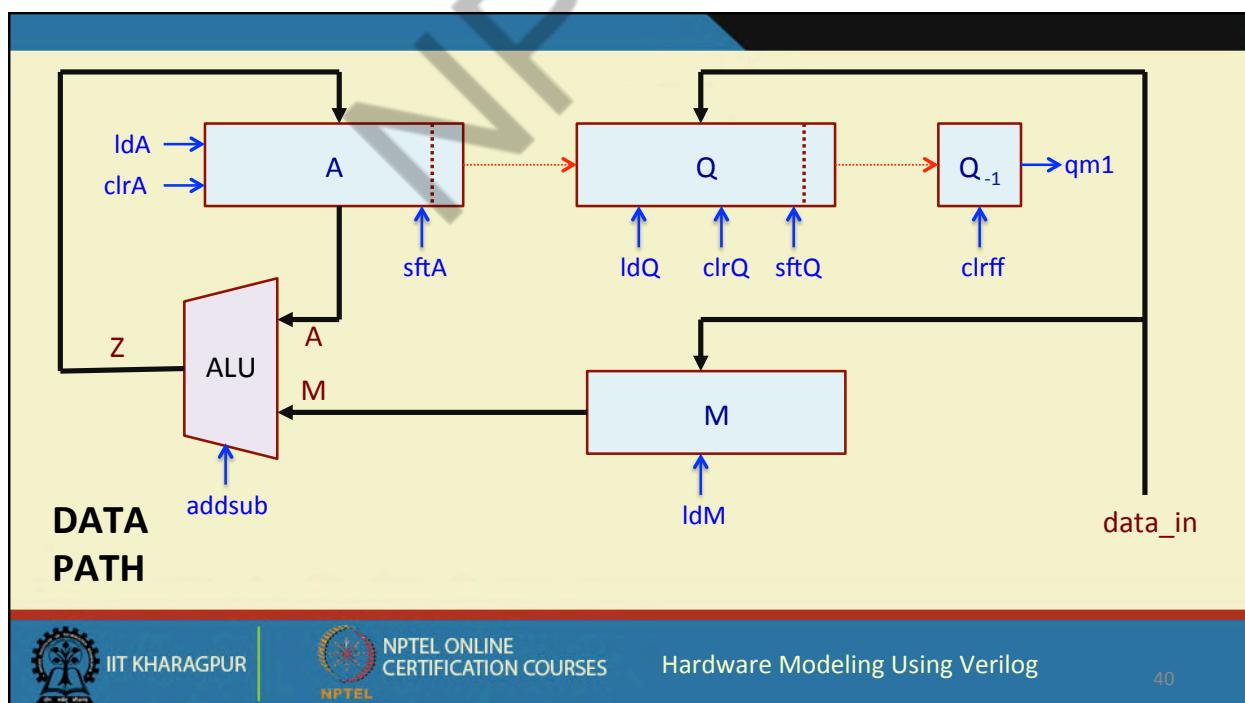


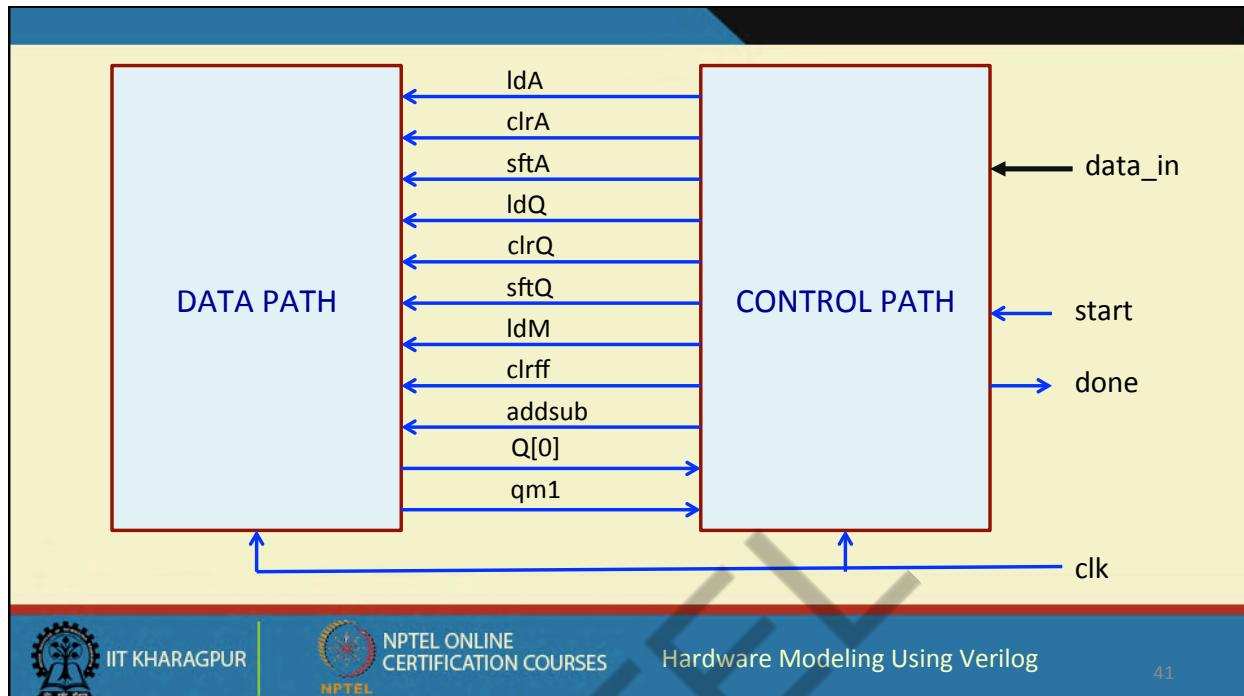
IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

39



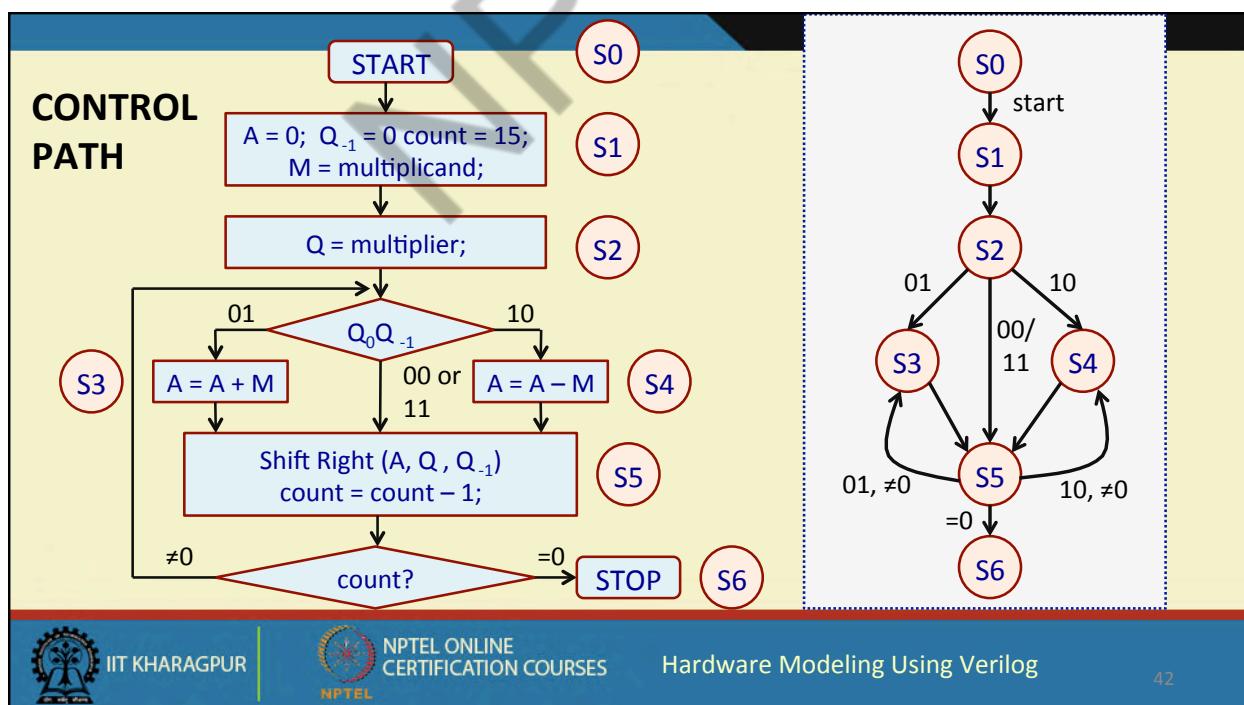


IIT KHARAGPUR

NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41



```

module BOOTH (ldA, ldQ, ldM, clrA, clrQ, clrrff, sftA, sftQ,
              addsub, decr, ldcnt, data_in, clk, qml, eqz);

    input ldA, ldQ, ldM, clrA, clrQ, clrrff, sftA, sftQ, addsub, clk;
    input [15:0] data_in;
    output qml, eqz;
    wire [15:0] A, M, Q, Z;
    wire [4:0] count;

    assign eqz = ~|count;

    shiftreg AR (A, Z, A[15], clk, ldA, clrA, sftA);
    shiftreg QR (Q, data_in, A[0], clk, ldQ, clrQ, sftQ);
    dff QM1 (Q[0], qml, clk, clrrff);
    PIP0 MR (data_in, M, clk, ldM);
    ALU AS (Z, A, M, addsub);
    counter CN (count, decr, ldcnt, clk);

endmodule

```

### THE DATA PATH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

43

```

module shiftreg (data_out,data_in,
                  s_in, clk, ld, clr, sft);
    input s_in, clk, ld, clr, sft;
    input [15:0] data_in;
    output reg [15:0] data_out;

    always @ (posedge clk)
        begin
            if (clr) data_out <= 0;
            else if (ld)
                data_out <= data_in;
            else if (sft)
                data_out <= {s_in,data_out[15:1]};
            end
        endmodule

```

```

module PIP0 (data_out,data_in, clk, load);
    input [15:0] data_in;
    input load, clk;
    output reg [15:0] data_out;

    always @ (posedge clk)
        if (load) data_out <= data_in;
    endmodule

    module dff (d, q, clk, clr);
        input d, clk, clr;
        output reg q;

        always @ (posedge clk)
            if (clr) q <= 0;
            else q <= d;
        endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

44

```

module ALU (out, in1, in2, addsub);
    input [15:0] in1, in2;
    input addsub;
    output reg [15:0] out;

    always @(*)
    begin
        if (addsub == 0) out = in1 - in2;
        else out = in1 + in2;
    end
endmodule

```

```

module counter (data_out, decr, ldcnt, clk)
    input decr, clk;
    output [4:0] data_out;

    always @ (posedge clk)
    begin
        if (ldcnt) data_out < 5'b10000;
        else if (decr) data_out <= data_out - 1;
    end
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

45

```

module controller (ldA, clrA, sftA, ldQ, clrQ, sftQ, ldM, clrrff, addsub, start,
                  decr, ldcnt, done, clk, q0, qm1);
    input clk, q0, qm1, start;
    output reg ldA, clrA, sftA, ldQ, clrQ, sftQ, ldM, clrrff, addsub, decr, ldcnt, done;
    reg [2:0] state;
    parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100, S5=3'b101, S6=3'b110;
    always @(posedge clk)
    begin
        case (state)
            S0:   if (start) state <= S1;
            S1:   state <= S2;
            S2:   #2 if ({q0,qm1}==2'b01) state <= S3;
                   else if ({q0,qm1}==1'b10) state <= S4;
                   else state <= S5;
            S3:   state <= S5;
            S4:   state <= S5;
            S5:   #2 if (({q0,qm1}==2'b01) && !eqz) state <= S3;
                   else if (({q0,qm1}==2'b10) && !eqz) state <= S4;
                   else if (eqz) state <= S6;
            S6:   state <= S6;
            default: state <= S0;
        endcase
    end

```

**THE CONTROL  
PATH**

```

always @(state)
begin
  case (state)
    S0:   begin clrA = 0; ldA = 0; sftA = 0; clrQ = 0; ldQ = 0; sftQ = 0;
          ldM = 0; clrf = 0; done = 0; end
    S1:   begin clrA = 1; clrf = 1; ldcnt = 1; ldM = 1; end
    S2:   begin clrA = 0; clrf = 0; ldcnt = 0; ldM = 0; ldQ = 1; end
    S3:   begin ldA = 1; addsub = 1; ldQ = 0; sftA = 0; sftQ = 0; decr = 0; end
    S4:   begin ldA = 1; addsub = 0; ldQ = 0; sftA = 0; sftQ = 0; decr = 0; end
    S5:   begin sftA = 1; sftQ = 1; ldA = 0; ldQ = 0; decr = 1; end
    S6:   done = 1;
    default: begin clrA = 0; sftA = 0; ldQ = 0; sftQ = 0; end
  endcase
end

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

47

- Test bench can be written similarly.
- Points to note:
  - The timing must be very clearly analyzed and signals activated at proper time instances in the test bench.
  - Otherwise, the simulation results will not come correct, though the module descriptions may be fine.
  - This cannot be taught ... requires practice and experience.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

48

## END OF LECTURE 27



## Lecture 28: SYNTHESIZABLE VERILOG

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## About the Verilog Language

- The language Verilog has a large number of features, most of which are supported by the simulation tools.
- Unfortunately, several of the language constructs are not supported by synthesis tools.
  - The language subset that can be synthesized is known as "*Synthesizable Verilog*" subset.
- Here we shall state the language features not supported by most of the synthesis tools.
  - Best be avoided if the objective is to map the design to hardware.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

51

## Synthesis Rules for Combinational Logic

- The output of a combinational logic circuit at time  $t$  should depend only upon the inputs applied at time  $t$ .
- Rules to be followed:
  - Avoid technology dependent modeling (i.e. implement functionality, not timing).
  - There must not be any feedback in the combinational circuit.
  - For "*if...else*" or "*case*" constructs, the output of the combinational function must be specified for all possible input cases.
  - If the rules are not followed, the circuit may be synthesized as sequential.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

52

## Styles for Synthesizable Combinational Logic

- The possible styles for modeling combinational logic are as follows:
  - Netlist of Verilog built-in primitives like gate instances (AND, OR, NAND, etc.).
  - Combinational UDP (*not all synthesis tools support this*).
  - Continuous assignments.
  - Functions.
  - Behavioral statements.
  - Tasks without event or delay control.
  - Interconnected modules of one or more of the above.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

53

### (a) As a Gate Netlist

```
module example (x1, x2, x3, x4, y);
  input x1, x2, x3, x4;
  output y;
  wire w1, w2, w3;
  or (w1, x1, x2);
  or (w2, x3, x4);
  xor (w3, x3, x4);
  nand (y, w1, w2, w3);
endmodule
```

Shall be synthesized in terms of gates from some target technology library.

The gate netlist is often optimized during synthesis.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

54

## (b) Using Continuous Assignment

```
module carry (cout, a, b, c);
    input a, b, c;
    output cout;

    assign cout = (a & b) | (b & c) |
                  (c & a);
endmodule
```

Shall be mapped to gates or cells from some target technology library.

The Boolean equations are optimized during synthesis.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

55

## (c) Using Procedural Blocking Assignment

```
module mux2to1 (f, in0, in1, sel);
    input in0, in1, sel;
    output reg f;

    always @(in0 or in1 or sel)
        if (sel) f = in1;
        else      f = in0;
endmodule
```

Inputs to the behavior (*here in0, in1, sel*) must be included in the event control expression; otherwise, a latch will be inferred at the output.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

56

## (d) Using Functions in Verilog

```
module fulladder (s, cout, a, b, cin);
    input a, b, cin;
    output s, cout;
    assign s = sum(a, b, cin);
    assign cout = carry(a, b, cin);
endmodule
```

```
function sum;
    input x, y, z;
    begin
        sum = x ^ y ^ z;
    end
```

```
function carry;
    input x, y, z;
    begin
        carry = (x&y) | (y&z) | (z|x);
    end
```

A function in Verilog returns a single value.

Can be used to make a code more readable.

Typically used with “assign”.

Input arguments appear in same order.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

57

```
module fulladder (s, cout, a, b, cin);
    input a, b, cin;
    output reg s, cout;

    always @(a or b or cin)
        FA (s, cout, a, b, cin);

    task FA;
        output sum, carry;
        input A, B, C;
        begin
            #2 sum = A ^ B ^ C;
            carry = (A&B) | (B&C) | (C|A);
        end
    endtask
endmodule
```

## (e) Using Tasks

The arguments must be specified in the same order as they appear in the task declaration.

More than one output value can be returned.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

58

## Difference between Function and Task

Function	Task
A function can call another function but not another task.	A task can call other tasks and functions.
A function executes in 0 simulation time.	A task may execute in nonzero simulation time.
A function cannot contain any delay, event, or timing control statement.	A task can contain delay, event, or timing control statements.
A function always return a single value.	A task can pass multiple values through " <i>output</i> " and " <i>inout</i> " type arguments.
A function must have at least one input argument.	A task can have zero or more arguments of type " <i>input</i> ", " <i>output</i> ", or " <i>inout</i> ".



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

59

## Constructs to Avoid for Combinational Synthesis

- Edge-dependent event control.
- Combinational feedback loops.
- Procedural or continuous assignment containing event or delay control.
- Procedural loops with timing.
- Data dependent loops.
- Sequential user defined primitives (UDPs).
- Other miscellaneous constructs like "*fork ... join*", "*wait*", "*disable*", etc.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

60

## Summary: Synthesizable Verilog Constructs

- “*module* ... *endmodule*”
- Instantiation of a synthesizable module
- “*always*” construct
- “*assign*” statements
- Built-in gate primitives
- User defined primitives – combinational only
- “*parameter*” statement
- “*functions*” and “*tasks*”
- “*for*” loop
- Almost all operators
- Blocking and non-blocking assignments
- “*if ... else*”, “*case*”, “*casex*” and “*casez*”
- Bits and part select of vectors



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

## Non-Synthesizable Constructs

- “*initial*” construct
- Delays in assignments and test benches.
- “*time*” construct
- “*real*” data type
- The operators “*==*” and “*!=*”
- “*fork ... join*” constructs
- “*force ... release*” constructs
- Variables in loop control



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

62

## END OF LECTURE 28



## Lecture 29: SOME RECOMMENDED PRACTICES

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Naming Conventions

- File Naming
  - A file must contain only one design unit, contained in a single “*module ... endmodule*” construct.
  - All Verilog files must have an extension of “*.v*”.
- Naming of variables, signals and other objects
  - Names must be composed of alphanumeric characters or underscores.
  - Names must start with a letter, and not a number or underscore.
  - All names must be unique irrespective of case.
  - Parameters and constant names must be given in UPPER CASE (e.g. *PI*, *DELAY*, etc.).
  - Signals and variable names must be in lower case, and must be meaningful names.
  - A constant name must describe the purpose of the constant (e.g. *reg\_a\_enable*).



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

65

- Construct names such as modules, functions and tasks must also be meaningful.
- For names composed of several words, underscore must be used (e.g. *load\_input*).
- When a signal uses active low polarity, it must use the suffix “*\_b*” (e.g. *clear\_b*).
- Signals that are used for clocking that do not have the word “*clock*” or “*clk*” already in their names, must use the suffix “*\_clk*” (e.g. *bus\_clk*).
- Unrelated signals must not be bundled into buses.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

66

## Comments

- Comments are required to describe the functionality of a design unit.
  - Each file must contain a file header, that follows some convention.
  - The header must include the name of the file.
  - The header must include the highest level construct contained in the file.
  - Every file header must include the originating section or department, author, and author's email address.
  - Header must include release history whenever such a change is registered.
  - The header must contain a purpose section explaining the functionality of the module.
  - The header must contain information describing the parameters being used in the construct.
  - The number of clock domains and clocking strategy must be documented.
  - Critical timing including external timing relationships must be documented.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

67

## Coding Style

- Good coding style helps in easy understanding of the code and maintainability.
  - Write code with proper indentation in a tabular format.
  - A constant indentation of *2 to 4 spaces* must be used for code alignment; do not use tab stops (tab stops interpretation varies from system to system).
  - Use spaces and empty lines to increase the readability of code.
  - One line must not contain more than one Verilog statement.
  - Use one line comments using “//”; avoid multi-line comments using “/\* ... \*/”.
  - Keep line length less than 80 characters, so as to avoid line wraps.
  - When declaring ports, declare *one port per line*. Descriptive comment must follow each port listing.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

68

## Module Partitioning

- Used for reducing complexity, and also to minimize the chances of error.
  - Procedure, tasks and functions must not access or modify signals / variables not passed as parameter to the module.
  - If we use a gated clock, internally generated clock, or use both edges of a clock, the clock generation circuitry must be kept in a separate module at the top level or at the same logical level in the hierarchy as the block to which the clocks apply.
  - Separate clock domains (e.g. *slow clock* and *fast clock*) must be partitioned into separate blocks.
  - The design should be partitioned so as to minimize the number of interface signals.
  - Do not mix structural and behavioral RTL code within a construct.
  - State machines and asynchronous logic must be partitioned in a separate block.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

69

## General Coding Techniques

- Some of the general guidelines for coding are as follows.
  - The expression in a condition must be a 1-bit value.  
Replace “*if (bus) data\_avail = 1*” by “*if (bus > 0) data\_avail = 1*”.
  - Do not assign signals to “*x*”.
  - Do not infer latches in functions; a function is supposed to synthesize combinational logic.
  - Operand sizes should match.
  - Use parentheses in complex expressions.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

70

## General Guidelines for Synthesis

- Some guidelines for synthesizable blocks are as follows.
  - All “*always*” blocks inferring combinational logic or a latch must have a sensitivity list containing all input signals.
  - Only one clock must be used in an “*always*” block.
  - “*wait*” and “#*delay*” statements must not be used.
  - Conditional expressions must be specified completely; i.e. value must be assigned to a variable under all conditions.
  - The “*initial*” statement must not be used.
  - Expressions must not be used in port connections.
  - Verilog user defined primitives must not be used.
  - Use non-blocking assignments in edge-sensitive constructs.

In *wait(expression)*, the expression is evaluated. If false, execution is suspended until it becomes true.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

71

- The internal “*wire*” declarations must follow the port I/O declarations at the top of the module.
- Use explicit port references in module instantiations.

```
full_adder FA (.sum(S),  
                .cout(Co),  
                .in1(A),  
                .in2(B),  
                .cin(Ci) );
```

- Use of “*cased*” statement is not allowed, which treats “*x*” and “*z*” states as don’t cares in synthesis.
- Use parameters for state encoding.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

72

- Only use ports of type “*input*” or “*output*”; ports of type “*inout*” (bidirectional) should be avoided.
- In procedural blocks, blocking assignments should be used for modeling combinational logic.
- The “*default*” case assignment must be used for all combinational logic case statement descriptions.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

## An Example

```
// Copyright (c) 2017 IIT Kharagpur
// -----
// FILE NAME: counter.v
// TYPE: module
// DEPARTMENT: Computer Sceience and Engineering
// AUTHOR: Prof. Indranil Sengupta
// AUTHOR'S EMAIL: indranil@cse.iitkgp.ernet.in
// -----
// Release history
// VERSION DATE AUTHOR DESCRIPTION
// 1.0 10/08/2016 indranil Initial version
// 2.0 12/07/2017 subir Updated version with clear
// 2.1 16/08/2017 indranil Asynchronous clear
// -----
// KEYWORDS: binary counter, asynchronous clear
// -----
// PURPOSE: 16-bit binary counter
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74

```
module counter (
    data,
    clear,
    clock
);

output reg [15:0] data; // The 16-bit count value

input clear;           // Asynchronous clear
input clock;          // Counter clock

// 16-bit binary counter with asynchronous clear
always @(posedge clock or negedge clear)
    if (!clear)
        data <= 16'b0000000000000000;
    else
        data <= data + 1;

endmodule
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

75

## END OF LECTURE 29



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

76



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 30: MODELING MEMORY

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# How to Model Memory?

- Memory is typically included by instantiating a pre-designed module from a design library.
- Alternatively, we can model memories using two-dimensional arrays.
  - Array of register variables (behavioral model).
  - Mainly used for simulation purposes.
  - Even used for the synthesis of small-size memories.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module memory_model ( ..... )
...
reg [7:0] mem [0:1023];
...
endmodule
```

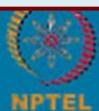
```
module memory_model ( ..... )
reg [7:0] mem [0:1023];
initial begin
mem[0] = 8'b01001101;
mem[4] = 8'b00000000;
end
endmodule
```

## Typical Example

- Each memory word is of type [7:0], i.e. 8 bits.
- The memory words can be accessed as `mem[0]`, `mem[1]`, ..., `mem[1023]`.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# How to Initialize Memory?

- By reading memory data patterns from a specified disk file.
  - Used for simulation.
  - Used in test benches.
- Two Verilog functions can be used:

**\$readmemb (filename, memname, startaddr, stopaddr)**

*(Data is read in binary format)*

**\$readmemh (filename, memname, startaddr, stopaddr)**

*(Data is read in hexadecimal format)*

- If “*startaddr*” and “*stopaddr*” are omitted, the entire memory is read.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 1: Initializing a memory from file

```
module memory_model ( ..... );
    reg [7:0] mem[0:1023];
    initial
        begin
            $readmemh ("mem.dat", mem);
        end
    endmodule
```

```
module memory_model ( ..... );
    reg [7:0] mem[0:1023];
    initial
        begin
            $readmemb ("mem.dat", mem,
                      200, 50);
        end
    endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 2: Single-port RAM with synchronous read/write

```
module ram_1 (addr, data, clk, rd, wr, cs);
    input [9:0] addr;      input clk, rd, wr, cs;
    inout [7:0] data;
    reg [7:0] mem [1023:0];   reg [7:0] d_out;

    assign data = (cs && rd) ? d_out : 8'bz;
    always @(posedge clk)
        if (cs && wr && !rd) mem[addr] = data;
    always @(posedge clk)
        if (cs && rd && !wr) d_out = mem[addr];
endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 3: Single-port RAM with asynchronous read/write

```
module ram_2 (addr, data, rd, wr, cs);
    input [9:0] addr;      input rd, wr, cs;
    inout [7:0] data;
    reg [7:0] mem[1023:0];   reg [7:0] d_out;

    assign data = (cs && rd) ? d_out : 8'bz;
    always @(addr or data or rd or wr or cs)
        if (cs && wr && !rd) mem[addr] = data;
    always @(addr or rd or wr or cs)
        if (cs && rd && !wr) d_out = mem[addr];
endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 4: A ROM / EPROM

```
module rom (addr, data, rd_en, cs);
    input [2:0] addr;      input rd_en, cs;
    output reg [7:0] data;
    always @(addr or rd_en or cs)
        case (addr)
            0: data = 22;
            1: data = 45;
            .....
            7: data = 12;
        endcase
    endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# An Important Point to Note

- Some simulation or synthesis tools give inconsistent behavior when using the “*inout*” data type.
  - Such “*inout*” bidirectional data should be avoided.
- A better way to design a memory unit is to keep the data input and data output bus signal lines separate.
  - An example memory description with separate data buses is shown on the next slide.



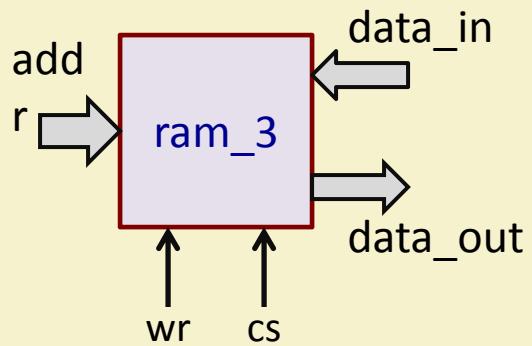
IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 4



```
module ram_3 (data_out, data_in, addr, wr, cs);  
    parameter addr_size = 10, word_size = 8,  
            memory_size = 1024;  
    input [addr_size-1:0] addr;  
    input [word_size-1:0] data_in;  
    input wr, cs;  
    output [word_size-1:0] data_out;  
    reg [word_size-1:0] mem [memory_size-1:0];  
  
    assign data_out = mem[addr];  
    always @(wr or cs)  
        if (wr) mem[addr] = data_in;  
endmodule
```



IIT KHARAGPUR

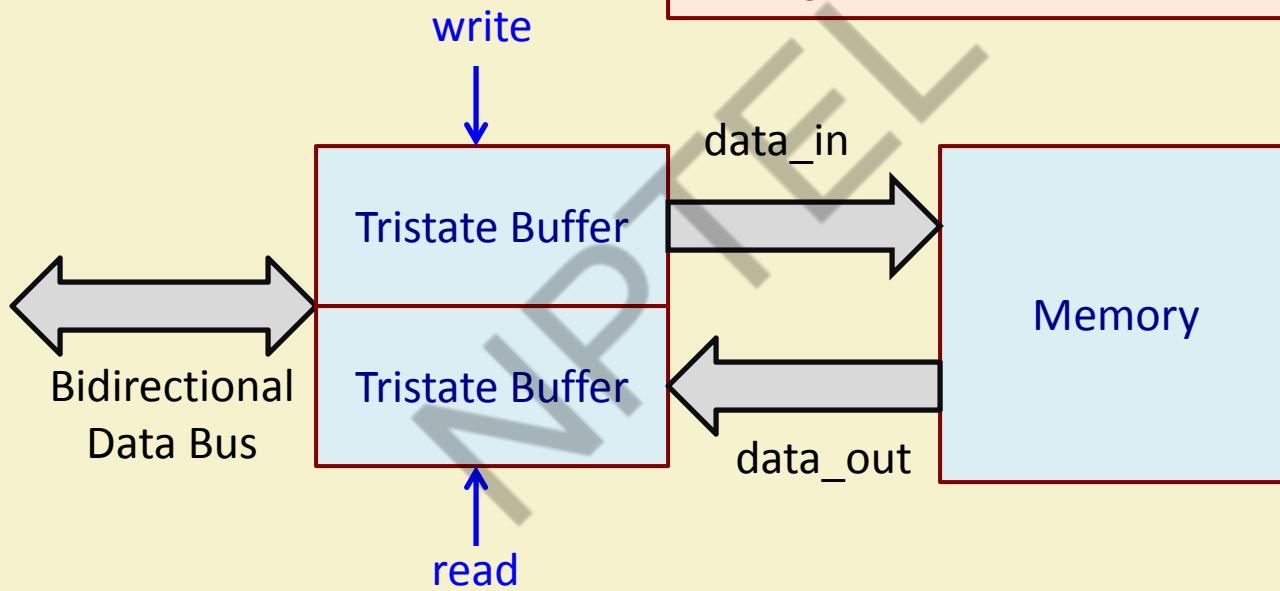


NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- For bidirectional data bus, tristate buffers can be included explicitly.

```
tri [7:0] Bus;
wire [7:0] data_out, data_in;
assign Bus = read ? data_out : 8'hzz;
assign data_in = write ? Bus : 8'hzz;
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module RAM_test;
    reg [9:0] address;
    wite [7:0] data_out;
    reg [7:0] data_in;
    reg write, select;
    integer k, myseed;

    ram_3 RAM (data_out, data_in, address, write, select);

initial
begin
    for (k=0; k<=1023; k=k+1)
        begin
            address = k;
            data = (k + k) % 256; read = 0; write = 1; select = 1;
            #2 write = 0; select = 0;
        end
end
```

## Simple Test Bench using ram\_3



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

repeat (20)
begin
    #2 address = $random(myseed) % 1024;
    write = 0; select = 1;
    $display ("Address: %5d, Data: %4d", address,
              data);
    #2 select = 0;
end
end

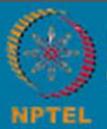
initial myseed = 35;
endmodule

```

Address:	0	Data:	0
Address:	960	Data:	128
Address:	482	Data:	196
Address:	693	Data:	106
Address:	246	Data:	236
Address:	228	Data:	200
Address:	148	Data:	40
Address:	767	Data:	254
Address:	355	Data:	198
Address:	259	Data:	6
Address:	21	Data:	42
Address:	872	Data:	208
Address:	758	Data:	236
Address:	193	Data:	130
Address:	909	Data:	26
Address:	632	Data:	240
Address:	719	Data:	158
Address:	214	Data:	172
Address:	67	Data:	134
Address:	908	Data:	24



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 30



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 31: MODELING REGISTER BANKS

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Introduction

- A register bank or register file is a group of registers, any of which can be randomly accessed.
  - Commonly used in computers to store the user-accessible registers.
  - For example, in the MIPS32 processor, there are 32 32-bit registers, referred to as *R<sub>0</sub>, R<sub>1</sub>, ..., R<sub>31</sub>*.
- Can be implemented in Verilog as independent registers, or as an array of registers similar to a memory.
- Registers banks often allow concurrent accesses.
  - MIPS32 allows *2 register reads and 1 register write every clock cycle*.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- We show various ways of designing a register bank that supports two reads and one write simultaneously.
  - It is assumed that the same register is not read and written simultaneously.

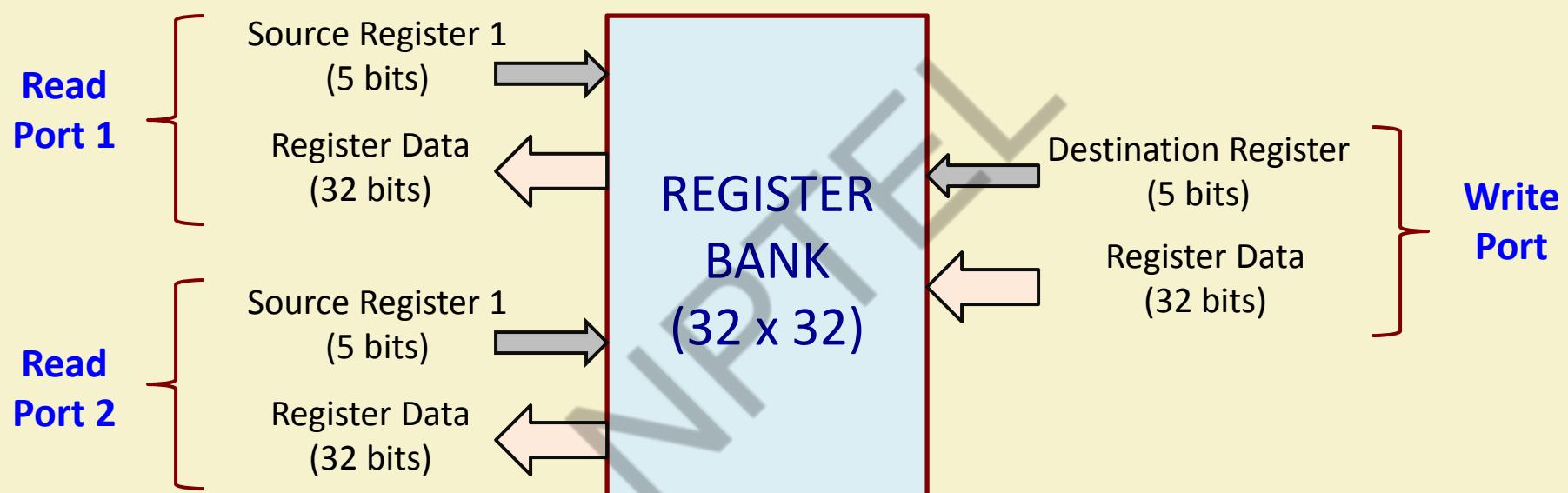


IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
// 4 x 32 register file

module regbank_v1 (rdData1, rdData2, wrData, sr1, sr2, dr, write, clk);
    input clk, write;
    input [1:0] sr1, sr2, dr; // Source and destination registers
    input [31:0] wrData;
    output reg [31:0] rdData1, rdData2;
    reg [31:0] R0, R1, R2, R3;

    always @(*)
        begin
            case (sr1)
                0: rdData1 = R0;
                1: rdData1 = R1;
                2: rdData1 = R2;
                3: rdData1 = R3;
                default: rdData1 = 32'hxxxxxxxxx;
            endcase
        end

```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
always @(*)
begin
  case (sr2)
    0: rdData2 = R0;
    1: rdData2 = R1;
    2: rdData2 = R2;
    3: rdData2 = R3;
    default: rdData2 = 32'hxxxxxxxxx;
  endcase
end
always @(posedge clk)
begin
  if (write)
    case (dr)
      0: R0 <= wrData;
      1: R1 <= wrData;
      2: R2 <= wrData;
      3: R3 <= wrData;
    endcase
  end
endmodule
```

This way of modeling is feasible if the number of registers is small.



```

// 4 x 32 register file

module regbank_v2 (rdData1, rdData2, wrData, sr1, sr2, dr, write, clk);
    input clk, write;
    input [1:0] sr1, sr2, dr; // Source and destination registers
    input [31:0] wrData;
    output [31:0] rdData1, rdData2;
    reg [31:0] R0, R1, R2, R3;

    assign rdData1 = (sr1 == 0) ? R0 :
                    (sr1 == 1) ? R1 :
                    (sr1 == 2) ? R2 :
                    (sr1 == 3) ? R3 : 0;
    assign rdData2 = (sr2 == 0) ? R0 :
                    (sr2 == 1) ? R1 :
                    (sr2 == 2) ? R2 :
                    (sr2 == 3) ? R3 : 0;

    always @(posedge clk)
        begin
            if (write)
                case (dr)
                    0: R0 <= wrData;
                    1: R1 <= wrData;
                    2: R2 <= wrData;
                    3: R3 <= wrData;
                endcase
        end
endmodule

```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
// 32 x 32 register file

module regbank_v3 (rdData1, rdData2, wrData, sr1, sr2, dr, write, clk);
    input clk, write;
    input [4:0] sr1, sr2, dr; // Source and destination registers
    input [31:0] wrData;
    output [31:0] rdData1, rdData2;

    reg [31:0] regfile[0:31];

    assign rdData1 = regfile[sr1];
    assign rdData2 = regfile[sr2];

    always @(posedge clk)
        if (write) regfile[dr] <= wrData;
endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

// 32 x 32 register file with reset facility

module regbank_v4 (rdData1, rdData2, wrData, sr1, sr2, dr, write, reset, clk);
    input clk, write, reset;
    input [4:0] sr1, sr2, dr;      // Source and destination registers
    input [31:0] wrData;
    output [31:0] rdData1, rdData2;
    integer k;

    reg [31:0] regfile[0:31];

    assign rdData1 = regfile [sr1];
    assign rdData2 = regfile [sr2];

```

```

always @(posedge clk)
begin
    if (reset) begin
        for (k=0; k<32; k=k+1) begin
            regfile[k] <= 0;
        end
    end
    else begin
        if (write)
            regfile[dr] <= wrData;
    end
end
endmodule

```



```
module regfile_test;

reg [4:0] sr1, sr2, dr;
reg[31:0] wrData;
reg write, reset, clk;
wire [31:0] rdData1, rdData2;
integer k;

regbank_v4 REG (rdData1, rdData2, wrData, sr1, sr2, dr, write, reset, clk);

initial clk = 0;

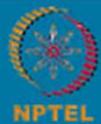
always #5 clk = !clk;

initial
begin
$dumpfile ("regfile.vcd"); $dumpvars (0, regfile_test);
#1 reset = 1; write = 0;
#5 reset = 0;
end
```

A test bench to verify  
operation of the register file



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
initial
begin
#7
for (k=0; k<32; k=k+1)
begin
    dr = k; wrData = 10* k; write = 1;
    #10 write = 0;
end

#20
for (k=0; k<32; k=k+2)
begin
    sr1 = k; sr2 = k+1;
    #5;
    $display ("reg[%2d] = %d, reg[%2d] = %d", sr1, rdData1, sr2, rdData2);
end

#2000 $finish;
end
endmodule
```



IIT KHARAGPUR

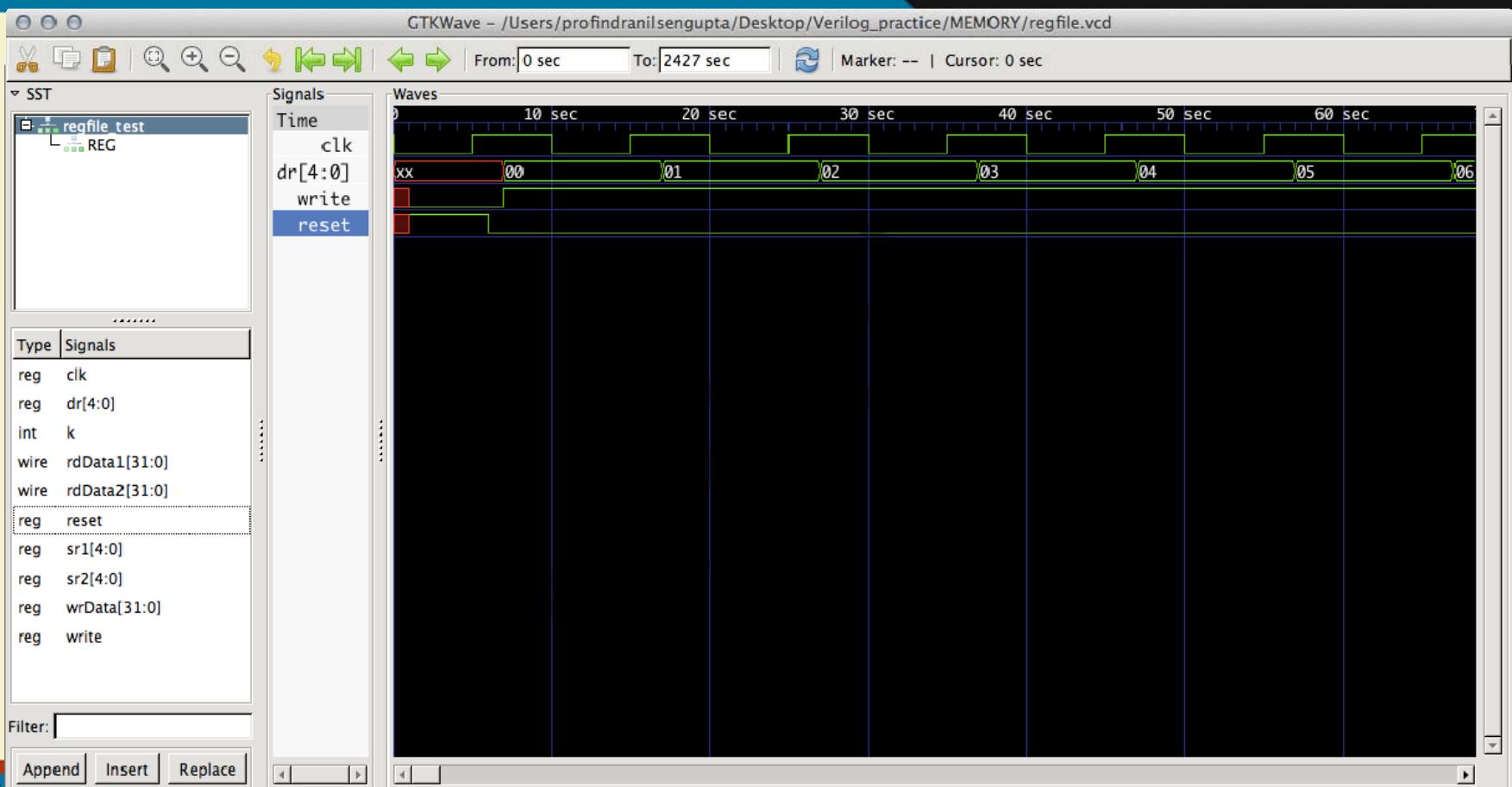


NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

reg[ 0 ] =	0 , reg[ 1 ] =	10
reg[ 2 ] =	20 , reg[ 3 ] =	30
reg[ 4 ] =	40 , reg[ 5 ] =	50
reg[ 6 ] =	60 , reg[ 7 ] =	70
reg[ 8 ] =	80 , reg[ 9 ] =	90
reg[10] =	100 , reg[11] =	110
reg[12] =	120 , reg[13] =	130
reg[14] =	140 , reg[15] =	150
reg[16] =	160 , reg[17] =	170
reg[18] =	180 , reg[19] =	190
reg[20] =	200 , reg[21] =	210
reg[22] =	220 , reg[23] =	230
reg[24] =	240 , reg[25] =	250
reg[26] =	260 , reg[27] =	270
reg[28] =	280 , reg[29] =	290
reg[30] =	300 , reg[31] =	310



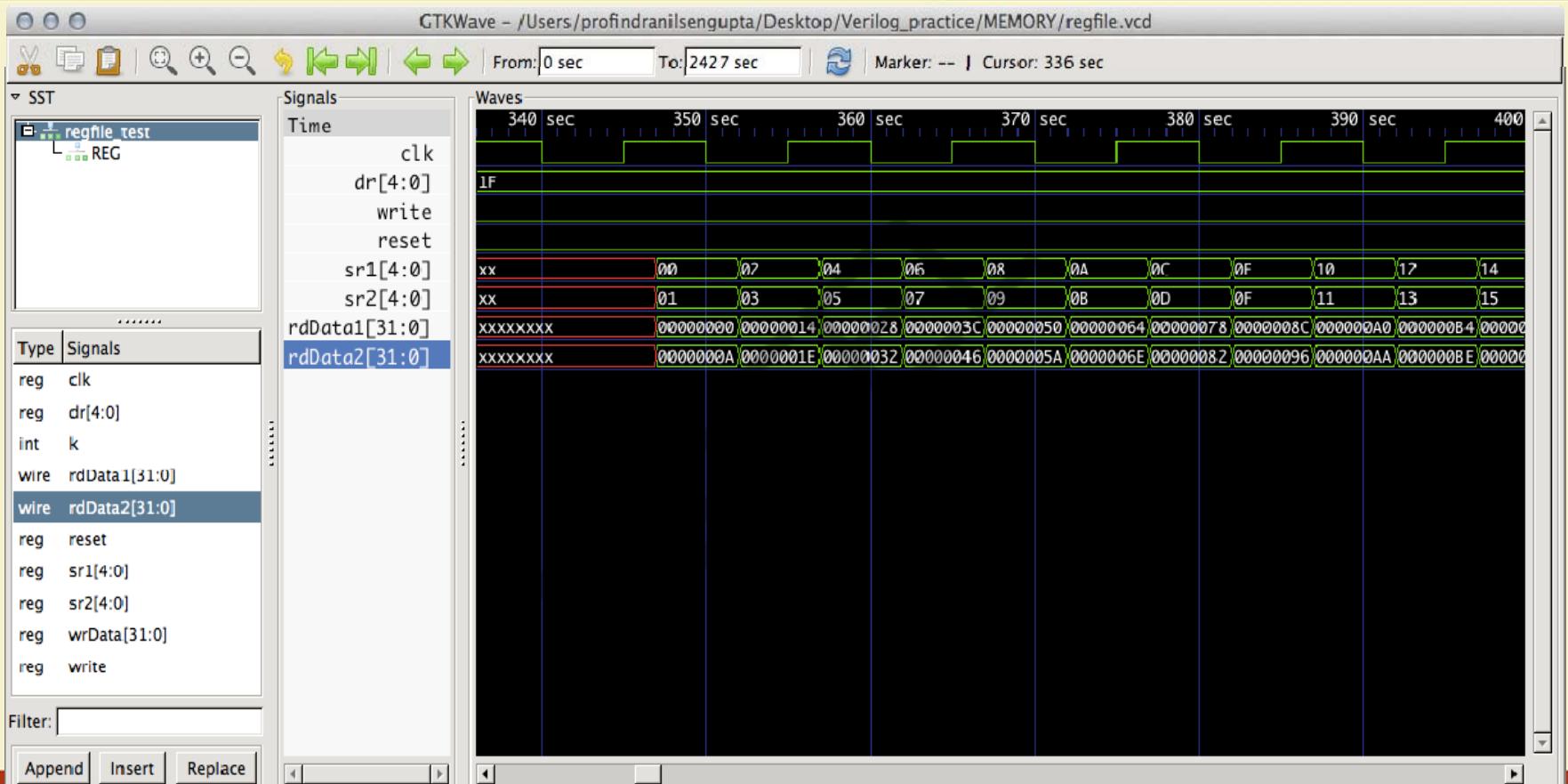


IIT KHARAGPUR

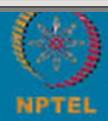


NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 31



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 32: BASIC PIPELINING CONCEPTS

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# What is Pipelining?

- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of  $k$  sub-computations (or stages).
  - Very nominal increase in the cost of implementation.
  - Very significant speedup (ideally,  $k$ ).
- Where are pipelining used in a computer system?
  - **Instruction execution**: Several instructions executed in some sequence.
  - **Arithmetic computation**: Same operation carried out on several data sets.
  - **Memory access**: Several memory accesses to consecutive locations are made.



IIT KHARAGPUR

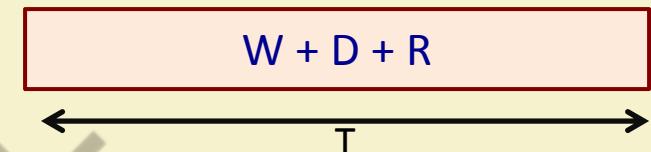


NPTEL  
ONLINE  
CERTIFICATION COURSES

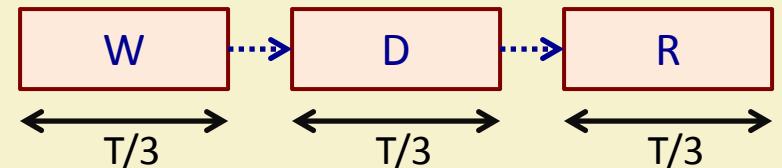
Hardware Modeling Using Verilog

## A Real-life Example

- Suppose you have built a machine  $M$  that can wash ( $W$ ), dry ( $D$ ), and iron ( $R$ ) clothes, one cloth at a time.
  - Total time required is  $T$ .
- As an alternative, we split the machine into three smaller machines  $M_W$ ,  $M_D$  and  $M_R$ , which can perform the specific task only.
  - Time required by each of the smaller machines is  $T/3$  (say).



For  $N$  clothes, time  $T_1 = N \cdot T$



For  $N$  clothes, time  $T_3 = (2 + N) \cdot T/3$



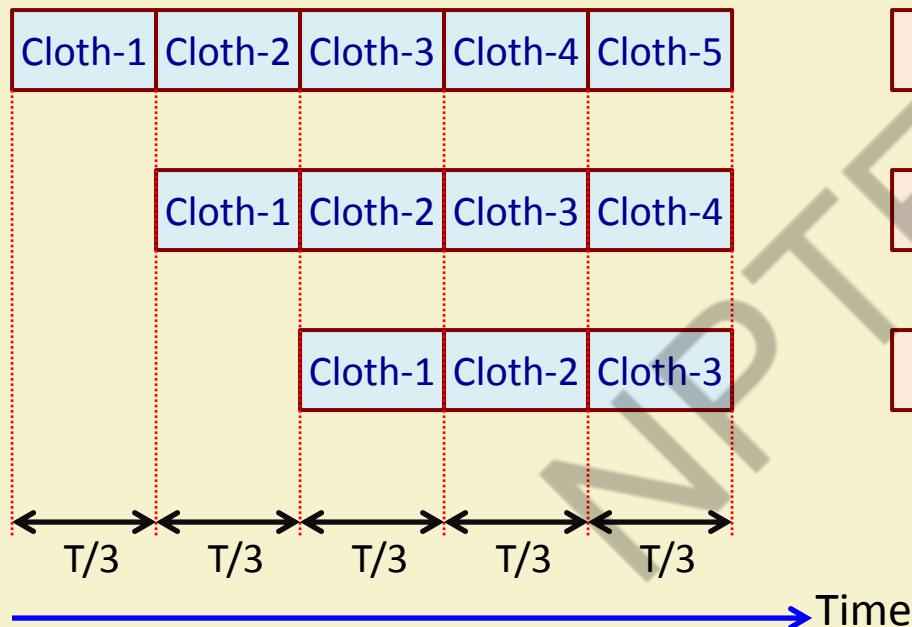
IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# How does the pipeline work?



Finishing times:

- Cloth-1 –  $3.T/3$
- Cloth-2 –  $4.T/3$
- Cloth-3 –  $5.T/3$
- ...
- Cloth-N –  $(2 + N).T/3$



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Extending the Concept to Processor Pipeline

- The same concept can be extended to hardware pipelines.
- Suppose we want to attain  $k$  times speedup for some computation.
  - Alternative 1: Replicate the hardware  $k$  times → cost also goes up  $k$  times.
  - Alternative 2: Split the computation into  $k$  stages → very nominal cost increase.
- Need for buffering:
  - In the washing example, we need a tray between machines (W & D, and D & R) to keep the cloth temporarily before it is accepted by the next machine.
  - Similarly in hardware pipeline, we need a *latch* between successive stages to hold the intermediate results temporarily.



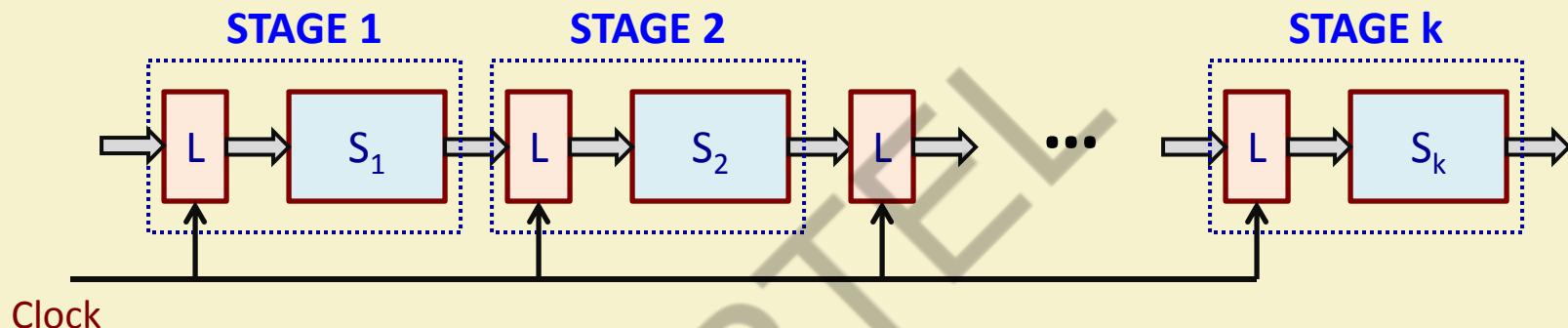
IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Model of a Synchronous k-stage Pipeline



- The latches are made with master-slave flip-flops, and serve the purpose of isolating inputs from outputs.
- The pipeline stages are typically combinational circuits.
- When **Clock** is applied, all latches transfer data to the next stage simultaneously.



IIT KHARAGPUR

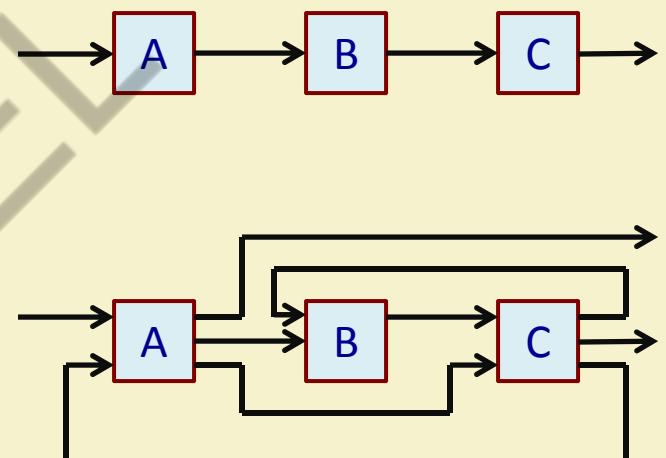


NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Structure of the Pipeline

- **Linear Pipeline:** The stages that constitute the pipeline are executed one by one in sequence (say, from left to right).
- **Non-linear Pipeline:** The stages may not execute in a linear sequence (say, a stage may execute more than once for a given data set).



A possible sequence: A, B, C, B, C, A, C, A



IIT KHARAGPUR

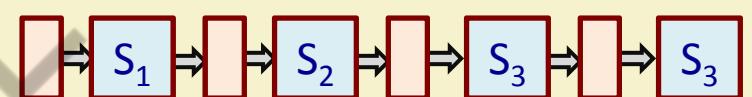


NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Reservation Table

- The *Reservation Table* is a data structure that represents the utilization pattern of successive stages in a synchronous pipeline.
  - Basically a space-time diagram of the pipeline that shows precedence relationships among pipeline stages.
    - X-axis shows the time steps
    - Y-axis shows the stages
  - Number of columns give evaluation time.
  - The reservation table for a 4-stage linear pipeline is shown.



	1	2	3	4
S <sub>1</sub>	X			
S <sub>2</sub>		X		
S <sub>3</sub>			X	
S <sub>4</sub>				X



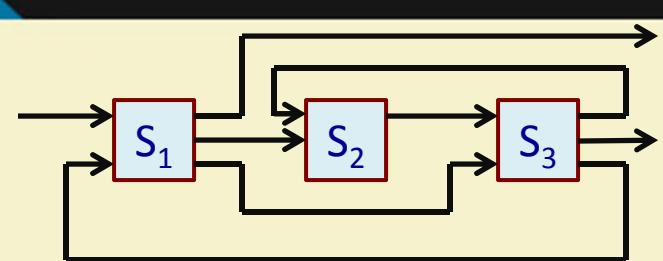
IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- Reservation table for a 3-stage dynamic multi-function pipeline is shown.
  - Contains feedforward and feedback connections.
  - Two functions X and Y.
- Some characteristics:
  - *Multiple X's in a row* :: repeated use of the same stage in different cycles.
  - *Contiguous X's in a row* :: extended use of a stage over more than one cycles.
  - *Multiple X's in a column* :: multiple stages are used in parallel during a clock cycle.



	1	2	3	4	5	6	7	8
$S_1$	X					X		X
$S_2$		X		X				
$S_3$			X		X		X	

	1	2	3	4	5	6
$S_1$	Y				Y	
$S_2$			Y			
$S_3$		Y		Y		Y



# Speedup and Efficiency

Some notations:

$\tau$  :: clock period of the pipeline

$t_i$  :: time delay of the circuitry in stage  $S_i$

$d_L$  :: delay of a latch

Maximum stage delay

$$\tau_m = \max \{t_i\}$$

Thus,

$$\tau = \tau_m + d_L$$

Pipeline frequency

$$f = 1 / \tau$$

- If one result is expected to come out of the pipeline every clock cycle,  $f$  will represent the maximum throughput of the pipeline.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- The total time to process  $N$  data sets is given by

$$T_k = [(k - 1) + N] \cdot \tau$$

$(k - 1) \tau$  time required to fill the pipeline  
 1 result every  $\tau$  time after that  $\rightarrow$  total  $N \cdot \tau$

- For an equivalent non-pipelined processor (i.e. one stage), the total time is

$$T_1 = N \cdot k \cdot \tau$$

(ignoring the latch overheads)

- Speedup of the  $k$ -stage pipeline over the equivalent non-pipelined processor:

$$S_k = \frac{T_1}{T_k} = \frac{N \cdot k \cdot \tau}{k \cdot \tau + (N - 1) \cdot \tau} = \frac{N \cdot k}{k + (N - 1)}$$

As  $N \rightarrow \infty$ ,  $S_k \rightarrow k$



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- Pipeline efficiency:
  - How close is the performance to its ideal value?

$$E_k = \frac{S_k}{k} = \frac{N}{k + (N - 1)}$$

- Pipeline throughput:
  - Number of operations completed per unit time.

$$H_k = \frac{N}{T_k} = \frac{N}{[k + (N - 1)].\tau}$$

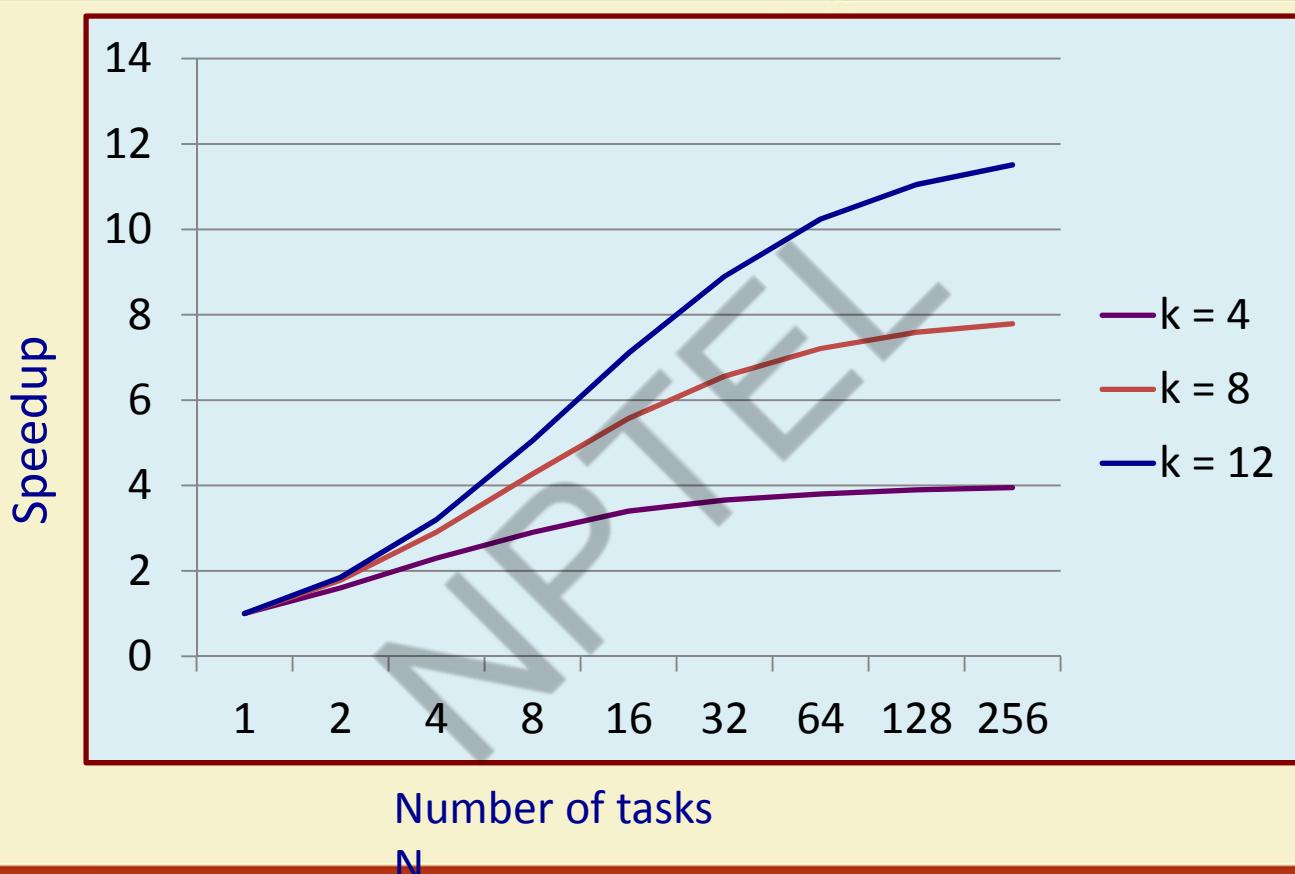


IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



# Clock Skew / Jitter / Setup time

- The minimum clock period of the pipeline must satisfy the inequality:

$$\tau \geq t_{\text{skew+jitter}} + t_{\text{logic+setup}}$$

- Definitions:
  - *Skew*: Maximum delay difference between the arrival of clock signals at the stage latches.
  - *Jitter*: Maximum delay difference between the arrival of clock signal at the same latch.
  - *Logic delay*: Maximum delay of the slowest stage in the pipeline.
  - *Setup time*: Minimum time a signal needs to be stable at the input of a latch before it can be captured.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 32



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 33: PIPELINE MODELING (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# A Simple Example

- We consider the example of a very simple 3-stage pipeline.
  - Four  $N$ -bit unsigned integers  $A$ ,  $B$ ,  $C$  and  $D$  as inputs.
  - An  $N$ -bit unsigned integer  $F$  as output.
  - The following computations are carried out in the stages:
    - a)  $S1: \quad x1 = A + B; \quad x2 = C - D;$
    - b)  $S2: \quad x3 = x1 + x2;$
    - c)  $S3: \quad F = x3 * D;$
  - Point to note:
    - Input  $D$  is used in  $S1$  as well as  $S3$ .
    - So the value of  $D$  must be forwarded to  $S2$  and then to  $S3$ .

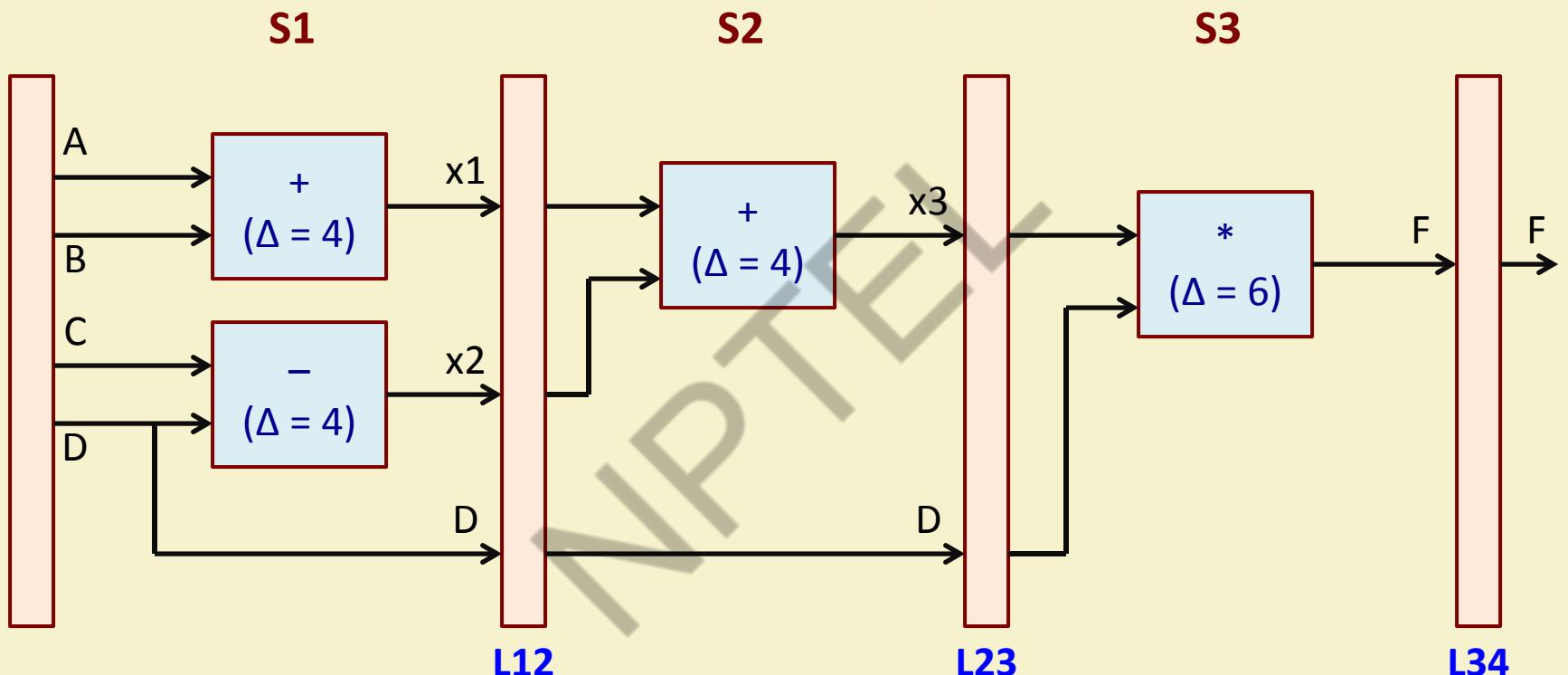


IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

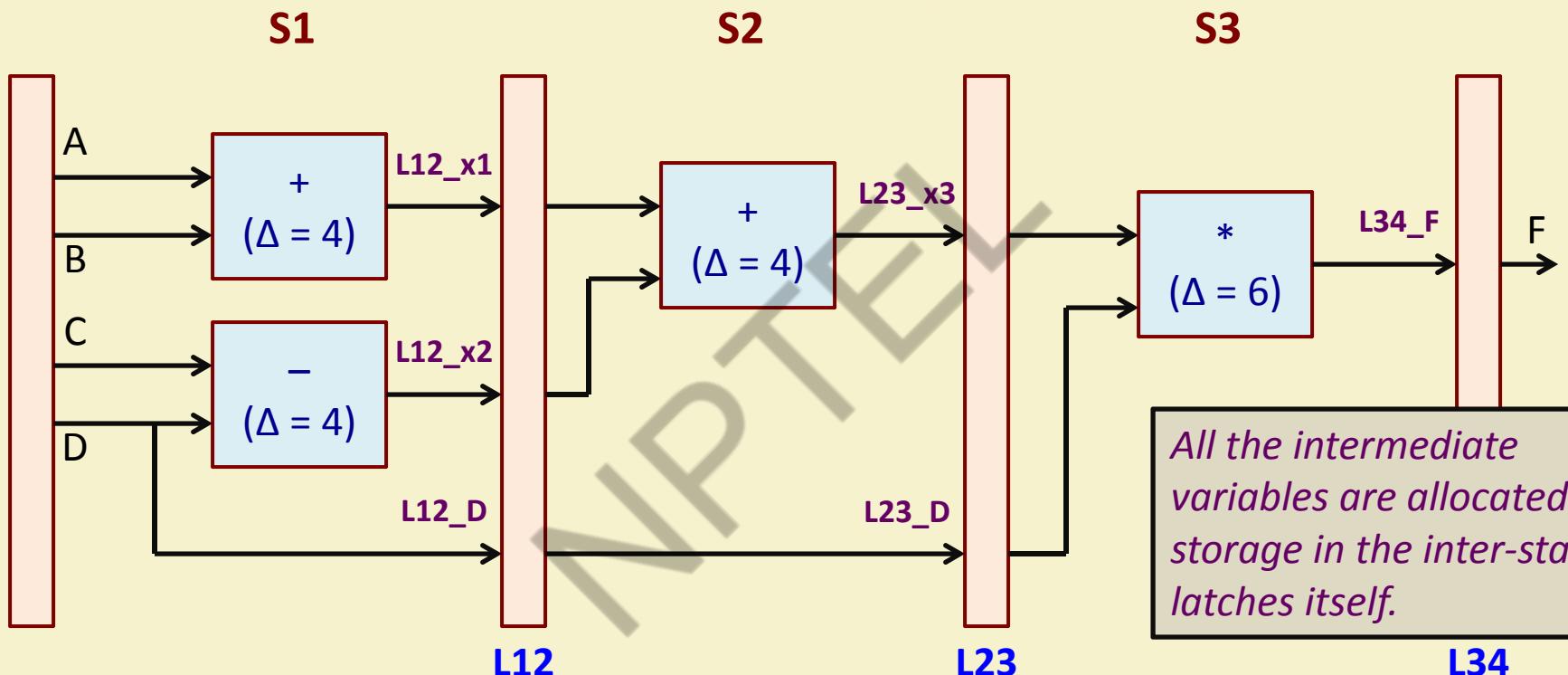


IIT KHARAGPUR



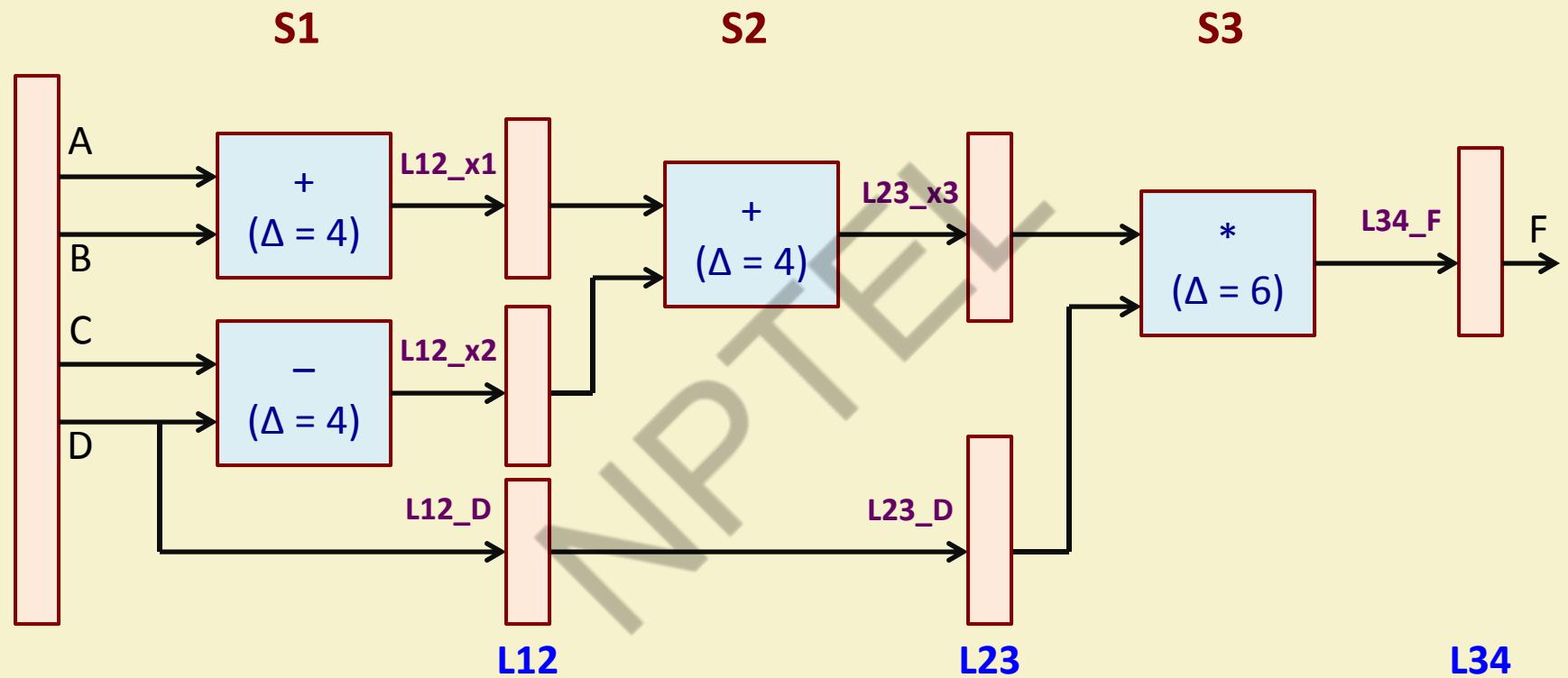
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



*All the intermediate variables are allocated storage in the inter-stage latches itself.*





IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Pipeline Modeling

```
module pipe_ex (F, A, B, C, D, clk);
parameter N = 10;
input [N-1:0] A, B, C, D;
input clk;
output [N-1:0] F;
reg [N-1:0] L12_x1, L12_x2, L12_D, L23_x3, L23_D, L34_F;

assign F = L34_F;

always @(posedge clk)
begin
    L12_x1 <= #4 A + B;
    L12_x2 <= #4 C - D;
    L12_D   <= D;                                // ** STAGE 1 **

```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
L23_x3 <= #4 L12_x1 + L12_x2;  
L23_D  <= L12_D;                      // ** STAGE 2 **  
  
L34_F  <= #6 L23_x3 * L23_D;    // ** STAGE 3 **  
end  
  
endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

module pipe_ex (F, A, B, C, D, clk);
parameter N = 10;

input [N-1:0] A, B, C, D;
input clk;
output [N-1:0] F;
reg [N-1:0] L12_x1, L12_x2, L12_D, L23_x3, L23_D, L34_F;

assign F = L34_F;

always @(posedge clk)
begin
    L12_x1 <= #4 A + B;
    L12_x2 <= #4 C - D;
    L12_D <= D;
end

```

Alternate way of coding:

- One stage per “always” block.
- Code is more readable.

```
always @(posedge clk)           // ** STAGE 2 **
begin
    L23_x3 <= #4 L12_x1 + L12_x2;
    L23_D   <= L12_D;
end

always @(posedge clk)           // ** STAGE 3 **
    L34_F   <= #6 L23_x3 * L23_D;

endmodule
```



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Pipeline Test Bench

```
module pipe1_test;  
  
    parameter N = 10;  
    wire [N-1:0] F;  
    reg [N-1:0] A, B, C, D;  
    reg clk;  
  
    pipe_ex MYPIPE (F, A, B, C, D, clk);  
  
    initial clk = 0;  
  
    always #10 clk = ~clk;
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

initial
begin
    #5 A = 10; B = 12; C = 6; D = 3; // F = 75 (4Bh)
    #20 A = 10; B = 10; C = 5; D = 3; // F = 66 (42h)
    #20 A = 20; B = 11; C = 1; D = 4; // F = 112 (70h)
    #20 A = 15; B = 10; C = 8; D = 2; // F = 62 (3Eh)
    #20 A = 8; B = 15; C = 5; D = 0; // F = 0 (00h)
    #20 A = 10; B = 20; C = 5; D = 3; // F = 66 (42h)
    #20 A = 10; B = 10; C = 30; D = 1; // F = 49 (31h)
    #20 A = 30; B = 1; C = 2; D = 4; // F = 116 (74h)
end

initial
begin
    $dumpfile ("pipe1.vcd");
    $dumpvars (0, pipe1_test);
    $monitor ("Time: %d, F = %d", $time, F);
    #300 $finish;
end
endmodule

```



```
Time: 0, F = x
Time: 56, F = 75
Time: 76, F = 66
Time: 96, F = 112
Time: 116, F = 62
Time: 136, F = 0
Time: 156, F = 96
Time: 179, F = 49
Time: 196, F = 116
```

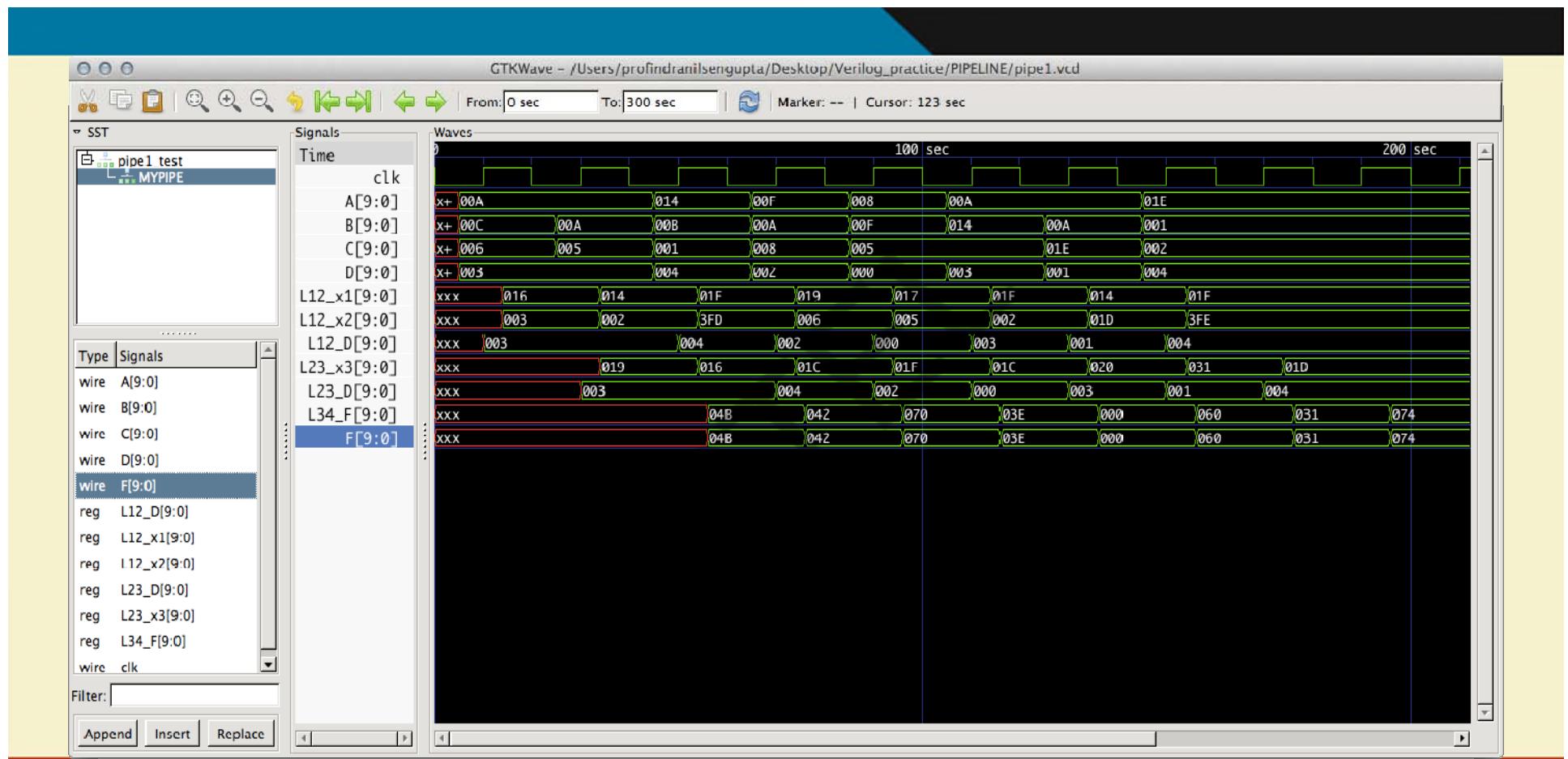


IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# A Warning

- In this example, we have used a single phase clock to load all the inter-stage registers in the pipeline.
- In a real pipeline, this may lead to race condition.
- Possible solution:
  - Use master-phase flip-flops in the registers.
  - Use a two-phase clock to clock the alternate stages.
- We shall illustrate the two-phase clocking approach in the next example.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 33



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 34: PIPELINE MODELING (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# A More Complex Example

- Consider a pipeline that carries out the following stage-wise operations:
  - *Inputs*: Three register addresses ( $rs1$ ,  $rs2$  and  $rd$ ), an ALU function ( $func$ ), and a memory address ( $addr$ ).
  - *Stage 1*: Read two 16-bit numbers from the registers specified by “ $rs1$ ” and “ $rs2$ ”, and store them in  $A$  and  $B$ .
  - *Stage 2*: Perform an ALU operation on  $A$  and  $B$  specified by “ $func$ ”, and store it in  $Z$ .
  - *Stage 3*: Write the value of  $Z$  in the register specified by “ $rd$ ”.
  - *Stage 4*: Also write the value of  $Z$  in memory location “ $addr$ ”.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# The Assumptions

- There is a register bank containing 16 16-bit registers.
  - 4-bits are required to specify a register address.
  - 2 register reads and 1 register write can be performed every clock cycle.
  - Register addresses are “*rs1*”, “*rs2*”, and “*rd*”.
- Assume that the memory is organized as *256 x 16*.
  - 8-bits are required to specify memory address.
  - Every memory location contains 16 bits of data, which can be read in a single clock cycle.
  - Memory address specified as “*addr*”.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

- The ALU function is selected by a 4-bit field “*func*”, as follows:

0000: ADD

0001: SUB

0010: MUL

0011: SELA

0100: SELB

0101: AND

0110: OR

0111: XOR

1000: NEGA

1001: NEGB

1010: SRA

1011: SLA

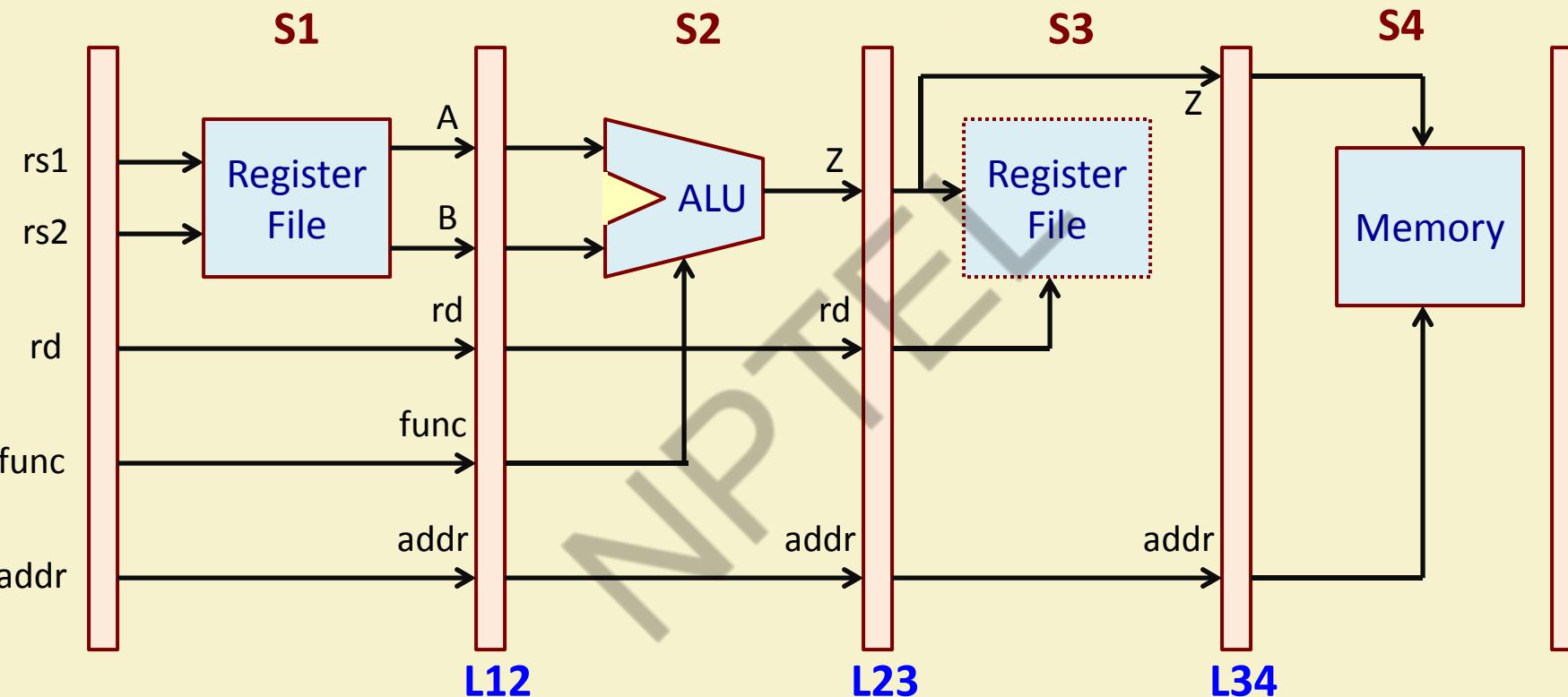


IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



IIT KHARAGPUR

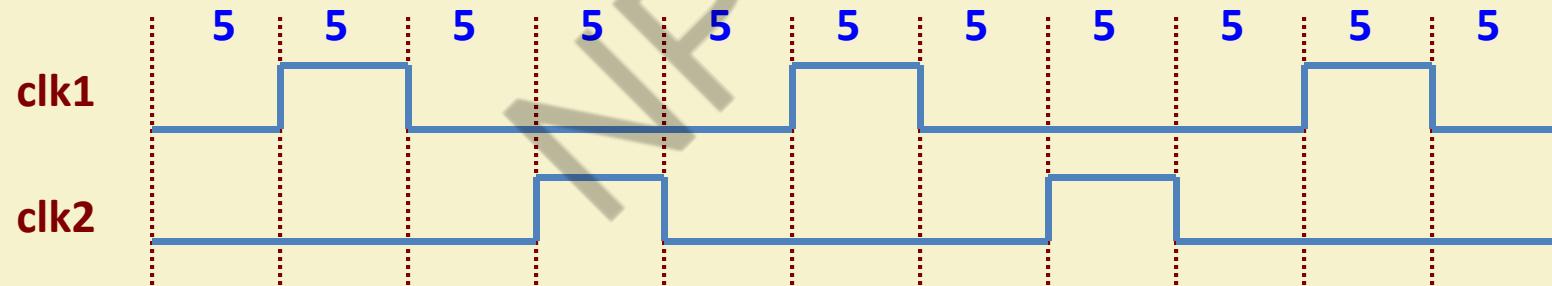


NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Clocking Issue in Pipeline

- It is important that the consecutive stages be applied suitable clocks for correct operation.
- Two options:
  - a) Use master/slave flip-flops in the latches to avoid race condition.
  - b) Use non-overlapping two-phase clock for the consecutive pipeline stages.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

module pipe_ex2 (Zout, rs1, rs2, rd, func, addr, clk1, clk2);

    input [3:0] rs1, rs2, rd, func;
    input [7:0] addr;
    input write, clk1, clk2;      // Two-phase clock
    output [15:0] Zout;

    reg [15:0] L12_A, L12_B, L23_Z, L34_Z;
    reg [3:0]  L12_rd, L12_func, L23_rd;
    reg [7:0]  L12_addr, L23_addr, L34_addr;

    reg [15:0] regbank [0:15];   // Register bank
    reg [15:0] mem [0:255];     // 256 x 16 memory

    assign Zout = L34_Z;

```

## Pipeline Modeling



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
always @(posedge clk1)
begin
    L12_A      <= #2 regbank[rs1];
    L12_B      <= #2 regbank[rs2];
    L12_rd     <= #2 rd;
    L12_func   <= #2 func;
    L12_addr   <= #2 addr;
end
// ** STAGE 1 **
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

always @(negedge clk2)
begin
    case (func)
        0: L23_Z <= #2 L12_A + L12_B;
        1: L23_Z <= #2 L12_A - L12_B;
        2: L23_Z <= #2 L12_A * L12_B;
        3: L23_Z <= #2 L12_A;
        4: L23_Z <= #2 L12_B;
        5: L23_Z <= #2 L12_A & L12_B;
        6: L23_Z <= #2 L12_A | L12_B;
        7: L23_Z <= #2 L12_A ^ L12_B;
        8: L23_Z <= #2 - L12_A;
        9: L23_Z <= #2 - L12_B;
        10: L23_Z <= #2 L12_A >> 1;
        11: L23_Z <= #2 L12_A << 1;
        default: L23_Z <= #2 16'hxxxx;
    endcase
    L23_rd <= #2 L12_rd;
    L23_addr <= #2 L12_addr; // ** STAGE 2 **
end

```



```
always @(posedge clk1)
begin
    regbank[L23_rd] <= #2 L23_Z;
    L34_Z        <= #2 L23_Z;
    L34_addr     <= #2 L23_addr;           // ** STAGE 3 **
end

always @(negedge clk2)
begin
    mem[L34_addr] <= #2 L34_Z;           // ** STAGE 4 **
end

endmodule
```



## Pipeline Test Bench

```
module pipe2_test;

    wire [15:0] Z;
    reg [3:0] rs1, rs2, rd, func;
    reg [7:0] addr;
    reg clk1, clk2;
    integer k;

    pipe_ex2 MYPIPE (Z, rs1, rs2, rd, func, addr, clk1, clk2);

    initial
        begin
            clk1 = 0; clk2 = 0;
            repeat (20)           // Generating two-phase clock
                begin
                    #5 clk1 = 1; #5 clk1 = 0;
                    #5 clk2 = 1; #5 clk2 = 0;
                end
        end

    initial
        for (k=0; k<16; k=k+1)
            MYPIPE.regbank[k] = k;      // Initialize registers
```

```

initial
begin
    #5    rs1 = 3;  rs2 = 5;  rd = 10; func = 0;  addr = 125; // ADD
    #20   rs1 = 3;  rs2 = 8;  rd = 12; func = 2;  addr = 126; // MUL
    #20   rs1 = 10; rs2 = 5;  rd = 14; func = 1;  addr = 128; // SUB
    #20   rs1 = 7;   rs2 = 3;  rd = 13; func = 11; addr = 127; // SLA
    #20   rs1 = 10; rs2 = 5;  rd = 15; func = 1;  addr = 129; // SUB
    #20   rs1 = 12; rs2 = 13; rd = 16; func = 0;  addr = 130; // ADD

    #60 for (k=125; k<131; k=k+1)
        $display ("Mem[%3d] = %3d", k, MYPIPE.mem[k]);
end

initial
begin
    $dumpfile ("pipe2.vcd");
    $dumpvars (0, pipe2_test);
    $monitor ("Time: %3d, F = %3d", $time, Z);
    #300 $finish;
end

endmodule

```



## Simulation Results

```
Time: 0, F = x
Time: 27, F = 8
Time: 47, F = 24
Time: 67, F = 3
Time: 87, F = 14
Time: 107, F = 3
Time: 127, F = 38
Mem[125] = 8
Mem[126] = 24
Mem[127] = 14
Mem[128] = 3
Mem[129] = 3
Mem[130] = 38
```

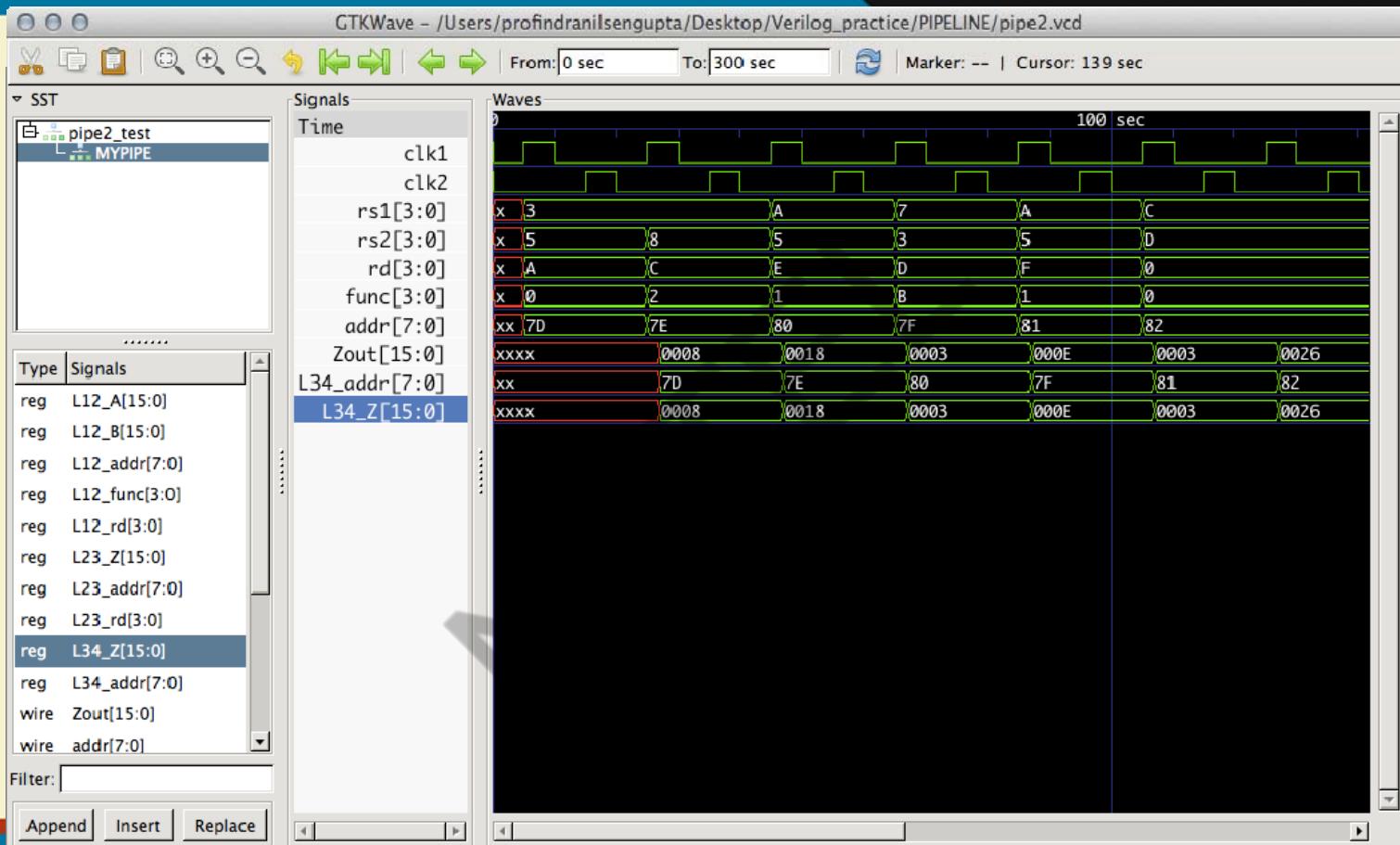


IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 34



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 35: SWITCH LEVEL MODELING (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Introduction

- A switch level circuit comprises of a netlist of MOS transistors.
- It is not very common for a designer to design modules using transistors.
  - May be required in very specific cases, for designing leaf-level modules in a hierarchical design.
- Verilog provides the ability to model digital circuits at the MOS transistor level.
  - Transistors function as switches; they either conduct (ON) or are open (OFF).
- The four logic levels 0, 1, X, Z and the associated signal drive strengths help in the modeling.
- Two types of switches supported: *ideal* or *resistive*.



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Various Switch Primitives in Verilog

- Ideal MOS switches
  - nmos, pmos, cmos
- Resistive MOS switches
  - rnmos, rpmos, rcmos
- Ideal Bidirectional switches
  - tran, tranif0, tranif1
- Resistive Bidirectional switches
  - rtran, rtranif0, rtranif1
- Power and Ground nets
  - supply1, supply0
- Pullup and Pulldown
  - pullup, pulldown



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

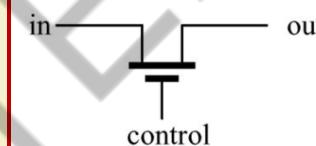
## (a) NMOS and PMOS Switches

- Declared with keywords “*nmos*” and “*pmos*”.
- Format for instantiation:

```
nmos (or pmos) [instance_name] (output, input, control);
```

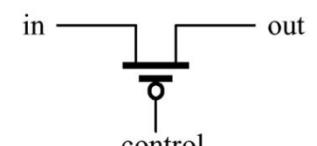
Here, “*instance\_name*” is optional.

*Also called pass transistors.*



nmos	control			
	0	1	x	z
.in	z	0	L	L
0	z	1	H	H
1	z	x	x	x
x	z	z	z	z
z	z	z	z	z

(a) nMOS switch



pmos	control			
	0	1	x	z
.in	0	z	L	L
0	1	z	H	H
1	x	z	x	x
x	z	z	z	z
z	z	z	z	z

(b) pMOS switch



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

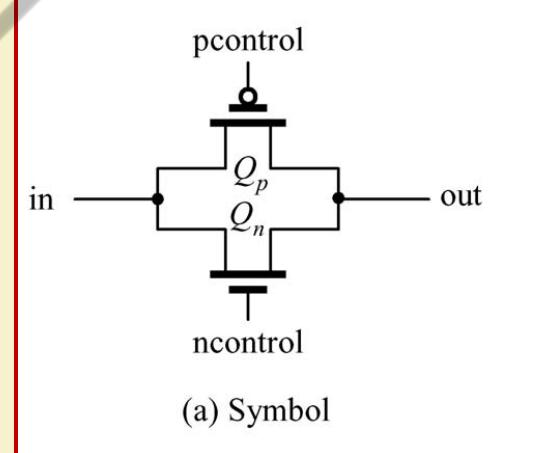
Hardware Modeling Using Verilog

## (b) CMOS Switch (Transmission Gate)

- Declared with keywords “*cmos*”.
- Format for instantiation:

```
cmos [instance_name] (output, input, ncontrol, pcontrol);
```

Here also, “*instance\_name*” is optional.



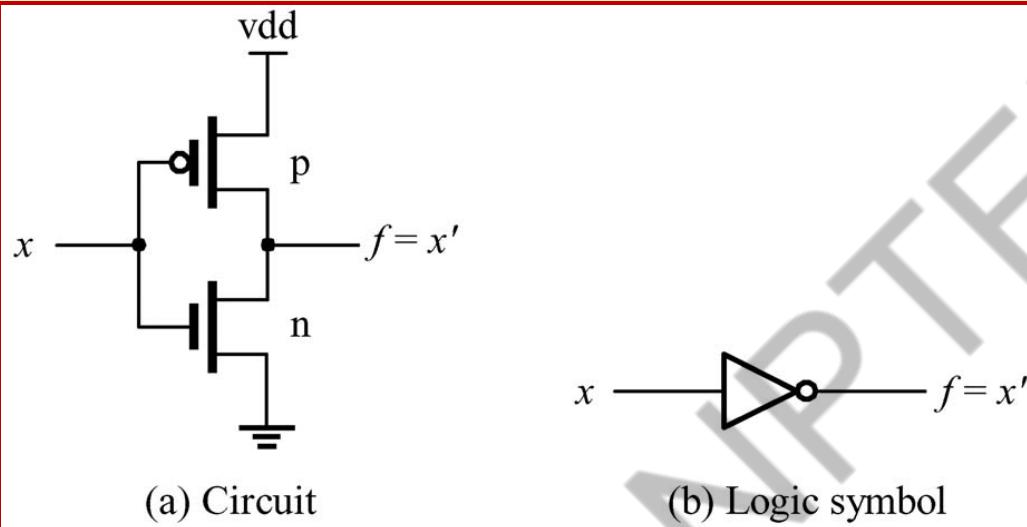
IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 1: CMOS Inverter



```
module cmosnot (x, f);  
    input x;  
    output f;  
    supply1 vdd;  
    supply0 gnd;  
    pmos p1 (f, vdd, x);  
    nmos n1 (f, gnd, x);  
endmodule
```



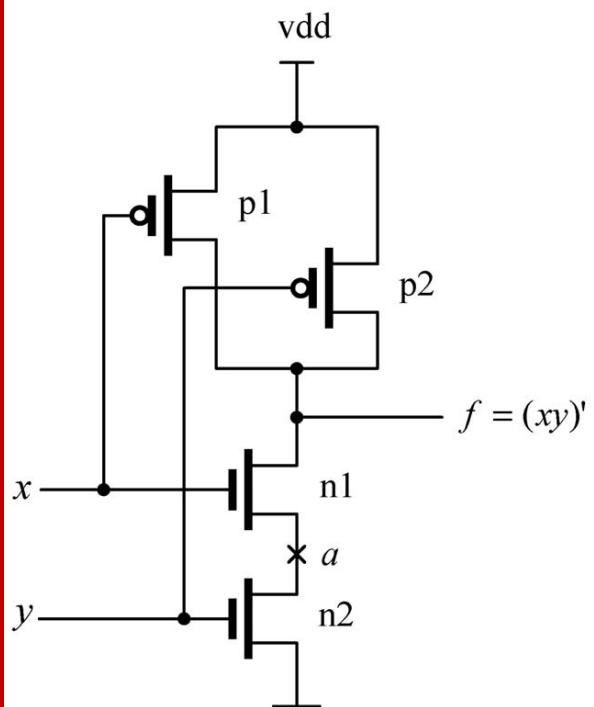
IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 2: CMOS NAND Gate



```
module cmosnand (x, y, f);  
  input x, y;  
  output f;  
  supply1 vdd;  
  supply0 gnd;  
  wire a;  
  pmos p1 (f, vdd, x);  
  pmos p2 (f, vdd, y);  
  nmos n1 (f, a, x);  
  nmos n2 (a, gnd, y);  
endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module cmosnand_test;

    reg in1, in2;
    wire out;
    integer k;

    cmosnand MYNAND2 (in1, in2, out);

    initial
        begin
            for (k=0; k<4; k=k+1)
                begin
                    #5 {in1,in2} = k;
                    $display ("In1: %b, In2: %b, Out: %b", in1, in2,
out);
                end
        end
    endmodule
```

In1: 0, In2: 0, Out: 1  
In1: 0, In2: 1, Out: 1  
In1: 1, In2: 0, Out: 1  
In1: 1, In2: 1, Out: 0



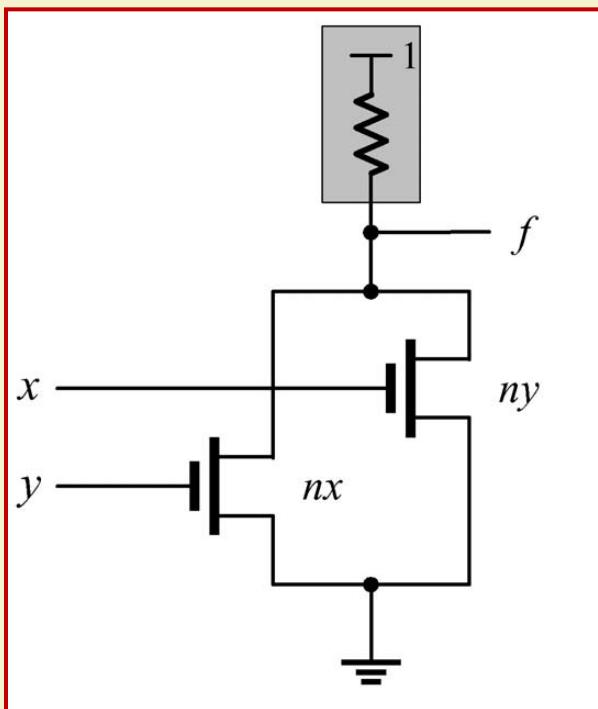
IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 3: Pseudo-NMOS NOR Gate



```
module pseudonor (x, y, f);  
    input x, y;  
    output f;  
    supply0 gnd;  
    nmos nx (f, gnd, x);  
    nmos ny (f, gnd, y);  
    pullup (f);  
endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module pseudonor_test;

    reg in1, in2;
    wire out;
    integer k;

    pseudonor MYNOR2 (in1, in2, out);

    initial
        begin
            for (k=0; k<4; k=k+1)
                begin
                    #5 {in1,in2} = k;
                    $display ("In1: %b, In2: %b, Out: %b", in1, in2,
out);
                end
        end
    endmodule
```

In1: 0, In2: 0, Out: 1  
In1: 0, In2: 1, Out: 0  
In1: 1, In2: 0, Out: 0  
In1: 1, In2: 1, Out: 0



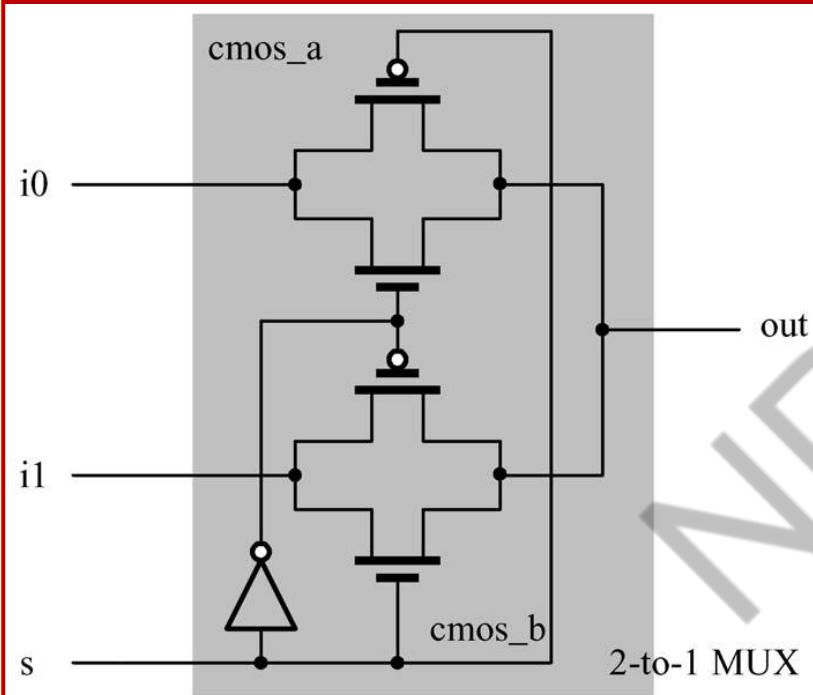
IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Example 4: CMOS 2x1 Multiplexer



```
module mux_2to1 (out, s, i0, i1);
    input s, i0, i1;
    output out;
    wire sbar;
    not (sbar, s);
    cmos cmos_a (out, i0, sbar, s);
    cmos cmos_b (out, i1, s, sbar);
endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```

module cmosmux_test;

    reg sel, in0, in1;
    wire out;
    integer k;

    cmosmux MUX21 (out, sel, in0, in1);

    initial
        begin
            for (k=0; k<8; k=k+1)
                begin
                    #5 {sel,in0,in1} = k;
                    $display ("Sel: %b, In0: %b, In1: %b, Out: %b",
                              sel, in1, in1,
                              out);
                end
        end
    endmodule

```

```

Sel: 0, In0: 0, In1: 0, Out: 0
Sel: 0, In0: 0, In1: 1, Out: 0
Sel: 0, In0: 1, In1: 0, Out: 1
Sel: 0, In0: 1, In1: 1, Out: 1
Sel: 1, In0: 0, In1: 0, Out: 0
Sel: 1, In0: 0, In1: 1, Out: 1
Sel: 1, In0: 1, In1: 0, Out: 0
Sel: 1, In0: 1, In1: 1, Out: 1

```



IIT KHARAGPUR

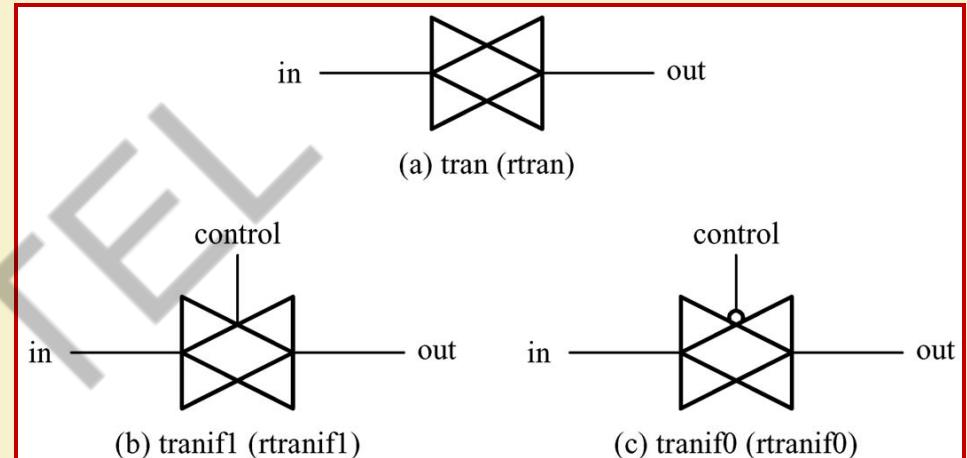


NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## (c) Bidirectional Switches

- “*nmos*”, “*pmos*”, and “*cmos*” gates conduct in one direction (drain to source).
- When it is required for devices to conduct in both direction, we use bidirectional switches.
- Three types of bidirectional switches: “*tran*”, “*tranif0*”, and “*tranif1*”.



### Syntax:

```
tran    [instance_name] (inout1, inout2);
tranif0 [instance_name] (inout1, inout2, cntl);
tranif1 [instance_name] (inout1, inout2, cntl);
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 35



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

# Lecture 36: SWITCH LEVEL MODELING (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Example 5: CMOS 4x1 Multiplexer using “tran” Switches

```
module mux_4to1 (out, s0, s1, i0, i1, i2, i3);  
    input s0, s1, i0, i1, i2, i3;  
    output out;  
    wire t0, t1, t2, t3;  
  
    tranif0 (i0, t0, s0);      tranif0 (t0, out, s1);  
    tranif1 (i1, t1, s0);      tranif0 (t1, out, s1);  
    tranif0 (i2, t2, s0);      tranif1 (t2, out, s1);  
    tranif1 (i3, t3, s0);      tranif1 (t3, out, s1);  
endmodule
```



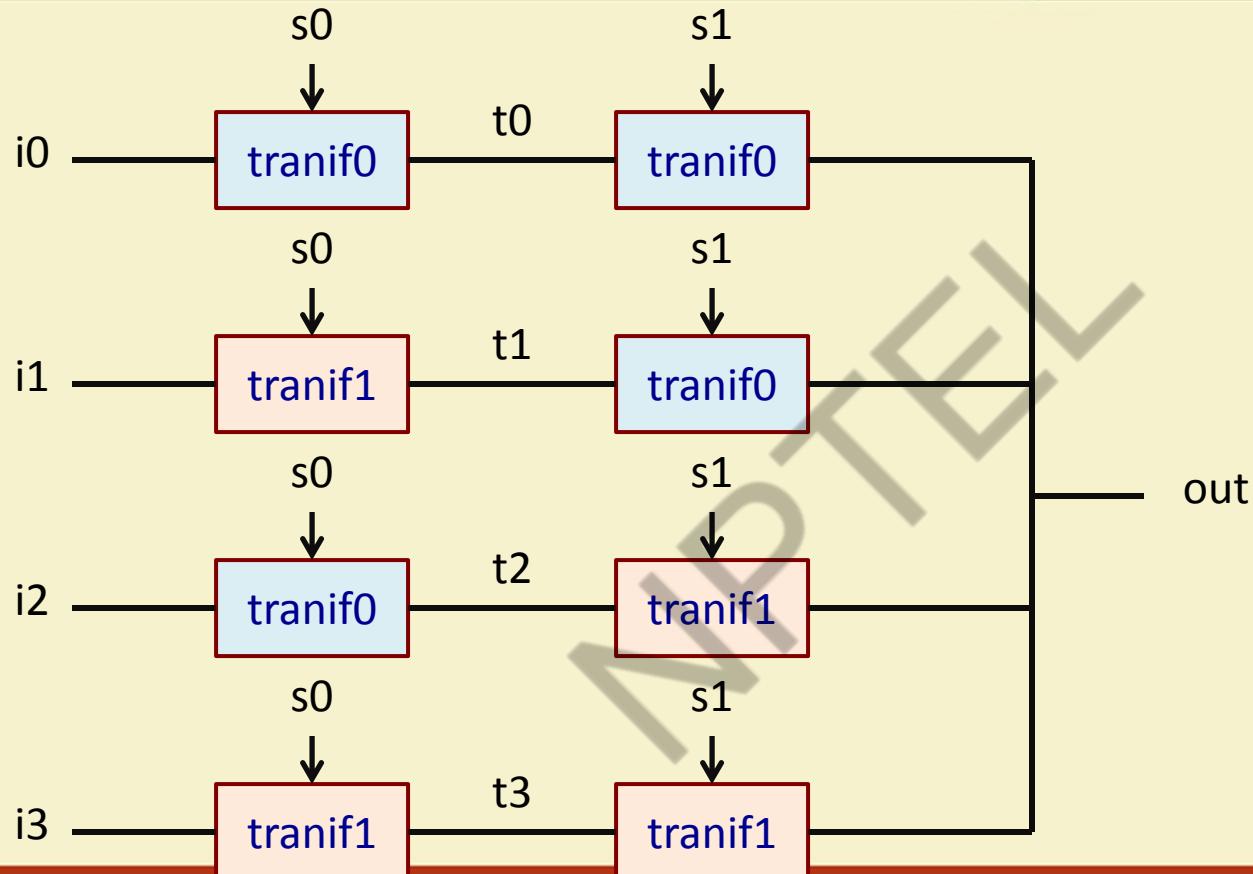
IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Schematic Diagram



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Test Bench

```
module mux41_test;  
    reg s0, s1, i0, i1, i2, i3;  
    wire out;  
    integer k;  
    mux_4to1 MYMUX41 (out, s0, s1, i0, i1, i2, i3);  
    initial  
        begin  
            for (k=0; k<64; k=k+1)  
                begin  
                    #5 {s0,s1,i0,i1,i2,i3} = k;  
                    $display ("Sel: %2b, In: %4b, Out: %b",  
                            {s0,s1}, {i0,i1,i2,i3}, out);  
                end  
        end  
    endmodule
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Simulation Results

```
Sel: 00, In: 0000, Out: x
Sel: 00, In: 0001, Out: 0
Sel: 00, In: 0010, Out: 0
Sel: 00, In: 0011, Out: 0
Sel: 00, In: 0100, Out: 0
Sel: 00, In: 0101, Out: 0
Sel: 00, In: 0110, Out: 0
Sel: 00, In: 0111, Out: 0
Sel: 00, In: 1000, Out: 0
Sel: 00, In: 1001, Out: 1
Sel: 00, In: 1010, Out: 1
Sel: 00, In: 1011, Out: 1
Sel: 00, In: 1100, Out: 1
Sel: 00, In: 1101, Out: 1
Sel: 00, In: 1110, Out: 1
Sel: 00, In: 1111, Out: 1
```

```
Sel: 01, In: 0000, Out: 1
Sel: 01, In: 0001, Out: 0
Sel: 01, In: 0010, Out: 0
Sel: 01, In: 0011, Out: 1
Sel: 01, In: 0100, Out: 1
Sel: 01, In: 0101, Out: 0
Sel: 01, In: 0110, Out: 0
Sel: 01, In: 0111, Out: 1
Sel: 01, In: 1000, Out: 1
Sel: 01, In: 1001, Out: 0
Sel: 01, In: 1010, Out: 0
Sel: 01, In: 1011, Out: 1
Sel: 01, In: 1100, Out: 1
Sel: 01, In: 1101, Out: 0
Sel: 01, In: 1110, Out: 0
Sel: 01, In: 1111, Out: 1
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Simulation Results

```
Sel: 10, In: 0000, Out: 1
Sel: 10, In: 0001, Out: 0
Sel: 10, In: 0010, Out: 0
Sel: 10, In: 0011, Out: 0
Sel: 10, In: 0100, Out: 0
Sel: 10, In: 0101, Out: 1
Sel: 10, In: 0110, Out: 1
Sel: 10, In: 0111, Out: 1
Sel: 10, In: 1000, Out: 1
Sel: 10, In: 1001, Out: 0
Sel: 10, In: 1010, Out: 0
Sel: 10, In: 1011, Out: 0
Sel: 10, In: 1100, Out: 0
Sel: 10, In: 1101, Out: 1
Sel: 10, In: 1110, Out: 1
Sel: 10, In: 1111, Out: 1
```

```
Sel: 11, In: 0000, Out: 1
Sel: 11, In: 0001, Out: 0
Sel: 11, In: 0010, Out: 1
Sel: 11, In: 0011, Out: 0
Sel: 11, In: 0100, Out: 1
Sel: 11, In: 0101, Out: 0
Sel: 11, In: 0110, Out: 1
Sel: 11, In: 0111, Out: 0
Sel: 11, In: 1000, Out: 1
Sel: 11, In: 1001, Out: 0
Sel: 11, In: 1010, Out: 1
Sel: 11, In: 1011, Out: 0
Sel: 11, In: 1100, Out: 1
Sel: 11, In: 1101, Out: 0
Sel: 11, In: 1110, Out: 1
Sel: 11, In: 1111, Out: 0
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

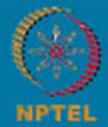
## Example 6: Full Adder using Transistor Level Modeling

```
module fulladder (sum, cout, a, b, cin);  
  
    input a, b, cin;  
    output sum, cout;  
    fa_sum SUM (sum, a, b, cin);  
    fa_carry CARRY (cout, a, b, cin);  
  
endmodule
```

```
module fa_carry (cout, a, b, cin);  
  
    input a, b, cin;  
    output cout;  
  
    wire t1, t2, t3, t4, t5;  
  
    cmosnand N1 (t1, a, b);  
    cmosnand N2 (t2, a, cin);  
    cmosnand N3 (t3, b, cin);  
    cmosnand N4 (t4, t1, t2);  
    cmosnand N5 (t5, t4, t4);  
    cmosnand N6 (cout, t5, t3);  
  
endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module myxor2 (out, a, b);
    input a, b;
    output out;
    wire t1, t2, t3, t4;

    cmosnand N1 (t1, a, a);
    cmosnand N2 (t2, b, b);
    cmosnand N3 (t3, a, t2);
    cmosnand N4 (t4, b, t1);
    cmosnand N5 (out, t3, t4);
endmodule
```

```
module fa_sum (sum, a, b, cin);
    input a, b, cin;
    output sum;
    wire t1, t2;

    myxor2 X1 (t1, a, b);
    myxor2 X2 (sum, t1, cin);
endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

```
module cmosnand (f, x, y);  
    input x, y;  
    output f;  
    supply1 vdd;  
    supply0 gnd;  
    pmos p1 (f, vdd, x);  
    pmos p2 (f, vdd, y);  
    nmos n1 (f, a, x);  
    nmos n2 (a, gnd, y);  
endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Test Bench

```
module fulladder_test;  
  
    reg a, b, cin;  
    wire sum, cout;  
    integer k;  
  
    fulladder FA (sum, cout, a, b, cin);  
  
    initial  
        begin  
            for (k=0; k<8; k=k+1)  
                begin  
                    #5 {a, b, cin} = k;  
                    $display ("Inputs: %3b Sum: %b, Carry: %b",
                               {a,b,cin}, sum, cout);  
                end  
        end  
    endmodule
```



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Simulation Results

```
Inputs: 000 Sum: 0, Carry: 0
Inputs: 001 Sum: 1, Carry: 0
Inputs: 010 Sum: 1, Carry: 0
Inputs: 011 Sum: 0, Carry: 1
Inputs: 100 Sum: 1, Carry: 0
Inputs: 101 Sum: 0, Carry: 1
Inputs: 110 Sum: 0, Carry: 1
Inputs: 111 Sum: 1, Carry: 1
```



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# Data Values and Signal Strengths

- Verilog supports 4 value levels and 8 strength levels to model the functionality of real hardware.
  - Strength levels are typically used to resolve conflicts between signal drivers of different strengths in real circuits.

Value Level	Represents
0	Logic 0 state
1	Logic 1 state
x	Unknown logic state
z	High impedance state

## Initialization:

- All unconnected nets are set to “z”.
- All register variables set to “x”.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

Strength	Type
supply	Driving
strong	Driving
pull	Driving
large	Storage
weak	Driving
medium	Storage
small	Storage
highz	High impedance

↑ Strength increases

- If two signals of unequal strengths get driven on a wire, the stronger signal will prevail.
- These are particularly useful for MOS level circuits, e.g. dynamic MOS.
- When a signal passes through a resistive switch, for example, its strength reduces.
- There is a convention followed for signal strength computation in MOS level circuits.
  - Details not discussed here.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

## Point to Note

- Most of the synthesis tools do not support switch level modeling.
  - Because it uses technology mapping from a given library.
  - Common gates and simple functional blocks are present in the library.
  - Thus it is not required to specify circuits at the transistor level.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

# END OF LECTURE 36



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

15



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 37: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Introduction

- We shall first look at the instruction set architecture of a popular *Reduced Instruction Set Architecture (RISC)* processor, viz. MIPS32.
  - It is a 32-bit processor, i.e. can operate on 32 bits of data at a time.
- We shall look at the instruction types, and how instructions are encoded.
- Then we can understand the process of instruction execution, and the steps involved.
- We shall discuss the pipeline implementation of the processor.
  - Only for a small subset of the instructions (and some simplifying assumptions).
- Finally, we shall present the Verilog design of the pipeline processor.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

## A Quick Look at MIPS32

- MIPS32 registers:
  - a) 32, 32-bit general purpose registers (GPRs), *R0* to *R31*.
    - Register *R0* contains a constant 0; cannot be written.
  - b) A special-purpose 32-bit program counter (*PC*).
    - Points to the next instruction in memory to be fetched and executed.
- No flag registers (zero, carry, sign, etc.).
- Very few addressing modes (register, immediate, register indexed, etc.)
  - Only load and store instructions can access memory.
- We assume memory word size is 32 bits (*word addressable*).



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

## The MIPS32 Instruction Subset Being Considered

- Load and Store Instructions
 

```
LW R2,124(R8) // R2 = Mem[R8+124]
SW R5,-10(R25) // Mem[R25-10] = R5
```
- Arithmetic and Logic Instructions (only register operands)

```
ADD R1,R2,R3 // R1 = R2 + R3
ADD R1,R2,R0 // R1 = R2 + 0
SUB R12,R10,R8 // R12 = R10 - R8
AND R20,R1,R5 // R20 = R1 & R5
OR R11,R5,R6 // R11 = R5 | R6
MUL R5,R6,R7 // R5 = R6 * R7
SLT R5,R11,R12 // If R11 < R12, R5=1; else R5=0
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

- Arithmetic and Logic Instructions (immediate operand)

```
ADDI R1,R2,25      // R1 = R2 + 25
SUBI R5,R1,150    // R5 = R1 - 150
SLTI R2,R10,10    // If R10<10, R2=1; else R2=0
```

- Branch Instructions

```
BEQZ R1,Loop        // Branch to Loop if R1=0
BNEQZ R5,Label     // Branch to Label if R5!=0
```

- Jump Instruction

```
J Loop              // Branch to Loop unconditionally
```

- Miscellaneous Instruction

```
HLT                  // Halt execution
```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

5

## MIPS Instruction Encoding

- All MIPS32 instructions can be classified into three groups in terms of instruction encoding.
  - **R-type** (Register), **I-type** (Immediate), and **J-type** (Jump).
  - In an instruction encoding, the 32 bits of the instruction are divided into several fields of fixed widths.
  - All instructions may not use all the fields.
- Since the relative positions of some of the fields are same across instructions, instruction decoding becomes very simple.



IIT KHARAGPUR

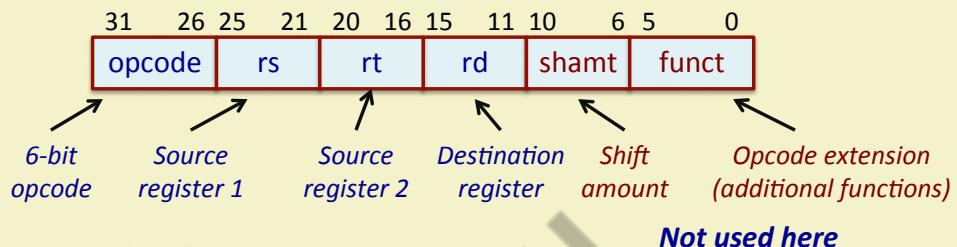
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

6

### (a) R-type Instruction Encoding

- Here an instruction can use up to three register operands.
  - Two source and one destination.
- In addition, for shift instructions, the number of bits to shift can also be specified (*we are not considering such instructions here*).



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

- R-type instructions considered with opcode:

Instruction	opcode
ADD	000000
SUB	000001
AND	000010
OR	000011
SLT	000100
MUL	000101
HLT	111111

```

SUB    R5,R12,R25
000001 01100 11001 00101 00000 000000
      SUB    R12    R25    R5
= 05992800 (in hex)
  
```



IIT KHARAGPUR

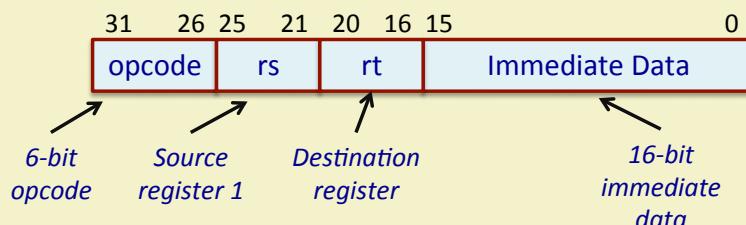
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

### (b) I-type Instruction Encoding

- Contains a 16-bit immediate data field.
- Supports one source and one destination register.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

9

- I-type instructions considered with opcode:

Instruction	opcode
LW	001000
SW	001001
ADDI	001010
SUBI	001011
SLTI	001100
BNEQZ	001101
BEQZ	001110

LW R20,84(R9)

```
001000 01001 10100 0000000001010100
      LW     R9     R20      offset
= 21340054 (in hex)
```

BEQZ R25,Label

```
001110 11001 00000 YYYYYYYYYYYYYYYY
      BEQZ   R25  Unused    offset
= 3b20YYYY (in hex)
```



IIT KHARAGPUR

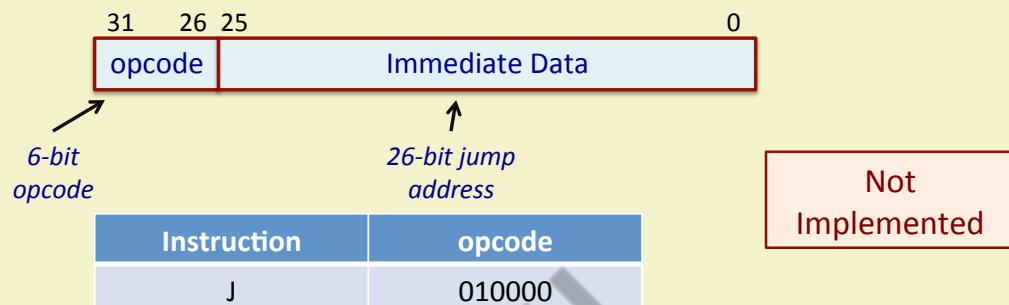
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

10

### (c) J-type Instruction Encoding

- Contains a 26-bit jump address field.
- Extended to 28 bits by padding two 0's on the right.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

11

### A Quick View

R-type	31	26	25	21	20	16	15	11	10	6	5	0
	opcode	rs	rt	rd	shamt	funct						
I-type	31	26	25	21	20	16	15					0
	opcode	rs	rt					Immediate Data				
J-type	31	26	25									0
	opcode							Immediate Data				

- Some instructions require two register operands *rs* & *rt* as input, while some require only *rs*.
- Gets known only after instruction is decoded.
- While decoding is going on, we can prefetch the registers in parallel.
  - May or may not be required later.

- Similarly, the 16-bit and 26-bit immediate data are retrieved and sign-extended to 32-bits in case they are required later.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

12

## Addressing Modes in MIPS32

- Register addressing                    ***ADD R1,R2,R3***
- Immediate addressing                ***ADDI R1,R2, 200***
- Base addressing                      ***LW R5, 150(R7)***
  - Content of a register is added to a “base” value to get the operand address.
- PC relative addressing                ***BEQZ R3, Label***
  - 16-bit offset is added to PC to get the target address.
- Pseudo-direct addressing             ***J Label***
  - 26-bit offset is added to PC to get the target address.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

13

**END OF LECTURE 37**



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 38: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### MIPS32 Instruction Cycle

- We divide the instruction execution cycle into five steps:
  - a) IF : Instruction Fetch
  - b) ID : Instruction Decode / Register Fetch
  - c) EX : Execution / Effective Address Calculation
  - d) MEM : Memory Access / Branch Completion
  - e) WB : Register Write-back
- We now show the generic micro-instructions carried out in the various steps.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

## (a) IF : Instruction Fetch

- Here the instruction pointed to by  $PC$  is fetched from memory, and also the next value of  $PC$  is computed.
  - Every MIPS32 instruction is of 32 bits.
  - Every memory word is of 32 bits and has a unique address.
  - For a branch instruction, new value of the  $PC$  may be the target address. So  $PC$  is not updated in this stage; new value is stored in a register  $NPC$ .

**IF:**       $IR \leftarrow \text{Mem}[PC];$   
                  $NPC \leftarrow PC + 1;$

For byte addressable memory, PC has to be incremented by 4.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

## (b) ID : Instruction Decode

- The instruction already fetched in  $IR$  is decoded.
  - $Opcode$  is 6-bits (bits 31:26).
  - First source operand  $rs$  (bits 25:21), second source operand  $rt$  (bits 20:16).
  - 16-bit immediate data (bits 15:0).
  - 26-bit immediate data (bits 25:0).
- Decoding is done in parallel with reading the register operands  $rs$  and  $rt$ .
  - Possible because these fields are in a fixed location in the instruction format.
- In a similar way, the immediate data are sign-extended.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18

**ID:**

```

A ← Reg [rs];
B ← Reg [rt];
Imm ← (IR15)16 ## IR15..0 // sign extend 16-bit immediate field
Imm1 ← (IR25)6 ## IR25..0 // sign extend 26-bit immediate field

```

*A, B, Imm, Imm1 are temporary registers.*



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

19

## (c) EX: Execution / Effective Address Computation

- In this step, the ALU is used to perform some calculation.
  - The exact operation depends on the instruction that is already decoded.
  - The ALU operates on operands that have been already made ready in the previous cycle.
    - A, B, Imm, etc.
- We show the micro-instructions corresponding to the type of instruction.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

20

<u>Memory Reference:</u>	ALUOut $\leftarrow$ A + Imm;	Example: LW R3, 100(R8)
<u>Register-Register ALU Instruction:</u>	ALUOut $\leftarrow$ A func B;	Example: SUB R2, R5, R12
<u>Register-Immediate ALU Instruction:</u>	ALUOut $\leftarrow$ A func Imm;	Example: SUBI R2, R5, 524
<u>Branch:</u>	ALUOut $\leftarrow$ NPC + Imm; cond $\leftarrow$ (A op 0);	Example: BEQZ R2, Label [op is ==]



## (d) MEM: Memory Access / Branch Completion

- The only instructions that make use of this step are loads, stores, and branches.
  - The load and store instructions access the memory.
  - The branch instruction updates *PC* depending upon the outcome of the branch condition.



**Load instruction:**

```
PC ← NPC;
LMD ← Mem [ALUOut];
```

**Store instruction:**

```
PC ← NPC;
Mem [ALUOut] ← B;
```

**Other instructions:**

```
PC ← NPC;
```

**Branch instruction:**

```
if (cond) PC ← ALUOut;
else PC ← NPC;
```



IIT KHARAGPUR

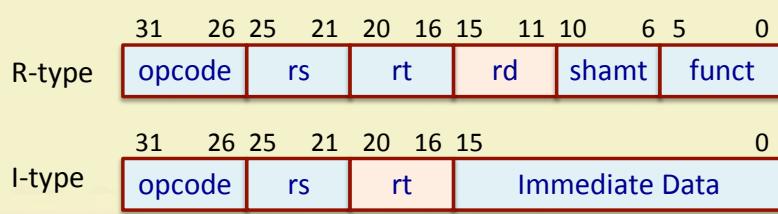
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

23

**(e) WB: Register Write Back**

- In this step, the result is written back into the register file.
  - Result may come from the ALU.
  - Result may come from the memory system (viz. a LOAD instruction).
- The position of the destination register in the instruction word depends on the instruction → *already known after decoding has been done.*



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

24

Register-Register ALU Instruction:

Reg [rd] ← ALUOut;

Register-Immediate ALU Instruction:

Reg [rt] ← ALUOut;

Load Instruction:

Reg [rt] ← LMD;



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

## SOME EXAMPLE INSTRUCTION EXECUTION



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

26

**ADD R2, R5, R10**

<b>IF</b>	IR $\leftarrow$ Mem [PC]; NPC $\leftarrow$ PC + 1;
<b>ID</b>	A $\leftarrow$ Reg [rs]; B $\leftarrow$ Reg [rt];
<b>EX</b>	ALUOut $\leftarrow$ A + B;
<b>MEM</b>	PC $\leftarrow$ NPC;
<b>WB</b>	Reg [rd] $\leftarrow$ ALUOut;

**ADDI R2, R5, 150**

<b>IF</b>	IR $\leftarrow$ Mem [PC]; NPC $\leftarrow$ PC + 1;
<b>ID</b>	A $\leftarrow$ Reg [rs]; Imm $\leftarrow$ (IR <sub>15</sub> ) <sup>16</sup> ## IR <sub>15..0</sub>
<b>EX</b>	ALUOut $\leftarrow$ A + Imm;
<b>MEM</b>	PC $\leftarrow$ NPC;
<b>WB</b>	Reg [rt] $\leftarrow$ ALUOut;



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27

**LW R2, 200 (R6)**

<b>IF</b>	IR $\leftarrow$ Mem [PC]; NPC $\leftarrow$ PC + 1;
<b>ID</b>	A $\leftarrow$ Reg [rs]; Imm $\leftarrow$ (IR <sub>15</sub> ) <sup>16</sup> ## IR <sub>15..0</sub>
<b>EX</b>	ALUOut $\leftarrow$ A + Imm;
<b>MEM</b>	PC $\leftarrow$ NPC; LMD $\leftarrow$ Mem [ALUOut];
<b>WB</b>	Reg [rt] $\leftarrow$ LMD;

**SW R3, 25 (R10)**

<b>IF</b>	IR $\leftarrow$ Mem [PC]; NPC $\leftarrow$ PC + 1;
<b>ID</b>	A $\leftarrow$ Reg [rs]; B $\leftarrow$ Reg [rt]; Imm $\leftarrow$ (IR <sub>15</sub> ) <sup>16</sup> ## IR <sub>15..0</sub>
<b>EX</b>	ALUOut $\leftarrow$ A + Imm;
<b>MEM</b>	PC $\leftarrow$ NPC; Mem [ALUOut] $\leftarrow$ B;
<b>WB</b>	-



IIT KHARAGPUR

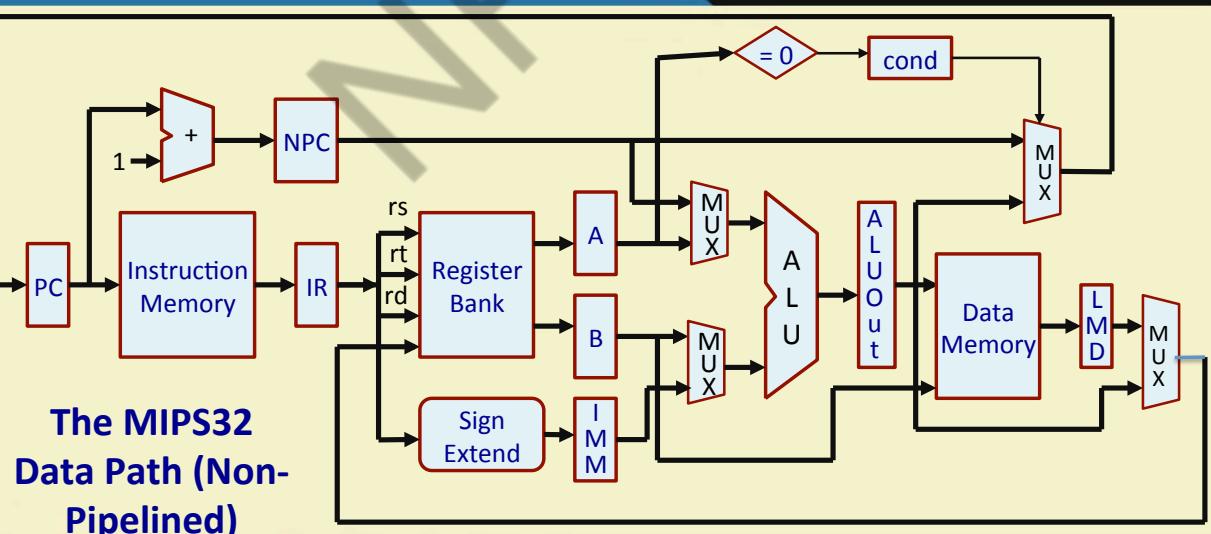
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

## BEQZ R3, Label

IF	$IR \leftarrow Mem[PC];$ $NPC \leftarrow PC + 1;$
ID	$A \leftarrow Reg[rs];$ $Imm \leftarrow (IR_{15})^{16} \# IR_{15..0}$
EX	$ALUOut \leftarrow NPC + Imm;$ $cond \leftarrow (A == 0);$
MEM	$PC \leftarrow NPC;$ if ( $cond$ ) $PC \leftarrow ALUOut;$
WB	-



The MIPS32 Data Path (Non-Pipelined)

## END OF LECTURE 38



IIT KHARAGPUR



NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

31



IIT KHARAGPUR



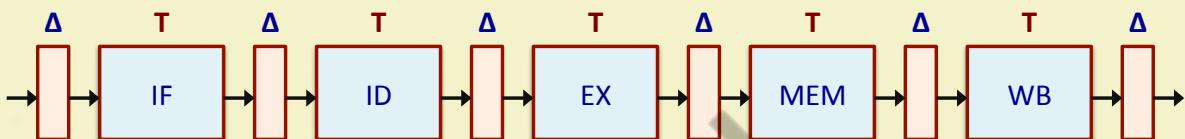
NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

## Lecture 39: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 3)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Introduction

- Basic requirements for pipelining the MIPS32 data path:
    - We should be able to start a new instruction every clock cycle.
    - Each of the five steps mentioned before (IF, ID, EX, MEM and WB) becomes a pipeline stage.
    - Each stage must finish its execution within one clock cycle.



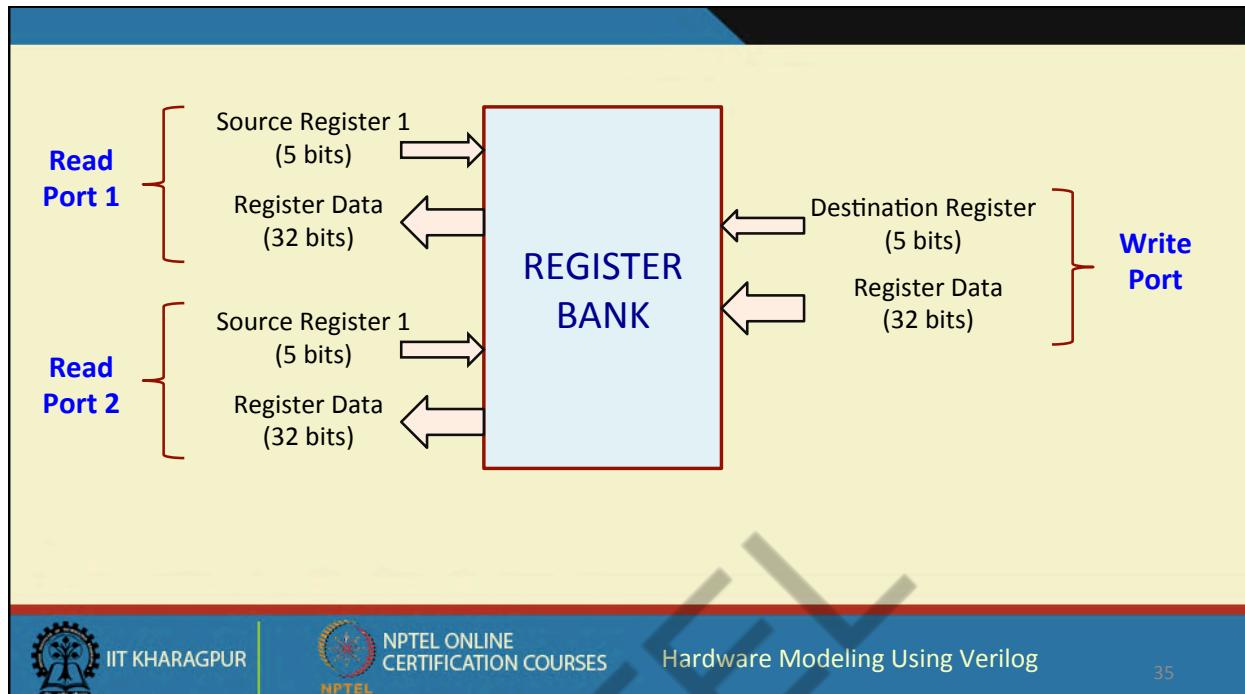
Instruction	1	2	3	4	5	6	7	8
	$i$	IF	ID	EX	MEM	WB		
$i + 1$		IF	ID	EX	MEM	WB		
$i + 2$			IF	ID	EX	MEM	WB	
$i + 3$				IF	ID	EX	MEM	WB

**Instr-i  
finishes**

*Instr-(i+  
finishe*

$\downarrow$   
*Instr-(i+2)  
finishes*

1) *Instr-(i+3)*  
finishes



## Micro-operations for Pipelined MIPS32

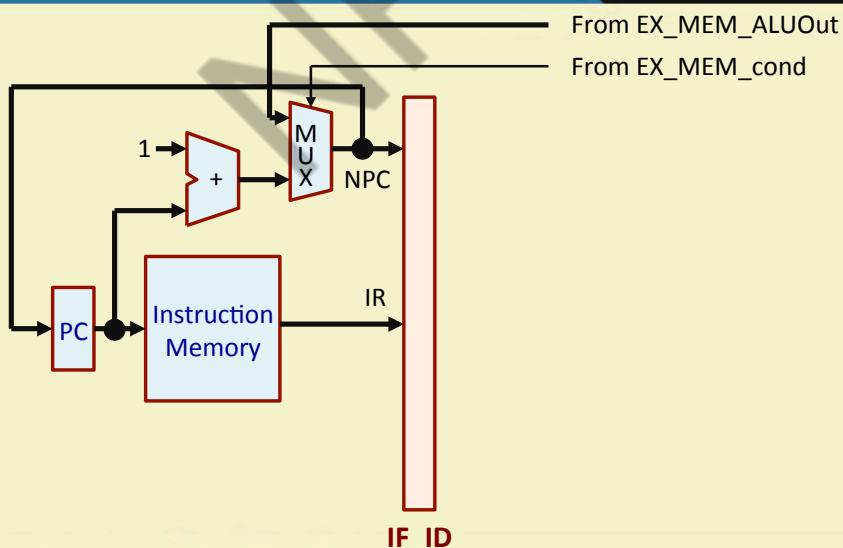
- Convention used:
  - Most of the temporary registers required in the data path are included as part of the inter-stage latches.
  - IF\_ID:** denotes the latch stage between the IF and ID stages.
  - ID\_EX:** denotes the latch stage between the ID and EX stages.
  - EX\_MEM:** denotes the latch stage between the EX and MEM stages.
  - MEM\_WB:** denotes the latch stage between the MEM and WB stages.
- Example:
  - ID\_EX\_A** means register **A** that is implemented as part of the **ID\_EX** latch stage.



## (a) Micro-operations for Pipeline Stage IF

```

IF_ID_IR      ← Mem [PC];
IF_ID_NPC,PC ← ( if ((EX_MEM_IR[opcode] == branch) & EX_MEM_cond)
                  { EX_MEM_ALUOut}
                  else {PC + 1} );
    
```

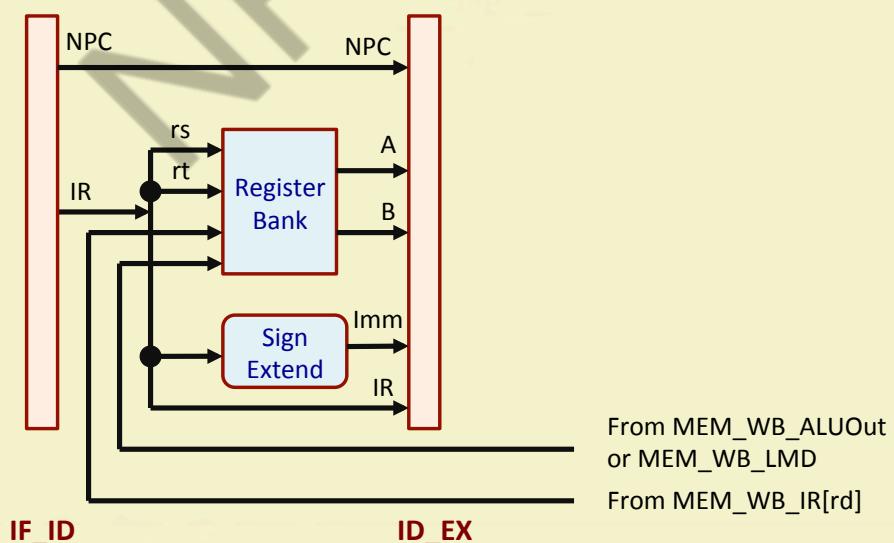


## (b) Micro-operations for Pipeline Stage ID

```

ID_EX_A   ← Reg [IF_ID_IR [rs]];
ID_EX_B   ← Reg [IF_ID_IR [rt]];
ID_EX_NPC ← IF_ID_NPC;
ID_EX_IR   ← IF_ID_IR;
ID_EX_Imm  ← sign-extend (IF_ID_IR15..0);

```



### (c) Micro-operations for Pipeline Stage EX

EX\_MEM\_IR  $\leftarrow$  ID\_EX\_IR;  
 EX\_MEM\_ALUOut  $\leftarrow$  ID\_EX\_A func ID\_EX\_B;

#### R-R ALU

EX\_MEM\_IR  $\leftarrow$  ID\_EX\_IR;  
 EX\_MEM\_ALUOut  $\leftarrow$  ID\_EX\_A func ID\_EX\_Imm;

#### R-M ALU

EX\_MEM\_IR  $\leftarrow$  ID\_EX\_IR;  
 EX\_MEM\_ALUOut  $\leftarrow$  ID\_EX\_A + ID\_EX\_Imm;  
 EX\_MEM\_B  $\leftarrow$  ID\_EX\_B;

EX\_MEM\_ALUOut  $\leftarrow$  ID\_EX\_NPC +  
 ID\_EX\_Imm;  
 EX\_MEM\_cond  $\leftarrow$  (ID\_EX\_A == 0);

#### BRANCH

#### LOAD / STORE

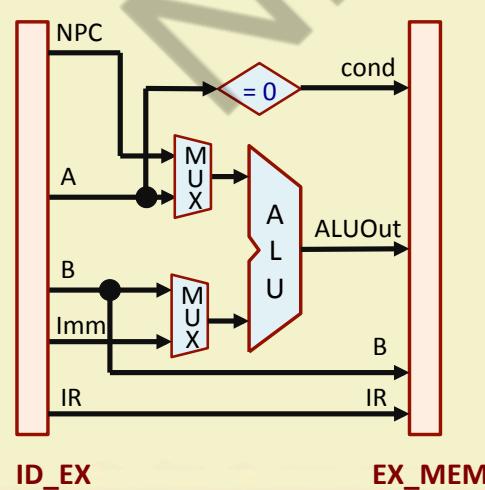


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

### (d) Micro-operations for Pipeline Stage MEM

MEM\_WB\_IR  $\leftarrow$  EX\_MEM\_IR;  
 MEM\_WB\_ALUOut  $\leftarrow$  EX\_MEM\_ALUOut;

ALU

MEM\_WB\_IR  $\leftarrow$  EX\_MEM\_IR;  
 MEM\_WB\_LMD  $\leftarrow$  Mem [EX\_MEM\_ALUOut];

LOAD

MEM\_WB\_IR  $\leftarrow$  EX\_MEM\_IR;  
 Mem [EX\_MEM\_ALUOut]  $\leftarrow$  EX\_MEM\_B;

STORE

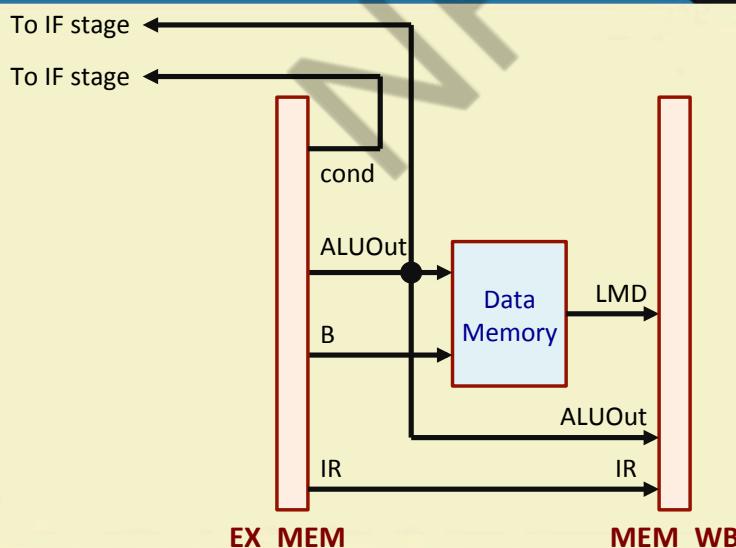


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

43



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

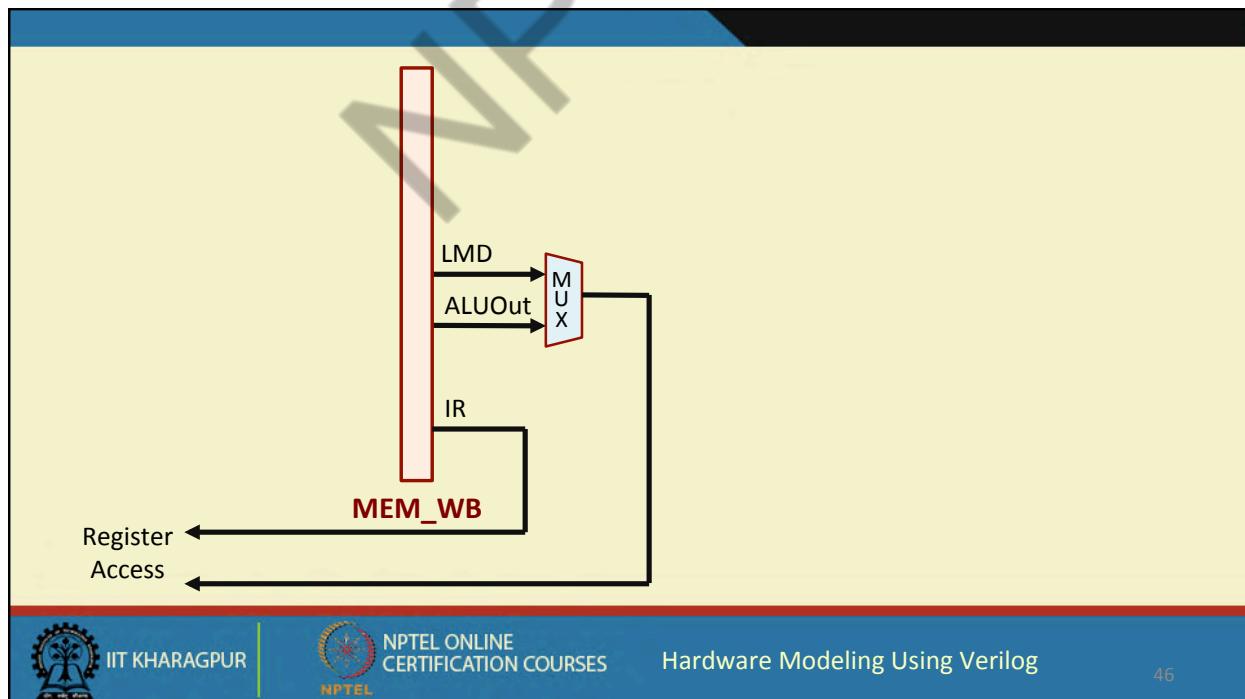
44

## (e) Micro-operations for Pipeline Stage WB

Reg [MEM\_WB\_IR [rd]]  $\leftarrow$  MEM\_WB\_ALUOut; **R-R ALU**

Reg [MEM\_WB\_IR [rt]]  $\leftarrow$  MEM\_WB\_ALUOut; **R-M ALU**

Reg [MEM\_WB\_IR [rt]]  $\leftarrow$  MEM\_WB\_LMD; **LOAD**



## PUTTING IT ALL TOGETHER :: MIPS32 PIPELINE

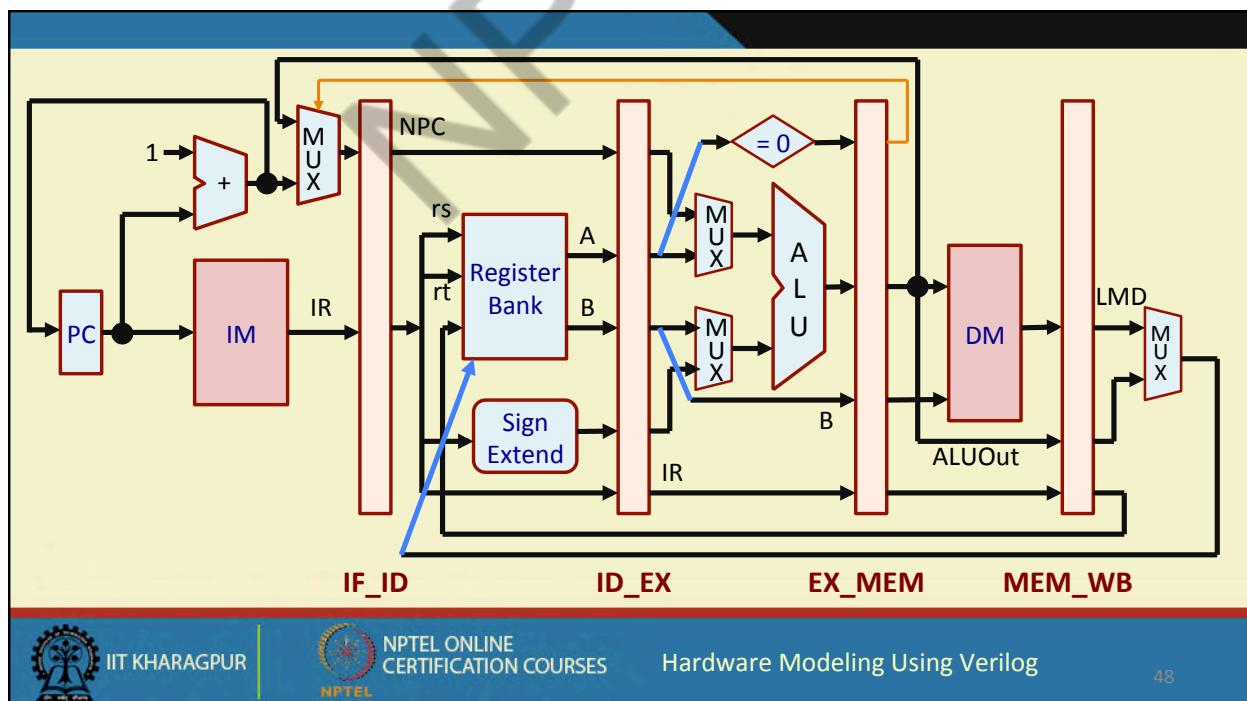


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

47



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

48

## END OF LECTURE 39



## Lecture 40: VERILOG MODELING OF THE PROCESSOR (PART 1)

PROF. INDRANIL SENGUPTA  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Verilog Implementation of MIPS32 Pipeline

- Here we show the behavioral Verilog code to implement the pipeline.
- Two special 1-bit variables are used:
  - **HALTED** :: Set after a HLT instruction executes and reaches the WB stage.
  - **TAKEN\_BRANCH** :: Set after the decision to take a branch is known. Required to disable the instructions that have already entered the pipeline from making any state changes.



```
module pipe_MIPS32 (clk1, clk2);
    input clk1, clk2;          // Two-phase clock

    reg [31:0] PC, IF_ID_IR, IF_ID_NPC;
    reg [31:0] ID_EX_IR, ID_EX_NPC, ID_EX_A, ID_EX_B, ID_EX_Imm;
    reg [2:0]  ID_EX_type, EX_MEM_type, MEM_WB_type;
    reg [31:0] EX_MEM_IR, EX_MEM_ALUOut, EX_MEM_B;
    reg       EX_MEM_cond;
    reg [31:0] MEM_WB_IR, MEM_WB_ALUOut, MEM_WB_LMD;

    reg [31:0] Reg [0:31];    // Register bank (32 x 32)
    reg [31:0] Mem [0:1023]; // 1024 x 32 memory

    parameter ADD=6'b000000, SUB=6'b000001, AND=6'b000010, OR=6'b000011,
              SLT=6'b000100, MUL=6'b000101, HLT=6'b111111, LW=6'6001000,
              SW=6'b001001, ADDI=6'b001010, SUBI=6'b001011, SLTI=6'b001100,
              BNEQZ=6'b001101, BEQZ=6'b001110;
```



```

parameter RR_ALU=3'b000, RM_ALU=3'b001, LOAD=3'b010, STORE=3'b011,
BRANCH=3'b100, HALT=3'b101;

reg HALTED;
    // Set after HLT instruction is completed (in WB stage)

reg TAKEN_BRANCH;
    // Required to disable instructions after branch

```



```

always @(posedge clk1)          // IF Stage
if (HALTED == 0)
begin
    if (((EX_MEM_IR[31:26] == BEQZ) && (EX_MEM_cond == 1)) ||
        ((EX_MEM_IR[31:26] == BNEQZ) && (EX_MEM_cond == 0)))
    begin
        IF_ID_IR      <= #2 Mem[EX_MEM_ALUOut];
        TAKEN_BRANCH <= #2 1'b1;
        IF_ID_NPC    <= #2 EX_MEM_ALUOut + 1;
        PC            <= #2 EX_MEM_ALUOut + 1;
    end
    else
    begin
        IF_ID_IR      <= #2 Mem[PC];
        IF_ID_NPC    <= #2 PC + 1;
        PC            <= #2 PC + 1;
    end
end

```



```

always @(posedge clk2)           // ID Stage
  if (HALTED == 0)
    begin
      if (IF_ID_IR[25:21] == 5'b00000)  ID_EX_A <= 0;
      else ID_EX_A     <= #2 Reg[IF_ID_IR[25:21]]; // "rs"

      if (IF_ID_IR[20:16] == 5'b00000)  ID_EX_B <= 0;
      else ID_EX_B     <= #2 Reg[IF_ID_IR[20:16]]; // "rt"

      ID_EX_NPC    <= #2 IF_ID_NPC;
      ID_EX_IR     <= #2 IF_ID_IR;
      ID_EX_Imm    <= #2 {{16{IF_ID_IR[15]}}, {IF_ID_IR[15:0]}};
    end
  end
end

```



```

case (IF_ID_IR[31:26])
  ADD,SUB,AND,OR,SLT,MUL: ID_EX_type <= #2 RR_ALU;
  ADDI,SUBI,SLTI:          ID_EX_type <= #2 RM_ALU;
  LW:                      ID_EX_type <= #2 LOAD;
  SW:                      ID_EX_type <= #2 STORE;
  BNEQZ,BEQZ:              ID_EX_type <= #2 BRANCH;
  HLT:                     ID_EX_type <= #2 HALT;
  default:                 ID_EX_type <= #2 HALT;
                           // Invalid opcode
endcase
end

```



```

always @(posedge clk1)          // EX Stage
  if (HALTED == 0)
  begin
    EX_MEM_type <= #2 ID_EX_type;
    EX_MEM_IR   <= #2 ID_EX_IR;
    TAKEN_BRANCH <= #2 0;

    case (ID_EX_type)
      RR_ALU: begin
        case (ID_EX_IR[31:26]) // "opcode"
          ADD:     EX_MEM_ALUOut <= #2 ID_EX_A + ID_EX_B;
          SUB:     EX_MEM_ALUOut <= #2 ID_EX_A - ID_EX_B;
          AND:     EX_MEM_ALUOut <= #2 ID_EX_A & ID_EX_B;
          OR:      EX_MEM_ALUOut <= #2 ID_EX_A | ID_EX_B;
          SLT:     EX_MEM_ALUOut <= #2 ID_EX_A < ID_EX_B;
          MUL:     EX_MEM_ALUOut <= #2 ID_EX_A * ID_EX_B;
          default: EX_MEM_ALUOut <= #2 32'hxxxxxxxxx;
        endcase
      end
  end

```

```

RM_ALU: begin
  case (ID_EX_IR[31:26]) // "opcode"
    ADDI:    EX_MEM_ALUOUT <= #2 ID_EX_A + ID_EX_Imm;
    SUBI:    EX_MEM_ALUOUT <= #2 ID_EX_A - ID_EX_Imm;
    SLTI:    EX_MEM_ALUOUT <= #2 ID_EX_A < ID_EX_Imm;
    default: EX_MEM_ALUOUT <= #2 32'hxxxxxxxxx;
  endcase
end

```



```

LOAD, STORE:
begin
    EX_MEM_ALUOut <= #2 ID_EX_A + ID_EX_Imm;
    EX_MEM_B      <= #2 ID_EX_B;
end

BRANCH: begin
    EX_MEM_ALUOut <= #2 ID_EX_NPC + ID_EX_Imm;
    EX_MEM_cond   <= #2 (ID_EX_A == 0);
end
endcase
end

```



```

always @(posedge clk2)          // MEM Stage
if (HALTED == 0)
begin
    MEM_WB_type <= EX_MEM_type;
    MEM_WB_IR   <= #2 EX_MEM_IR;

    case (EX_MEM_type)
        RR_ALU, RM_ALU:
            MEM_WB_ALUOut      <= #2 EX_MEM_ALUOut;

        LOAD:      MEM_WB_LMD      <= #2 Mem[EX_MEM_ALUOut];

        STORE:    if (TAKEN_BRANCH == 0) // Disable write
                    Mem[EX_MEM_ALUOut] <= #2 EX_MEM_B;
    endcase
end

```



```

always @(posedge clk1)          // WB Stage
begin
  if (TAKEN_BRANCH == 0)      // Disable write if branch taken
    case (MEM_WB_type)
      RR_ALU:   Reg[MEM_WB_IR[15:11]] <= #2 MEM_WB_ALUOut; // "rd"
      RM_ALU:   Reg[MEM_WB_IR[20:16]] <= #2 MEM_WB_ALUOut; // "rt"
      LOAD:     Reg[MEM_WB_IR[20:16]] <= #2 MEM_WB_LMD; // "rt"
      HALT:    HALTED <= #2 1'b1;
    endcase
  end
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

## END OF LECTURE 40



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

62



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 41: VERILOG MODELING OF THE PROCESSOR (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Running Example Programs on the Processor

- We shall show how test benches can be written for verifying the operation of the processor model.
- What will be the test benches like?
  - Load a program from a specific memory address (say, 0).
  - Initialize PC with the starting address of the program.
  - The program starts executing, and will continue to do so until the HLT instruction is encountered.
  - We can print the results from memory locations or registers to verify the operation.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

64

## Example 1

- Add three numbers 10, 20 and 30 stored in processor registers.
- The steps:
  - Initialize register R1 with 10.
  - Initialize register R2 with 20.
  - Initialize register R3 with 30.
  - Add the three numbers and store the sum in R4.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

65

Assembly Language Program	Machine Code (in Binary)
ADDI R1,R0,10	001010 00000 00001 0000000000001010
ADDI R2,R0,20	001010 00000 00010 00000000000010100
ADDI R3,R0,25	001010 00000 00011 00000000000011001
ADD R4,R1,R2	000000 00001 00010 00100 00000 000000
ADD R5,R4,R3	000000 00100 00011 00101 00000 000000
HLT	111111 00000 00000 00000 00000 000000



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

66

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (20)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

67

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h2801000a; // ADDI R1,R0,10
    mips.Mem[1] = 32'h28020014; // ADDI R2,R0,20
    mips.Mem[2] = 32'h28030019; // ADDI R3,R0,25
    mips.Mem[3] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[4] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[5] = 32'h00222000; // ADD  R4,R1,R2
    mips.Mem[6] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[7] = 32'h00832800; // ADD  R5,R4,R3
    mips.Mem[8] = 32'hfc000000; // HLT

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

68

```

mips.HALTED = 0;
mips.PC = 0;
mips.TAKEN_BRANCH = 0;

#280
for (k=0; k<6; k++)
    $display ("R%ld - %2d", k, mips.Reg[k]);
end

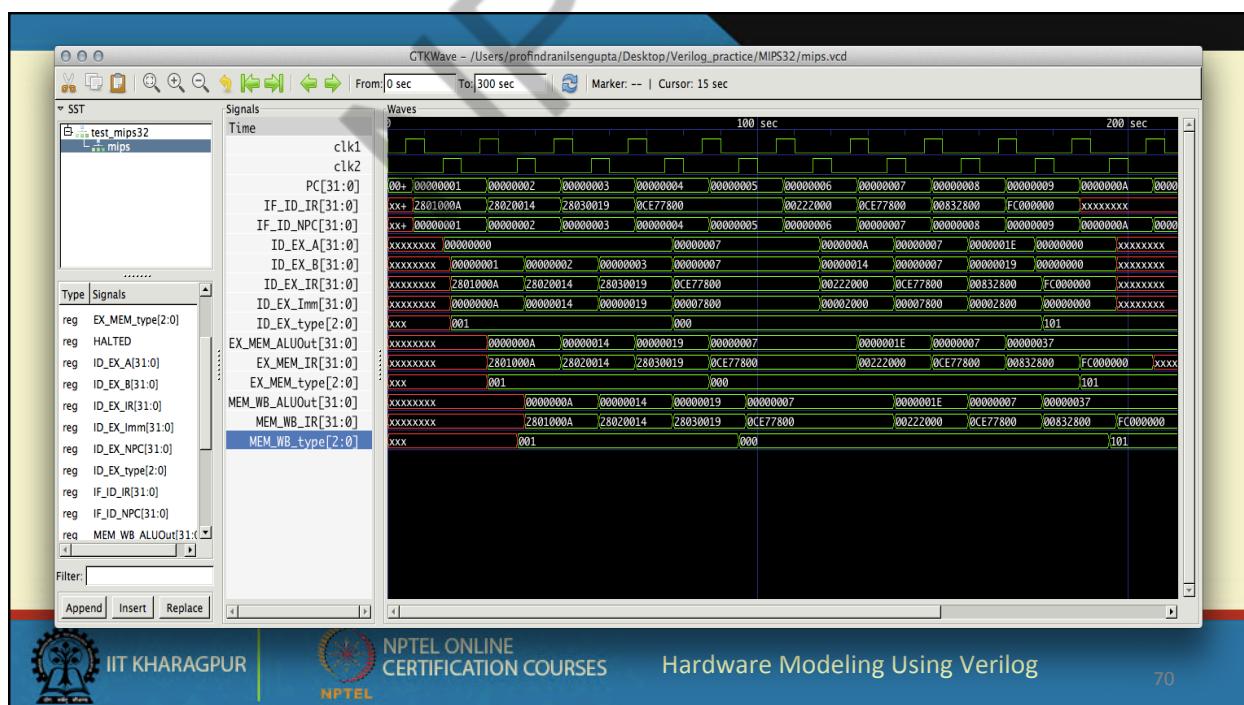
initial
begin
    $dumpfile ("mips.vcd");
    $dumpvars (0, test_mips32);
    #300 $finish;
end

endmodule

```

**SIMULATION OUTPUT**

R0	- 0
R1	- 10
R2	- 20
R3	- 25
R4	- 30
R5	- 55



## Example 2

- Load a word stored in memory location 120, add 45 to it, and store the result in memory location 121.
- The steps:
  - Initialize register R1 with the memory address 120.
  - Load the contents of memory location 120 into register R2.
  - Add 45 to register R2.
  - Store the result in memory location 121.

Assembly Language Program	Machine Code (in Binary)
ADDI R1,R0,120	001010 00000 00001 0000000001111000
LW R2,0(R1)	001000 00001 00010 0000000000000000
ADDI R2,R2,45	001010 00010 00010 0000000000101101
SW R2,1(R1)	001001 00010 00001 0000000000000001
HLT	111111 00000 00000 00000 00000 00000

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (50)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h28010078; // ADDI R1,R0,120
    mips.Mem[1] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[2] = 32'h20220000; // LW   R2,0(R1)
    mips.Mem[3] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[4] = 32'h2842002d; // ADDI R2,R2,45
    mips.Mem[5] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[6] = 32'h24220001; // SW   R2,1(R1)
    mips.Mem[7] = 32'hfc000000; // HLT

    mips.Mem[120] = 85;

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74

```

mips.PC = 0;
mips.HALTED = 0;
mips.TAKEN_BRANCH = 0;

#500 $display ("Mem[120]: %4d \nMem[121]: %4d",
               mips.Mem[120], mips.Mem[121]);
end

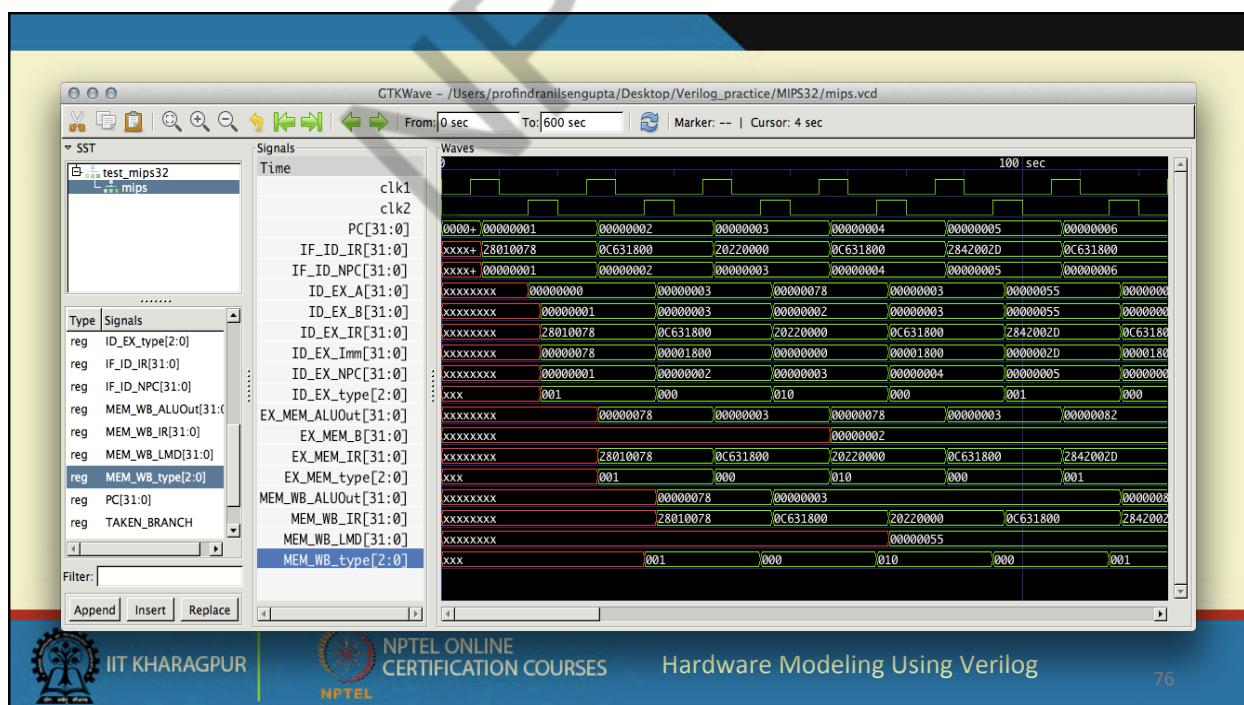
initial
begin
$dumpfile ("mips.vcd");
$dumpvars (0, test_mips32);
#600 $finish;
end

endmodule

```

**SIMULATION OUTPUT**

**Mem[120]: 85**  
**Mem[121]: 130**



## Example 3

- Compute the factorial of a number N stored in memory location 200. The result will be stored in memory location 198.
- The steps:
  - Initialize register R10 with the memory address 200.
  - Load the contents of memory location 200 into register R3.
  - Initialize register R2 with the value 1.
  - In a loop, multiply R2 and R3, and store the product in R2.
  - Decrement R3 by 1; if not zero repeat the loop.
  - Store the result (from R3) in memory location 198.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

77

Assembly Language Program		Machine Code (in Binary)
ADDI	R10,R0,200	001010 00000 01010 0000000011001000
ADDI	R2,R0,1	001010 00000 00010 0000000000000001
LW	R3,0(R10)	001000 01010 00011 0000000000000000
Loop:	MUL R2,R2,R3	000101 00010 00011 00010 00000 00000
SUBI	R3,R3,1	001011 00011 00011 0000000000000001
BNEQZ	R3,Loop	001101 00011 00000 1111111111111101
SW	R2,-2(R10)	001001 00011 01010 1111111111111110
HLT		111111 00000 00000 00000 00000 00000



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

78

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (50)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

79

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h280a00c8; // ADDI R10,R0,200
    mips.Mem[1] = 32'h28020001; // ADDI R2,R0,1
    mips.Mem[2] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[3] = 32'h21430000; // LW R3,0(R10)
    mips.Mem[4] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[5] = 32'h14431000; // Loop: MUL R2,R2,R3
    mips.Mem[6] = 32'h2c630001; // SUBI R3,R3,1
    mips.Mem[7] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[8] = 32'h3460ffff; // BNEQZ R3,Loop (i.e. -4 offset)
    mips.Mem[9] = 32'h2542ffff; // SW R2,-2(R10)
    mips.Mem[10] = 32'hfc000000; // HLT

```



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

80

```

mips.Mem[200] = 7;      // Find factorial of 7

mips.PC = 0;
mips.HALTED = 0;
mips.TAKEN_BRANCH = 0;

#2000 $display ("Mem[200] = %2d, Mem[198] = %6d",
                 mips.Mem[200], mips.Mem[198]);
end

initial
begin
$dumpfile ("mips.vcd");
$dumpvars (0, test_mips32);
$monitor ("R2: %4d", mips.Reg[2]);
#3000 $finish;
end
endmodule

```



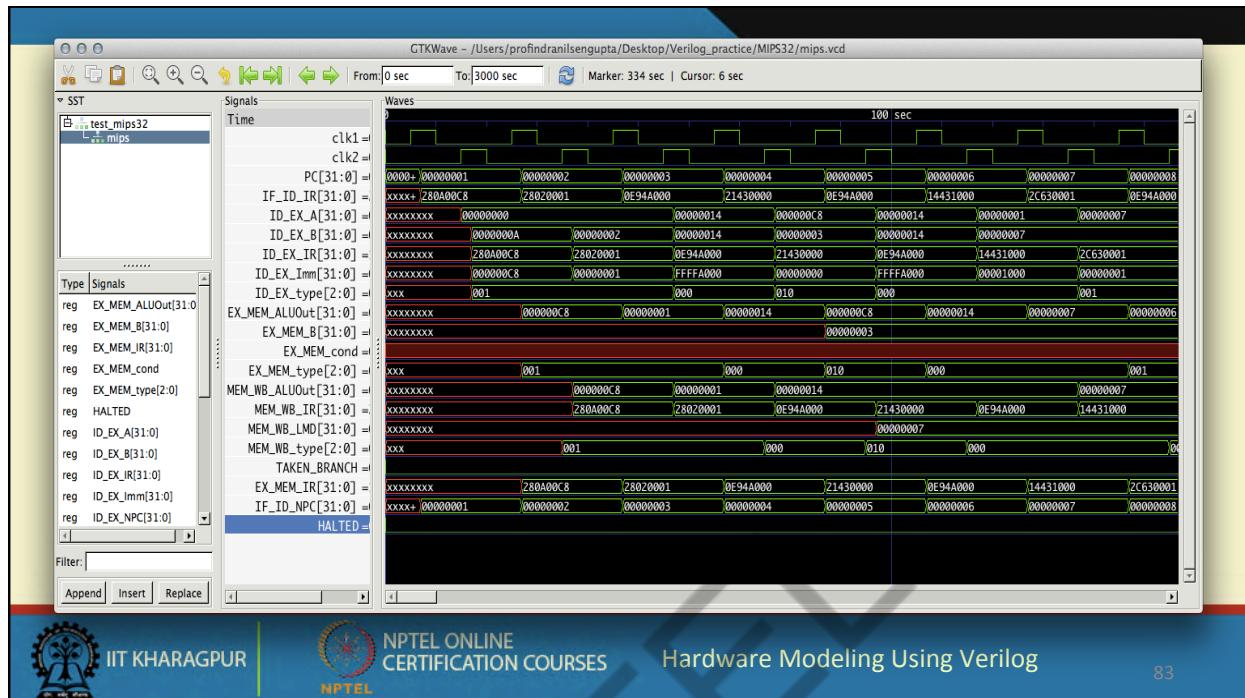
### SIMULATION OUTPUT

```

R2:      2
R2:      1
R2:      7
R2:     42
R2:    210
R2:   840
R2: 2520
R2: 5040
R2: 5040
Mem[200] = 7, Mem[198] = 5040

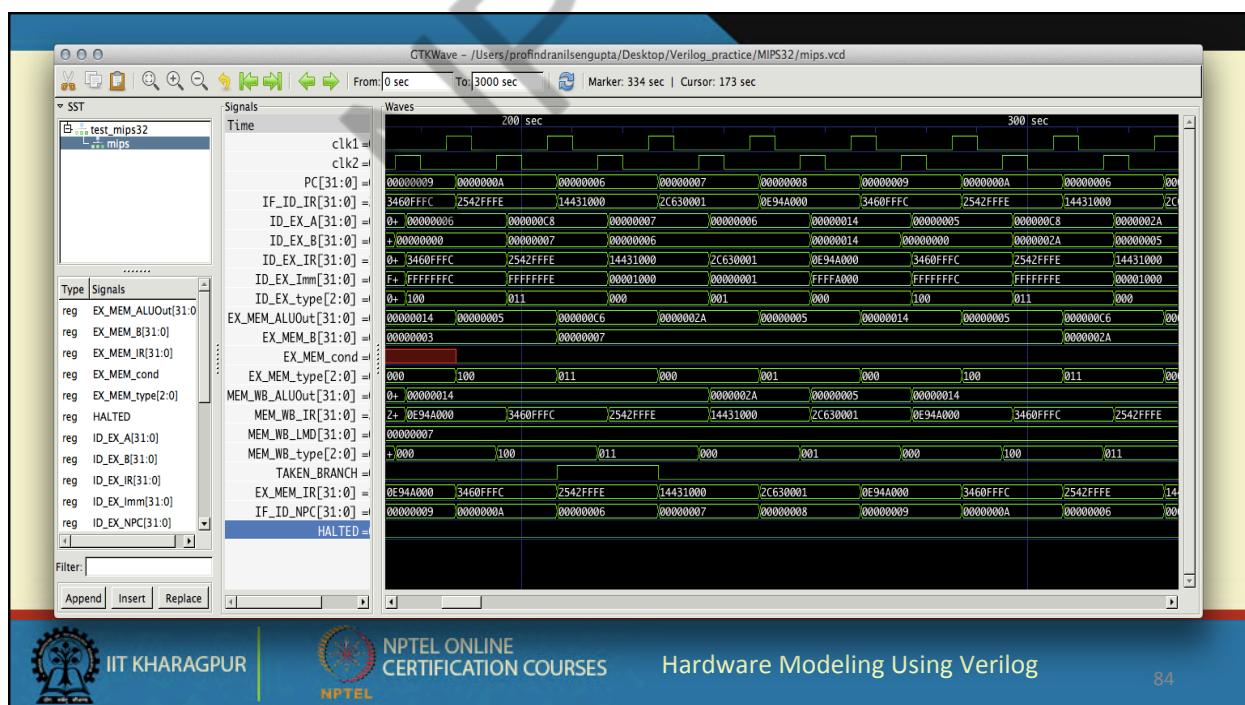
```



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

83

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

84

## Point to Note

- We have not considered the methods for avoiding hazards in pipelines.
- For the examples shown, we have inserted dummy instructions between dependent pairs of instructions.
  - So that data hazard does not lead to incorrect results.
- Also, we have modeled the processor using behavioral code.
  - In a real design where the target is to synthesize into hardware, structural design of the pipeline stages is usually used.
  - The Verilog code will be generating the control signals for the pipeline data path in the proper sequence.



IIT KHARAGPUR

NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

85

## END OF LECTURE 41



IIT KHARAGPUR

NPTEL  
NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

86



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

NPTEL

## Lecture 42: SUMMARIZATION OF THE COURSE

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Topics Covered

- Basic introduction to Verilog language and features.
- Difference between behavioral and structural representations.
- Using continuous dataflow assignments in Verilog code.
- Modeling combinational and sequential circuits.
- Procedural blocks, blocking and non-blocking assignments.
- Writing test benches.



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

88

## Other Topics Discussed

- Modeling finite state machines and controllers.
- Partitioning a design into data and control paths.
- User defined primitives.
- Switch level modeling.
- Basic concepts of pipelining.
- Pipelined design and implementation of a subset of the MIPS32 instruction set.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

89

## What has not been covered!!

- Use of synthesis tools, for FPGA and/or ASIC implementations.
- Complex structural designs, as they are difficult to discuss on slides.
- Expertise comes from experience.
  - Designing for complex problems gives a lot more insight than a text book or a course can teach.



IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

90

**END OF THE COURSE**

**THANK YOU FOR ATTENDING**



IIT KHARAGPUR



NPTEL  
ONLINE  
CERTIFICATION COURSES

Hardware Modeling Using Verilog

91

NPTEL