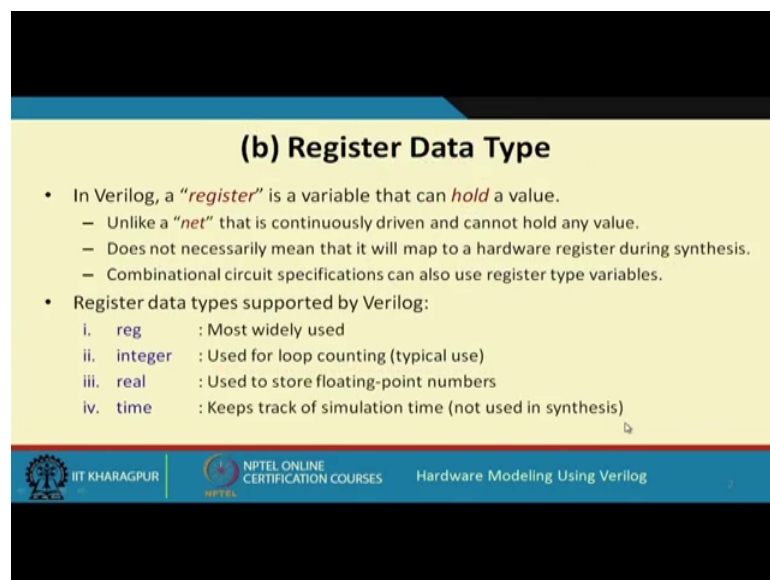


**Hardware Modeling using Verilog**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 07**  
**Verilog Language Features (Part 2)**

So, in the last lecture, if you recall, we were talking about some of the features of the Verilog language in particular, we are discussing the net type variable. So, what are the different types of net type variables; now we continue our discussion here in this lecture.

(Refer Slide Time: 00:40)



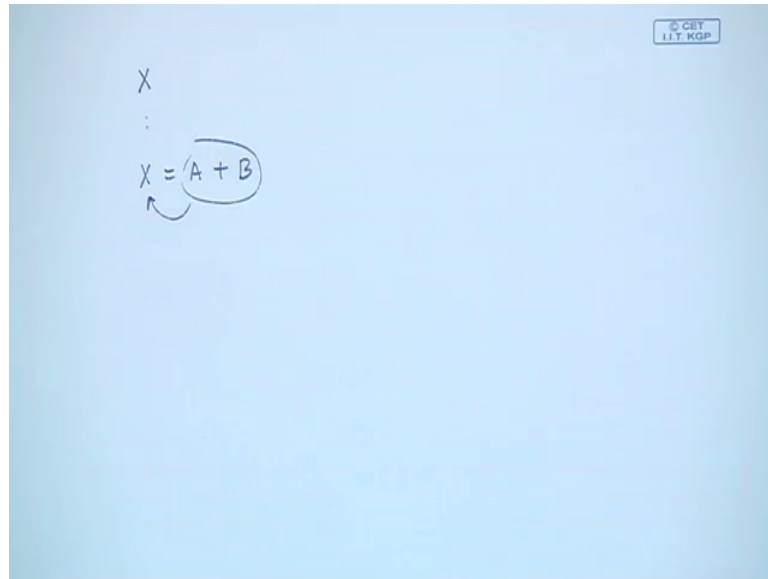
**(b) Register Data Type**

- 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:
  - i. `reg` : Most widely used
  - ii. `integer` : Used for loop counting (typical use)
  - iii. `real` : Used to store floating-point numbers
  - iv. `time` : Keeps track of simulation time (not used in synthesis)

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

So, here we shall be first talking about the other data type that we talked about in addition to net, the register data type. Now register data type by its very semantic, it means that it is a variable which can hold a value like suppose I have defined a register X, then I can write a statement like X equal to a something let X equal to A plus B.

(Refer Slide Time: 01:13)



So, I can write something; something will get assigned to X. Now this assignment will not be done using an assign statement; not continuous assignment because I told in the last lecture that in an assign statement the left hand side must be a net type variable it cannot be a register type variable.

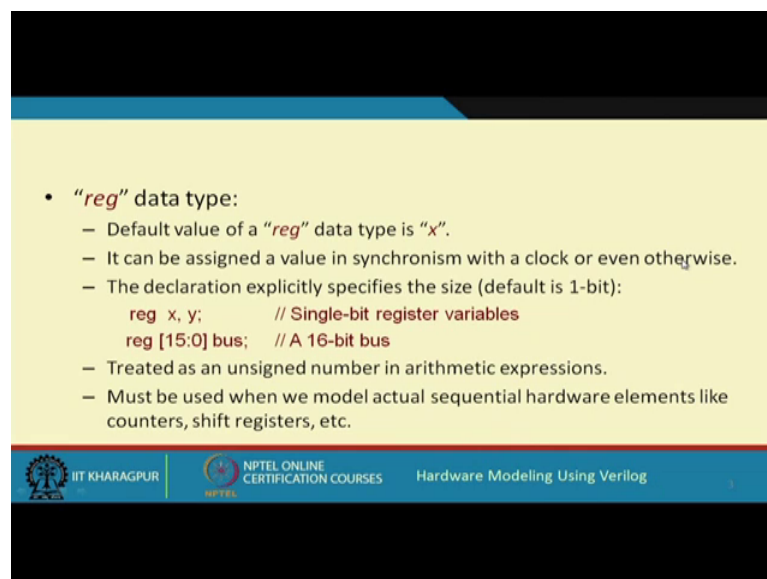
So, whenever you use a register type variable there you are actually storing a value, it is supposed to hold the value till you are using it again. So, a register is a variable that can hold a value and this is different from a net which is continuously driven and cannot hold any value because if the line that is driving the net changes. The net variable immediately changes, it cannot hold it, but I also mentioned that this does not mean that a register variable will always map to a hardware register. This may or may not be true, there will be some cases which we shall be illustrating with examples where you may be using a register variable in a Verilog code, but ultimately you are circuit that we synthesized will be a combinational circuit, right.

So, the following register data types are supported in Verilog, the most widely used is something called REG, this is short form for register, then you have integer real and time integer is typically used for application where you want to count something like number of times you are looping and so on and real as the name implies this is used to store some floating point number with fractional parts and time is a special variable which is used to keep track of simulation time.

Now, you say let me just emphasize one thing here that we shall be talking about many features of Verilog which are simulation only features that make sense only when you are doing simulation, when you are carrying out synthesis, those features does not mean anything and the synthesis tool will be simply ignoring them.

Let me take some examples, see whenever we instantiate a gate, we had seen some examples where we had specified some delays of these gates, right, like you can use using that hash symbol, I can right hash 5, hash 2 hash 1, but the synthesis tool will be ignoring that those will be only for simulation purposes because in during synthesis whenever an and gate is actually implemented. So, whatever is the actual delay of the and gate that will be the delay, right not 1 or 2 or 3; whatever you are saying similarly this time this time is something that relates to the time of simulation this is never used during synthesis such variables will be ignored during synthesis.

(Refer Slide Time: 04:41)



• “*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):  

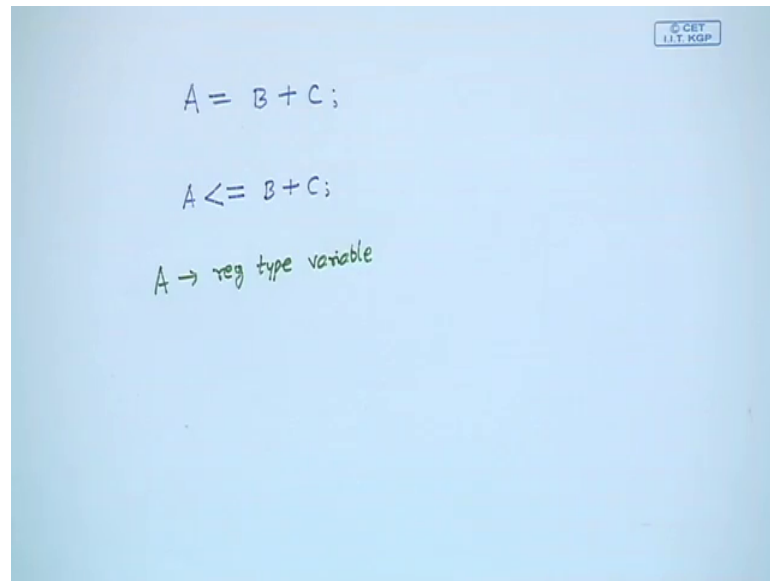
```
reg x, y; // Single-bit register variables
```

```
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

Let us start with reg, the first thing is that in the reg data type. So, just after you declare, but you have not initialized it to any value. So, the default value will start with undefined or x a reg value can be assigned in several different ways, we shall be seeing that they can be assigned in synchronism with a clock or even without a clock you can do it.

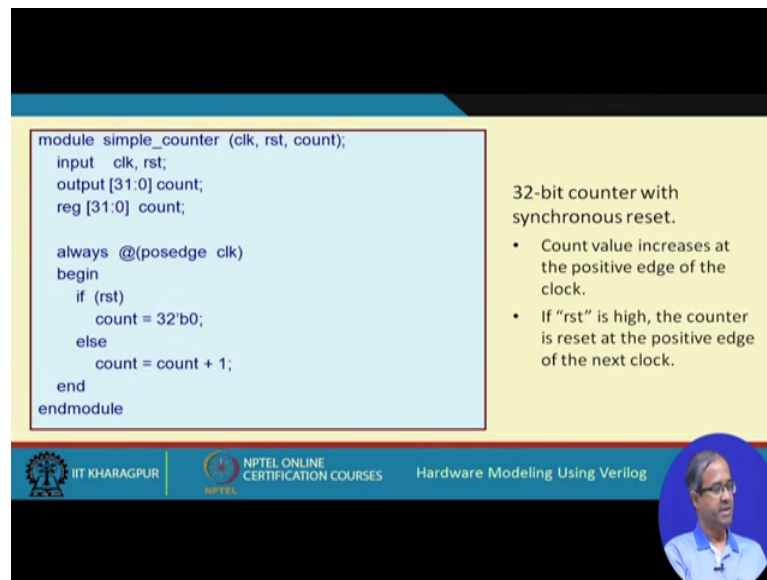
(Refer Slide Time: 05:14)



Now, there are 2 kinds of assignments, we should be learning like for example, a reg type variable; a, you can either write an assignment like this or you can write an assignment like this less than equal to. So, their meanings are different, we shall be explaining these things later, but the point to notice that here A is a reg type variable. So, for these kind of assignments the left hand side must be a reg type variable, right.

So, some of the example declarations are shown here reg x y, well if you simply specify the name of some variable like here x and y they will be by default taken to be one bit variables single bit variables. So, x is a 1 bit variable, y is a 1 bit variable, well you can also define a vector, we shall be talking about vector separately again this is one example reg within square bracket 15 colon 0 bus, this means I am declaring a variable bus where there are 16 bits 0 is the least significant bit, 15 is the most significant bit. So, I can define a 16 bit bus variable called bus using a statement like this. Now a reg type variable when it is used in an arithmetic expression, it is treated as an unsigned number without any sign and another point is that when you are actually implementing some hardware register base design like counter shift register etcetera then you must use reg, there is no other alternative.

(Refer Slide Time: 07:14)




```
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



So, here there is one example which uses reg, of course, here there are many constructs which I have not discussed yet, but this example you can appreciate what it is here we are trying to implement a counter, there are 3 ports clock reset and count clock and reset or the input and count in the is the output as you can see count is a 32 bit vector 31 0; that means, this is a 32 bit counter, well, ignore this reg for the time being and this is the main body of the counter description, there is a statement called always at positive edge of the clock. So, in English, it means this always whenever there is a positive edge of the clock coming, you do this what you do you check, if the reset signal is one on high or not if it is high you initialize count value 32 B 0 means it is a 32 bit number value is 0. So, you are initializing it to 0.

If reset is one count is 0, else count equal to count plus 1; you do this every time clock edge, positive edge is coming. Now as I said such expression with equal to. So, these are examples where you are using that, but you can have only a reg type value in the left hand side that is why I have to separately declare the same variable count which was in the output also as a reg because if you do not do this your compiler will give you an error that left hand side variable type is wrong, right. So, here the count value is increasing at the positive edge of the clock and if reset is high the counter will be reset whenever next clock comes, it will reset to 0.

(Refer Slide Time: 09:26)


```
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.

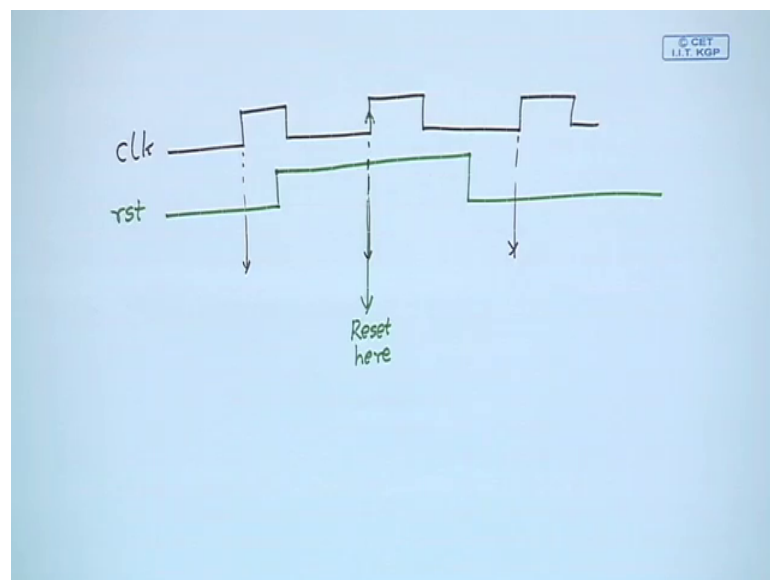
- Here reset occurs whenever "rst" goes high.
- Does not synchronize with clock.

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



Now, there is another way of specifying this will look at this previous one here, this always block was executing begin end whenever the positive edge of clock was coming.

(Refer Slide Time: 09:41)



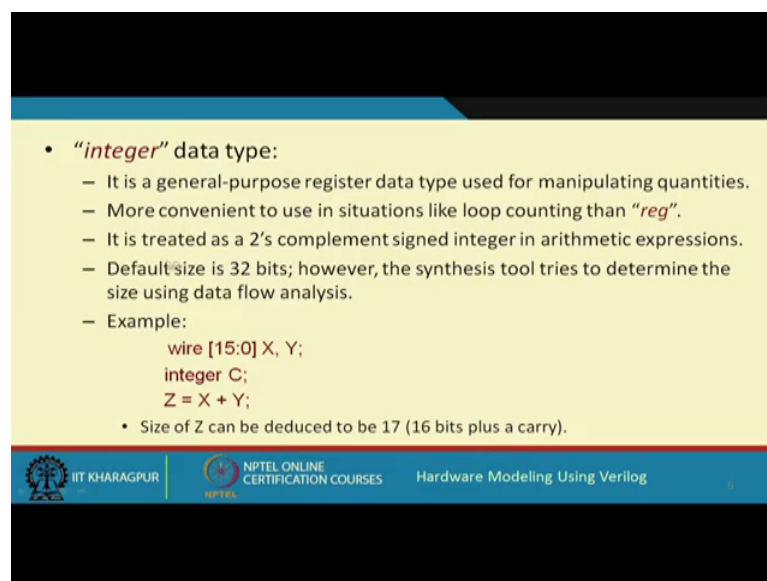
So, you think of a scenario like this. So, I have a clock, clock is coming like this, this is one positive edge, this is another positive edge, this is another post positive edge means 0 to 1; 0 to 1 positive edge. So, this always block will get executed here, here and here, but suppose my reset signal, I have made it high here. Let us say I have made it high here and have kept it high for some time and they have then I made it low. So, here even

though I have made the research signal high here, but according to our previous design the counter will be reset only here whenever the next positive edge of the clock comes.

So, counter will be reset here because the always block will be executing only when positives coming. So, even if your reset has been activated here, it will be waiting for some time before the counter is actually reset, but there can be application where you may want that whenever reset is going high you reset the counter immediately, right. So, our next design which is a slight modification of this one what we do we change this always statement we add another clause.

What I say; we say always act positive edge of the clock or positive edge of reset. So, if the reset signal goes from 0 to 1 that will also cause the begin end to execute and it will see that reset has become one it will immediately reset, it to 0. So, this is an example of asynchronous reset which means resetting is not happening in synchronism with the clock.

(Refer Slide Time: 11:46)



• *“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).

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

So, you can have both these descriptions in Verilog as you want, next comes the integer data type, the integer is a more general purpose type which you heard you can use it to store any arbitrarily integer values mainly for counting some situations like loop counting. So, reg may not be very convenient there because whenever you are counting something the count value may sometimes become negative, but as I said in reg, a

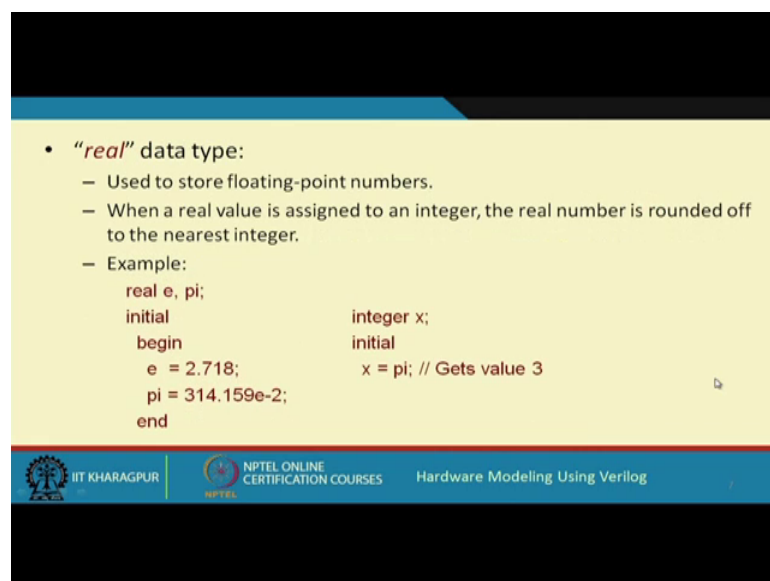
variable declared of type reg is always considered to be an unsigned quantity it cannot have a negative value, fine.

So, integer data type another feature is that it cannot have a size defined to it. So, the default size is 32 bits, but the point to note is that, but if you are doing synthesis again not for simulation if you are doing synthesis the synthesis tool will try to apply some intelligence it will do some kind of data flow analysis and try to guess that what should be the value of the integer should I keep whole 32 bits or less than 32 bits should also be like you take an example. Suppose, I have declared 2 wires x and y vectors of size 16 each 0 to 15; 16 bit, 16 bits and I declare an integer of size C.

So, as I said that whenever you declare an integer you cannot specify the size then we write sorry this not z this is C, let me correct this, this is C. So, here if I write C equal to x plus y, this is also C yeah. So, if I write C is equal to x plus y, what will happen? Two 16 bit numbers are added. So, the synthesis tool will see that x and y are 16 bit numbers you are adding it. So, this sum can be maximum 17 bits 16 bits plus a carry.

So, when it realizes the hardware for C it will not be implementing it as a 32 bit counter rather, it will be using it only as a 17 bit register, 17 bit is sufficient in this particular case, this is what I mean is that this synthesis tool will try to determine the size by carrying out some data flow analysis wherever possible.

(Refer Slide Time: 14:32)



• *“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.
- Example:

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

integer x;
initial
x = pi; // Gets value 3
```

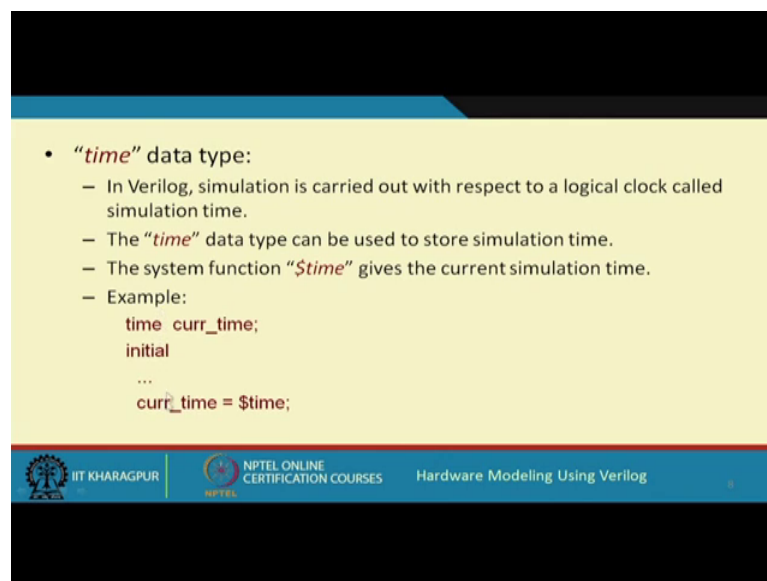
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Talking about real data types as the name implies they are used to store floating point numbers; numbers with fractional parts and just like in a high level language. So, whenever you assign a real value to an integer, there is little difference normally in a high level language like C the number is truncated and the integer part is stored, but here it is rounded off. So, a small example, suppose I define 2 variables e and pi as real and in the initial block which you have used seen in the test benches, this initial is normally used in the test benches.

So, in an initial block let us say I have initialized e to 2.718 and pi 2, this is just a notation f. So, I could have written 3.14159, but I have written 314.159 into 10 to the power minus 2 e; minus 2 e; minus 2 means 10 to the power minus 2, right. Now suppose I declare an integer x and somewhere again inside an initial block, I write x equal to pi. So, pi is 3.14 say to be rounded up and only the value 3 will be assigned to X right; this is what I mean here.

(Refer Slide Time: 16:08)



• “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;
```

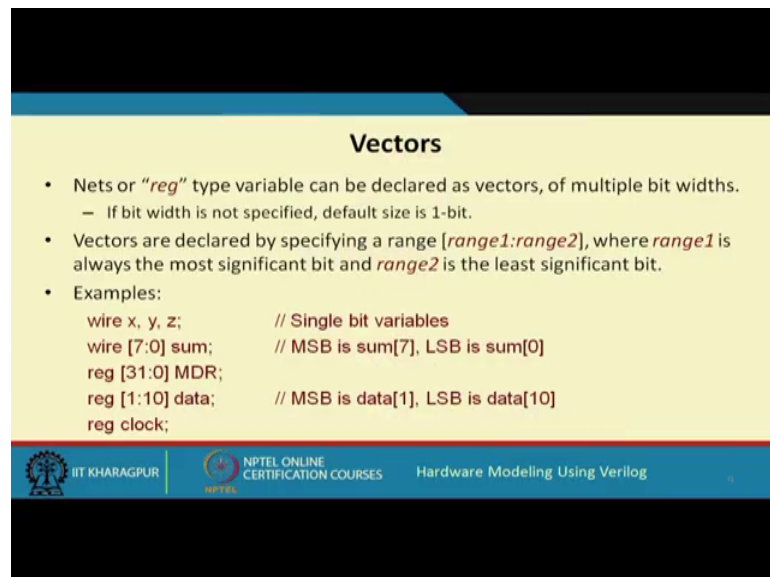
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now time is a data type which is used to keep track of simulation time. So, when simulation is carried out well you can use that I Verilog that; I had mentioned even you can say means you can use I Verilog or any other tool which are available to simulate a Verilog design. So, when you do simulation there is a notion of simulation time you specify time you specify gate delays lot of things you do. So, just a small example in the

Verilog code you can define a variable, let us say current time is a variable of type time and this special system function dollar time is there.

So, anything starting with dollar is a system function this system function actually gives you the current simulation time. So, in the initial block somewhere if you want to assign the current time to this variable you can write a statement like this curr time equal to dollar time. So, you can use the time data type in some applications where you actually want to find out that to execute a block how much time it has taken it took. So, you can print the time before you can print the time after and see the difference like that fine.

(Refer Slide Time: 17:43)



**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;
```

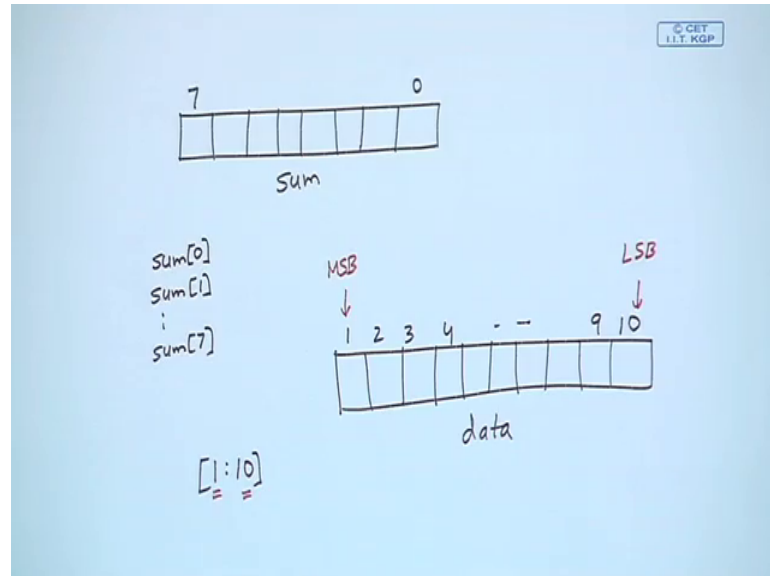
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, let us come to vectors you have seen single bit variables. Now let us see how can group such single bit variables to create something called vectors. So, nets or reg types variables both of them they can be declared as vectors. Vectors are nothing, but multiple bit quantities and if you do not specify this bit width then by default it is taken to be one bit. So, some examples we have already seen vectors can be declared by specifying a range within square bracket range one colon range 2, the convention is that the first one range one is always the most significant bit of the number and the rightmost one range 2 is the least significant bit.

Well, I will explain what it means; let us see some examples, suppose I write simply wire x, y, z; x, y, z are simple variables without any vector notation, these are single bit

variables. So, if I write wire 7 colon 0 sum; what does this mean? This will mean that I have an 8 bit variable called sum this is my sum there are 8 bits.

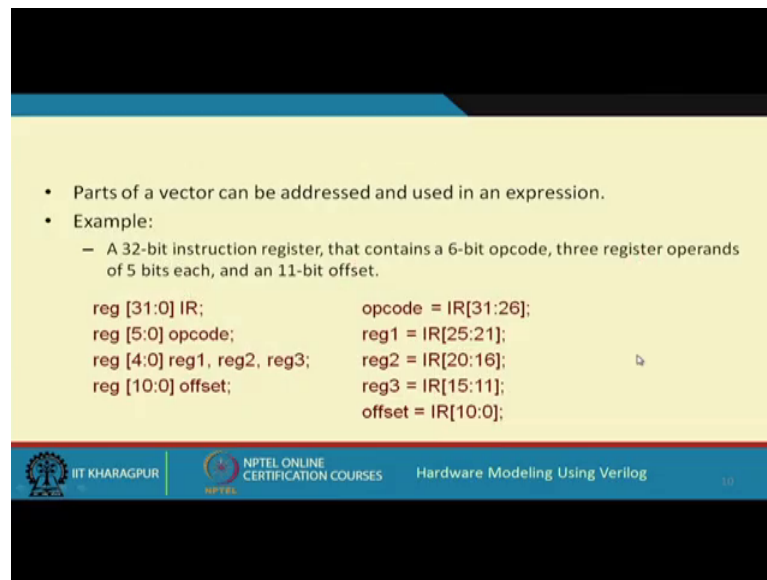
(Refer Slide Time: 19:13)



The index is 7 is the most significant, 0 is the least significant the individual bits, I can access by writing sum 0, sum 1 up to sum 7, this is just like an array this just like an array of bits, similarly, another example m d r this of course, I have defined of type range well, now the point is a range 1, range 2 can be any arbitrary values. So, it is not necessary that range one must be greater than range 2 like in this example I have shown reg 1 colon 10 data. So, how many numbers are there; 1 to 10, there are 10 numbers. So, actually here also we are declaring a register of size ten, but how this is data let us see. So, the name of the variable is data and the way we have declared the range one was one range 2 was 10. So, the index values are defined like this 1, 2, 3, 4 up to 10. So, in this case.

So, if you use this data in an arithmetic expression this particular bit will be treated as the most significant bit this particular bit will be treated as the least significant bit. So, by specifying your range one and range 2 in a suitable way, you can choose to swap LSB-MSB like here in this case, the 7 was the MSB and 0 was the LSB, but if you write one to 10, 1 become MSB, 10 becomes LSB. So, this convention has to be kept in mind, the first value of the range will be the bit on the left, the second value on the range will be the index bit index of the further on the right.

(Refer Slide Time: 21:47)



• 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];
```

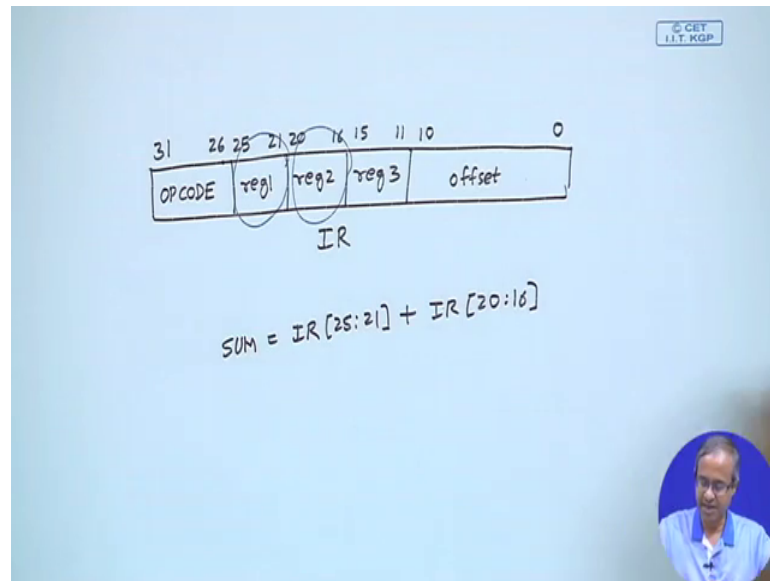
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 10

So, that will be your word. So, whenever you are using it, you should keep that in mind, all right.

So, after you have seen this vector let us see how we can use sections of a vector in an expression. So, what do you mean by sections of a vector suppose I have declared a vector of size something, let us say 32. So, the example I have shown is the vector of size 32 and I want to extract 5 bits from the middle of the vector and assign it to some variable now that can I do it? Yes, I can do it. So, I can take out any cross section of a vector by specifying the starting and the ending index and I can assign it to some variable I can use it in an arithmetic expression whatever I can do I want let me take an example. So, here I am taking an example of an instruction encoding. So, the example I have taken is a 32 bit instruction register that contains a 6 bit opcode, 3 register operands 5 bits each and an 11 bit offset.

So, what I mean is something like this suppose I have a 32 bit instruction register.

(Refer Slide Time: 23:09)



So, bit let us say from 0 to 31. So, what are the different fields? So, I have the first 6 fields or the opcode; first 6 fields means bit number 31 to bit number 26, then I have 3 register operands. So, I call them reg 1, reg 2, reg 3, these are all 5 bit fields. So, the bit number will be 21 to 25, 16 to 20 and 11 to 15 and the last field is a field which is called offset. Offset is an 11 bit field 0 to 10, suppose my instruction register has this format, there are 5 fields, right. So, what I can do is something like this I can declare this register IR 32 bit and I can declare the individual fields also opcode is a 6 bit field; reg 1, reg 2, reg 3 are 5 bit fields, 4 to 0 offset is an 11 bit field, then I can simply write opcode equal to IR 31 colon 26. So, you see here. So, we mentioned 31 to 26 opcode will be assigned 31 colon 26.

Similarly, reg 1 will be 25 to 21, you see reg 1 will be IR 25 to 21, reg 2 will be 20 to 16, reg 3 15 to 11; offset 10 to 0. So, in this way, I can use this cross sections in an assignment statement even mean I can use them in an expression like let us say, I can write an expression like this also next I can write sum equal to let us say IR 25 to 21 plus IR 20 to 16. So, I can write like this also. So, here what I am meaning is that 25 to 21 means this number 20 to 16 means this number; I am adding these 2 5 bit numbers and I am storing the result in another variable sum such things I can also do, right. So, I can take out a cross section from a given vector and I can do any kind of manipulations on it.

(Refer Slide Time: 26:00)

**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
```

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

Now, talking about multi-dimensional arrays and memories, now we can declare multi-dimensional arrays in Verilog of any dimension let us take 2 examples here reg 31 to 0 register bank again 15 to 0, what does this mean? First is reg 31 to 0, this is the declaration of a register of width 32; 32 bit register. Now we are saying that I have a register bank where there are 16 elements. So, we are actually talking about 16; 32 bit registers, right. So, and when you access this elements will be accessing them using the 2 index values like here if I just write.

(Refer Slide Time: 27:04)

register-bank [5]  
↓  
32-bit value

CET I.I.T. KGP

Let us say register bank if I just write 5 register bank 5, this will mean a 32 bit value, the content of the register number 5 because here I have declared register bank as an array of size 16; 0 to 15. So, when I say register bank 5. So, it is one of those elements and each of this element is of type reg 31 0; that means, a 32 bit number.

But, but here for integer for example, integer is already a 32 bit by default. So, I need not have to specify this. So, I write metric 7 0, 15 0 this is a 2 dimensional matrix with 8 rows and 16 columns, now will see later that when we use let us say memories in a design. So, one way there are other ways also we have see. So, one way is to declare a memory as a 2 dimensional array multi-dimensional array. So, some examples of memory declaration is this. So, you can simply write reg some name mem bit 0 to 2, 0 4; 7, this is a 2 kilo bit memory, total number of location is 2 0 4 8 and each location contains one bit, but if you declare it like this, similar to this register bank. So, there are 0 to 1 0 2 3; one kilo words each word is of size 16. So, you can declare a memory like this.

(Refer Slide Time: 29:00)

**Specifying Constant Values**

- A constant value may be specified in either the *sized* or the *unsized* form.
  - Syntax of sized form: `<size>'<base><number>`
  - Examples:
 

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

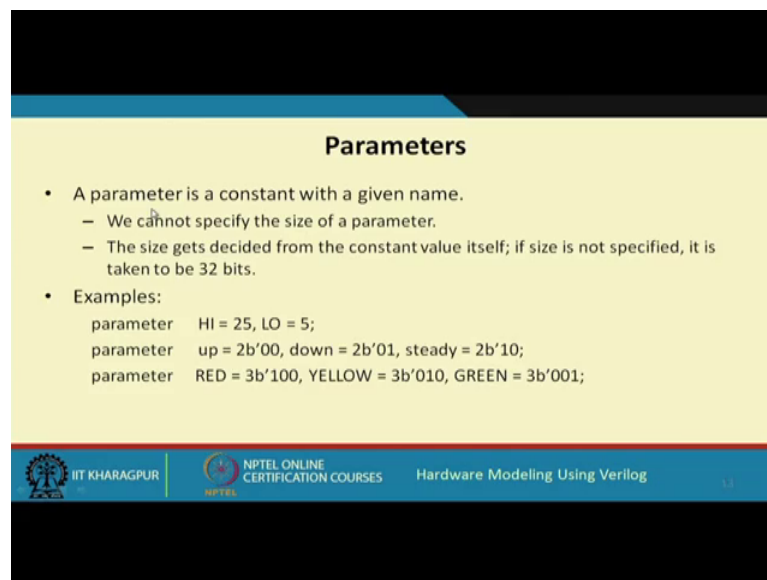
Variables of type integer and real are typically expressed in unsized form.

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

So, specifying constant values you can specify constant values like this in either sized or un-sized form, the syntax is you specify a size then a single quote followed by a base then a number here I have shown some examples of the base I have shown as binary or hexadecimal h 4 B 0 1 0 one means a 4 bit binary number 0 1 0 1 1; B 0 means logic 0 1 bit 12 hexadecimal means a 12 bit hexadecimal number 12 bit B 3 C.

So, this is B 3 C well out of this I can have some x also do not care also 8xf this is 8x means all xxxx f and if I write only 25, this is a sign number and by default, it will be put in 32 bits. So, whenever you are using integer or real numbers you typically do not use the size specifier; they are typically expressed in un-size form, right.

(Refer Slide Time: 30:19)



**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 | 13

Now, talking about parameters, we shall be seeing parameters again later. So, parameter is something similar to hash define that you use in a C program, it is like a constant we define a constant with a given name and in the program whenever we use that name we replace that name by that constant. So, this parameter is something like that.

But when you declare parameter we do not specify the size the size automatically get deduced by the constant values that we are defining like we have given some examples it will be clear the first example, I am saying parameter high equal to 25 low equal to 5. So, in the program whenever there is an HI, this HI will be replaced by 25 and whenever there is LO, LO will be replaced by 5, similarly next example up down steady.

So, I am declaring 2 bit number 0 0 means up, 0 1 is down, 1 0 is steady. So, if we use parameter in the program it will make your program more readable. So, instead of writing 0 0 and 0 1; if you write up and down, it will be easier to understand, right. So, that is the main purpose of using this kind of constructs parameter, similarly for a traffic light controller system, let us say you can define red as a 3 bit number 1 0 0 yellow as 0 1 0 green as 0 0 1.



(Refer Slide Time: 31:57)

```
// 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
```

Any variable assigned within the "always" block must be of type "reg".

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

This some examples and just one Verilog complete module, I am showing that uses a parameter this is an n bit counter.

So, I am declaring it as an N clear clock count clear and clock are the inputs count is the output this is 0 to N and because I am using it on the left hand side I am also declaring it as reg. So, this counter counts at a negative edge of the clock. So, it is reg edge. So, very similar to the previous one; if clear count you see here; we use the other kind of assignment less than equal to. So, the difference will explain later 0, else count equal to count plus 1. So, here if you just change this one line in the program your module can change from a means from a 7 bit counter to something else not 7 bit, 8 bit counter n plus one bit 0 to n parameter n equals 7 means it is a 8 bit counter. So, if I make it 15, it will become a 16 bit counter.

So, you change just one line that is the advantage of using parameters. So, with this we come to the end of this lecture again we shall be looking at some more features of the Verilog language in our next lectures.

Thank you.