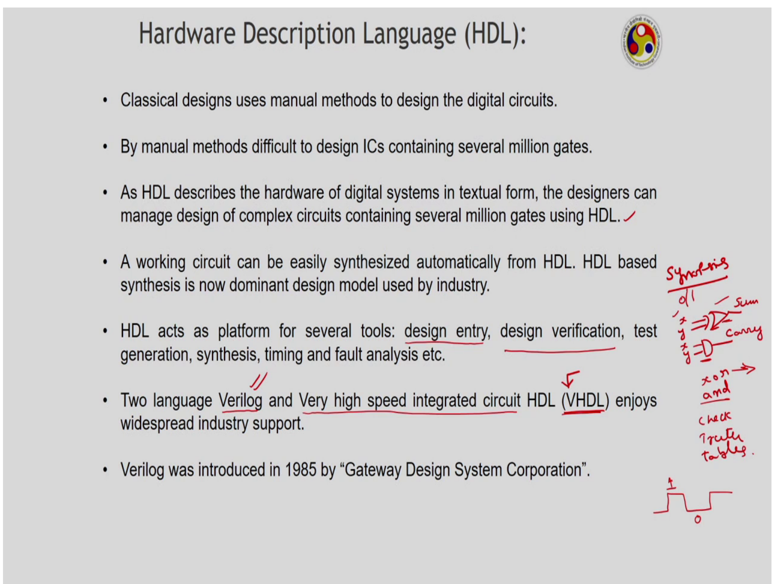
System Design using Verilog
Dr. Shaik Rafi Ahamed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Introduction to Verilog
Lecture - 01
Verilog Operators and Modules

Ok. The title of the course is a System Design Using Verilog. So, in this lecture first, I will discuss about the need of Verilog for the system design, and what is the design flow to design any system using Verilog, and then some basics of Verilog ok.

(Refer Slide Time: 00:57)



So, if you this any digital system design normally, so we will use the manual procedure to design any circuit let us say for example half adder. If you want to design a half adder, so we will take the number of inputs are two. Because the half adder is capable of adding two bits of the information and the output will be sum and carry.

So, we will form the truth table. So, we will simplify the Boolean expressions for the outputs. We will draw the logic diagram. Then we will connect this logic diagram on a breadboard. We will give the inputs logic 0 and logic 1 through a power supply.

So, 0 volts corresponds to 0 logic 0, and 5 volts corresponds to logic 1. And to verify the outputs, we will connect the LEDs at the sum and carry. If sum bit is 1, the LED will glow; if sum bit is 0, the LED will be off. This is the normal classical method ok.

But the drawback of this normal classical designs is if a circuit consisting of the millions of gates. So nowadays we are using VLSI and ULSI, Very Large-Scale Integration, and Ultra Large-Scale Integration. So, in this, the ICs will be containing several millions of the gates. So, in order to design those ICs containing several millions of the gates, the manual methods fails because it is very difficult to I mean connect these many number of gates on the breadboard and check the functionality of the system ok.

So, one of the solutions to design such complex ICs containing several millions of the gates is a hardware description language HDL. So, using HDL, so this HDL is actually it describes the hardware of the digital system in textual form. So, instead of using the actual hardware components, so we will use some sort of the textural form like C program and all ok.

So, because of that the designer can manage the design of the complex circuits containing the millions of gates. This is the one of the important advantage of this HDL. And another advantage of this HDL is a working circuit can be easily synthesized. So, we are going to discuss what is meant by synthesize.

So, synthesize means if you write a program using HDL , so we will get a net list means which will give the physical connections between the different electronic components. So, that synthesize circuits can be done by using this HDL. So, synthesis of the given HDL can be automatically done using Verilog ok. So, that is why this HDL based synthesize is now dominant design in many of the industries ok.

So, one of the important or famous synthesis tool is synopsys, synopsys is one of the important synthesize tool. And coming for the general operation that can be performed by using the HDL. So, HDL can perform the design entry. Once if you design a circuit say full adder ok, so the entry can be done by using if I take say for example, if you have a exclusive OR gate for sum if I give the inputs as x, y sum is exclusive OR operation, and carry is AND operation.

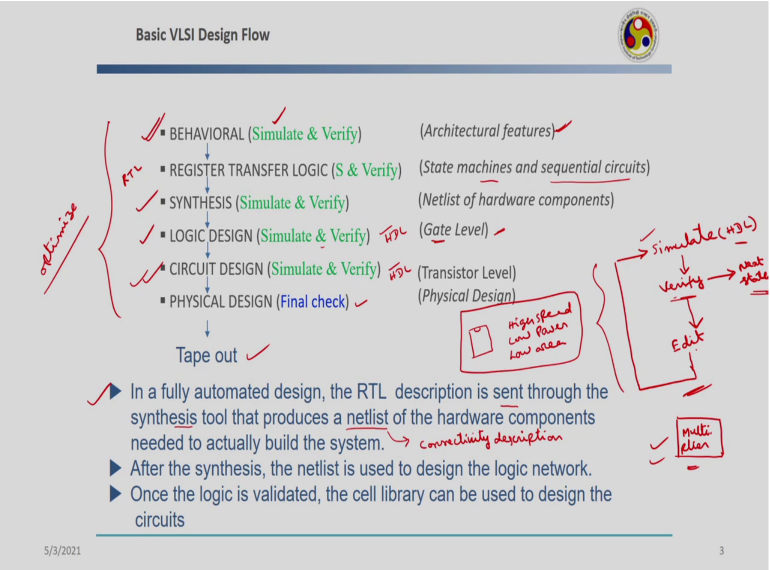
So, this design can be entered. This can be entered by using a keyword called XOR this can be entered by using AND ok. So, whatever the design that you are going to get for a system can be easily entered using HDL. So, the second function of this HDL is design verification. We can verify whether this particular gate is performing exclusive OR operation or not, this particular AND gate is performing AND operation or not.

Basically, we are going to check the truth tables. And we can generate the tests means here this x can be 0 or 1. This can be generated using HDL, x can be 0 or 1 – we can generate the wave form type of thing, we can generate some sort of this one this is logic 1, this is logic 0 ok. And also we can do the synthesis as I have told. So, synthesis is given an HDL, we can obtain the actual the circuit diagram.

Then we can do the timing and fault analysis, there are some uses of this HDL. So, there are two HDLs which are more popular. So, one is Verilog, another is VHDL. The full form of the VHDL is Very High Speed Integrated Circuit Hardware Description Language. This V stands for this entire thing Very High Speed Integrated Circuit, and HDL is Hardware Description Language. But this Verilog is more popular.

So, in this course, we will discuss only about Verilog ok. Once if you understand this Verilog language, we can easily learn this VHDL also. Again between these two, Verilog is one of the popular language. So, this Verilog was introduced by a Gateway Design System Corporation in 1985.

(Refer Slide Time: 07:21)



Now, coming for the basic design flow. So, in order to design any system, what is the flow that we have to follow so that finally, we can fabricate a IC. So which can operate at high speeds, which occupies the less area, which consumes less power ok. So, given a specification, from the specifications first we have to I mean obtain the behavioral features of that particular specification, we have to draw the behavioral architecture ok. This is basically architectural feature ok.

So, here at architectural level, we will stimulate and we will check the functionality, this is higher level of the design module. So, we will simulate that particular system ok. So, if the design is ok, then we will proceed for the next step. This simulation can be done using HDL.

Then we will verify the functionality. If functionality is ok, we will go for the next stage; otherwise we will edit, again we will go for the simulation. So, this process is repeated until you will get a correct design ok.

So, this behavioral I mean modeling is basically it consisting of the architecture without giving the actual details of the circuitry. Suppose, if we have the multiplier, so we will represent by a block diagram. This is a multiplier. But here at this behavioral level, we will not discuss about the internal details of the multiplier. So, what is the internal circuitry and all this details will be not covered at behavioral level ok.

And once this behavioral level is passed using this process, then we will go to the next level which is called as Register Transfer Logic RTL which is called RTL level. So, this RTL level basically consisting of different state machines and sequential circuits ok. So, again here also we will do the same procedure. So, we will simulate using HDL, we will verify the functionality. If ok, we will go to the next stage; otherwise you will edit and we will repeat this process until the functionality is correct ok.

Once if this RTL is available, so in a fully automated design this RTL description will be sent to the synthesis tool ok. This synthesis tool basically produces the netlist of hardware components. So, what is netlist is? Netlist will give this description of the connections. This will give connectivity description, how the different electronic components will be connected ok.

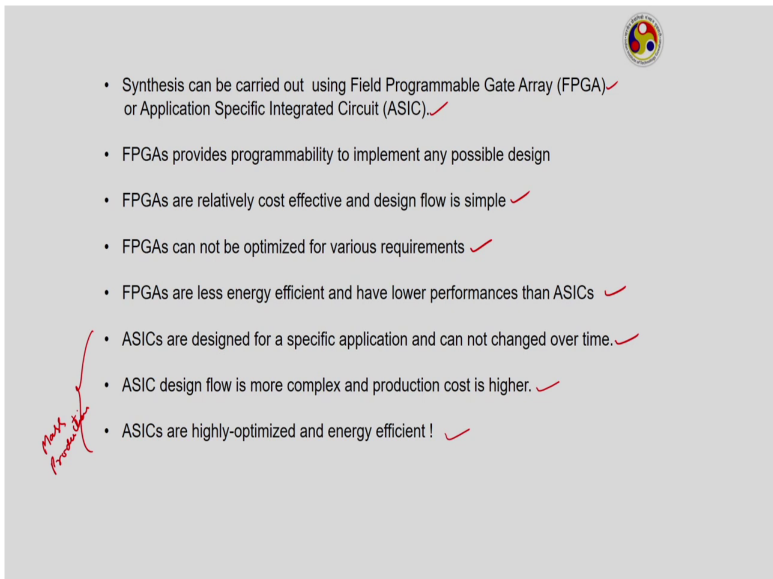
So, once if you know this connectivity of this one, connectivity of the circuit, then we will do the synthesis ok. So, there are two important synthesis tools are there one is FPGA and ASIC that we are going to discuss in the coming slides. So, after that, we will get the logic design this is what is called the gate level ok.

And each gate consisting of the transistors ok, then we will discuss about the circuit design. So, here at each and every level we will simulate and verify. This also we can simulate and verify by using HDL such as Verilog. Once all the stages have been passed, so finally, we will go for the physical design and that physical design will be sent to the tape out for the fabrication of the IC.

So, in order to I mean obtain an IC which can operate at high speed, low power consumption, low area, we have to optimize the problem at each and every level. So, we can optimize the problem at behavioral level, we can optimize the problem at RTL level, synthesis level, logic design, circuit design. Even in the layout also we can optimize the problem.

Once if you optimize this, we will get an optimal IC which can operate at high speeds, low power and low area. So, this is about the basic VLSI design flow.

(Refer Slide Time: 12:49)



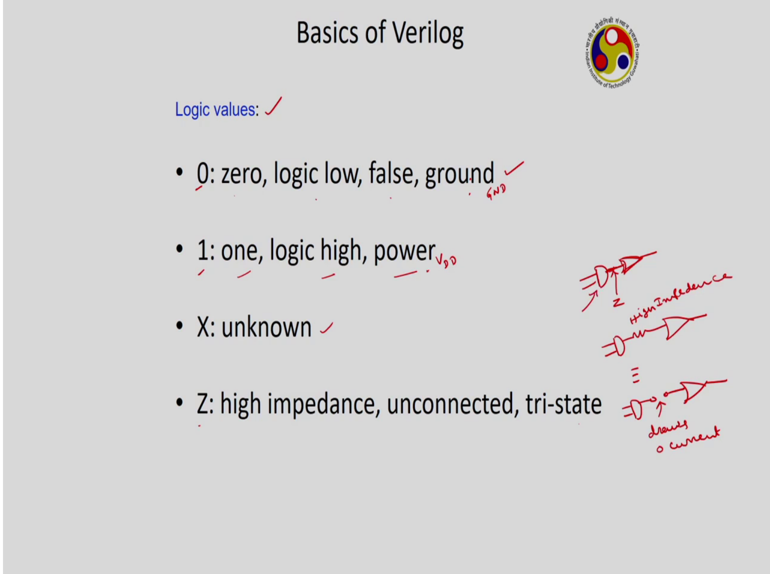
So, as I have told there are two I mean synthesis tools one is we can use the field programmable gate array, FPGA, or we can use application specific integrated circuit ok. So, each one is having relative advantages and disadvantages. If we go for the FPGAs, FPGA are cost effective and design flow is simple ok.

So, in FPGA we will be having some standard libraries ok. So, once if you dump this synthesize circuit into the FPGA, it will use the internal blocks, and it will simulate ok, but they are not optimized for the various requirements ok. So, if you want to design a particular IC say for example, FFT – Fast Fourier Transform ok. We cannot optimize this FFT design using FPGA. For that, we have to go for the ASIC ok.

So, they are less energy efficient and have lower performance than the ASICs ok. So, ASICs are designed for the application specific for a particular specific application. Even though this ASIC design flow is more complex, so this will give the optimal design and also it is more highly energy efficient.

So, in conclusion, normally if we want to have the mass production we will go for the ASICs, ASIC for mass production. Whereas, for designing a fewer systems, then we will go for the FPGA ok. So, you see about this synthesis tools ok. This is basic introduction to the HDL. So, with this we will go for some basics of Verilog how to write the codes using Verilog.

(Refer Slide Time: 14:53)



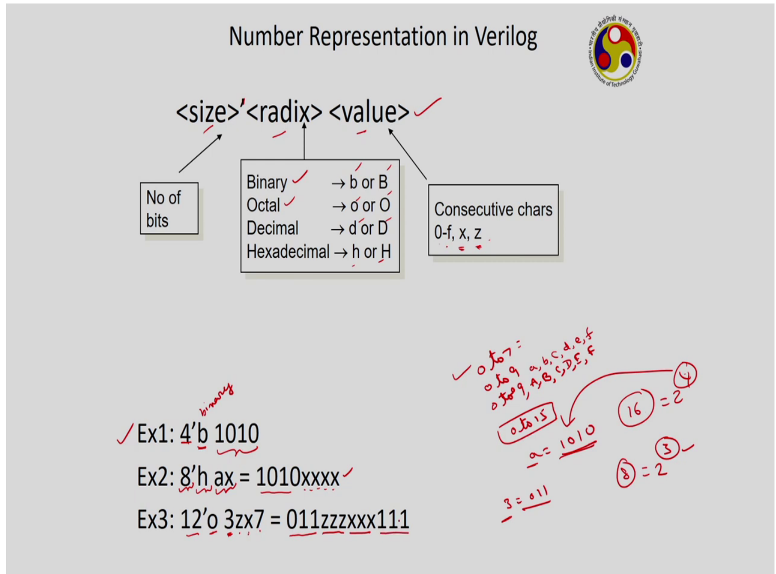
So, first I will discuss about the logic values. So, logic 0 can be zero, logic low, false, ground, so all these are the names for the logic 0. Similarly, logic 1, it can be one, logic high or power. This is ground and this is power ok. Like this is GND, this is VDD in case of CMOS circuits. X is a do not care which is unknown; it can be 0 or 1 ok.

So, there is one more state which is called as Z which is high impedance state which is unconnected or tri-state. So, basically this is something like floating ok. Suppose, if I connect the output of this gate to the input of some other gate, so if state of this output is Z that is something like this high impedance state.

So, in between these connections, we have very high impedance which is almost high impedance means if I take the infinite impedance, it will be open circuit which is almost equivalent to open circuit between these two means virtually it draws almost 0 current.

Means, if this connection if we do not want to use this connection; this gate will not draw any current from this output, so that the driving capability of this gate will increase, so it is a high impedance state.

(Refer Slide Time: 16:45)



Now, how to represent the numbers in the Verilog ok? So, the basic format of this number representation is we have to first specify the size followed by radix, then you have to specify the value. So, we have a different number of systems like binary. We have only 0 or 1; octal – we can have 0 to 7; decimal – 0 to 9; hexadecimal – 0 to 9, then we have A for 10, B for 11, C for 12, D – 13, E – 14, and F for 15.

So, totally we have 0 to 15 in hexadecimal ok. So, normally, this binary it can be represented with either lower case letter, upper case letter B, this with 0 either lower or upper, this is also D, this is also H.

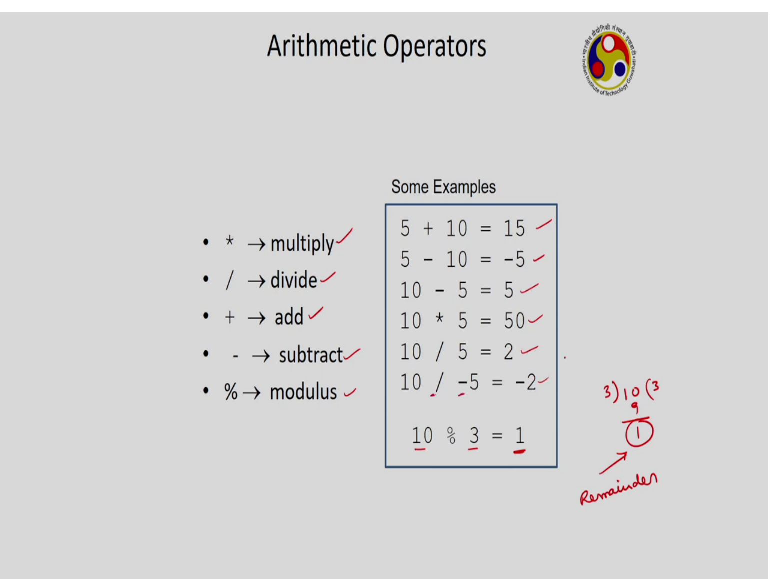
Then the value consisting of any of these values like 0 to F in case of hexadecimal; x it can be a do not care, it can be high impedance. Also as I have told in the earlier slide, so the logical values can be 0, 1, x, z. So, here also this value will be value can be either any one of this 0 to F, or it can be do not care, or it can be high impedance state.

Here I have given some examples. So, what does it mean? This 4 there will be a dash here. So, this 4 represents 4 bits. And this b represents binary. And what is the value of that binary number 1010. This 8 represents 8 bits; h represents hexadecimal; A x, A is nothing but as I have told this can be either lowercase or uppercase letters A is 10, 10 means 1 0 1 0. This is 1 0 10.

And this is x is do not care. So, we have 4 bit equivalent of the x is xxxx. So, total 8 bits. So, this is 12 bits – octal. This is octal can have digit from 0 to 7, so 3zx7. So, because this octal we have eight different characters, we use 3 bits to represent each octal digit. 3, 3 will be represented by a 3-bit number. In hexadecimal, 4 bits are required because total 16 different combinations, whereas in octal we have 8 combinations.

So, this 8 is 2 cube; so, each digit can be represented by 3 bits. In hexadecimal, 16 is 2 raise to the power of 4. So, each bit will be represented by 4 bits ok. So, this 3 is, this is 3, and z is 3z high impedance, x is do not care, and 7 is 3 ones ok. This is about the number representation in Verilog.

(Refer Slide Time: 20:06)

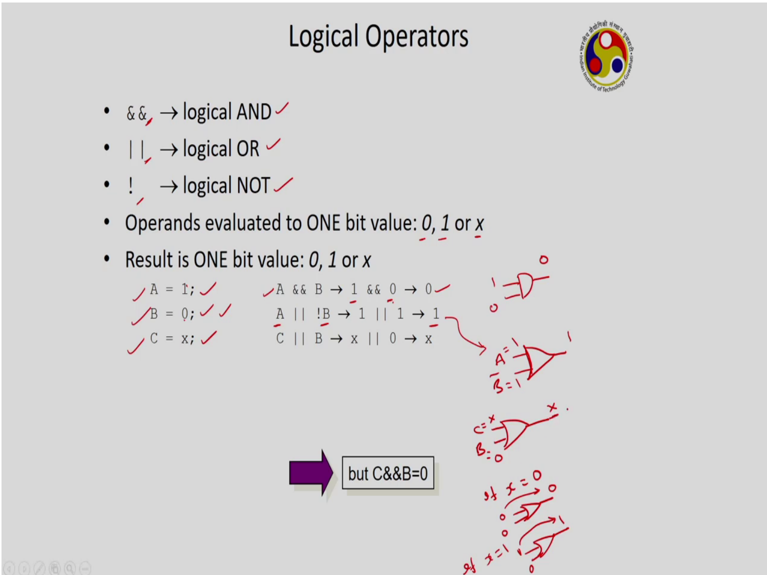


So, coming for the arithmetic operations similar to the conventional arithmetic. So, this star will be used to represent the multiplier, this symbol is used for divide, plus is for add, minus is for subtract, percentage is for modulus. So, modulus is nothing but the remainder of the operation.

For example, if I take 10 percentage of 3, so if I divide 10 with 3, 3 3's are 9, 1 is the reminder. So, this reminder is represented here ok that is what is called the modulus. So, these are some examples. This is plus operation 15. This is minus, minus 5; this is plus 5. This is

multiplication; this is division. This is multiplication, and you have minus value also this. So, these are the arithmetic operators.

(Refer Slide Time: 21:18)

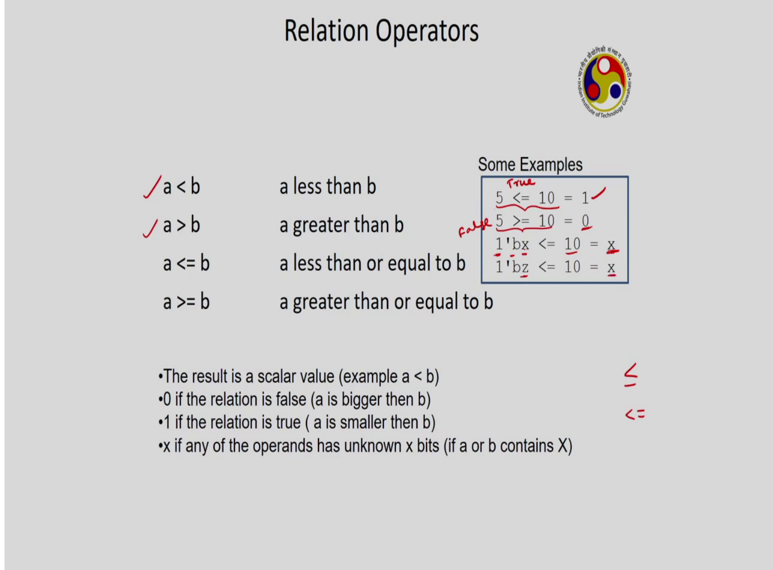


Similarly, we have logical operators. So, in order to do logical AND, we will use this symbol ok; and logical OR – this symbol; logical NOT – this symbol. So, in all these cases, each bit can be either 0 or 1 or x ok. And we are going to perform this operation bitwise if I take say for example, if A is 1, B is 0, C is do not care. So, this is A and B. So, A is 1, B is 0. So, 1 AND with 0 is 0. If I give for AND gate, one, one input is 0, another input is 1; output will be 0.

So, this will be equivalent to this logical OR operation we have A, and this is NOT, B bar, this is something like A,Bbar we are going to give. So, A is 1, B is 0, but B bar is equal to 1 ok, output is 1. This will give output is equal to 1. This is OR operation. And the third example is C OR with B.

So, C is do not care, B is 0. If you perform the OR operation with do not care, we will get do not care only because do not care can be 0 or 1 ok. If x is equal to 0, 0 OR with 0, so if you have 0 OR with 0 is 0; if do not care is 1, this is one the second bit is 0 only. So, output is 1. So, this one means 1; do not care is 0 means 0. So, basically this is do not care ok. So, these are three logical operations.

(Refer Slide Time: 23:38)



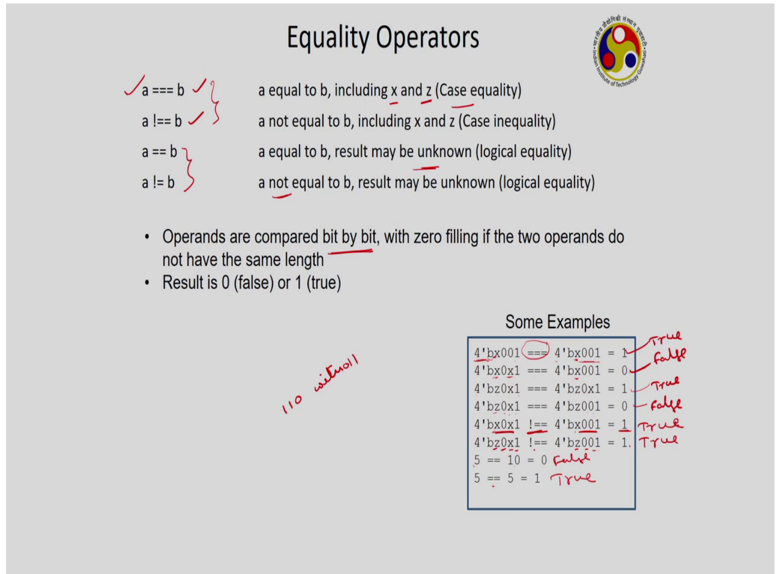
There are some relation operations. So, this, these are I mean similar to the conventional a less than b, a greater than b, and less than or equal to in conventional normally we will write this one, but in Verilog we will write less than and then is equal to side by side. Similarly, this is greater than or equal to ok.

So, since I have given some examples like a 5 less than or equal to 10, 5 less than or equal to 10 is this operation true or false because 5 is less than 10 ok, this is true. So, the value is 1. As I have told a logic one can be used for the true also ok. The second statement is false, this is false. So, the logic value is 0. As I have told 0 can be used for representing the false operation ok.

Then this is one bit as I have discussed in the number representation this is 1 bit binary do not care is less than or equal to 10. So, this is two-bit number. This is one-bit number you say do not care ok. So, this is again do not care because this operation we cannot discuss because this is do not care unless otherwise we know the value we cannot compare that value with the other value that is why the outcome of this one is do not care ok.

Similarly, if I use z also, we will get a do not care. So, here about some relation operations.

(Refer Slide Time: 25:27)



Then equality operation. So, a is equivalent to b will be represented by this. If a is not equal to b is represented like this. So, here this is case inequality here the, this include x and z also ok, whereas, here this x and z are not included. And this is x and z are not included ok.

So, two with x and z are included two with x and z are not included. Here also we are going to perform the comparison operation bit by bit, with zero filling if two operands do not have the same length. Suppose, if I want to I mean compare 110 with 11 so, we will add 0 here.

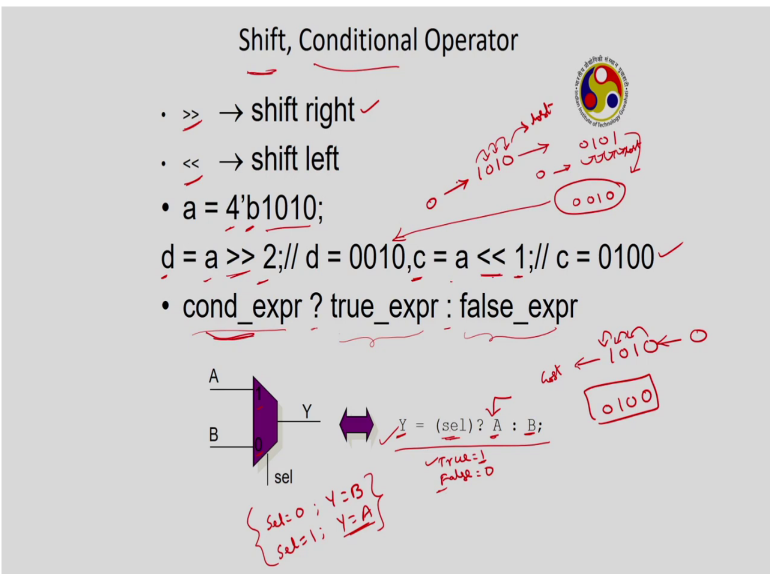
Here if I take these 4 bits, binary and this is do not care 0 0 1. This is 4 bits binary do not care 0 0 1; because we are used these three equalities this notation this will include x and z also because these two are same. This is also do not care, this is also do not care. So, this is giving true that is why logic 1 ok.

And the second one is this is x, two do not cares are there, whereas here we have only one do not care, so that is false. This is logic 0. Whereas, here this is 4 bit binary, z is there, z 0 x 1. This is also z 0 x 1, true. This is z 0 x 1, but this is z 0 0 0, so false. This is false, this is true. This is true, this is false.

And this is for not equality. So, this is x 0 x 1, this is x 0 0 1. So, these two are not equal including x and z. So, this is true. This is z 0 x 1, this is z 0 0 0 0 1. So, this is again not

equal, this is true. So, this is without z and x, 5 is equal to 10 – false; 5 is equal to 5 – true. So, these are some equality operators.

(Refer Slide Time: 28:09)



Then we have some shift and conditional operations. So, the shift right will be represented by two greater than symbols; shift left by two less than symbols ok. If I take a is equal to 4-bit binary 1010 which is 10. If I write d is equal to a right shift by 2 bits, you have to specify the number of bits ok.

So, what we will get? This is 1010 if we shift by 1 bit, this will become this 1 will go here, 0 will go here, this 1 will go here, and this 0 will be comes to the initial. So, this will come to this is 0101. And if I shift one more bit here, so 0's will be inserted here actually basically not this 0.

So, this will be lost and so many 0s will be inserted here. So, after the shift, so this bit will move here, this bit will move here, this bit will move here, this will be lost, and a 0 will be inserted here So, what will be the result? So, 0 0 1 0. So, this is how we can perform the shift right operation.

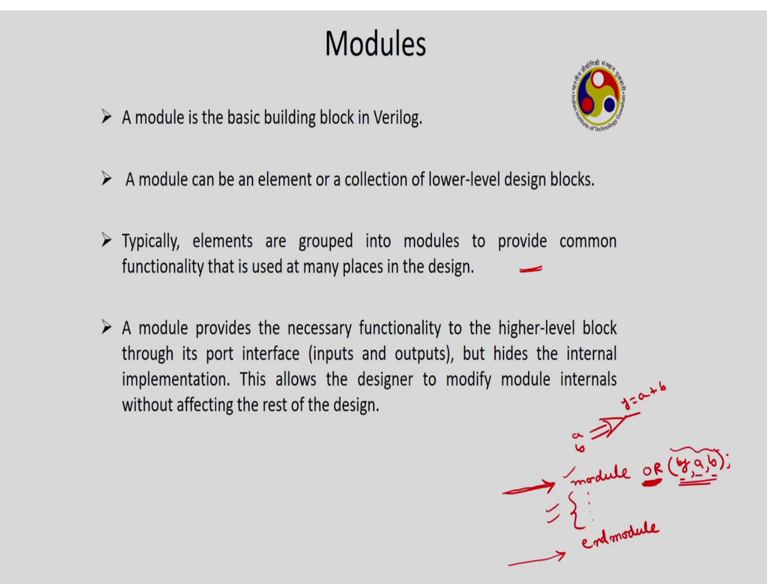
Similarly, shift left operation if I write c a less than 2 less than equal this is I mean shift left by one-bit position. So, shift left means 1010. Shift left means this will be lost, this will come

here, this will come here, this will go here this will be lost, and a 0s will be inserted here. So, what will be this one? 0 1 0 0, this is 0 1 0 0 ok. So, this is about shift operator.

Then we have some conditional operator also. So, we have to give some condition with question mark, we have to give here the expression for the true. After that we have to give the false expression ok. Like if I take the 2 by 1 multiplexer, so what is the operation here? If selection is 0, output Y will be the input that is connected to 0 will be B. If selection is 1, output will be the input that is connected to 1 signal which is A ok.

So, this can be represented using single statement this. Y, selection is the condition this is the condition if this is true, true means 1, false means 0. So, this true condition is A. If this is true means if this is equal to 1, output Y becomes A. If this is false means 0, output Y becomes B ok. This thing can be represented using single statement like this. This is what is called the conditional operator.

(Refer Slide Time: 31:43)



Then the basic building block of Verilog, if you want to write the code ok, the basic building block is module ok. So, a module can be an element or collection of the lower level design blocks ok. For example, if I want to write a simple program for a OR gate simple two input OR gate, you have to first define the module.

You have to give some name say OR this is optional, and you have to specify the inputs and outputs. Output is Y inputs are, a and b. So, this is the first statement that you have to write for the Verilog code. This is the Verilog code correspond to simple OR gate. After that you have to write inside this body you have to write the different statements. And the last one is end module here.

You have to describe your circuit ok. The starting of any Verilog code is module, and the ending of that one is end module. So, in the module, you have to specify all the inputs and outputs ok. So, typically the elements are grouped into the module to provide the common functionality.

So, this functionality will be explained here. So, the module provides the necessary functionality to the higher-level blocks through its port interfaces. So, these are the ports we have one output port, two input ports ok. So, this is the general structure of the module ok. This will be clear while writing the Verilog codes.

(Refer Slide Time: 33:25)

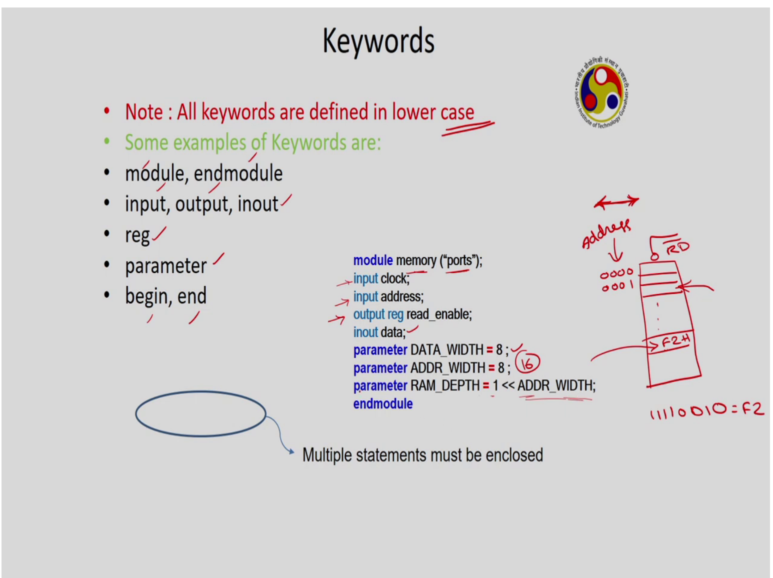
- Everything you write in Verilog must be inside a **module**
  exception: compiler directives
- In Verilog, a module is declared by the keyword **module.** A corresponding keyword **endmodule** must appear at the end of the module definition.
- Each module must have a ***module_name***, which is the identifier for the module, and a ***module_terminal_list***, which describes the input and output terminals of the module.
- **modules** cannot be nested.

So, continuation of these modules. Everything you write in Verilog must be inside the module as I have already described. So, in Verilog, a module is declared by the keyword module, and then you have end module ok. So, each module must have the module name which is of

course an optional one, and then we have to give terminal list as I have explained in the previous example.

So, here we have to write module some module name, and then the inputs and outputs. This is terminal list. So, but the modules cannot be nested this is one of the important property ok.

(Refer Slide Time: 34:17)



So, there are several keywords in the Verilog using which we will write the Verilog code ok. So, you note that all the keywords are defined in lowercase only this is one of the important points to be noted. So, there are some examples of the keywords like module is one keyword, end module, input, output, inout.

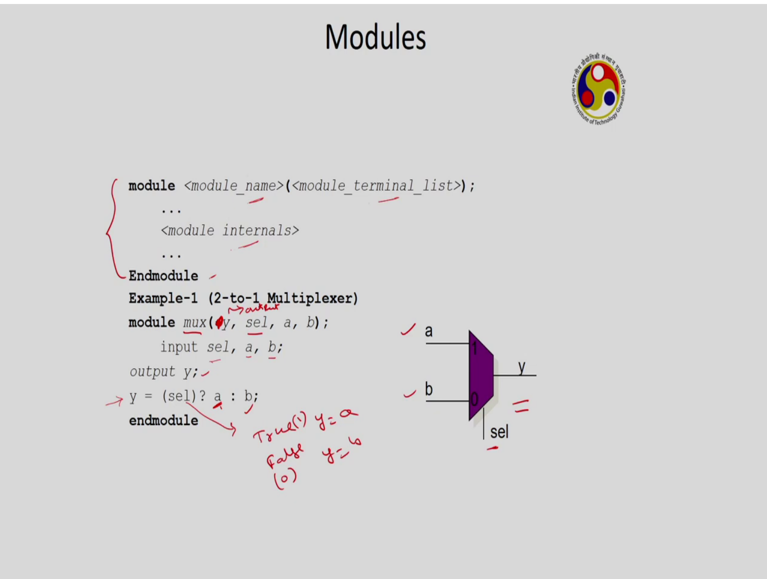
If I want to use a dual port which is which will access input as well as output port we will write inout, and register this reg stands for the register if we want to store some data, parameter, begin, end, these are some few keywords. So, the keywords, all the keywords will be understood while writing the modules or the Verilog codes ok.

So, here I have given a simple program like module for the memory we have to specify the ports. So, this memory will have some clock. So, we are inputting the clock; we are inputting the address also. So, in order to access this memory, we require the address. Each memory location will be having some addresses.

Here we have to specify the address this is some 0 0 0 0, 0 0 0 1 these are the addresses. Then we can have read enable if we want to read the data from this location, we should have read enable for this signal through read bar. And if you want to write, so you have to mention some data in any location I can write this data. Suppose, if I want to write the data into this one F2H hexadecimal ok.

So, what is the width of the data? This is 8 bits, I have given here 1 1 1 1 0 0 1 0, this is F 2 ok. So, this data width is 8. And what is the width of the address? Of course, in this example I have given as a 16 ok, depth will be one-bit shifted version by address width ok. This is some example of a module.

(Refer Slide Time: 36:40)



There are some other examples ok. So, this is the general structure of the module as I have told. So, module, module name, module terminal list, then the internals, then endmodule. If I take a simple 2 by 1 multiplexer which I have discussed earlier ok, so I am giving the name module name as mux, output is y, this is not there, output is y, then the inputs are one selection line, a, b.

So, first we have to mention this output. These are the input. Later we can explain in detail inputs are selection a b, and then output is y. And as I have told the operation can be explained using a single conditional operator which is y is equal to select a, b.

As I have told if this is true, this value will be the output, output y is equal to the first value. If it is false means 0, true means 1, output y will be the second value b ok. This is the main example of a Verilog code for 2 by 1 multiplexer.

Thank you.