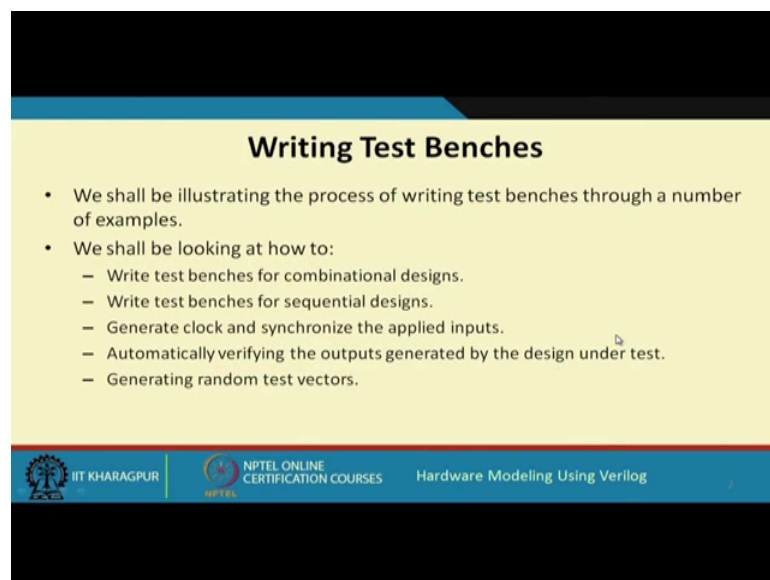


Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 22
Writing Verilog Test Benches

So, continuing with our previous lecture, in this lecture, we shall actually be seeing some examples and we shall see how we can write test benches in various different ways. So, writing Verilog test benches is the topic of the lecture.

(Refer Slide Time: 00:38)



Writing Test Benches

- We shall be illustrating the process of writing test benches through a number of examples.
- We shall be looking at how to:
 - Write test benches for combinational designs.
 - Write test benches for sequential designs.
 - Generate clock and synchronize the applied inputs.
 - Automatically verifying the outputs generated by the design under test.
 - Generating random test vectors.

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

So, in this lecture, as I said, we shall be illustrating various ways of applying the stimulus using examples. Specifically, we shall be looking at the few things; the following things; we shall see how we can write test benches for combinational designs, not only that how we can apply the test patterns; how we can generate the test patterns. Then similar things for sequential designs, generation of the clock and synchronization of the inputs is very important; how we can do that. And also we shall talk about that how could automatically verify the output generated.



Like for example: let us say; I am designing an adder my test bench should be such that it is applying the inputs, it is reading out the output and it will automatically compare whether the output is coming correctly or not and it will just tell me that the design is good or bad. So, it will not just show me a list of all the inputs and outputs it can also tell


me; whether the output is coming as per the specification or not and lastly we shall also look at how to generate random test vectors which may be useful in many test bench applications, fine.

(Refer Slide Time: 02:07)

Example 1: Full Adder

```
module full_adder (s, co, a, b, c);
  input a, b, c;
  output s, co;
  assign s = a ^ b ^ c;
  assign co = (a & b) | (b & c) | (c & a);
endmodule
```

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





Let us start with a very simple example; the simplest example a full adder. So, we have a behavioral description of the full adder using assign statement. So, s and co are the someone carry and a, b, c are the inputs.


(Refer Slide Time: 02:30)

```
module testbench;
  reg a, b, c; wire sum, cout;
  full_adder FA (sum, cout, a, b, c);

  initial
  begin
    $monitor ($time, " a=%b, b=%b, c=%b, sum=%b, cout=%b",
              a, b, c, sum, cout);
    #5 a=0; b=0; c=1;
    #5 b=1;
    #5 a=1;
    #5 a=0; b=0; c=0;
    #5 $finish;
  end
endmodule
```

0	a=x	b=x	c=x	sum=x	cout=x
5	a=0	b=0	c=1	sum=1	cout=0
10	a=0	b=1	c=1	sum=0	cout=1
15	a=1	b=1	c=1	sum=1	cout=1
20	a=0	b=0	c=0	sum=0	cout=0

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

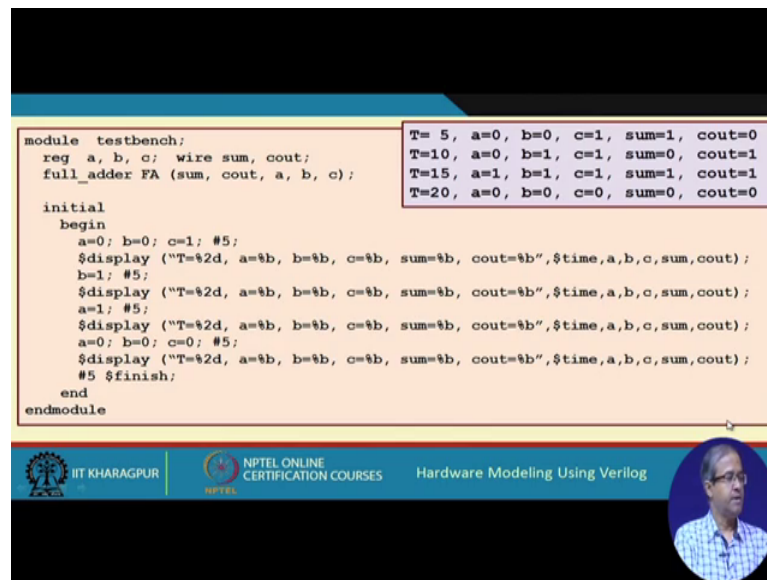


So, some is the xor and carries a b or b c or c a. So, let us see different ways of doing it, this is the first approach and this kind of test bench, you have already seen earlier. So, what we have done? This was our full adder module, we have instantiated the full adder model we call it FA. So, we have specified the parameters; the same order some carry a, b, c; here we are calling it sum cout; carry out; a, b, c are the inputs. So, the inputs were declaring as rich as usual and the output sum and c out; we are declaring as wire. So, here because it is a combinational circuit; we are making it simple, we are using a single initial block.

So, we are executing monitor only once, here you are saying we want to display whether also give it in the beginning dollar time, comma you can also give it like that. So, there the time will be displayed in the beginning like this, then within quote a equal to b equal to c equal to sum equal to and c out equal to all b means binary. We are displaying these 5 things. So, you recall monitor means these 5 variables who will display it whenever at least one of the variables they change state, fine. So, the first input, I am applying is a 0 b 0 c 1, then I change b to 1. So, delay is 5. So, after delay of 5, I am doing this, then a equal to 1 and last factor is 0 0 0 and then after another 5, I finish. So, let us see the simulation output what we get on the screen if we run it.

So, initially; so, whenever the time is initialized to 0 that will also be a change. So, a, b, c some x; all are undefined all are x at time 5, I have initialized at time 5, a becomes 0, b 0, c 1, my sum becomes 1 carries 0, then 10 I mean b equal to 1. So, now, b is 1, a 0, b 1, c 1. So, now, sum is 0 carries 1. So, again after time 5 a is 1, this is at time 15, a is also become one 1, 1, 1, 1. So, sum is one carry is one and at the end all are 0, 0, 0, 0 sum is 0 carry is also 0.

(Refer Slide Time: 05:09)




```
module testbench;
  reg a, b, c; wire sum, cout;
  full_adder FA (sum, cout, a, b, c);

  initial
  begin
    a=0; b=0; c=1; #5;
    $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b", $time, a, b, c, sum, cout);
    b=1; #5;
    $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b", $time, a, b, c, sum, cout);
    a=1; #5;
    $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b", $time, a, b, c, sum, cout);
    a=0; b=0; c=0; #5;
    $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b", $time, a, b, c, sum, cout);
    #5 $finish;
  end
endmodule
```

T= 5	a=0	b=0	c=1	sum=1	cout=0
T=10	a=0	b=1	c=1	sum=0	cout=1
T=15	a=1	b=1	c=1	sum=1	cout=1
T=20	a=0	b=0	c=0	sum=0	cout=0

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



So, this is a very simple way of generating these tests. Now the same example I am showing in a slightly different way, since earlier case I use a monitor of course, this is more convenient, but I am just illustrating the use of display instead of monitor here what I am doing using same full adder instantiation I am just declaring inputs and reg, the outputs as well here I apply the first input a 0, b 0, c 1, then I apply a delay of 5, then I display what time a, b, c some c out t equal to percentage 2 d a, b, c, sum; we are these are all b binary display will immediately display this line t equal to 5 a 0 0 1 1 0, then b equal to 1 again a delay of 5. So, now, b becomes 1, again after a delay of 5, I put another display same line.

So, the next line is displayed, then a equal to 1; there is again a delay of 5; again display third line, then again a 0, b 0, c 0, again delay of 5, again display fourth line and then finish. So, see that if you give monitor, we need to give only once and whenever the variable changes, this simulator will automatically understand that well we are supposed to monitor the variables and we want to print it and display means wherever I want the values to be printed I insert a display command like that just like print f in c. So, in this example, I have inserted for print f statements corresponding to the 4 display statements and for the displaced statements 4 lines are printed this is the difference between monitor and display. So, in a test bench I can use display if I want I can use monitor I do not see one drawback of monitor is that if over a long period during simulation variables do not change their values nothing will get printed.

But I want to see that every after every gap of fifty what is the outputs coming I want to see then I want to give a display that irrespective of the values whether they are changing or not changing I want to see their values then I have to use the display command rather than monitors, all right.

(Refer Slide Time: 07:54)

```

module testbench;
  reg a, b, c; wire sum, cout;
  integer i;
  full_adder FA (sum, cout, a, b, c);

  initial
  begin
    for (i=0; i<8; i=i+1)
    begin
      (a,b,c) = i; #5;
      $display ("T=%2d, a=%b, b=%b, c=%b, sum=%b, cout=%b",
               $time, a, b, c, sum, cout);
    end
    #5 $finish;
  end
endmodule

```

T= 5, a=0, b=0, c=0, sum=0, cout=0
T=10, a=0, b=0, c=1, sum=1, cout=0
T=15, a=0, b=1, c=0, sum=1, cout=0
T=20, a=0, b=1, c=1, sum=0, cout=1
T=25, a=1, b=0, c=0, sum=1, cout=0
T=30, a=1, b=0, c=1, sum=0, cout=1
T=35, a=1, b=1, c=0, sum=0, cout=1
T=40, a=1, b=1, c=1, sum=1, cout=1

Now, the third example same the full adder here, let us say I want to generate all 7 inputs like the simulation output let us say, then and see how we have done it safer starting from 0 0 0 0 0 1 0; 1, 2, 3, 4, 5, 6, 7; all 8 combinations I want to generate and I want to print the outputs, here I have used a for loop for the purpose well how have we done this it is quite simple we have declared an integer of type i because this a, b, c, these are 3 variables, alright, but we can treat them as a 3 bit number if we if we concur take them a, b, c taken together like this within this concatenation operator they can be treated as a 3 bit number.

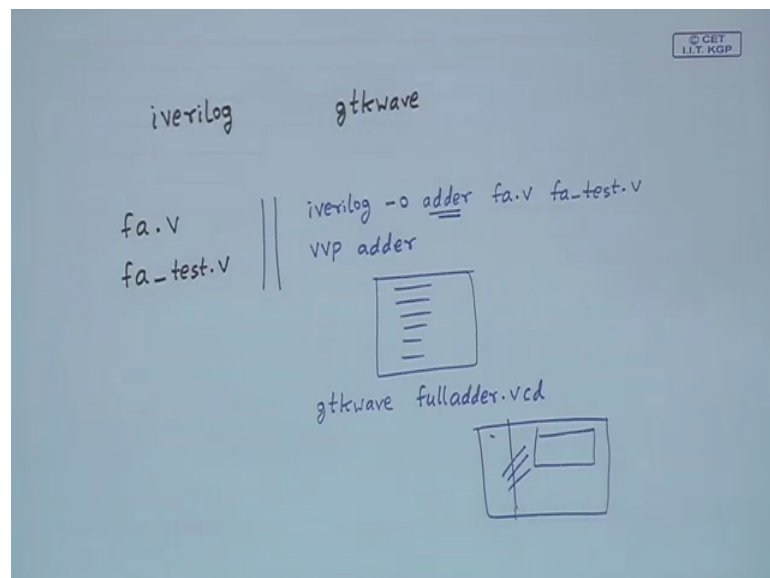
So, in 3 bits numbers can go from 0 up to seven. So, I am writing this fall loop as i equal to 0 up to i less than a; that means, up to i equal to 7 at the end of every loop i equal to i plus 1. So, i 0, 1, 2, 3, 4, up to 7 and I just write a statement like this concatenation; a comma b comma c equal to i. So, this will automatically convert this number i into binary and assign the corresponding last 3 bits to a, b, c; this will be done by the simulator automatically. So, the values will get assigned automatically this like this and

after delay of 5; I am displaying. So, in the display the time is showing us 5, 10, 15 like that I am displaying the current time a, b, c and some and crowd.

So, you can see this is quite convenient because the second approach was a little cumbersome I had to. So, many display statements, but in a loop if I used it is displaying I am just writing display only once, but in a loop the patterns are getting generated automatically there are 8 patterns I am generating 0 to 7, right, now this is the same for loop version, but we have just included dump file and dumpvars command now I want to dump them to a file also. So, we want to see the simulation output in addition this is the change we have done.

So, let us see we have specified that we want to dump them into a file whose name is full adder dot VCD and we have given a statement dumpvars 0 comma test bench. So, what is 0 comma test bench means 0 comma test bench means that in this module whose name is test bench whatever variables are there you dump all of them. So, when you write dumpvars 0 comma test bench. So, here the variables are a, b, c sum and c out all these 5 variables will be dumped.

(Refer Slide Time: 11:28)

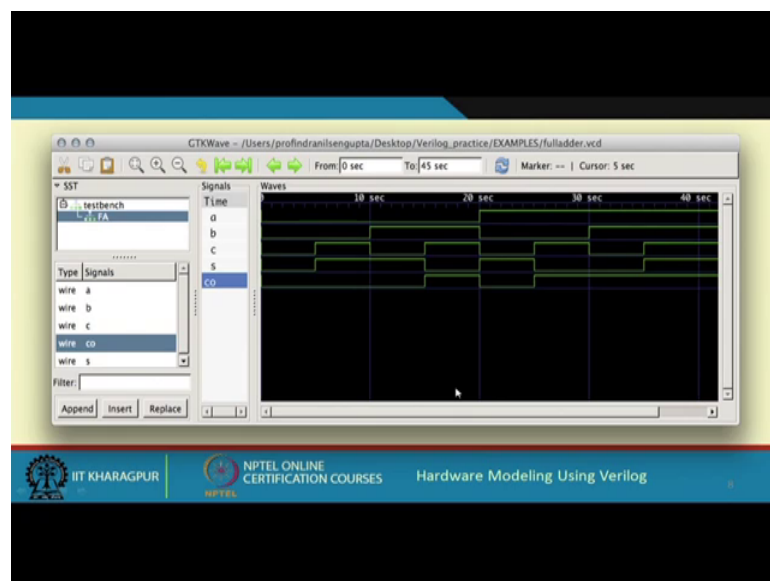


So, well just I recall one more thing that I asked you earlier that you can carry out simulation using the i Verilog tool the i karas Verilog and you can view the waveforms using g t k wave tool. So, just for this example, so let us say my original file was full adder dot v and the test bench that I wrote was full adder underscore test dot v. So, for

simulation what are the commands I am giving first I am writing or I am invoking i Verilog this is a command line parameter. So, I have to open a command window I have to give it. So, I have to give an output file name, let us give the output file name as adder then I give the name of these 2 files FA dot v and FA underscore test dot v, but after this well if there any errors it will show errors there is no errors then the next command will have to give is VVP, this file you see generated adder.

So, once you give this then this simulation output will be generated you can see it on this screen now if you want to say it on the waveform you want to see the waveforms then you will have to invoke gtk wave gtk wave see here we have given a dump file called full adder dot vcd. So, you will have to give the name of the vcd file, gtk wave full adder dot v c d, and then the window will be opened. So, you can click here you can pull down the variables you want to display and you will see the waveforms here.

(Refer Slide Time: 13:48)



Let us see how. So, the first thing is that once you simulate it once you give the vvp command, you will be getting the simulation output like this.

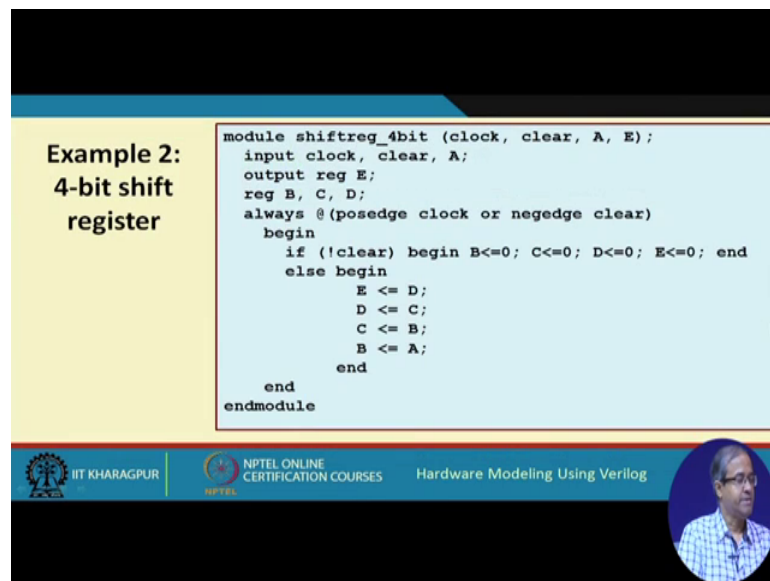
Now, when you run gtk wave, you will be getting the waveform like this let us try to understand this waveform generation here you see on the left here you see the test bench which is the name of the module under it there was a module which is instantiated call FA full adder, the variables which are there they are all mentioned here a, b, c co and s and these 5 variables are being displayed in the axis of time let us go back to the

previous one; you see this was the simulation output at time 5 you applied a, b, c, 0 0 0 at 10, we applied 0 0 1 15 0 1 0 and so on.

Let us see whether it is happening this is my time t equal to 0 you see a, b, c, they are all 0 0 0 here at time t equal to 5; this c is becoming one at time 10; b is becoming 1 and c become. So, 0 0 0 0 0 0; here it is 0 0 1, here it is 0 1 0, then it is 0 1 1, then it is 1 0 0 our f 1 0 1 5 1 1 0 6 like that it will go on right and the sum and carry you can select and let us for example, if it is let us say 1 1; here a is 0 1 1 0 1 1.

So, sum is 0 and carries 1 and here 1 1 1; a, b, c all are one sum is 1 carries also 1. So, you can actually look at the waveform and you can verify the functionality you can see the time source, right.


(Refer Slide Time: 15:40)



**Example 2:
4-bit shift
register**

```
module shiftreg_4bit (clock, clear, A, E);
input clock, clear, A;
output reg E;
reg B, C, D;
always @(posedge clock or negedge clear)
begin
if (!clear) begin B<=0; C<=0; D<=0; E<=0; end
else begin
E <= D;
D <= C;
C <= B;
B <= A;
end
end
endmodule
```

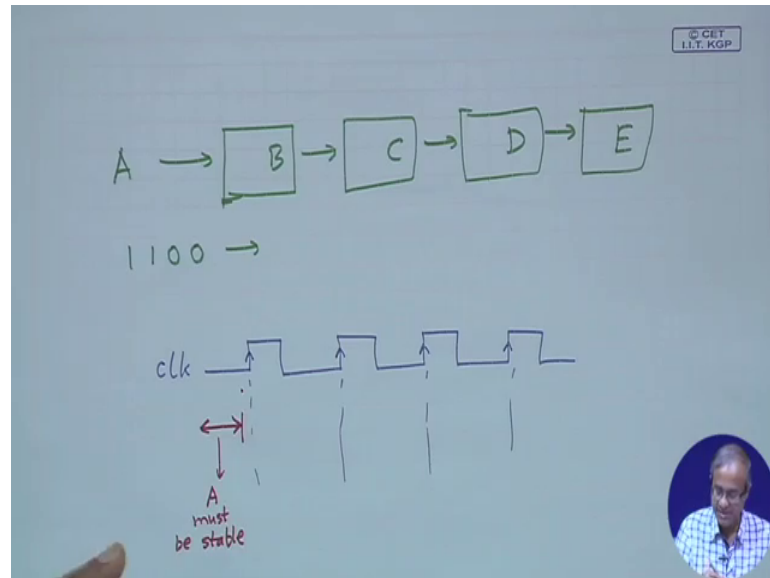
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Let us take another example which is a sequential circuit this is a 4 bit shift register. Now earlier we have seen that how we can use the non blocking assignment statement like this to create a shift register. So, I am showing the same example here. So, it is a 4 bit shift register where the input is a and the final output is e and the intermediate flip flop outputs are b, c and d. So, the description was I am repeating. So, whenever there is a positive edge of the clock or the clear is negative it is negative clear 0; clear you execute this block.

So, if the clear is 0, then you initialize all the flip flops to 0; else you carry out a shifting d will go to e, c will go to d, b will go to c, a equal to b all together because this is non blocking.

(Refer Slide Time: 16:49)



So, the 4 flip flops they are appearing. So, they are B, C, D and E; this is B, this is C, this is D and this is E and you are applying A from outside, right, this is how the shift register is supposed to work.

(Refer Slide Time: 17:15)

```
module shift_test;
  reg clk, clr, in;  wire out;  integer i;
  shiftreg_4bit SR (clk, clr, in, out);

  initial
    begin clk = 1'b0; #2 clr = 0; #5 clr = 1; end

  always #5 clk = ~clk;

  initial begin #2;
    repeat (2)
      begin #10 in=0; #10 in=0; #10 in=1; #10 in=1; end
    end

  initial
    begin
      $dumpfile ("shifter.vcd");
      $dumpvars (0, shift_test);
      #200 $finish;
    end
endmodule
```

The slide shows a Verilog code snippet for a shift register testbench. The code includes module declarations, register and wire declarations, and timing constraints for the clock and clear signals. It also includes a sequence of input values for the shift register. A small inset video of a speaker is visible in the bottom right corner of the slide.

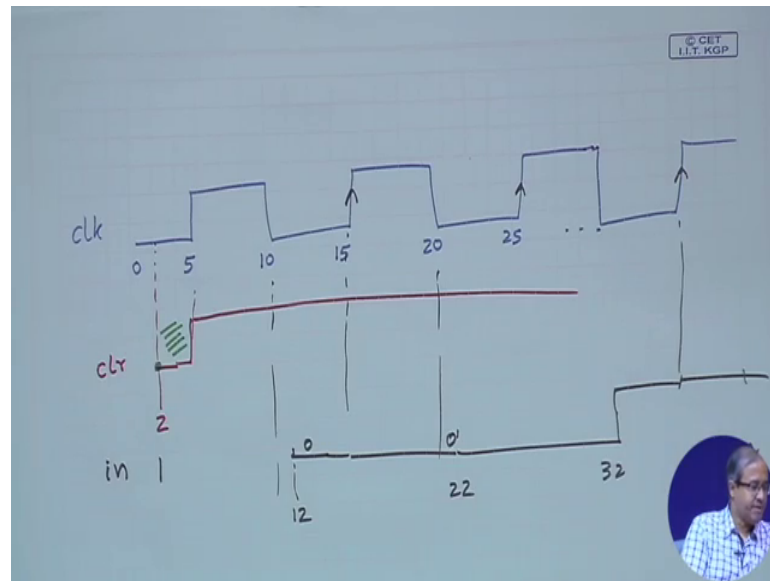
Now, for this; we want to write a test bench this is a test bench, we wrote let us try to understand what we are trying to go here see in this shift register the inputs are clock and clear and the other side the outputs A, B, C, D.

So, we have instantiated this shift register clock and clear this A and E here, we are calling it in and out, right, fine. So, this part I am explaining here, let us see what we are trying to do we are trying to apply some input 0 0 1 and 1 like we have a 4 bit shift register, we are trying to apply first a 0, then a 0, then 1, then I want and we want to shift these 4 bits inside the register now you see one thing we have to ensure in the shift register description where here you assume that is triggered at the positive edge of the clock pause edge here I am calling it clk. So, you see suppose my clock signal is coming like this. So, the shift register is supposed to be shifting and the positive edge of the clock. So, what I must ensure is that before the positive edge comes before that here before that my input must be stable here I have only one input a; a must be stable .

So, what I mean to say is that it must not be like this that the signal a, let us say this clock; we are applying and at the same time, this signal a which is there; it should not happen that a is also changing at the same time the clock is coming because whenever the clock is coming if a is also changing at the same time there will be confusion because this a and the clock changing maybe the previous value will be taken. So, a must be applied a little before the clock is coming stabilized and then you apply the clock then only the correct value of a will be shifted into register right this is why I mean I mean to say.

So, now, let us see this. So, what I have done here I have initialized clock to 0 because I am not given in delay it is at time t equal to 0 at time 2 I have set clear to 0 at time 5, I again has set clear to 1. So, I have cleared the register at time 5 because you see clear was a neage clear. So, here it was neage at time 0 at time t equal to 0 the counter be cleared. So, at 5 again I am releasing the clear making it one well and then I am saying at 5 clock equal to not clock.

(Refer Slide Time: 20:49)



So, let us try to see what is happening here I am generating a clock signal like this is clock. So, at time 0, I am making it at 0 this is time 0 and at 5 always at after delay 5 clock equal to naught clock. So, after delay of 5, I am making it one after another delay of 5, 10, I am making it 0, again at 15, I am making it again 1 20 again 25 and so on like this; the clock is changing, right, now here in between what I have done; I have set clear to 0 at time 2.

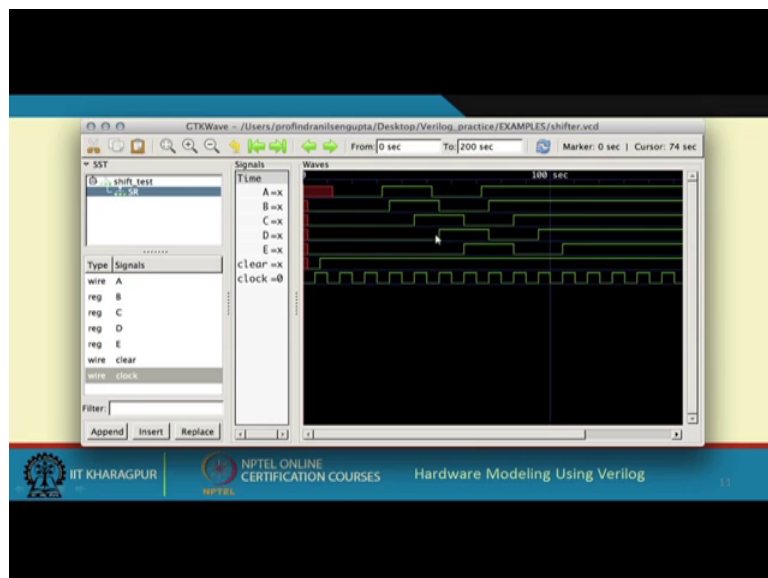
So, at time 2, let us say somewhere here; this is time 2, I have made clear equal to 0 and again at time 5, I have made clear equal to 1 at time 5, I have again made clear equal to 1 and I leave it at 1. So, what it means is that during this period the counter will be cleared because it has got a 0 edge here it has become 0 out here fine now what I am doing; hey let us see these 4 bits 0 0 1 1; I am shifting in this initial block, I am initially giving a delay of 2, right, then after delays of 10, 10, 10, 10, I am shifting before bits 0 0 1 1; what does this mean, let us see now. So, we are giving an initial delay of 2. So, initial deal of 2 means we are here 2 this is 10, this is 20, then you are giving a delay of 10 after that; that means, it will be here at time twelve then again we are giving a delay of 10. So, it will be here 22.

So, you see after the falling edge comes, I am applying the next input here. So, this in the first in equal to 0, I am applying here at 12, I am applying 0, then at 22, I will be applying again as 0, then at 32, later on I will be applying a one then again at 42, later I

will again apply a 1 0 0 1 1; you say the points I am applying; I am ensuring that when the next clock edge comes the input is absolutely stable when the next clock edge comes it is stable you see just look at the next one also 20. So, when the next clock edge comes this input is stable. So, there is no confusion of this edge with the input varying, right.

So, we have written our test bench in such a way that this delays are automatically adjusted the inputs are stable adjust well before the clock edge comes clock edge is coming at 15, but at time twelve of my input is stable clock edge is coming at twenty 5 at time 22, my input is stable it is coming at 35, at 32 this stable and in the here in the last initial block of my code, we have just specified the dump file and said that at 200 will finish.

(Refer Slide Time: 24:23)

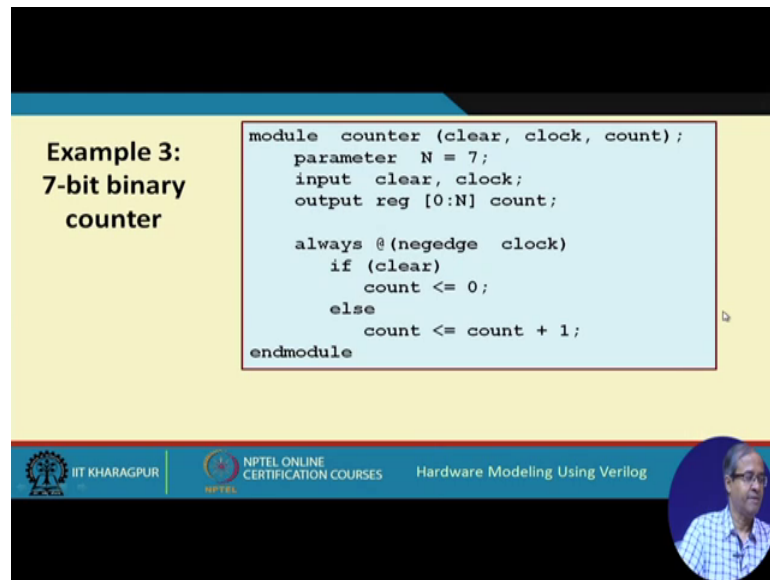


So, you just see the waveform this is what I wanted you to see; you see clock, this is the clock signal which I am showing clear you see clear there is a small gap here. So, at time 2, it has become 0. So, when it has become 0 all A, B, C, D has become 0s cleared; A is 0 not A B 0 C D; this for shift register outputs have been cleared. Now the thing will start from time 12. So, this is 10 next clock, when it comes you see here; it will start counting this a is 0 0 will be shifted.

So, you apply 0; here 0 will be shifted, apply 0 again, it will be shifted, then you apply a 1 1 will be shifted is we apply a one you see these were these are getting shifted 0 0 1 1 0 0 1 1 0 0 1 1, they are getting shifted; A is getting B, B to C, C to D, D to E. So,

this will go on. So, you see it this 0 0 1 1; I am repeating 2 times repeat 2. So, 0 0 1 1 0 0 1 1 that is why you see 0 0 1 1 0 0 1 1, then it remains one and it nicely gets shifted, right, fine.

(Refer Slide Time: 26:04)




**Example 3:
7-bit binary counter**

```
module counter (clear, clock, count);
    parameter N = 7;
    input clear, clock;
    output reg [0:N] count;

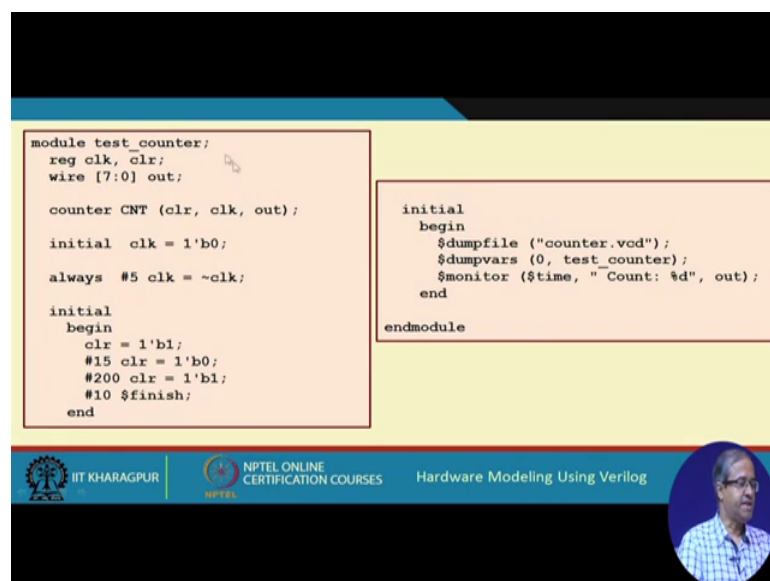
    always @(negedge clock)
        if (clear)
            count <= 0;
        else
            count <= count + 1;
endmodule
```

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



So, now let us take another example. So, it is a 7 bit binary counter. So, clear clock and count. So, at neage clock whenever clock is going from 1 to 0, if synchronous clear; clear is 1, I am clearing otherwise incrementing. So, the test bench you are writing like this.

(Refer Slide Time: 26:26)



```
module test_counter;
    reg clk, clr;
    wire [7:0] out;


    counter CNT (clr, clk, out);

    initial clk = 1'b0;
    always #5 clk = ~clk;

    initial
        begin
            clr = 1'b1;
            #15 clr = 1'b0;
            #200 clr = 1'b1;
            #10 $finish;
        end

    initial
        begin
            $dumpfile ("counter.vcd");
            $dumpvars (0, test_counter);
            $monitor ($time, "Count: %d", out);
        end
endmodule
```

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



So, for counter it is very simple. So, initially we are saying clock equal to one in one initial block. So, in another always block after delay of 5 clock equal to not clock.

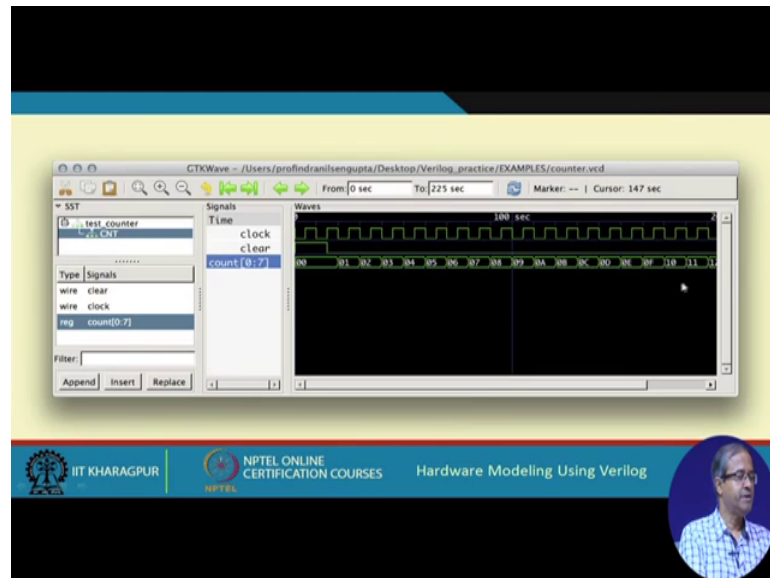
So, in the last initial block, we are specifying the dump files and you are saying the monitor, we want to see time and count value and here we are specifying that initially we are clearing the counter and at time 15 you are making it 0 again. So, the counting should start from time 15; till then clear was 1 and it will continue until 200. So, 200; again I am setting clear to 1; it will be again cleared; then after time 10 I will finish.

(Refer Slide Time: 27:17)

0 Count: 0	140 Count: 13
20 Count: 1	150 Count: 14
30 Count: 2	160 Count: 15
40 Count: 3	170 Count: 16
50 Count: 4	180 Count: 17
60 Count: 5	190 Count: 18
70 Count: 6	200 Count: 19
80 Count: 7	210 Count: 20
90 Count: 8	220 Count: 0
100 Count: 9	
110 Count: 10	
120 Count: 11	
130 Count: 12	

So, if we again simulate this code, similarly you will see that the output generated will be like this you see you are monitoring time and count. So, there is a time and there is the count the time will go from 0 and count will start you see here before 15 count will not start bigger is cleared. So, only after 15 next clock comes, it will start counting. So, from twenty the count starts increasing then clock rate is 10, 30, 40, 50; 1 1 1; it is incrementing.

(Refer Slide Time: 27:57)



So, it will continue till 200; 200 we are again clearing it. So, after 200 after time 10, it becomes 0. So, the timing diagram if you view, it will look like this you see clock is coming, this is clear; the clear will be cleared at time 15 till then the count value as 0 you see 0 0 written here then with every clock falling edge; it becomes 0 1 next falling at 0 2, 0 3, these are the count values, this is shown in hexadecimal 3, 4, 5, 6, 7, 8, 9; a, b, c, d, e, f.

(Refer Slide Time: 28:38)

**Example 4:
Automatic
verification of
output**

```
module fulladder (a, b, c, s, cout);  
  input a, b, c;  
  output s, cout;  
  
  assign s = a ^ b ^ c;  
  
  assign cout = (a&b) | (b&c) | (c&a);  
  
endmodule
```

The slide displays a Verilog module for a full adder. The module has three inputs: 'a', 'b', and 'c'. It has two outputs: 's' (sum) and 'cout' (carry out). The sum 's' is calculated as the XOR of 'a', 'b', and 'c'. The carry out 'cout' is calculated as the OR of the ANDs of 'a' and 'b', 'b' and 'c', and 'c' and 'a'. The module is named 'fulladder' and ends with 'endmodule'.

And so on; it will continue. So, the counter is counting correctly you can see verify now we are taking an example where we want to verify the output automatically. So, we take the example of the simple full adder again full adder sum and carry.

(Refer Slide Time: 28:53)

```

module fulladder_test;
  reg a,b,c;
  wire s, cout;
  integer correct;

  fulladder FA (a,b,c,s,cout);

  initial
  begin
    correct = 1;

    #5 a=1; b=1; c=0; #5;
    if ((s != 0) || (cout != 1))
      correct = 0;

    #5 a=1; b=1; c=1; #5;
    if ((s != 1) || (cout != 1))
      correct = 0;

    #5 a=0; b=1; c=0; #5;
    if ((s != 1) || (cout != 0))
      correct = 0;

    #5 $display ("%d", correct);
  end
endmodule

```

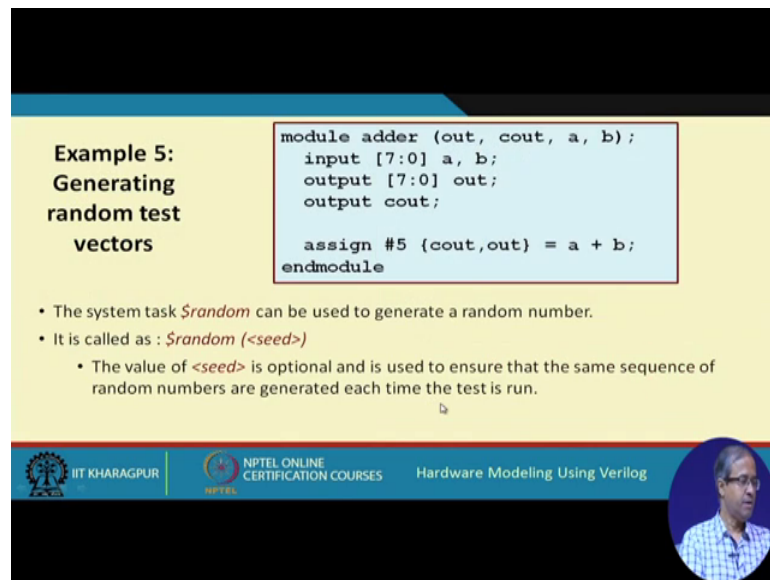
Shall display 1 if outputs are correct; and display 0 otherwise.

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

Let us say we write the test bench like this, we write the test bench as follows we instantiate the full adder we declare a variable called correct of type integer initialize it to one in the initial block correct equal to one then at time 5, let us say after delay 5, we apply a 1 b 1 c 0; right, well, we give a delay then we check for a, b, c, 1 1 0, this sum is supposed to be 0 and carries supposed to be 1, right. So, we are checking if sum is not equal to 0 or carry is not equal to one then you set the variable correct to 0 then apply the next vector 1 1 1 again give a delay, then if sum is not equal to 1 which is supposed to be carrier is not equal to 1; then again you said carry to the correct to 0 and then the last vector 0 1 0 0 1 0 supposed to be give someone and carry 0. So, if sum is not equal to 1 carry not equal to 0; correct; equal to 0 at the end you just display correct. So, it will be displaying either 0 or 1.

So, I mean if everything is correct, the full adder design is correct then you should get a one in the output, but if any of the outputs are not matching then you should get a 0 as the output, right. So, this is a self checking test bench you can verify the output and compare and print this summary result directly.

(Refer Slide Time: 30:32)




**Example 5:
Generating
random test
vectors**

```
module adder (out, cout, a, b);  
  input [7:0] a, b;  
  output [7:0] out;  
  output cout;  
  
  assign #5 {cout,out} = a + b;  
endmodule
```

- The system task *\$random* can be used to generate a random number.
- It is called as : *\$random (<seed>)*
 - The value of *<seed>* is optional and is used to ensure that the same sequence of random numbers are generated each time the test is run.

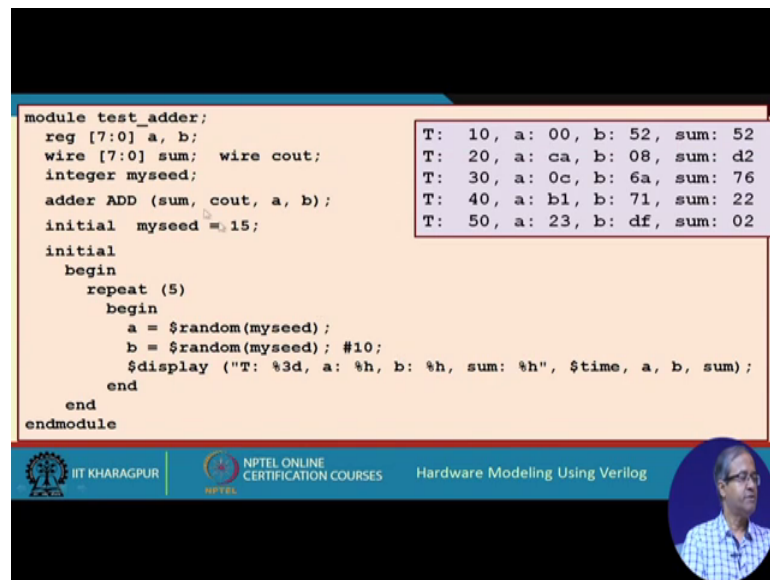
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Now, the last example, I take is one of generating random test vectors. Now here let us say we take an adder example that 2 2 8 bit numbers a and b the output is a is an 8 bit sum and c out is the carry. So, this is a behavioral description carry out and out the 9 bit sum with the carry is equal to a plus b. Now we want to test it with some random input pattern say b, there is a function called dot dollar random or task sometimes called also which can be used to generate a random number this function can be called with a parameter seed also this is optional you may or may not give this seed.

So, if you give a particular seed it means that the same sequence of random number will be generated every time.


(Refer Slide Time: 31:47)



```
module test_adder;
  reg [7:0] a, b;
  wire [7:0] sum; wire cout;
  integer myseed;
  adder ADD (sum, cout, a, b);
  initial myseed = 15;
  initial
  begin
    repeat (5)
    begin
      a = $random(myseed);
      b = $random(myseed); #10;
      $display ("T: %3d, a: %h, b: %h, sum: %h", $time, a, b, sum);
    end
  end
endmodule
```

T: 10, a: 00, b: 52, sum: 52
T: 20, a: ca, b: 08, sum: d2
T: 30, a: 0c, b: 6a, sum: 76
T: 40, a: b1, b: 71, sum: 22
T: 50, a: 23, b: df, sum: 02

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



So, that you can rerun and reproduce the test you can verify whether it is working in the same way, but if you do not give a seed then it will be really a random thing if you run it again some other input will come. So, let us see the example here see here we have instantiated the adder add a b these are reg; this will be the inputs to the other and some I have declare as vars, this c out is also a vars and I declare an integer called my seed. So, which I initialize to 15 initial my seat 15. So, in this initial block I repeat 5 times I apply 5 random patterns repeat 5 is what I do a equal to dollar random my seed I generate a random number put it to a.

So, I put another random number put it to b give a deal of 10 then display, what I displayed time t a b in hexadecimal and some also in hexadecimal. So, you see that time 10 the first random number that that was generated was 0 0 and 5 2. So, if we add them, sum is 5 to second time, it was c a and 0, it say a and 8 is b, c, d, e, f 0 1 2 root 2 and carry to making d. Similarly 0 c and 6 is 7 6 b one and 7, if we add some will be 2; 2 there will be a carry out to 3 and d f if we add some would be 0 2. So, in this way, you can generate random numbers and you can just apply to your design under test.

So, with this we come to the end of this lecture, well, we have seen a number of examples of writing test benches. Of course, this is not the end of thing; we will be taking more examples later and whenever we talk about a design. We shall also be showing the test bench side by side. So, you will always be getting a flavor of something

new on the test benches that will be discussing the future also, but till then we stop for now.

Thank you.