Date:	**02/6/2020**	**Name:**	**SAFIYA BANU**
Course:	**DIGITAL DESIGN USING HDL**	**USN:**	**4AL16EC061**
Topic:	FPGA Basics: Architecture, Applications and Uses Verilog HDL Basics by Intel Verilog Testbench code to verify the design under test (DUT)	**Semester & Section:**	**8th , B**
Github Repository:	**Safiya-Courses**		

FPGA Basics: Architecture, Applications and Uses

The field-programmable gate array (FPGA) is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application.

What is FPGA?

The **field-programmable gate array (FPGA)** is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application. The interconnects can readily be reprogrammed, allowing an FPGA to accommodate changes to a design or even support a new application during the lifetime of the part.

The FPGA has its roots in earlier devices such as programmable read-only memories (PROMs) and programmable logic devices (PLDs). These devices could be programmed either at the factory or in the field, but they used fuse technology (hence, the expression “burning a PROM”) and could not be changed once programmed. In contrast, FPGA stores its configuration information in a re-programmable medium such as static RAM (SRAM) or flash memory. FPGA manufacturers include Intel**, Xilinx,** Lattice Semiconductor**,** Microchip Technology **and** Microsemi**.**

FPGA Architecture

A basic FPGA architecture (Figure 1) consists of thousands of fundamental elements called configurable logic blocks (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs. Input/output (I/O) blocks interface between the FPGA and external devices.

Depending on the manufacturer, the CLB may also be referred to as a logic block (LB), a logic element (LE) or a logic cell (LC)

An individual CLB (Figure 2) is made up of several logic blocks. A lookup table (LUT) is a characteristic feature of an FPGA. An LUT stores a predefined list of logic outputs for any combination of inputs: LUTs with four to six input bits are widely used. Standard logic functions such as multiplexers (mux), full adders (FAs) and flip-flops are also common.

*(*The number and arrangement of components in the CLB varies by device; the simplified example in Figure 2 contains two three-input LUTs (1), an FA (3) and a D-type flip-flop (5), plus a standard mux (2) and two muxes, (4) and (6), that are configured during FPGA programming.

This simplified CLB has two modes of operation. In normal mode, the LUTs are combined with Mux 2 to form a four-input LUT; in arithmetic mode, the LUT outputs are fed as inputs to the FA together with a carry input from another CLB. Mux 4 selects between the FA output or the LUT output. Mux 6 determines whether the operation is asynchronous or synchronized to the FPGA clock via the D flip-flop.

Current-generation FPGAs include more complex CLBs capable of multiple operations with a single block; CLBs can combine for more complex operations such as multipliers, registers, counters and even digital signal processing (DSP) functions.

CPLD vs FPGA

Originally, FPGAs included the blocks in Figure 1 and little else, but now designers can choose from products with a large range of features. Less complex devices such as simple programmable logic devices (SPLDs) and complex programmable logic devices (CPLDs) bridge the gap between discrete logic devices and entry-level FPGAs.

Entry-level FPGAs emphasize low power consumption, low logic density and low complexity per chip. Higher-function devices add functional blocks dedicated to specific functions: Examples include clock management components, phase-locked loops (PLLs), high-speed serializers and deserializers, Ethernet MACs, PCI express controllers and high-speed transceivers. These blocks can either be implemented with CLBs—termed soft IP—or designed as separate circuits; i.e., hard IP. Hard IP blocks gain performance at the expense of reconfigurability.

At the high end, the FPGA product family includes complex system-on-chip (SoC) parts that integrate the FPGA architecture, hard IP and a microprocessor CPU core into a single component. Compared to separate devices, a SoC FPGA provides higher integration, lower power, smaller board size and higher-bandwidth communication between the core and other blocks.

FPGA Applications

Many applications rely on the parallel execution of identical operations; the ability to configure the FPGA's CLBs into hundreds or thousands of identical processing blocks has applications in image processing, artificial intelligence (AI), data center hardware accelerators, enterprise networking and automotive advanced driver assistance systems (ADAS).

Many of these application areas are changing very quickly as requirements evolve and new protocols and standards are adopted. FPGAs enable manufacturers to implement systems that can be updated when necessary.

A good example of FPGA use is high-speed search: Microsoft is using FPGAs in its data centers to run Bing search algorithms. The FPGA can change to support new algorithms as they are created. If needs change, the design can be repurposed to run simulation or modeling routines in an HPC application. This flexibility is difficult or impossible to achieve with an ASIC.

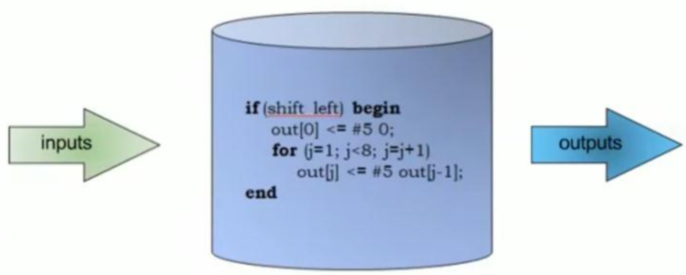
Other FPGA uses include aerospace and defense, medical electronics, digital television, consumer electronics, industrial motor control, scientific instruments, cybersecurity systems and wireless communications.

Verilog History

- Introduced in 1984 by Gateway Design Automation
- 1989 Cadence purchased Gateway and subsequently released Verilog to the public
- Open Verilog International (OVI) was formed to control the language specifications
- 1995 IEEE accepted OVI Verilog as a standard
- 2001 and 2005 IEEE revised standard
- 2009 Merged with SystemVerilog becoming IEEE Standard 1800-2009

Behavior Modeling

- Only the functionality of the circuit, no structure
- Synthesis tool creates correct logic

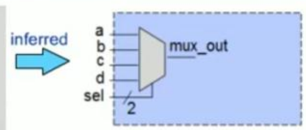


More Terminology

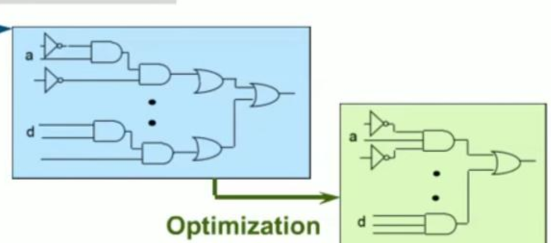
- **Register Transfer Level (RTL): A type of behavioral modeling, for the purpose of synthesis**
  - Hardware is implied or inferred
  - Synthesizable
- **Synthesis: Translating HDL to a circuit and then optimizing the represented circuit**
- **RTL Synthesis: Translating a RTL model of hardware into an optimized technology specific gate level implementation**

RTL Synthesis

```
always @(a, b, c, d, sel)
   case (sel)
      2'b00: mux_out = a;
      2b'01: mux_out = b;
      2b'10: mux_out = c;
      2'b11: mux_out = d;
   endcase
```



Translation



Verilog - Basic Modeling Structure

```
module module_name (port_list);
    port declarations
    data type declarations
    circuit functionality
    timing specifications
endmodule
```

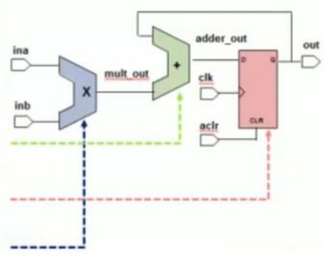
- Begins with keyword **module** & ends with keyword **endmodule**
- Case-sensitve
- All keywords are lowercase
- Whitespace is used for readability
- Semicolon is the statement terminator
- **//** : Single line comment
- **/* */** : Multi-line comment
- Timing specification is for simulation (not discussed)

ALTERA. MEASURABLE ADVANTAGE™

Verilog HDL Model: Demonstration Example

```
module mult_acc (out, ina, inb, clk, aclr);

input [7:0] ina, inb;
input clk, aclr;
output [15:0] out;

wire [15:0] mult_out, adder_out;
reg [15:0] out;

assign adder_out = mult_out + out;

always @ (posedge clk or posedge aclr)
        if (aclr)   out = 16'h0000;
        else        out = adder_out;

multa u1(.in_a(ina), .in_b(inb),
                    .m_out(mult_out));

endmodule
```

Ports and data types



ALTERA. MEASURABLE ADVANTAGE™

Arithmetic Operators

Operator Symbol	Functionality	Examples ain = 5 ; bin = 10 ; cin = 2'b01 ; din = 2'b0z		
+	Add, Positive	bin + cin ⇒ 11	+bin ⇒ 10	ain + din ⇒ x
–	Subtract, Negate	bin – cin ⇒ 9	-bin ⇒ -10	ain – din ⇒ x
*	Multiply	ain * bin ⇒ 50		
/	Divide	bin / ain ⇒ 2		
%	Modulus	bin % ain ⇒ 0		
**	Exponent*	ain ** 2 ⇒ 25		

- Treats vectors as a whole value
- Results unknown if any operand is Z or X
- Carry bit(s) handled automatically if result wider than operands
- Carry bit lost if operands and results are same size

** Check synthesis tool for support*

Blocking vs. Nonblocking Assignments

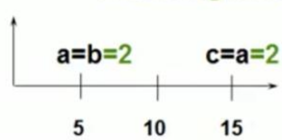
Blocking (=)

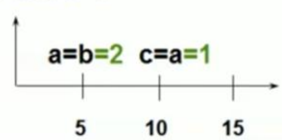
```
initial
   begin
      a = #5 b;
      c = #10 a;
   end
```

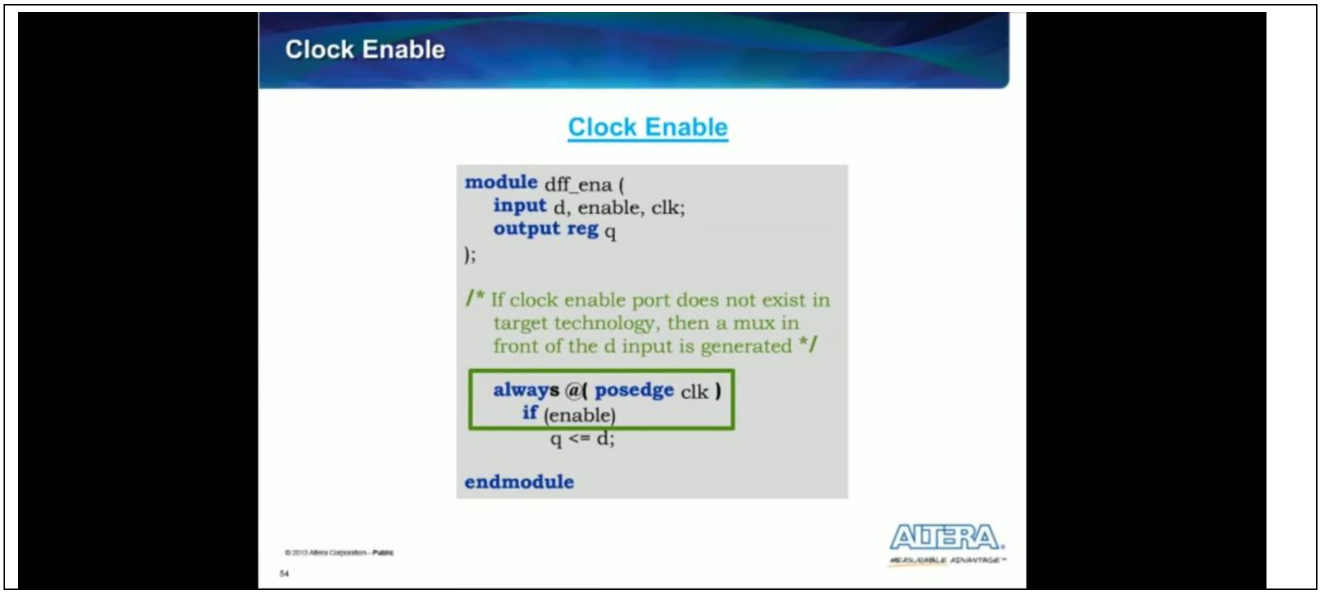
Nonblocking (<=)

```
initial
   begin
      a <= #5 b;
      c <= #10 a;
   end
```

Assuming initially a=1 and b=2






Clock Enable
Clock Enable
module dff_ena (
input d, enable, clk;
output reg q
);
/* If clock enable port does not exist in
target technology, then a mux in
front of the d input is generated */
always @(posedge clk)
if (enable)
q <= d;
endmodule
© 2013 Altera Corporation—Public
54
ALTERA.
MEASURABLE ADVANTAGE™

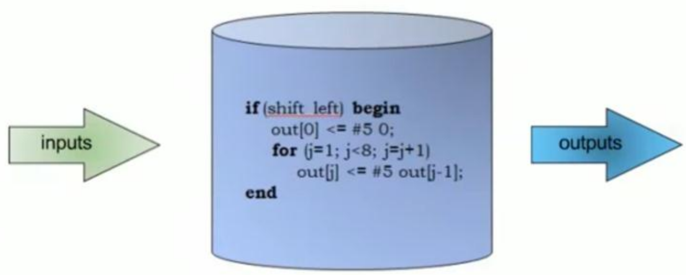
11:35 AM 47.7KB/s

Verilog History

- Introduced in 1984 by Gateway Design Automation
- 1989 Cadence purchased Gateway and subsequently released Verilog to the public
- Open Verilog International (OVI) was formed to control the language specifications
- 1995 IEEE accepted OVI Verilog as a standard
- 2001 and 2005 IEEE revised standard
- 2009 Merged with SystemVerilog becoming IEEE Standard 1800-2009

Behavior Modeling

- Only the functionality of the circuit, no structure
- Synthesis tool creates correct logic

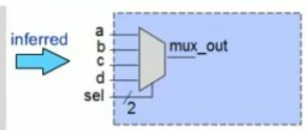


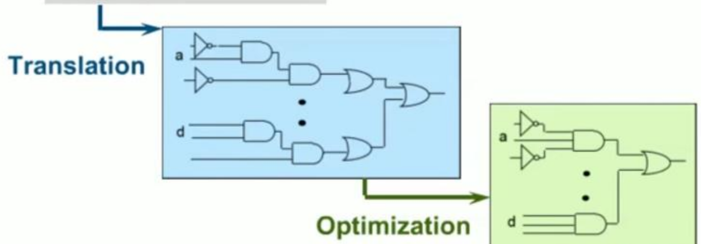
More Terminology

- **Register Transfer Level (RTL): A type of behavioral modeling, for the purpose of synthesis**
  - Hardware is implied or inferred
  - Synthesizable
- **Synthesis: Translating HDL to a circuit and then optimizing the represented circuit**
- **RTL Synthesis: Translating a RTL model of hardware into an optimized technology specific gate level implementation**

RTL Synthesis

```
always @(a, b, c, d, sel)
   case (sel)
      2'b00: mux_out = a;
      2b'01: mux_out = b;
      2b'10: mux_out = c;
      2'b11: mux_out = d;
   endcase
```

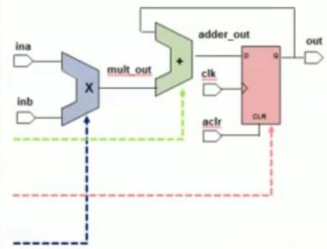




Verilog HDL Model: Demonstration Example

```
module mult_acc (out, ina, inb, clk, aclr);

input [7:0] ina, inb;
input clk, aclr;
output [15:0] out;

wire [15:0] mult_out, adder_out;
reg [15:0] out;

assign adder_out = mult_out + out;

always @ (posedge clk or posedge aclr)
        if (aclr)    out = 16'h0000;
        else         out = adder_out;

multa u1(.in_a(ina), .in_b(inb),
                    .m_out(mult_out));

endmodule
```

Ports and data types

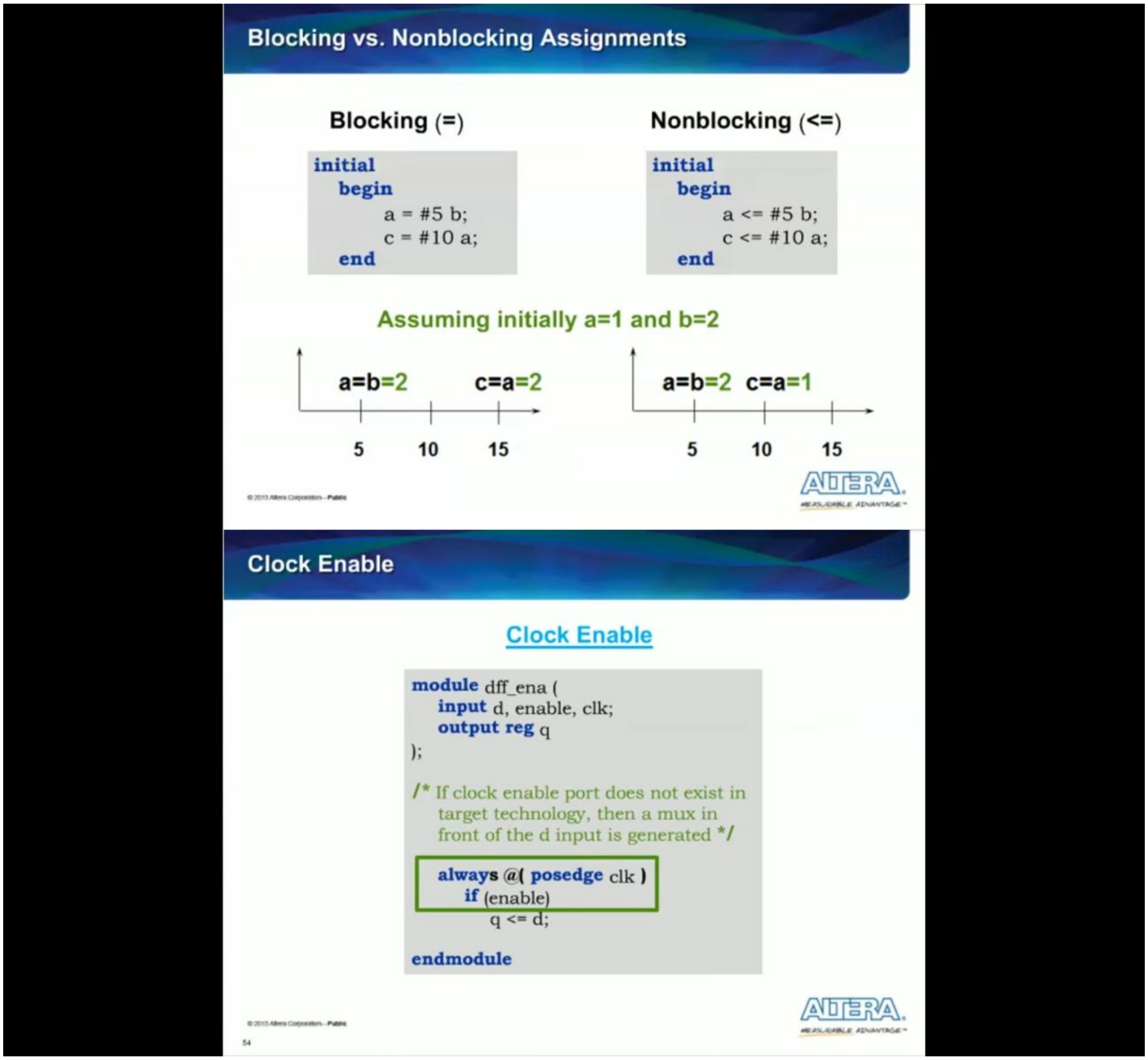


Arithmetic Operators

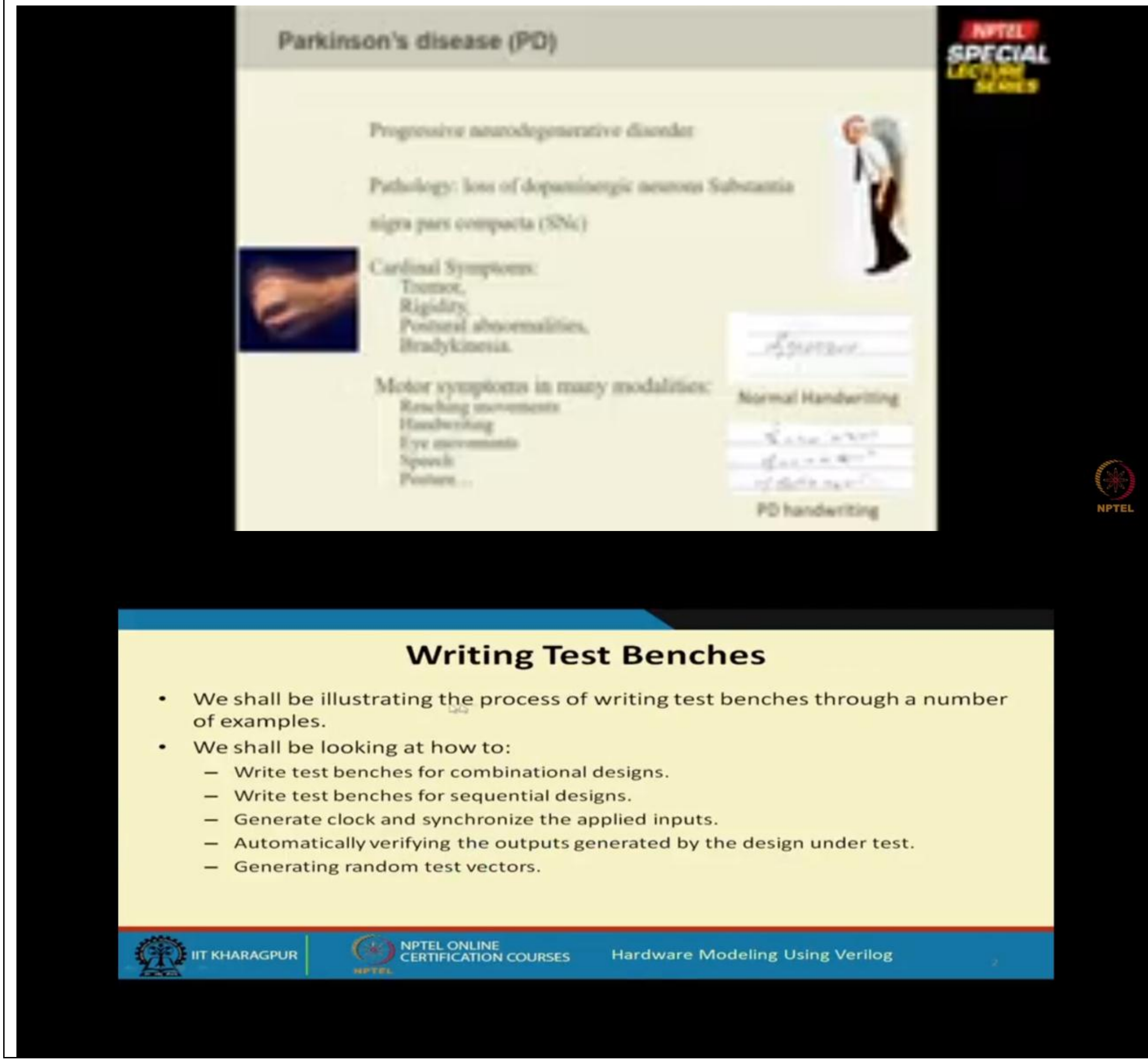
Operator Symbol	Functionality	Examples ain = 5 ; bin = 10 ; cin = 2'b01 ; din = 2'b0z		
+	Add, Positive	bin + cin ⇒ 11	+bin ⇒ 10	ain + din ⇒ x
–	Subtract, Negate	bin – cin ⇒ 9	-bin ⇒ -10	ain – din ⇒ x
*	Multiply	ain * bin ⇒ 50		
/	Divide	bin / ain ⇒ 2		
%	Modulus	bin % ain ⇒ 0		
**	Exponent*	ain ** 2 ⇒ 25		

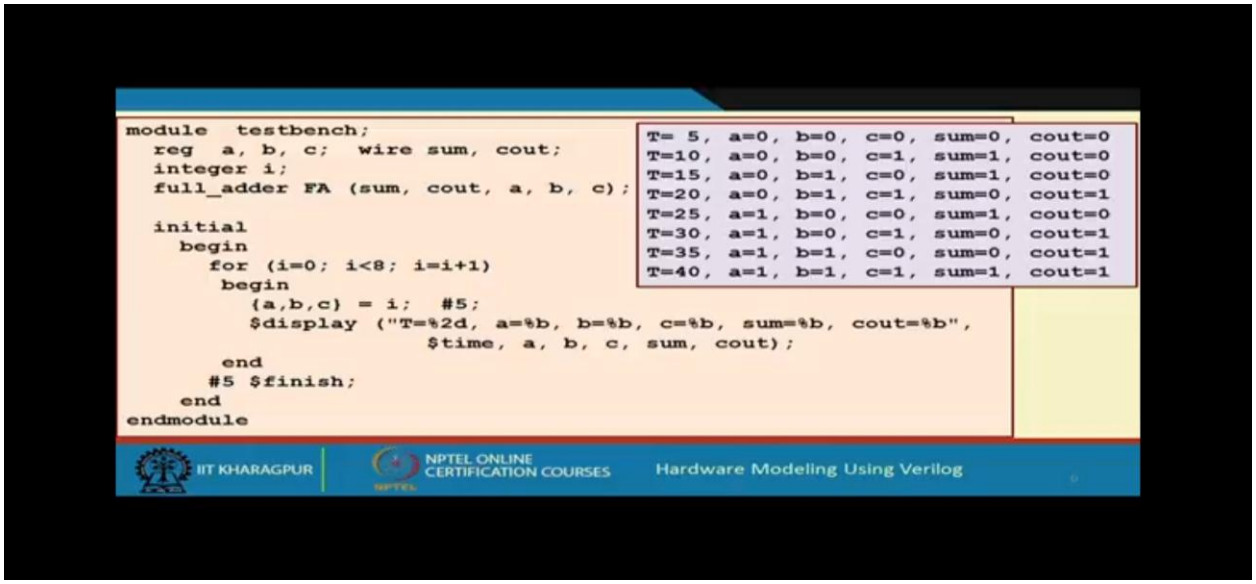
- **Treats vectors as a whole value**
- **Results unknown if any operand is Z or X**
- **Carry bit(s) handled automatically if result wider than operands**
- **Carry bit lost if operands and results are same size**

** Check synthesis tool for support*



Verilog Testbench code to verify the design under test (DUT)


Parkinson's disease (PD)
NPTEL SPECIAL LECTURE SERIES
Progressive neurodegenerative disorder
Pathology: loss of dopaminergic neurons Substantia nigra pars compacta (SNc)
Cardinal Symptoms:
Tremor,
Rigidity,
Postural abnormalities,
Bradykinesia.
Motor symptoms in many modalities:
Reaching movements
Handwriting
Eye movements
Speech
Posture ...
Normal Handwriting
PD handwriting
NPTEL
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



TASK

Implement a 4:1 MUX and write the test bench code to verify the module

```
module top;
wire  out;
reg  a;
reg  b;
reg  c;
reg  d;
reg s0, s1;

m41 name(.out(out), .a(a), .b(b), .c(c), .d(d), .s0(s0), .s1(s1));
 initial
 begin

 a=1'b0; b=1'b0; c=1'b0; d=1'b0;
 s0=1'b0; s1=1'b0;
```

```
 #500 $finish;

end

always #40 a=~a;
always #20 b=~b;
always #10 c=~c;
always #5 d=~d;
always #80 s0=~s0;
always #160 s1=~s1;

always@(a or b or c or d or s0 or s1)
$monitor("At time = %t, Output = %d", $time, out);

endmodule;
```