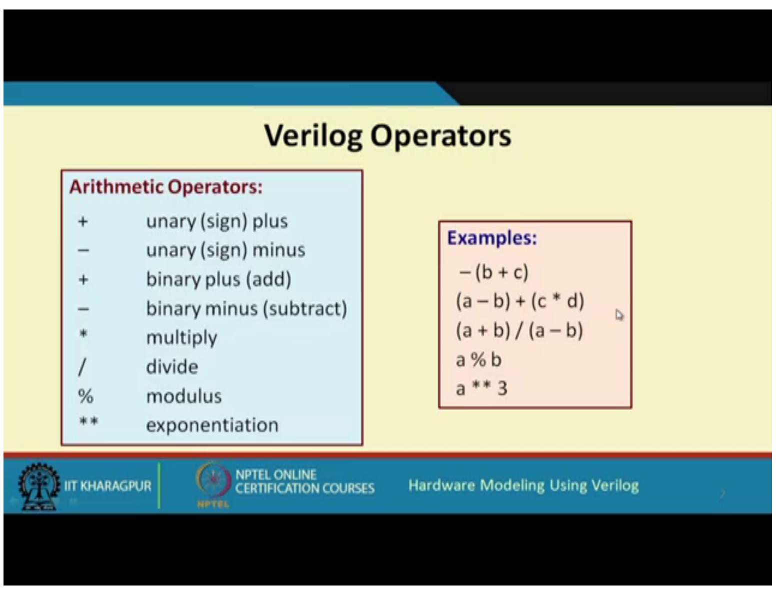
Hardware Modeling using Verilog
Prof. Indranil Senagupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 09
Verilog Operators

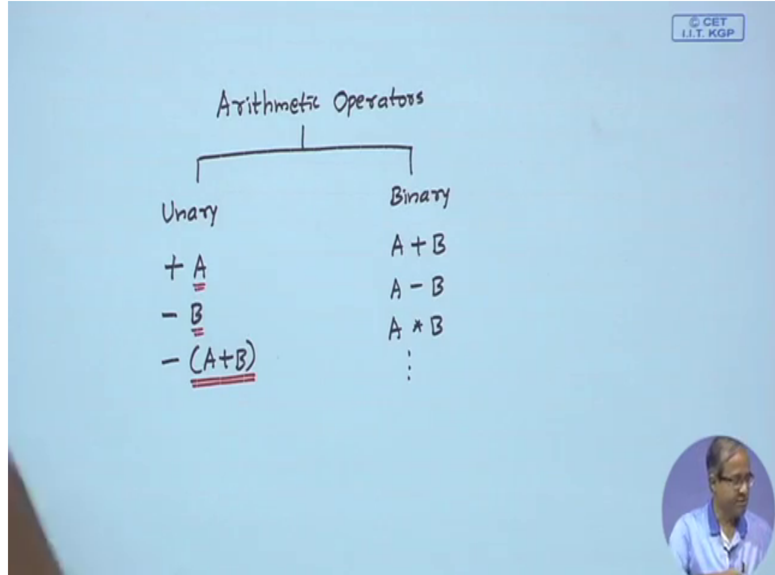
So, we continue with our discussion on the various features of the language verilog. So, in this lecture, we shall be discussing about the various verilog operators. Like in any programming language well you can take as an analogy C, C plus plus or java there are some operators which are built in the language which can operate on some data items, some numbers, some characters some values and they produce some results. So, in the same way in verilog, there are A set of operators as if you see some of them are similar to the operators which are available in high level languages like C or C plus plus. But there are A few others which are very specific, and they have some direct connection with the hardware that verilog is supposed to model.

(Refer Slide Time: 01:27)



So, we start with this lecture on verilog operators. So, we start with the most basic kind of operators which we see in any language. The operators that allow us to carry out some arithmetic, these are the so called arithmetic operators.

(Refer Slide Time: 01:50)



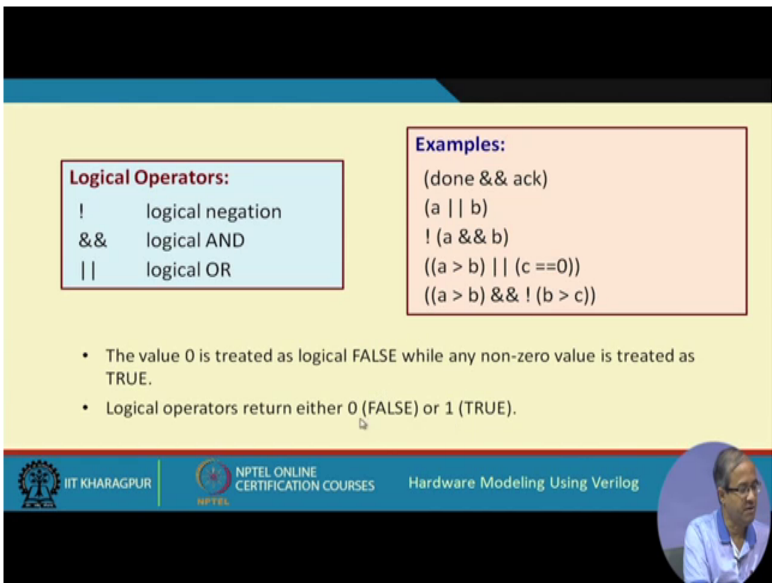
Now, broadly speaking if you are just think of the arithmetic operators, so these operators can be broadly classified into either unary or binary; Unary means this kind of operator will be operating on A single operand or data item binary means it will be operating on two data items. For some examples of unary are you are quite familiar with this kind of notation, you can write plus A or you can write minus B, these are something which is quite valid expression, we can write minus within parenthesis, let say A plus B. So, these minus and this plus these are unary operations, they are working on A single data item.

So, in the first case, this is A, here this is B, and here this is just whole thing this minus is operating on this whole thing. And in contrast the binary operators, they operate on two data items like A plus B when you write. So, this plus is operating on A plus B, A minus B, A multiplied B and so on there are several others. Let see. So, in verilog the operators that are supported are as follows. So, you have two unary operators one is plus, one is minus. And among the binary operators some of them I have already mentioned, we have addition, subtraction, multiplication, division, modulus, modulus means you divide by the second opponent and take the reminder, and double star means exponentiations. Some of the examples are shown here.

This is an example of unary operator minus, while this plus is a binary operator. Here minus star and plus, these are all binary operator. This minus is working on a and b; star

is working on c and d; and this plus is working on this first number and the second number. Similarly, this slash, mod exponentiations this means a to the power three these are the so called verilog operators. And as you see that the verilog operators are somewhat very similar to what you see in the other programming languages like C with the exception of the exponentiation operator the others are that means, all most the same as those available in the language C or C plus plus fine.

(Refer Slide Time: 04:55)

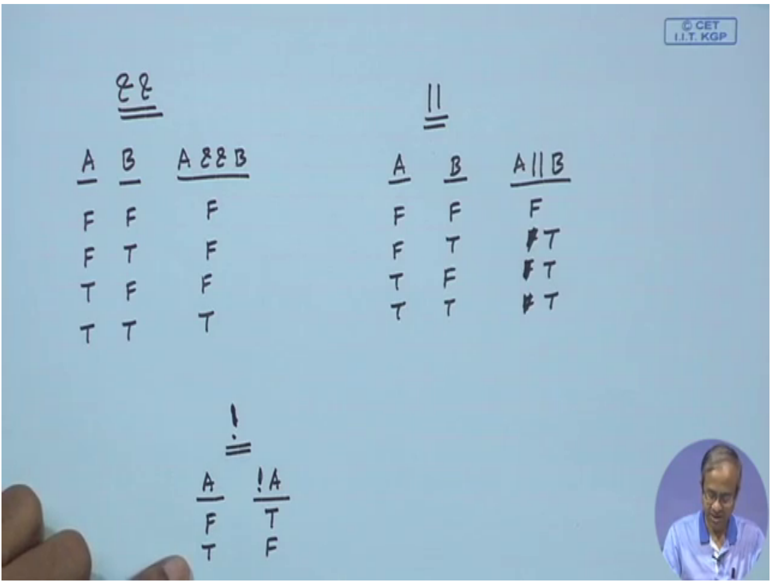


So, let us move on. So, we now look at logical operators. Well, when we talk about arithmetic operators, we are operating on some numbers, numbers can have some values, which we can add, subtract, multiply, divide, and so on. When you talk about logic operators, we are considering logic values true or false, and we can combine several such logic values into a single result, which is again a logic value true or false these are the so called logic operations or logical operations. So, in verilog, there are three kinds of logical operators that are supported negation which is not logical AND which is denoted by double ampersand and logical OR which is denoted by double bar. These are again very similar to the language C or C plus plus such things are available there also.

So, these operators are used in some conditionals while here I am not shown the complete examples, I have only shown the logical expressions, but actually we will be using this in some kind of control constructs like let say and if then else statement. So, you are aware of if then else kind of statements. So, when I specify A condition in an if

statement, so I can use this kind of logical operators in that condition expression. So, some examples of condition expressions are shown here done and ack. So, here it is assumed that done and ack are both logical expressions or of values true or false, and you are defining the logical.

(Refer Slide Time: 06:59)



And now the way this logical and this defined it is just follows the AND operator. Like say if you have two operators, two opponents A and B and the output which is A AND AND B then if both are false this will be false; if one of them is false then also it will be false; only when both of them are true, then only this expression will be true. So, this is how this AND operation is defined. Similarly you can have this or A or B where A is a logical expression, B is another logical expression which has again values true and false and they are combined with an OR.

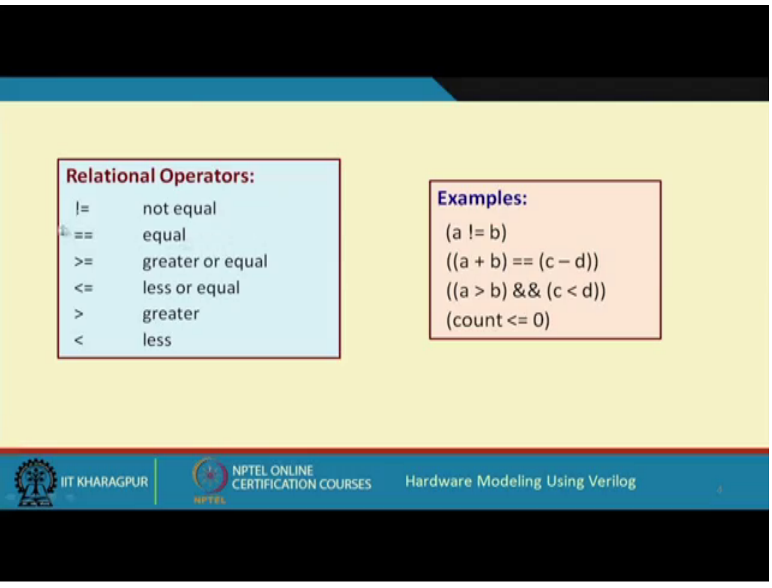
So, just to recall the OR operations will be defined as follows. So, when both of them are false, then the OR operation is false, but if at least to one of them is true, false true, true false, or true true then and then expression sorry this will be true this expression will be returning true, this is the definition of OR. So, if at least one of the inputs are true, then the expression will be true. And the NOT operation is just the logical negations. So, you have single input and you have NOT of A; if A is false, NOT A is true; if A is true, NOT A is false right, these are the three logical operators which are supported by verilog.

So, here you can see some other examples are given of logical expressions NOT a AND AND b. So, here this a and b will be done first then NOT of that. Here there are two expressions I means a greater than b, this will be either true or false; c equal to 0, this will be a true or false, and you are combining them using this OR. Similarly here there are two a greater then b, b greater than c, so you are first doing NOT of this, then you are combining these two using AND.

So, the point to notice that so while you are evaluating such expression like when I have written done AND AND ack, the values of this two variable done and ack, if the value is zero, then it will be treated as false, but any nonzero value will be treated as true. And as it said the logical operator will return the value again either true or false, but the values will be zero for false and one for true that is the convention which is followed in this kind of logical operations fine.

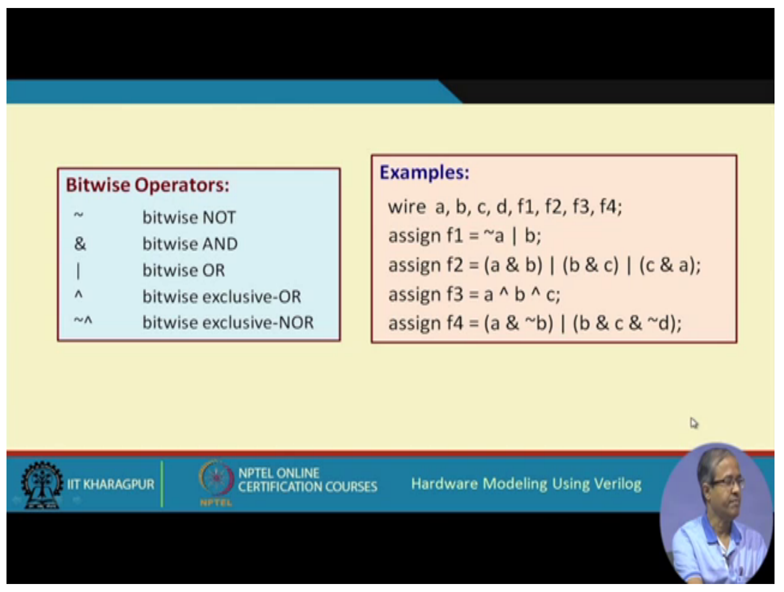
Now, let us come to the relational operators these are again a very standard feature in any programming language, we often have to compare two numbers to check whether they are greater, less than, equal and so on. So, these relational operators are used for that purpose. So, they will be typically taking two numbers as input they will be doing some kind of relational operation; and the result that they will be giving you will be true or false.

(Refer Slide Time: 10:43)



Let us see. So, in verilog six such relational operators are supported not equal, double equal to sign means equal, greater than or equal less than or equal, greater than, less than. So, some relational expressions are as follows a not equal to b, a plus b equal to c minus d, a greater than b and c less than d count less than zero. Well, these expressions you can use in some control constructs which will be studying later like if then else or while or for these kind of control constructs. So, here has I said, so typical relational operator let say a not equal to b they operator numbers. So, just assume that both a and b are numbers, and the result that is generated it will say whether this condition is true or false. So, it is a Boolean value right, these are true for relational operators.

(Refer Slide Time: 11:50)

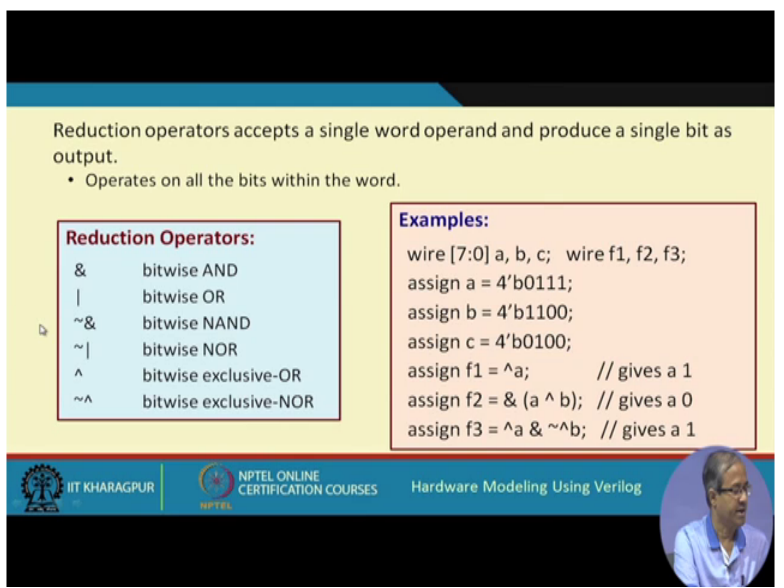


Next, they are some bitwise operator. Again these bitwise operators not exactly all of them, some of them at least are available in a language like C. So, verilog is supposed to modular hardware and at the level of hardware we talk about bits we talk about the logic operations. So, these bitwise operations are very important in verilog, because at times when we just model some logic expressions at the bit level, we use this kind of bit level operators. So, the bitwise operators that are supported are bitwise NOT is denoted by this tilde symbol. Single ampersand means bitwise AND, single bar is bitwise OR, hat is exclusive OR; and tilde hat is exclusive NOR.

So, we have seen some examples earlier where you use the assign statements to model some combinational functions. So, here again I am showing some examples where some

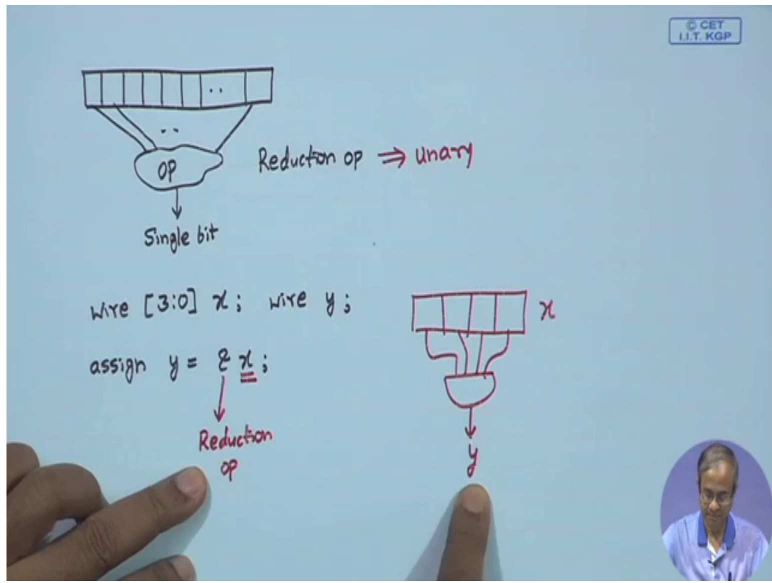
of this operators are used. You see we have defined some variables a, b, c, d, f 1, f 2, f 3, f 4 as wires. Let us in a first statement, we have said assigned f 1 equal to not a or b you see these are all bitwise operators. First the value of a is inverted not of a then that value is bitwise or with the value of b. Similarly, the second statement, which is actually similar to the evaluation of the carry of a full adder, so you take the end of a and b, b and c, c and A and then or them together. f 3 is take the exclusive or of three variables a, b, c, a XOR b XOR c; and f 4 says a AND NOT b or b and c AND NOT d. So, in terms of Boolean expression, this is similar to a b bar or plus b c d bar. So, bitwise operators as it said, the operates and bits all the operators are single bit variables, they are not vectors and the return a value f 1, f 2, f 3, f 4, those are also of type bit, single bit, these are something to remember.

(Refer Slide Time: 14:34)



So, here we have something which is normally not present in the kind of hardware or the high level languages with the familiar with like C or C plus plus, these are called reduction operators. Now, what is a reduction operator reduction operator is trying to reduce set of items into a single item, this is from for the name reduction comes.

(Refer Slide Time: 15:12)



So, let me give you an example. Suppose I have a word, you can treat it as a vector, multi bit word. So, I just operate on this bits using one of the reduction operators, this is a reduction operator. So, what the reduction operator will do, it will take all the bits of my input word as input, this is that are multiple bits, and it will be generating a single bit result as the output, this is what is the purpose of a deduction operator. It converts multiple bits in a variable into a single bit of result.

Let us see. So, this is actually what we have just said it accepts a single word as input and it will produce a single bit as output. So, it operates on all the bits of the word. So, what are the kind of reduction operators available, single AND is bitwise AND, this is bitwise OR, tilde and is NAND, tilde bar is NOR, this is XOR, tilde hat is exclusive NOR. Well, how we just use this kind of reduction operator it is simple. Suppose, I have a variable let us say I have declared a variable it is of type wire let say for bits I call it x. So, I can write somewhere let say I have defined another variable a single bit variable called y, I can write y equal to ampersand x, this reduction operators are all these are unary operators, this is something to also remember, they operate on a single data item or operant. So, this ampersand works on a single data item x and, so this is your reduction operator reduction.

So, what this reduction operators doing, it is taking the bitwise and of all the four bits of x, this x was four bits. So, actually logically speaking this will be like AND gate, this is

your x, and the output of the AND gate will be your y. So, this reduction operator actually models a gate with multiple number of inputs. This ampersand x means this is an AND gate, it will take all the bits of this variable x as input, and it will generate this y as the output. So, depending on the kind of operator we have used, so the type of this gate will differ; like as I said we can have so many different types of gates.

We will have some examples are shown here. So, you see. So, here we have defined three words a b c these are vectors of size 8 7 to 0. and we have also declared three single bit variables of type wires f 1, f 2, f 3. And a, b, cwe have just initialized them with some value. Well, here just take an assume for this example at least that this instead of 7, you can consider as 3. So, there is no harm in that is what me easier for your understand. So, let us assume these are four bit numbers. So, I am initializing a, b, c with this four bit constants 0111, 1100, 0100.

(Refer Slide Time: 19:26)

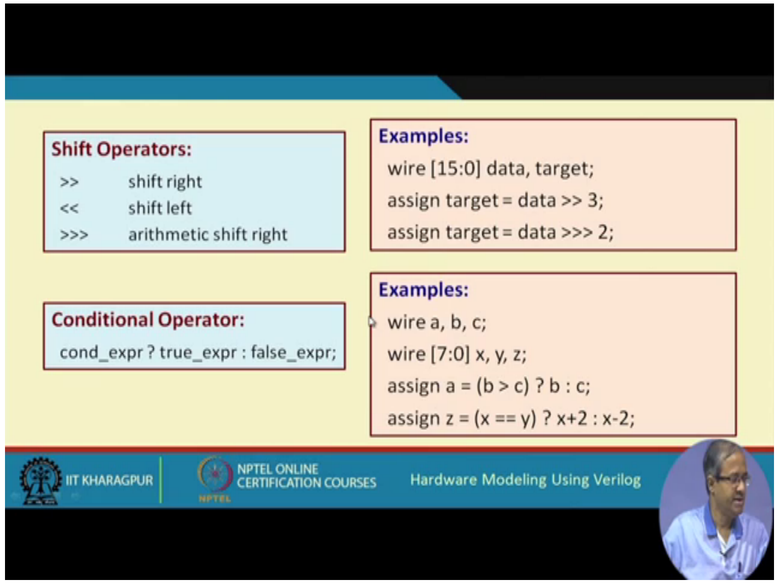
So, this initial values are a is 0111, b is 1100 and c is 0100. Now, the first reduction operator I am using here in this example this is a f 1 equal to XOR a, which means it will do a bit by bit XOR of all the bits of a and it will generate f 1. So, what will be the f 1? So, f 1 will actually the bit by bit XOR of these four bits. So, it will be 0 XOR 1 XOR 1 XOR 1. So, you know XOR actually counts the number of ones where it is odd or even. Here since it is odd number of ones the result of the XOR will be 1. Similarly here f two

is actually doing a XOR b and end of that; that means, this is your reduction operator first a and b you are taking an XOR.

So, let us see a is this, b is this. So, a was 0111, b was 1100 you take an XOR first XOR operation. So, 1 and 0 is 1, 1 and 0 is 1, 1 1 is 0 0, and 1 is 1. This is are XOR and then you are using a reduction operator ampersand you are taking and of all this four bits 1011, because at least one zero is there and operational give you a result zero right. So, your f 2 will be 0. The third example says you do a reduction operate on a, then you do and on another reduction operate on b. So, this is XOR and this is XNOR. So, XOR on a then XNOR on B and of that this is a logical operator, let us see this is bitwise logical operator.

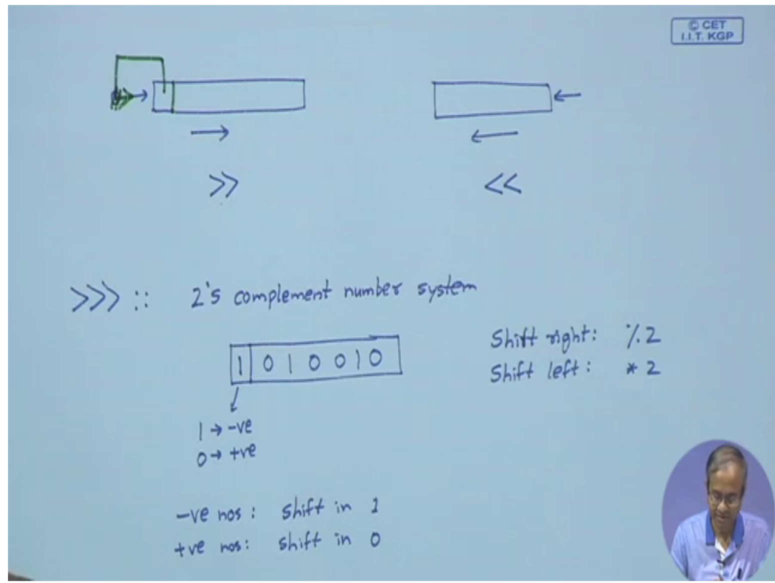
First let us do this XOR of a. So, XOR of a, what will be the value. So, already you have done it bit by bit XOR of a, it will be 1, and XNOR of b. So, you do XNOR. So, there is even number of ones in b. So, for even number of ones XNOR becomes one. So, you take and of these two XOR a and XNOR b, so 1 and 1, so result will be one. So, here f 3 will be one. So, this example as we show you how this reduction operators work.

(Refer Slide Time: 22:39)



Next comes shift and conditional operators. Now, the first two kind of shift operators are available in a language like C, which simply specifies right shift or left shift. So, here let us see the example I have given, we have define two variables data and target of vector of size 16. So, if I say data right shift by some number three, it means you shift it right.

(Refer Slide Time: 23:18)



So, when I have a data in a register in a vector, I shift it right by three positions, and when I shift it right zeros will be inserted on the left. Similarly, when we shift it left, then zeros will be inserted on the right. So, this is the shift right operator and the shift left operator. So, I can specify by how many bits I am shifting right. So, I can shift it here in the example I am sure and I am shifting by three positions. But there is a special version of right shift which is specified by three greater than symbols, three greater than means arithmetic right shift.

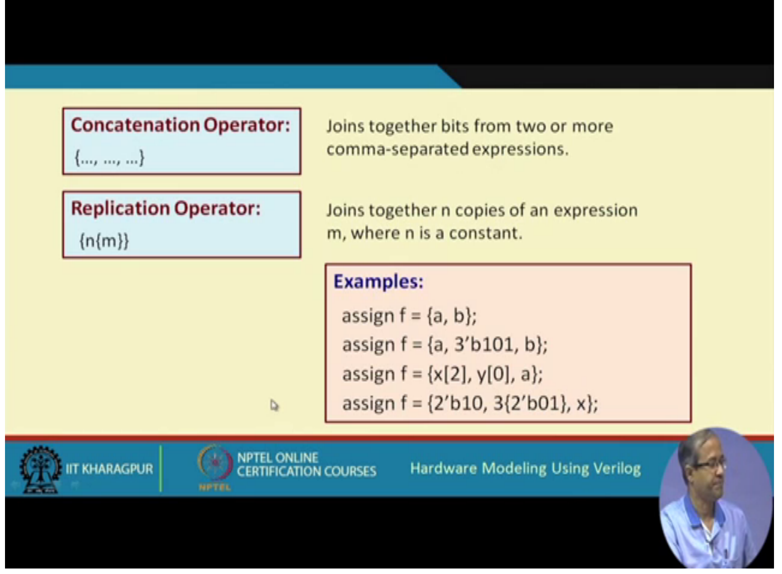
So, what does arithmetic right shift means, well in arithmetic right shift in order to understand its basic purpose we need to understand the 2's complement number system very clearly. So, just to refresh your memory, so here in the 2's complement number system, whenever you represent your number the most significant bit will indicate the sign one means negative zero means positive. So, the other bits will be something let us say the other bits will be something. Now, for a 2's compliment number for any number, in fact, if you do a shift right, shift right is actually equivalent to division by 2, shift right by one position; and shift left means multiplied by 2.

Now, for a 2's complement number, if you want implement shift right, the rule is that if the number is negative for negative numbers, you shift in 1; and for positive numbers you shift in 0. So, you see to implement such this is called arithmetic shift because we are doing some arithmetic operations here. To implement this, what we have said is there

you see for a normal right shift we said that always 0 will be inserted, but now I am making some changes here we are saying not 0, but whatever was this sign that same sign bit will be inserted. If it was negative, 1 will be inserted; if it was positive, 0 will be inserted right, so this is denoted by three greater than. So, here you have an example data arithmetic shift by two positions.

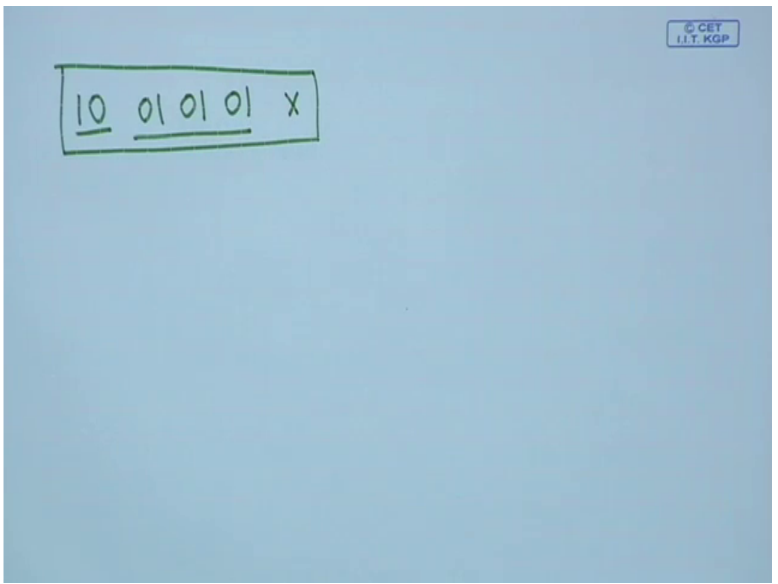
Well, here exactly this thing is available in a language like C you have a conditional operator. So, we specify a conditional expression then a question mark then an expression, colon, another expression and finally, semi colon. The first expression here will mean that if the condition is true then you take this if the condition is false then take this. So, some examples are shown here let say wire seven zero x, y, z these are eight bits. So, I have written some condition b greater than c. So, I have said if b is greater than c, the true expression is this then b else c, you assign this to a. What does this mean this means you assign the greater of b and c to the variable a. If b is greater than this b will be assigned; if b smaller it will be false then c will be assigned. Similarly, here another example if this lets say x and y, if these are equal, this is the condition then you take x plus 2 else you take x minus 2 and assign it to another variable z, well. This is just example this a, b, c, x, y, z you can have anything right. So, this kind of operator is also available.

(Refer Slide Time: 28:02)



And you have something called concatenation operation where just I am showing the example within curly brackets I can specify two or more items separable commas like a comma b, a comma 3 bit 101 comma b. This will mean all these three items in all these two items will be concatenated together, they will be concatenated together to form a single vector. Suppose a was a 4 bit vector, b was a three bit vector, if I write concatenate a comma b, it will become a seven bit vector, a and b will be joined together. And we can use this replication operator along it concatenation where I can use a constant n followed by something within curly bracket, this will mean repeat n times like the last example shows this is an example where I am concatenating three things first is two bits one 0, second is three times 2 bit 0 1.

(Refer Slide Time: 29:19)



So, what would you mean you start with one zero this is a first one, then I am saying three times zero one which means 01, 01, 01. And the last item is a undefined x. So, there will be in x in the end. So, my final in concatenated result will be this 10, 01 01 01 x, it will be a nine bit vector.

(Refer Slide Time: 29:46)

```
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
```

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

So, let us see some examples very quickly. So, here there is an example shows some of the operators in use. This is a module complete module description x and y are inputs; outputs are f 1 and f 2, x and y are defined as 10 bit vectors. So, f 1 is a 5 bit vector; f 2 is a single bit vector. So, here you see here I have taken cross sections from the vector x and y from bit number four to zero to make it 5 bits, I have done bit by bit AND, and I have assigned it to f 2 because the size of f 1 was 5. F 2 is the single bit. So, I can take any single bit from x any single bit from some other let say f 1 three I do or not then I do a bit by bit wise OR. This x, I do a reduction operation NAND I assign it to f 2, this is the conditional. Well, I am assigning something to f 1 is a 5 bit; if f 2 is true then these 5-bit is assign x bit number 5, 6, 7, 8, 9 this will be assigned. If f 2 false then these 5-bit 0, 1, 2, 3, 4 these will be assigned, here are some examples.

(Refer Slide Time: 31:11)

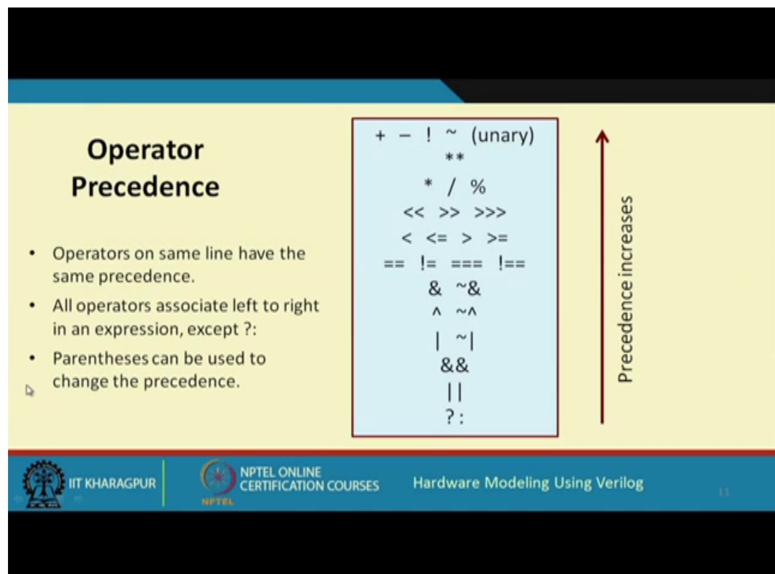
```
// An 8-bit adder description
module parallel_adder (sum, cout, in1, in2, cin);
   input  [7:0] in1, in2;
   input  cin;
   output [7:0] sum;
   output cout;

   assign #20 {cout,sum} = in1 + in2 + cin;
endmodule
```

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, here we are showing the behavioral description of an eight bit adder. So, here we are assuming that the number input numbers are in 1 and in 2, cin is the carry in, and this is the sum and this is the carry out. So, because it is an eight bit adder this in 1 and in 2 will be 8-bits. This C in is the carry in, sum is 8-bits, and C out is carry out. So, we can express this in behaviorally in a single statement like this, but of course, the edition will be this in 1 plus in 2 plus carry in, but you see when you are adding two 8-bit numbers, the result can be 9-bits because that can be one bit of carry coming out.

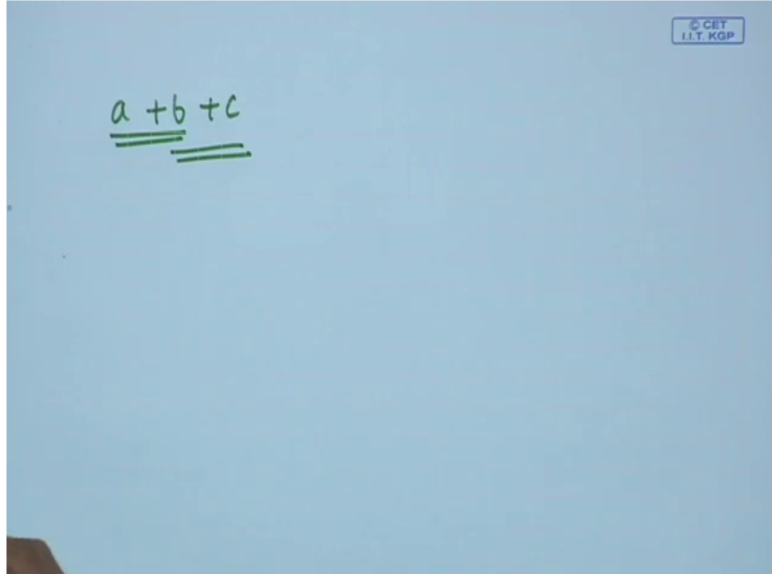
So, if you want to keep everything of the result intact, it will be one bit more. So, how have I how I specified it here I have just written C out comma sum concatenation which means the left hand side is a concatenated item which represents 9-bits. So, after this addition, the most significant bit will going to c out the remaining 8-bits will go in two sum that will be the result. And this is just a delay that can be used for simulation.

(Refer Slide Time: 32:31)



And just we talked about so many operators. Just talking about the operator precedence, this chart actually shows you just overall picture how the precedence varies. So, this is low priority, and this is highest priority, and operators on the same line have the same precedence. And excepting the last operator the conditional operator, so all other operators they are left to right associative.

(Refer Slide Time: 33:07)

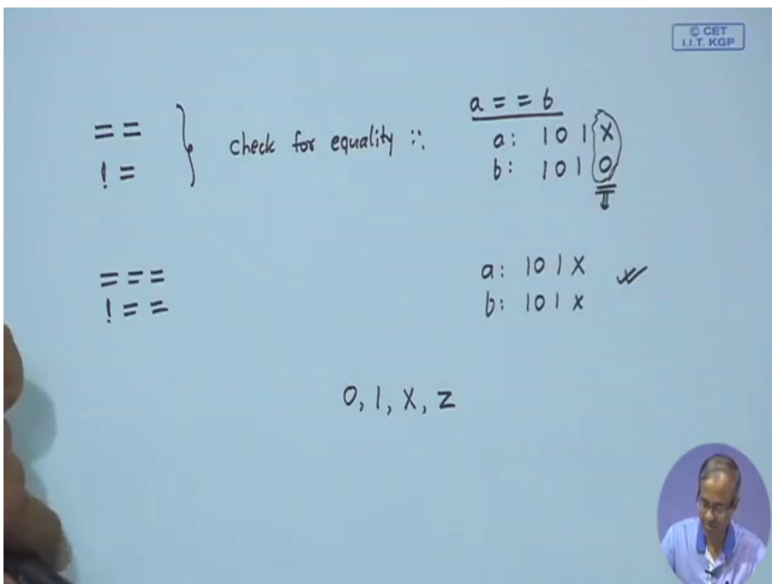


Means suppose I write a plus b plus c, this will mean first this left most edition will be done then the next one will be done left to right. Now, you see the highest priority are

unary operators plus minus NOT, NOT there are two different notations either this or tilde, then are the arithmetic operations. So, among this arithmetic operations we have exponentiation in the highest, then comes the other operators multiply, divide, mod. Well here are not shown then will come addition subtraction will be there. So, I have not shown all the operators in fact.

And then the shift operators, then the relational operators and relation operator less than greater than are the higher priority, and equal to, not equal to checking will be lower priority. Then comes the bitwise operators, and this highest priority, then XOR, then OR. This is what the priority values which verilog assumes then the logical operators, double ampersand, double OR and lastly the conditional. Now, here you see for equality checking, I have talked about double equal to not equal to; here you see there is another type of equal to triple equal to and not equal to.

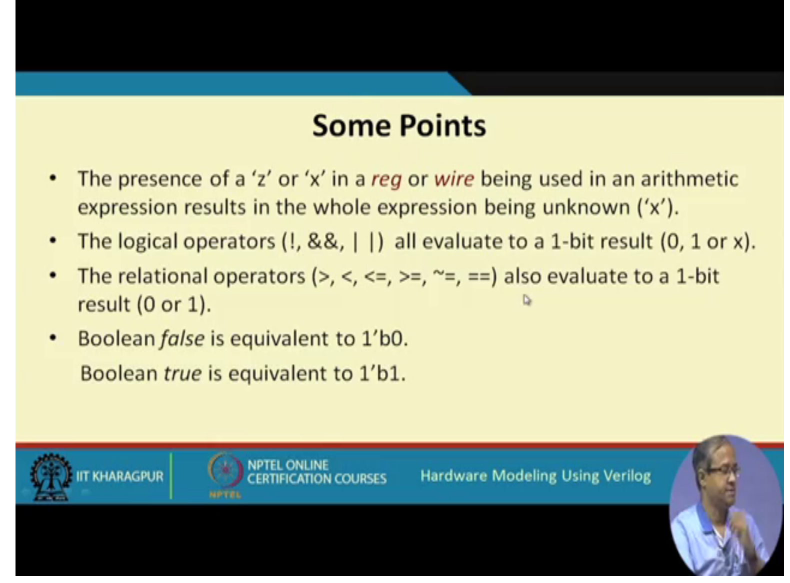
(Refer Slide Time: 34:45)



So, let us very quickly talk about this thing. So, so you have equal to and not equal to. So, we are saying there is another version where we can write it like this. Now, when you specify this, this means you check for equality falling a rule like this, suppose I am checking two numbers a and b, let say a is the current values 101 undefined x, b is 1010. So, if you make the comparison then you see this x and 0 will be compared, and x and 0 this will be considered to be matched.

But when you are using three equal to then this means exact equality. So, if a is 101x and B is also 101x then only this equality will be returning true you recall that in verilog we have a four valued logic system where the logic value supported at zero one do not care or undefined and high impedance z. So, if you write triple equal to then this values are matched in an exact way x and x, z and z, 0 and 0, one and one, but if it is double equal then it is not done that way this is just the only difference.

(Refer Slide Time: 36:35)



So, these are few things which have already mentioned most of them. And the first point is a little different that see when you are using an arithmetic expression. Let say we are using a and b, two numbers we are adding a plus b, let say a is a vector, b is also a vector. Suppose for one of the number, let say a one of the bits is having a state x. So, when you do the addition the entire sum will become x, this is a rule of arithmetic operation. So, the presence of either an x or a z in any variable which is used an arithmetic expression will result in the value of the whole expression to become unknown x. So, all the bits will become x and these we have mentioned logical operators generate one bit result, relation operators also generate one bit result; false is 0, true is 1.

So, with this we come to the end of this lecture where we have try to summaries the various kind of operators which are available in the verilog language with which we can specify the description or the model of the hardware system that we are trying to design

or build. So, in the next lecture, we shall looking at some of the examples based on the structures and the constructs which we have seen so far.

Thank you.