Hardware Modeling using Verilog

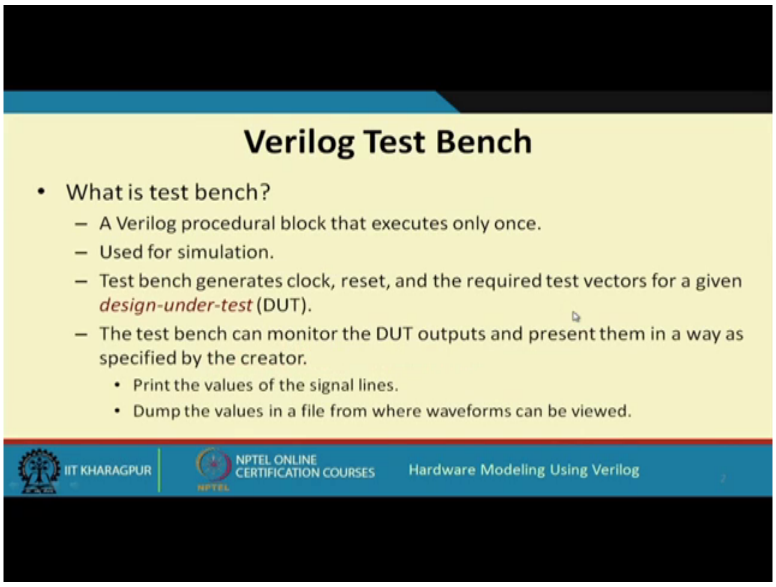
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 21
Verilog Test Bench

So, in our previous lectures we had looked at a number of examples in Verilog, and we have also seen how to write test benches for such modules that we have described. We had looked at some of the features of the test benches; how you can apply this stimulus values for combination circuits, how you can apply clocks for sequential circuits and so on.

Now, in this lecture, we shall be looking at the test benches, the different commands that are available in a more formal and elaborate way and in the next lecture, we shall be looking at some examples on some of the features that we will be discussing in the process. So, our lecture is titled Verilog test bench.

(Refer Slide Time: 01:09)



Verilog Test Bench

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

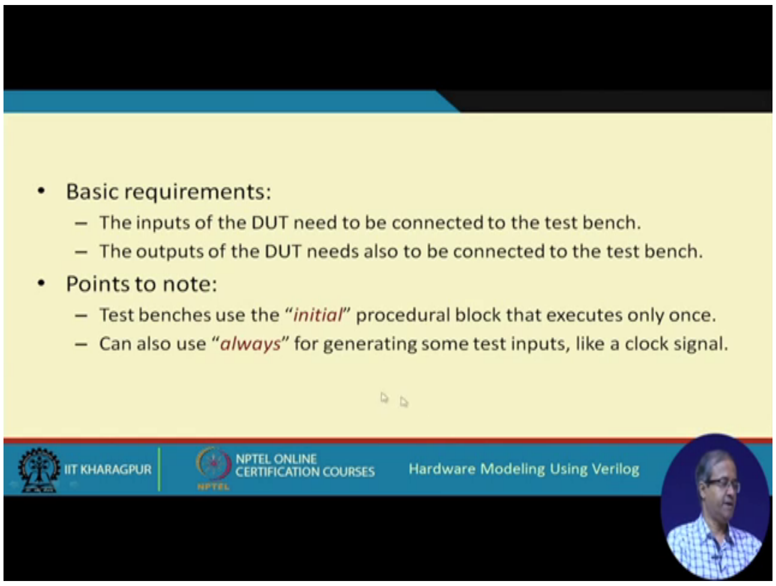
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Now, these are the things which I have already discussed earlier. So, let us have a quick recap. So, a test bench is a very log module, it is a procedural block that runs or executes only once, you recall a procedural block can be of 2 types; the always procedural block and the initial procedural block. Particularly, the initial procedural block is used for writing test benches because there are certain things which you need to run only once and the initial block is meant to be run only once; of course, there are some parts of the test bench like generating a clock which you want to do it repeatedly that part you can do using always if you want, fine.

So, such this initial blocks and these test benches written using those; they are used only for simulation. So, if your task is to synthesize a circuit, then you do not write the test bench you will be writing test bench, if you will not; if you want to verify your design by applying some test inputs and see what the outputs are coming then only you write the test bench, fine. So, roughly speaking what does the test bench do they generate the different inputs to your circuit or your design like clock reset and the other functional inputs.

So, suppose you have created a design, let us call it a design under test or DUT; you can apply such inputs to DUT and it can also monitor the outputs you see when you apply the inputs; after that you may want that I want to see what the outputs are. So, your test bench can just print those values on the screen or if you want it can also dump it in a file. So, that you can later on see the signal values in a nicely graphical interface; the waveforms the timing diagrams and so on in that form. So, you can view the outputs in whatever way you want to by specifying it appropriately, right.

(Refer Slide Time: 03:44)



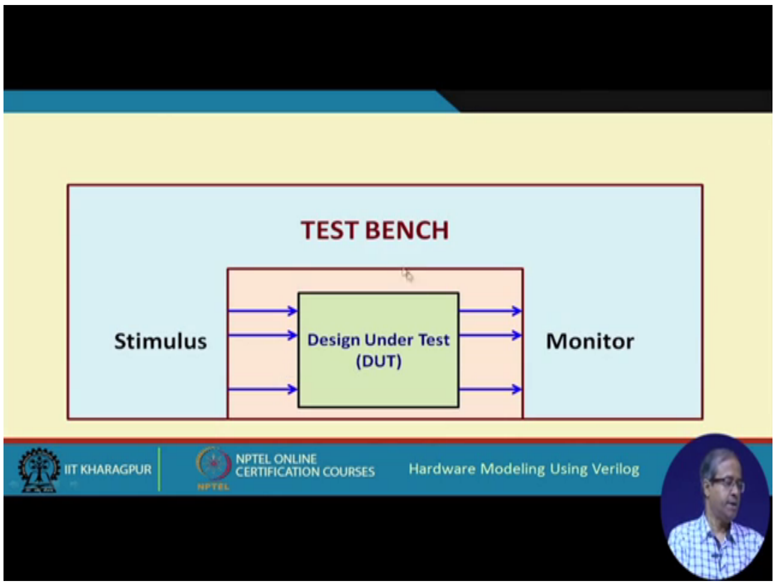
- Basic requirements:
  - The inputs of the DUT need to be connected to the test bench.
  - The outputs of the DUT needs also to be connected to the test bench.
- Points to note:
  - Test benches use the "*initial*" procedural block that executes only once.
  - Can also use "*always*" for generating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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

As I said, you can either print the values or you can dump them in a file from where you can view them in the form of waveforms fine.

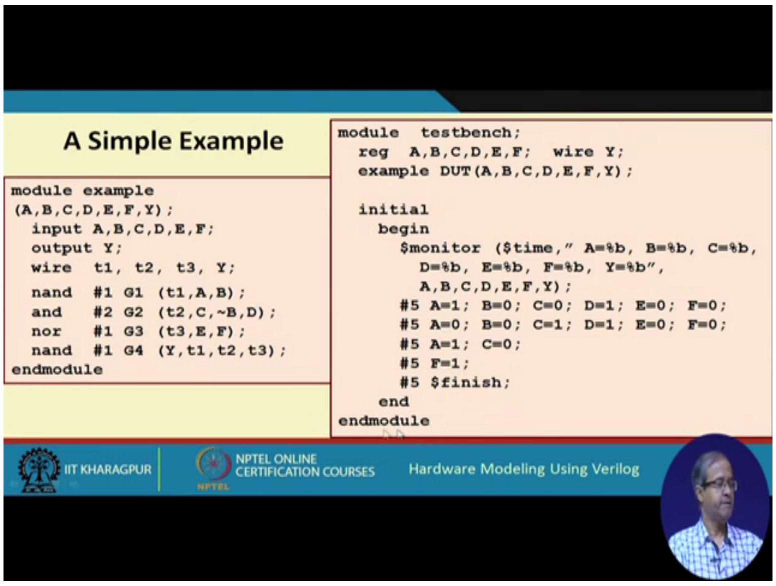
So, the basic requirement is the design that you have created the inputs of it should be connected to the test bench because the test bench will be applying the inputs to the dot. Similarly the outputs of the DUT should be connected to the test bench because the test bench would be reading those values and will be printing or displaying whatever you want to. So, these are the things I have already mentioned. So, in a test bench you can either use initial blocks or always blocks also. So, initial procedural block is the one which executes only once, but always block is repetitive. So, for generating some periodic signals like clock you can use always ok pictorially whatever I told just now you can visualize like this.

(Refer Slide Time: 04:38)


TEST BENCH
Stimulus
Design Under Test (DUT)
Monitor
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Let us say this is your design which you have written in very long, I am showing it as a box and this is the test bench you have written. So, the first thing; what you do is that you instantiate a copy of your design within the test bench, then you connect the input lines of your design; under test to some stimulus output lines from your test bench. So, your test bench outputs will be your inputs to your UDT, similarly whatever DUT generates as outputs that will be read by your test bench and it will be monitored you can either print it you can display it whatever.

(Refer Slide Time: 05:38)



A Simple Example

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

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

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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

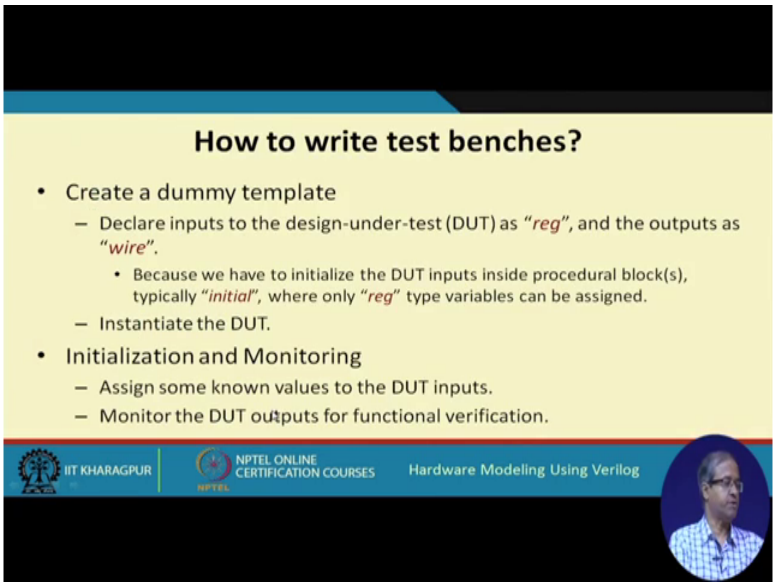
So, stimulus and monitoring these 2 mechanisms are also part of your test bench. This gives you the overall picture, fine, a very simple example to start let us say here; we have a simple module description where there are 4 gates in our design there are seven parameters A, B, C, D, E, F, Y where this 6 parameters A, B, C, D; A for the inputs and Y is the output and t 1, t 2, t 3 are and this Y or some wires t 1, t 2, t 3 are the intermediate lines and Y is the final output.

So, this is just an arbitrary description this is not any design which is meaningful, it doesnot compute any meaningful function just for illustration. So, you instantiate the 4 gates you appropriately specify the input values and some of them can be inverted also like not B, you can write and this t 1, t , t 3 are the outputs of G 1, G 2, G 3 which will be the inputs of the last gate here. So, finally, the output is Y. So, this kind of test bench you have already seen earlier. So, in the test bench what you do first thing is that you instantiate this module this A, B, C, D, E, F, Y, here let us say we will give the same names A, B, C, D, E, F, Y.

So, whatever were the inputs of here A, B, C, D, E, F; here there will be outputs actually and you have to declare them as reg because you will be assigning them inside a procedural block; you will be assigning some values to A, B, C, D, E, F that is why they will be reg and for a module, Y was the output, but for your test bench this why you declare just as a wire because this will be an input, you can just read it and you can just print it. So, inside the initial block, you can give several statements; we shall explain this meaning first is a monitor command monitor actually means that whenever any of these variables whatever you specify in this list they change you print them in this format first to print the time simulation time, then A equal to percentage B means binary A equal to this value B equal to C equal to and so on.

And with a delay of 5, 5, 5 where applied the test vectors C; first we have applied A 1 0 0 1 0 0 then you have set 0 0 1 1 0 0, then you have said only I mean A 1 C, it means the other variables are not changing, B is still 0, D till 1, E is still 0, F is still 0 and in the last line, we have written f equal to 1; that means, the remaining 5 variables are same only F is changing, F is 1 and then we finish our simulation. So, this test bench means if you just run you will get the outputs for this Y in this order.

(Refer Slide Time: 08:55)



How to write test benches?

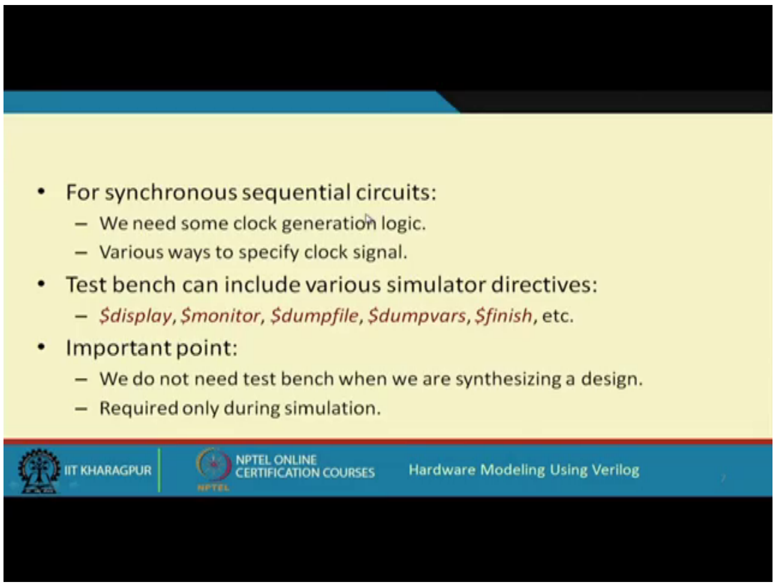
- Create a dummy template
  - Declare inputs to the design-under-test (DUT) as "*reg*", and the outputs as "*wire*".
    - Because we have to initialize the DUT inputs inside procedural block(s), typically "*initial*", where only "*reg*" type variables can be assigned.
  - Instantiate the DUT.
- Initialization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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Now, there are a few things points. So, when you write this test bench, here is an example, you create a dummy template module test bench n module, then you instantiate your DUT or design under test inside. So, first task for writing test bench is to create a dummy template. So, insert the dummy template of course, you will have to instantiate your design under test and whatever signals were inputs to your duty, they will now be declared as reg and whatever signals where outputs in your duty they will be declared as wire. So, you see exactly that we did here. So, whatever our inputs here A, B, C, D, E, F; they have declared as reg and Y was output Y, you have declared as wire, right, the reason I have already mentioned. So, these you have to declare reg because you will be assigning them to values inside the initial procedural block.

So, here let us say inside this initial block you are assigning values to A, B, C, D, E, F and we said earlier that inside any procedural block whenever you assign values the left hand side must be a reg type variable that is the reason. So, means after that you will have to initialize and monitor will have to apply some values to the inputs of your DUT.

(Refer Slide Time: 10:41)



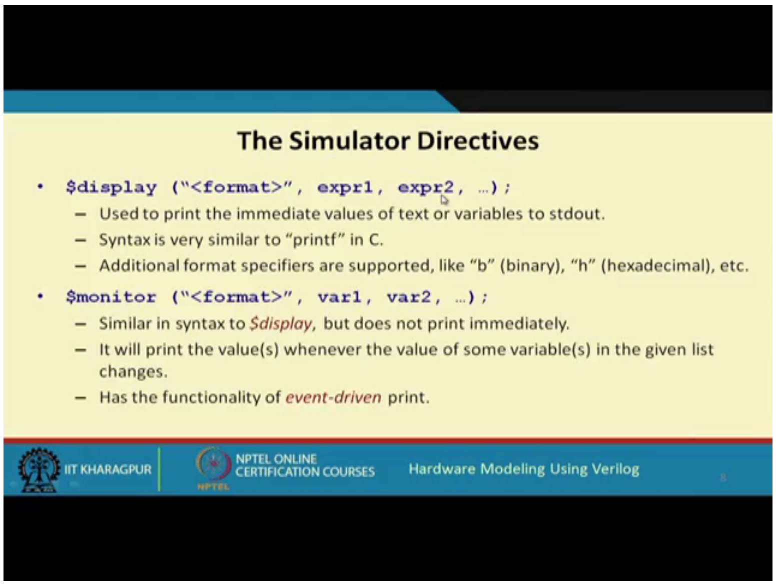
- For synchronous sequential circuits:
  - We need some clock generation logic.
  - Various ways to specify clock signal.
- Test bench can include various simulator directives:
  - *$display*, *$monitor*, *$dumpfile*, *$dumpvars*, *$finish*, etc.
- Important point:
  - We do not need test bench when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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

And you will have to monitor. The DUT outputs in a suitable way to verify whether it is working correctly or not. So, for synchronous sequential circuits in addition to your inputs you may be having some clocks reset clear this kind of inputs. So, you need something more for sequential circuits. So, whatever example I took just now that is for a combinational circuit, but for a synchronous sequential circuit you will also need some clock generation logic there are various ways to specify clock signals. We shall see later and through some examples how to do it and the test bench can include some simulated directive sometime.

These are called tasks also simulated directives, these are very common tasks, display it starts with a dollar symbol, display, monitor, dumpfile, dumpvars, finish, these you have already seen in our example which you saw earlier and again, I am repeating that you need to write a test bench only when you are carrying out simulation. So, if your objective is to synthesize a design then you do not need a test bench, right.

(Refer Slide Time: 11:53)



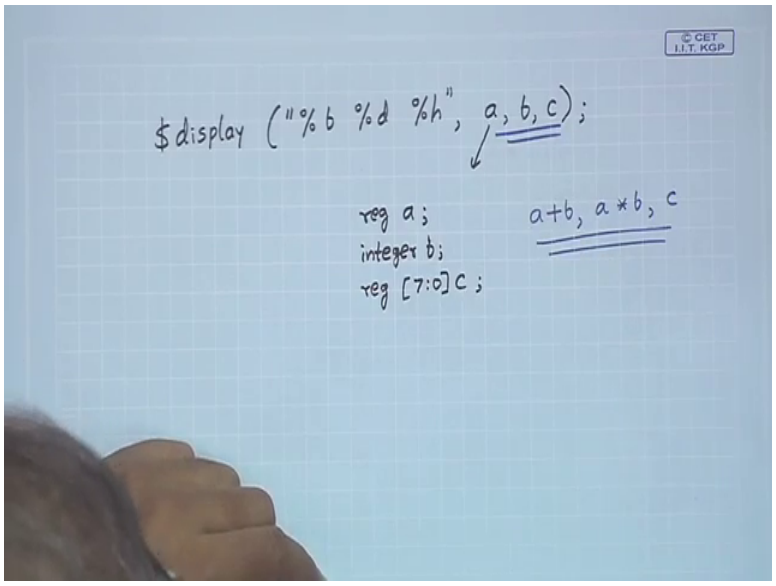
The Simulator Directives

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

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

Let us now explain the meaning of the simulator directives. The first similar directive we explain is display see display is exactly similar to the print f command that you have in language like C; in C whenever you want to print the value of some variables you give a print f, you specify the list of the variables you want to print and also you specify a format string that tells you how you want to print. So, exactly the same way in the display directive dollar display within bracket within double quotes you first specify the format string, then you specify the variables or the expressions that you want to print or display. So, some examples I am giving.

(Refer Slide Time: 12:55)



```
$display ("%b %d %h", a, b, c);

reg a;
integer b;
reg [7:0] c;
```

a+b, a*b, c
© CET I.I.T. KGP

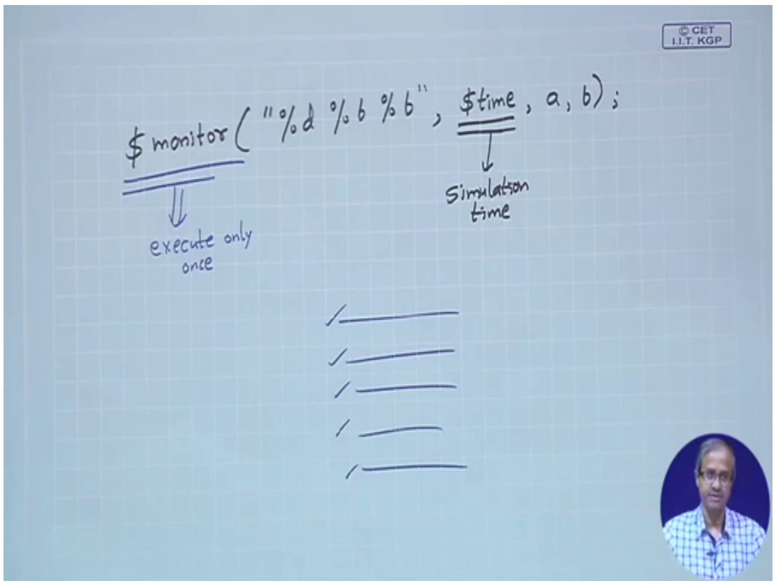
So, you can give a command like this; dollar display within double quote you can say percentage b, percentage d, percentage h, suppose I specify the names of 3 variables a, b and c where this a can be a bit variable, I can declare them; let us say as a reg a single bit variable b can be lets an integer b can be an integer variable.

So, I printed using the integer description d and h is a hex a hexadecimal. So, let us say c can be a vector that since c is an 8 bit vector. So, I am saying you display this 8 bit number in hexadecimal form. So, this will be displayed in hexadecimal this is what is meant by display you specify a list of it not only variables, you can also have an expression like instead of a, b, c, you can also write I want to print a plus b; I want to print a star b multiplied b and then I want to print c, I can also write like this. So, any expressions in general not necessarily only variables, right.

So, this display directive will be printing the variables whenever displays encountered and. So, whenever these executed used to print the immediate values whatever are the immediate values. So, whenever display command is encounter that line is encountered whatever values of these variables are there at that point in time they will be immediately displayed, fine.

So, as I said you can have specifies like binary hexadecimal and so on; well you have another directive which is very similar to display monitor. So, the format is also quite similar syntax. So, initially within double quotes you give the format how you want to display then you specify some well in monitor you typically give variables not expressions because you want to monitor the value of some variables, but there is one difference from display will print these values as soon as it is encountered, but monitor will not print the values, immediately, it will print the values whenever the value of at least one of the variables change.

(Refer Slide Time: 15:55)

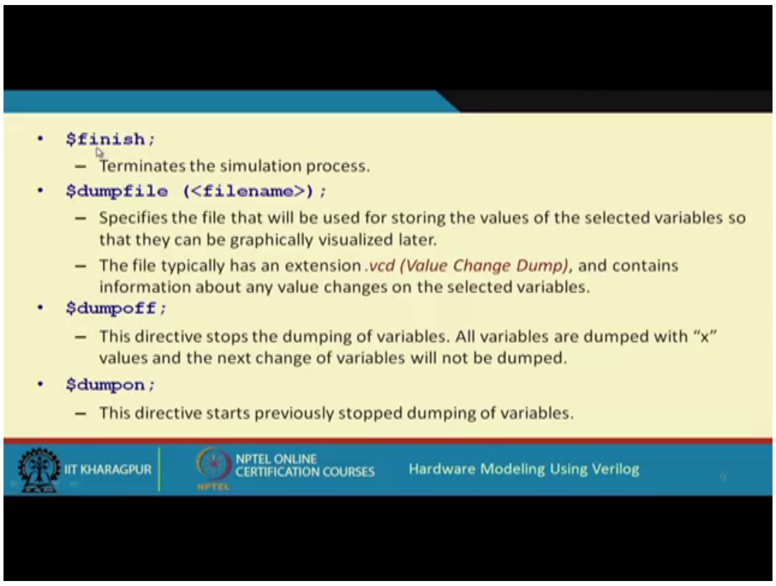


There is a concept of even driven printing approach like suppose I say dollar monitor percentage d, percentage , percentage b, let us say I first want to display my time simulation time, this will mean des dollar time means the present simulation time, this is a special variable, it indicates the current simulation time.

Now, let us say I want to display 2 scalar variables a and b, now monitor means it will not print the values immediately, you execute this monitor only once, this monitor statement will execute only once, this you can give inside an initial statement only once, but when you see the output which we will be generating you may see that output will be generating many lines. So, whenever any one of these variables change there will be a one line of output generated. So, you can see how the variables are changing with time during simulation.

So, this monitor command is very suitable or handy in such situation, right. So, this is the display or this is the mainly the difference between display and monitor. So, one is an immediate print other is an event driven print.

(Refer Slide Time: 17:34)

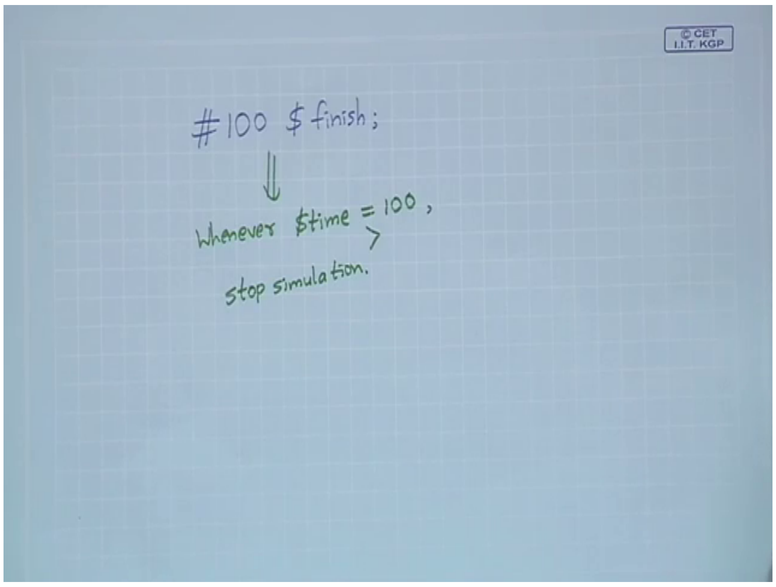


- `$finish;`
  - Terminates the simulation process.
- `$dumpfile (<filename>);`
  - Specifies the file that will be used for storing the values of the selected variables so that they can be graphically visualized later.
  - The file typically has an extension *.vcd (Value Change Dump)*, and contains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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Now, whenever you want to finish this simulation, you give this command dollar finish typically, you give a delay before that let us say hash 100 dollar finish; that means, at time 100. So, whenever this simulation time is has reached 100 you finish like.

(Refer Slide Time: 17:57)



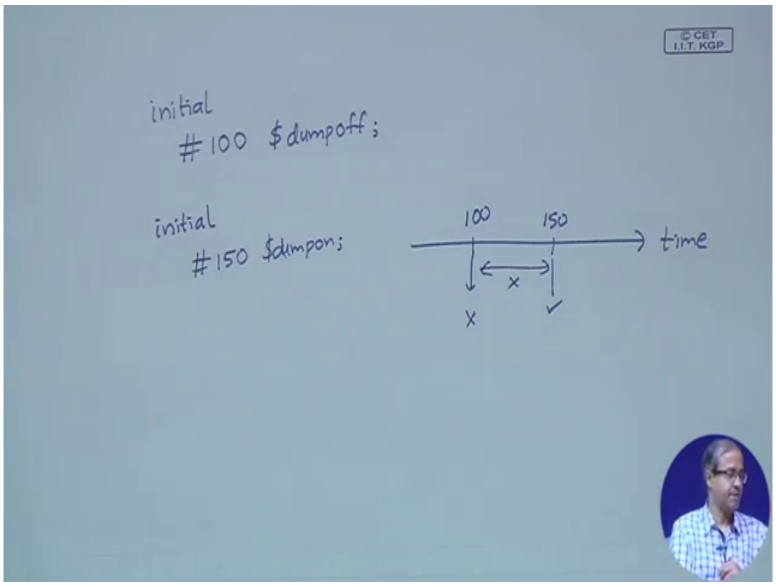
© CET I.I.T. KGP
#100 $finish;
$\Downarrow$
Whenever $time $\overset{=}{>}$ 100,
stop simulation.

You can give as I said hash 100 dollar finish. So, what will this statement mean; this statement will mean that whenever the current simulation time which is held in the variable dollar time this equals to 100 or is exceeding 100 or it is greater then stopped simulation the meaning is this, right.

So, there is another command called dumb file where you can specify the name of a file in the argument. So, here you are actually specifying that where to dump the values of all the variables which you can view later. So, this file name will specify specifies the name of that file which will be used to store the values of some selected variables we say we shall see how to select them variables there is another command for that some variables will be dumped into the file so that you can see them later in a suitable way and this file which is specified here this typically has an extension dot VCD.

This is quite commonly used file format this is the acronym for value changed and this file contains information about all changes that take place in the variables that you have specified and during simulation wherever you encounter dollar dump of this dumping into the file will stop and whenever you again encounter dump on.

(Refer Slide Time: 20:12)



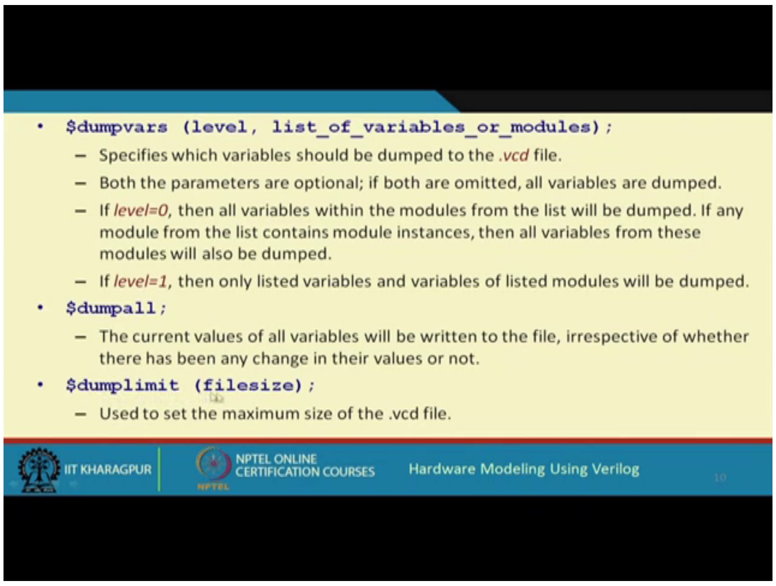
```
initial
  #100 $dumpoff;

initial
  #150 $dumpon;
```

So, again, this dumping will start like for example, if you give a command like this lets say suppose you give initial 100 dumb of there is another initial block.

Let us say you give 150 dumb on this will mean that in the axis of time. So, whenever time has reached this is time. So, whenever time has reached 100, you stop the dumping temporarily, pause, then again when the time has reached 150; you again resume dumping dump on. So, you can for certain period of time you can stop the dumping operation if you want all these controls you have with you, right fine.

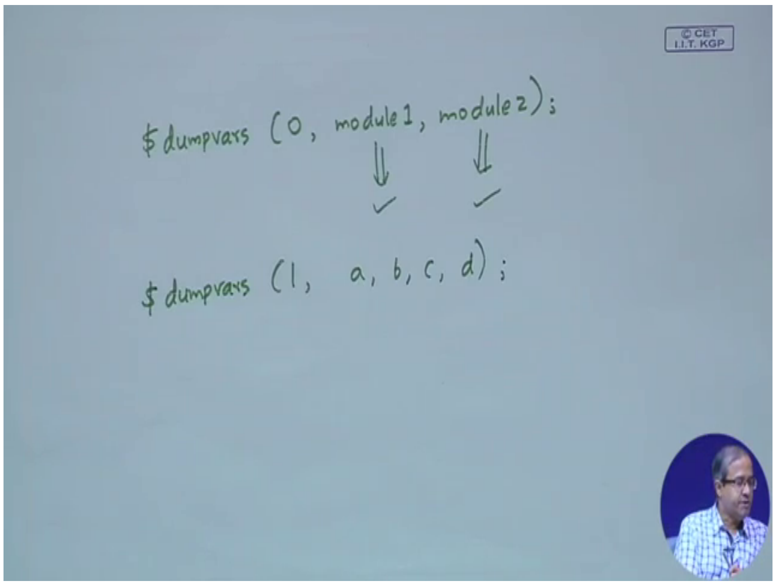
(Refer Slide Time: 21:09)


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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, dumb fires actually tells you see here we have seen dumb file means you are saying some selected variables will be dumped, but which variables that you can specify in dumpvars; dumpvars, how we can specify level is something which can be 0 or 1; say if level is equal to 0, if I give 0 in the first parameter, then in second parameter; I give the name of a module, then all the variable name of modern list of modules then all the variables within the modules will be dumped like.

(Refer Slide Time: 21:57)

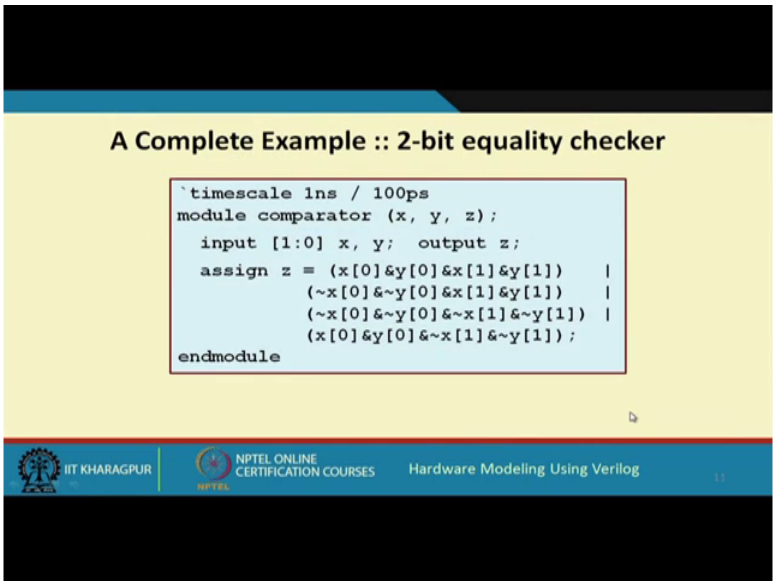


I can specify it either in this form I can give dumpvars ; the first parameter I can give 0 level equal to 0, then I can give a list of modules I can write; let us say module 1, module 2 like this whatever models are there.

So, what this will do? So, all the variables that are inside module one all the variables switches there is in model 2 they will all be dumped, dumpvars 0 means what means that and if I write dumpvars 1, then I can give a list of name of variables a, b, c, d, the variables that I want to dump. So, you see. So, if level equal to 0, then all variables within the models from this list will be dumped, if level equal to 1, then only the listed variables will be dumped right this is the main difference and if you give dollar dump all without giving in a list, then all variables will be written to the file.

So, irrespective of there is a change in the variable or not and sometimes when you are just handling a very complex design and your simulation is going on for a long time your dump file may become very large. So, there is also a facility where you can specify that well I will not allow my dump file to cross a certain limit like I can say it should not be more than 5 megabytes let us say. So, I can specify a file size limit in by using the command dump limit dump limit file size this can be used to set the maximum size of the dump file, right.

(Refer Slide Time: 23:58)



A Complete Example :: 2-bit equality checker

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

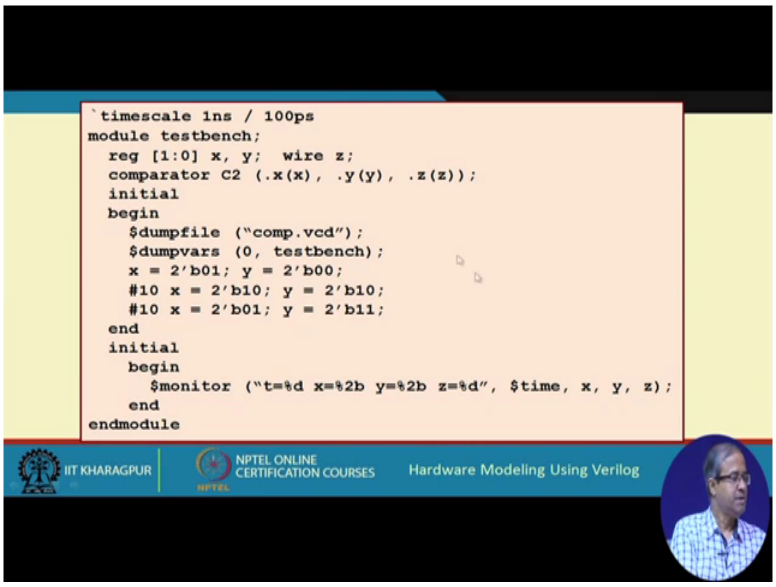
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, let us take a small example. So, here we are take an example of a 2 bit equality checker well we talked about the time scale command earlier you remember in this time scale which appears in the beginning of a module the first parameter indicates the unit like in the delay whenever if I write hash 5; it means 5 into 1 nano second and this second one; this 100 ps; picoseconds, this means the resolution. This simulation will be carrying out the or will be computing the results correct up to 100 picoseconds.

So, this is just module comparator which compares whether 2 numbers are equal or not there is one number is x 1 x 0 other is y 1 y 0, you see this assign z equal to there are 4 conditions separate by or this or this or this. So, what does the first condition say it says x 0 and y 0 and y one and x one it means all of them are one; that means, this x is also 1 1 y is also 1 1; second one says x 0 is not y 0 is naught x 1 is 1; y 1 is 1 which means 1 0 and 1 0 both are 1 0; 1 0 and this is all are naught; naught; naught; that means, both all are 0 0, x is also 0 0, y is also 0 0 and here x 0, y 0, are 1 1, x 1 y 0 or naught; that means, 0 1, 0 1.

So, for all the 4 combinations if they are equal, z will be 1, just we are writing down the expression for that.

(Refer Slide Time: 25:50)



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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
Hardware Modeling Using Verilog

This is a simple test bench using the commands that we have mentioned you see we have given a time scale at the beginning this is the module test bench this is the comparator module this was comparator we are giving it name C 2 2 bit comparator. Well, this is the recommended style of including or instantiating a module I mentioned earlier also instead of just writing x, y, z.

It is good to write using dot the name of these variables and the variables that are using here in well here in this example the names are the same, but you could have used the names a, b, c also or you could have included the variables in a different order like you put z first, then x, then y; like that it is always good to specified like this. So, see here we have given 2 initial blocks in the second initial block there is only a monitor statement you see t equal to percentage D we are displaying the time then x, y, z, 2 b means mint bit within a 2 space.

So, the first one will be blank. So, x, y, z will be z is integer you see z it is output we are printing as d; this you could have also given as b; no problem and this x and y are 2 bit number that said 2 b; 2 b and here we have given a dump file specified file name what the values will be dumped and 0 comma name of the module test bench is a name of the models inside test bench whatever variables are declaring and the modules; you are instantiated all those variables will be dumped then my stimulus. First I am applying x equal to 0 1 y equal to 0 0, then after time 10; I am applying x equal to 1 0 y equal to 1 0 then again after time 10; I apply x equal to 0 1 y equal to 1 1. So, only for this second one there would be a match, right.

So, with this I come to the end of this lecture. In the next lecture, we shall actually be seeing some examples of the constructs that we have seen in this lecture; how you can use them and what are the different ways; you can generate or apply the test stimulus to a design under test. This we shall be discussing in the next lecture.

Thank you.