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

Lecture No #07
Verilog: Part 6

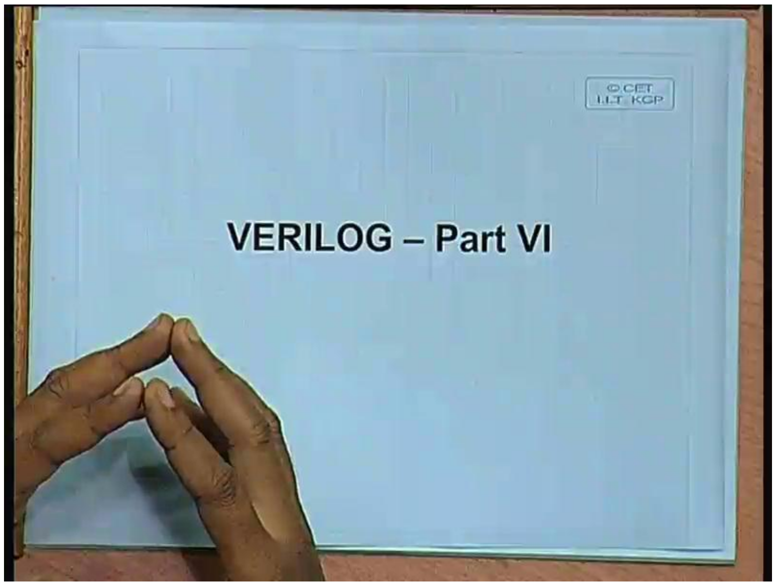
(Refer Slide Time: 01:11)

Lecture #7

Verilog : Part VI

So in this sixth lecture on the language verilog we will be discussing today some additional features of the language. Namely if you recall earlier we had mentioned that when you have memory as part of a design system, we usually incorporate it as an instantiated model. We take the memory block as a pre designed one available in the library. We can simply pick it up and put it in our design. So we will see today an alternate way of representing and using memory. For instance if we need a very small amount of memory in a design, we do not find a matching memory block in the libraries so in that case what to do?

(Refer Slide Time: 01:57)

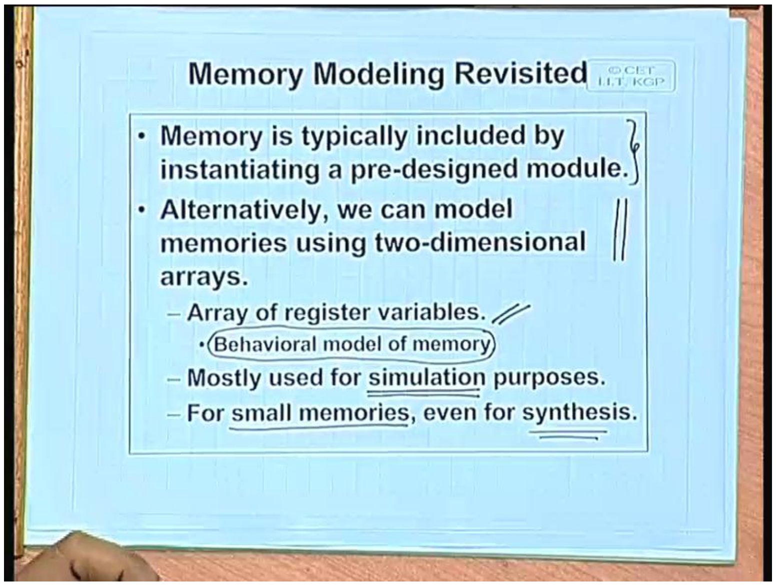


© CET I.I.T. KGP

VERILOG – Part VI

And after that we will be looking at how we write the Verilog test benches and then what are the different specific things that need to be understood whenever you want to write and use a verilog test bench along with a design module okay. So we start with having another look on the modeling of memory systems. Now as I had mentioned earlier if you recall memory is included in a design by instantiating predesigned module which can be picked up from a library okay.

(Refer Slide Time: 02:45)

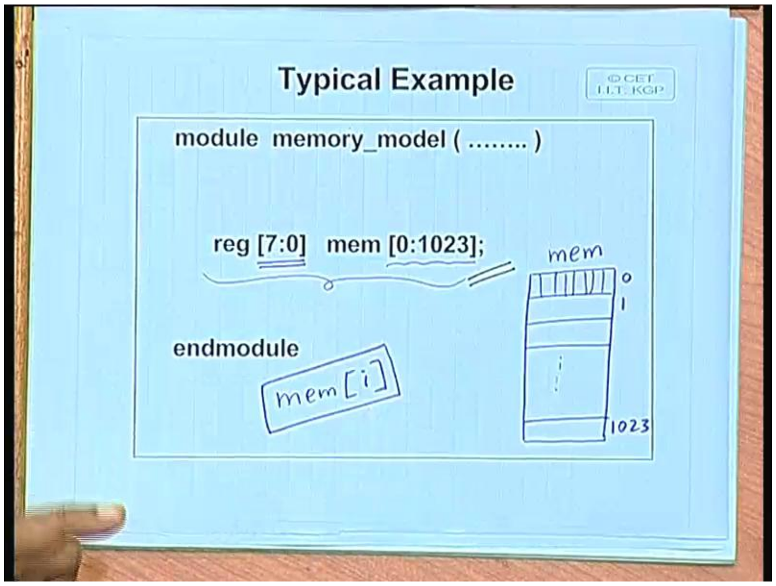


Memory Modeling Revisited

- Memory is typically included by instantiating a pre-designed module.
- Alternatively, we can model memories using two-dimensional arrays.
  - Array of register variables.
    - Behavioral model of memory
  - Mostly used for simulation purposes.
  - For small memories, even for synthesis.

Memory is designed or included by instantiating a predesigned module. But this is possible provided you have such a memory module available in the library and the corresponding size of the memory module matches with your requirement. So as an alternative what we can do, the language verilog also supports this. We can module memory using 2 dimensional arrays. Two dimensional arrays is essentially an array of register variables you can treat it as a register bank. So this is you can regard it as the behavioral model of the memory system well actual memory system is not just an array of registers it is something more than that there is some special layout and special way they are fabricated. But this view of the memory is often used for simulation purposes, but when you are talking of fabrication when your ultimate aim is to design and fabricate the chip. Then you would have to instantiate a proper memory block from the library. But if you have small memory requirements in a design, then you can use it even for synthesis. This alternate way modeling memory as a 2 dimensional array. Now let us see how we can do this.

(Refer Slide Time: 04:19)



Typical Example

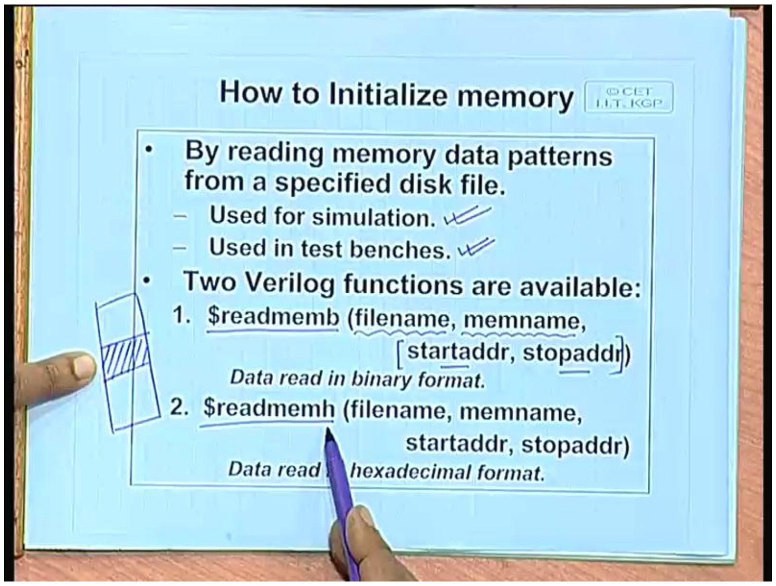
```
module memory_model ( …….. )

reg [7:0] mem [0:1023];

endmodule
```

© CET I.I.T. KGP
mem
0
1
1023
mem[i]

First a typical declaration. This is how we can declare a 2 dimensional array or you can say an array of register variables. Here this declaration says that this is a register type variable and the size of the register type variable is 8 bits and it is an array. So actually what will be there this will be treated as an array of 1023, 1024 registers with index going from 0 up to 1023. There will be so many registers the name of this 2 dimensional array will be given as mem and each of the words will be of size 8 okay. So this is how we can define a 2 dimensional array of storage locations which can be treated as an array of register variables.

So if we access this structure as say mem of say I this will actually return a register type variable of size 8 which logically. Means the contents of the ith location of the memory so we can model a memory system behaviorally by this and you can access the ith location using simply mem I. This is what you can do. But I have said that you typically use this kind of a memory modeling when your target is to simulate. That is the most common scenario. So next we shall see that once we have designed such a memory system. So how we can initialize it.

(Refer Slide Time: 06:24)



How to Initialize memory

- By reading memory data patterns from a specified disk file.
  - Used for simulation.
  - Used in test benches.
- Two Verilog functions are available:
  1. $readmemb (filename, memname, [startaddr, stopaddr])
     Data read in binary format.
  2. $readmemh (filename, memname, startaddr, stopaddr)
     Data read in hexadecimal format.

Initialization can go like this. See this initializing memory what we are talking here, This is from the point of view of simulation typically what happens in case of simulation is that, you have a memory system which you have included in your design and you have say some data that you want to load into the memory system for the purpose of initialization. So one alternative, of course you can make explicit assignments of the values to the memory locations and make the initialization. Now as an alternative and this is possible only for simulation again I am telling you that you can do 1 thing. You can specify the name of a file on the disc and you can read the values of the memory location from the disc for the disk and you populate and load the memory with those values.

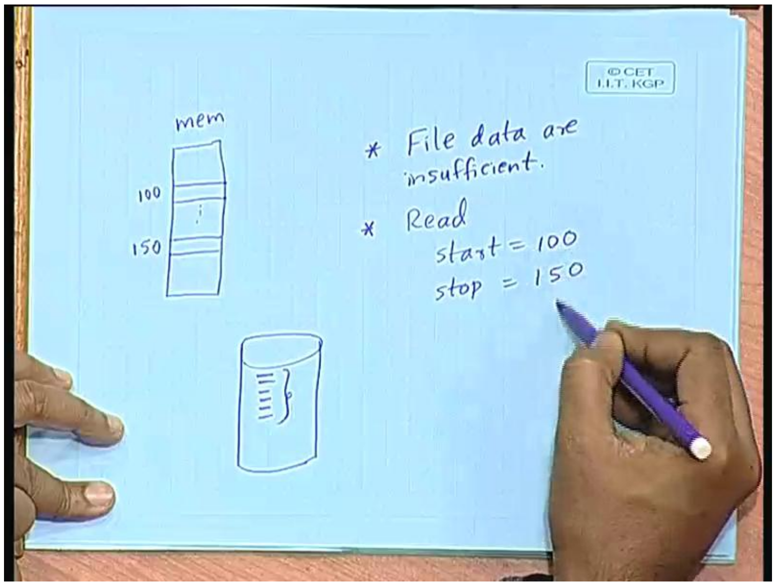
So this is one very common way of initializing memory by reading the data patterns from a specified disc file. This as I told you this is used for designs of where you only want to simulate something not synthesize or we will see later that when you are using a test bench means I want to generate test patterns that I need to apply to a circuit I have already designed for the purpose of testing. There also the test affecters can be available in a disc file I can read from the disc I can apply them to the circuit okay. So these are the 2 different situations where we can read the patterns from a disc file and load the memory with it in Verilog there are 2 functions that are available readmemb and readmemh b and h.

Basically refers the radix of the number that are ready b means binary h means x. Well if it is binary then your file will be containing a streams of zeroes and one's only. If it is h it will contain hexadecimal characters. Now the parameters are very similar parameter.

The first parameter specifies the name of the file from where the data has to be read the second parameter. Means the register or the 2 dimensional array that you have defined in your Verilog program, the last 2 parameters are optional. Well if you do not specify that then the entire memory what ever you have defined that will be populated by reading the data values. But you have an alternative you can specify the start address and stop address of the memory and you can load only that part of that of the memory.

So if you have a bigger memory you can choose to load only certain part of it. The best possible way is to use scratch pad you need not initialize that okay. So if you have such a requirement you can specify both the starting and the stopping address of the memory and the data which you get from disc they will be loaded in this area so readmemb and readmemh both are similar in terms of its usage. The difference again I am telling you is here we are using binary data here we are using hexadecimal data okay. So the question is that if the file search does not matches with the memory size other things okay. Generally 2 things the first is that you have a memory defined.

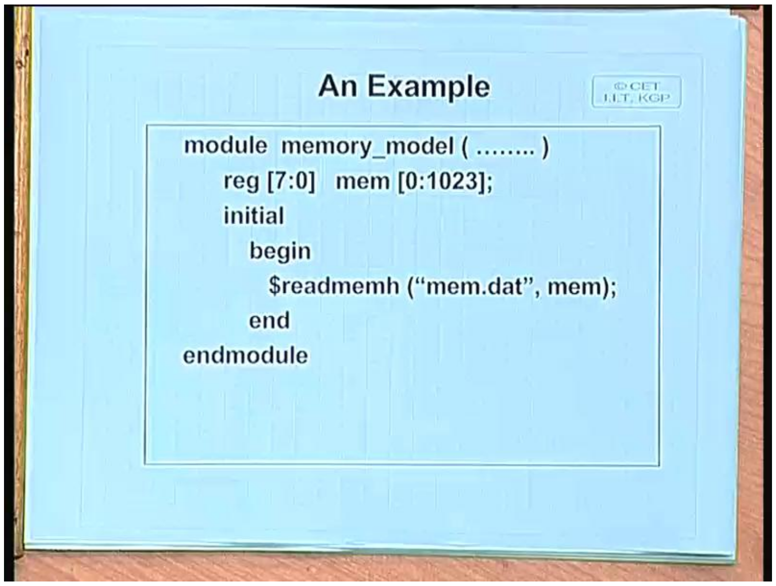
(Refer Slide Time: 10:08)



Say this is your memory lets say the name is mem. There may be 1 alternative where the file data the data which are there in the file are insufficient. Insufficient means whatever data you are asking to read from the file there are so many data. So this will give a file system error that you do not have so many data in the file. There will be error will be given. and alternatively, suppose you have the other way round the file content more data or in 1 particular line. You have possibly missed some digit. But it will not check further it will read sequentially the digits as they appear and moreover suppose you say this is a disc file and you give a command to read say from start address hundred to stop address say 1 50.

Now in the disc well you will be reading from the beginning not from offset 100. But you will be reading the first 51 words or numbers whatever size they have defined them to be and you will be loading them into this mem array starting from index 100 up to 150. So here there can be 2 things. One is that you do not have sufficient data in the file of course again that you have given flagged and the other thing is that start and stop what ever boundary you have given here the mem is not defined for somehow those range values there also similar to this error. That means you are trying to access something out of the boundary. So similar sense what can I interpreted way it can give all the error messages where ever you can count is determined sequentially right.

(Refer Slide Time: 12:19)



An Example

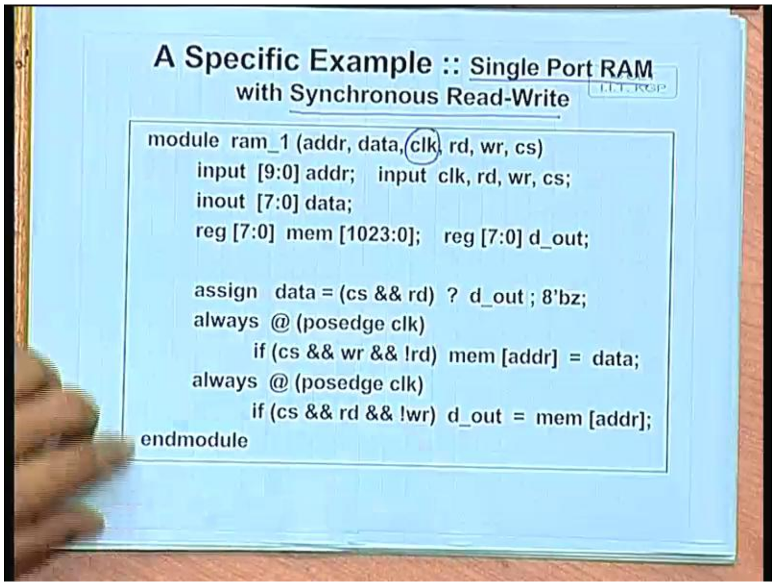
```
module  memory_model ( ........ )
      reg [7:0]   mem [0:1023];
      initial
         begin
            $readmemh (“mem.dat”, mem);
         end
endmodule
```

So a simple example. So you have defined a memory system 1024 words 8 bits each and you can give a command like this readmemh, filename, name of the variable in the third and fourth parameter. I am omitting those are optional. So entire memory will be populated by data from the given file okay fine. This is not the instantiation of the module. This I am defining the memory as a 2 dimensional array as an array of registers okay. Now let us,

Student: sir time alone is first memory memory is specialized memory or just suggested.

When you say that I am instantiating a memory from the library that is not a register array. That is the entire specialized memory design and fabricated. So that means at the high level you will be talking about the behavioral specification when you are simulating. But you can use that same module up to the layout level because it is available up to the layout. That detail is available in the library. For a example we will look at a couple of memory types how we can model them in Verilog.

(Refer Slide Time: 12:44)



A Specific Example :: Single Port RAM with Synchronous Read-Write

```
module  ram_1 (addr, data, clk, rd, wr, cs)
        input  [9:0] addr;    input  clk, rd, wr, cs;
        inout  [7:0] data;
        reg [7:0]  mem [1023:0];    reg [7:0] d_out;

        assign   data = (cs && rd)  ?  d_out ; 8'bz;
        always  @ (posedge clk)
                if (cs && wr && !rd)  mem [addr]  =  data;
        always  @ (posedge clk)
                if (cs && rd && !wr)  d_out  =  mem [addr];
endmodule
```

So the first example we take is a single port RAM single port. RAM means that there is only 1 read write access port with synchronous read write synchronous read write means all read write are taking place in synchronism with the clock. So here you can say in this module there is a signal called clock there are read write chip select as usual there are therefore memory chips we have the address and the data. Now in this example description we have defined this address to be of 10 bits and beta to be of 8 bits. So there are 1024 locations of 8 bits each clock read write c s is that all input signals data can be inputs or outputs. So other than input and output there is another kind of declaration in out in out means it is a bidirectional signal and verilog we can also define the signal as in out.

Now here we are defining memory as a behavioral block reg mem. So it is a 2 dimensional array since the address is of 10 bits we are defining mem of size 2 to the power 10 right. And the data which is which will be outputted during read operation that we are defining it as a reg value d out. Now you see why we have given first 1 is a continuous assignment statement data is assigned a value depending on chip select and read the data will be assigned a value only. If it is a read operation which means that the chip select is activated we are assuming that all the signals are active high in this example. So if chip select is true and read is true then what ever d out has been computed that will get assigned or else this is tri state 8 z or else you force it to the tri state value which means you do not assign anything leave it as it is.

Now you see there are 2 concurrent always block running both are triggered by the positive edge of the clock. The first 1 is for the memory write the second 1 is for memory read. Memory write means here is the condition chip select write is active and read is not active. Because my mistake both read and write may be active at the same time just by checking that you check that the read is low. If it is so then you the data which ever what ever is coming from outside that gets written into mem addr, that is memory write operation and for memory read you again check if it is chip select and read and not write. Then you read the contents of the memory and do not store in data rather store it in d out storage in data will be done synchronously here okay. So this is 1 typical example where you have synchronous read write. But if you have asynchronous read write which is common to you can say most of the chips which we use today. There are no clocks available with the memories. There we would not use these statements triggered by posedge clock but rather Yes.

Student: Instead of posedge can we use, we always use the sign d out.

Instead of d out we have used data here.

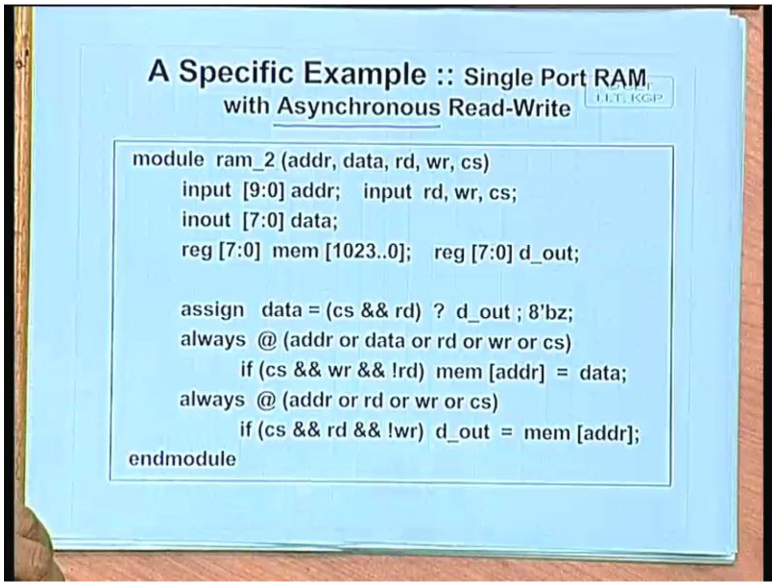
Student: Instead of d out in the always clock we use the data there yes sir.

Data out here.

Student: Sir, yes and usually we assign data out.

And here also you write data but under yeah it is fine. But there is only 1 thing this is explicitly done due to the reason that when you are not doing any read or write anything I want the data line to be tri stated. Somehow this is 1 way of doing it that you can do it in some other way also that only chip is not selected or neither it is read or write is not active. So we just want the data line to be tri stated. This is just 1 way of doing it you can do it some other way also it is not the only way and if you have asynchronous read write now, then you can make some small change to be designed.

(Refer Slide Time: 18:36)

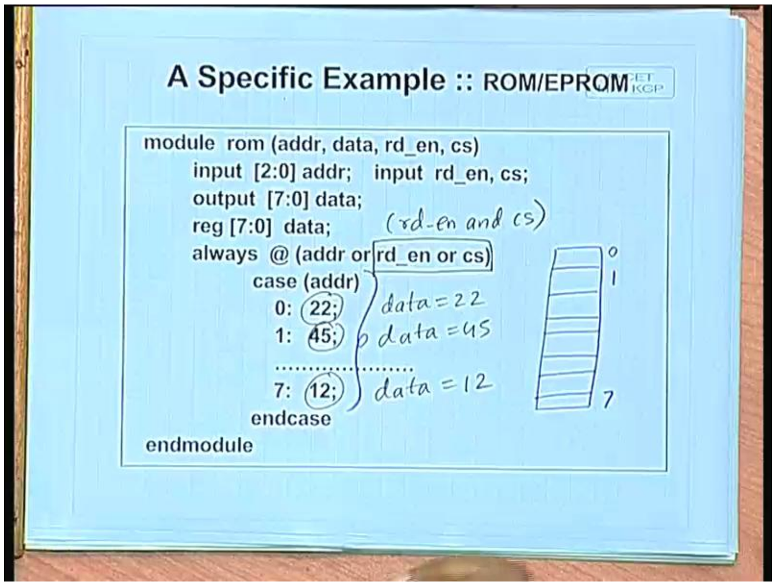


A Specific Example :: Single Port RAM with Asynchronous Read-Write

```
module  ram_2 (addr, data, rd, wr, cs)
        input  [9:0] addr;    input  rd, wr, cs;
        inout  [7:0] data;
        reg [7:0]  mem [1023..0];    reg [7:0] d_out;

        assign   data = (cs && rd)  ?  d_out ; 8'bz;
        always  @ (addr or data or rd or wr or cs)
                if (cs && wr && !rd)  mem [addr]  =  data;
        always  @ (addr or rd or wr or cs)
                if (cs && rd && !wr)  d_out  =  mem [addr];
endmodule
```

This is the same memory module. But with asynchronous read and write. So you observe the difference now in the parameter you have no clock only read write chip select this assign is as usual the same. This always blocks the body of the block is the same. But the triggering condition is different now here you execute this if either the address changes or the data changes or there is some change in read write or cs see this is what happens in a typical memory chip you use. If you change the address immediately the data out changes that is the behavior okay. So here you can modulate like this. Similarly for read if any of the signals change into the mem may be any of the signal changes it is not really a write then this condition will not evaluate to true anyway write will not take place. But you will be checking it whenever there is some change in 1 of the inputs right. So these are for RAM chips RAM systems. But if you have a ROM or EPROM where you only want to read something the data storage is permanent then you often modulate like this.

(Refer Slide Time: 19:58)



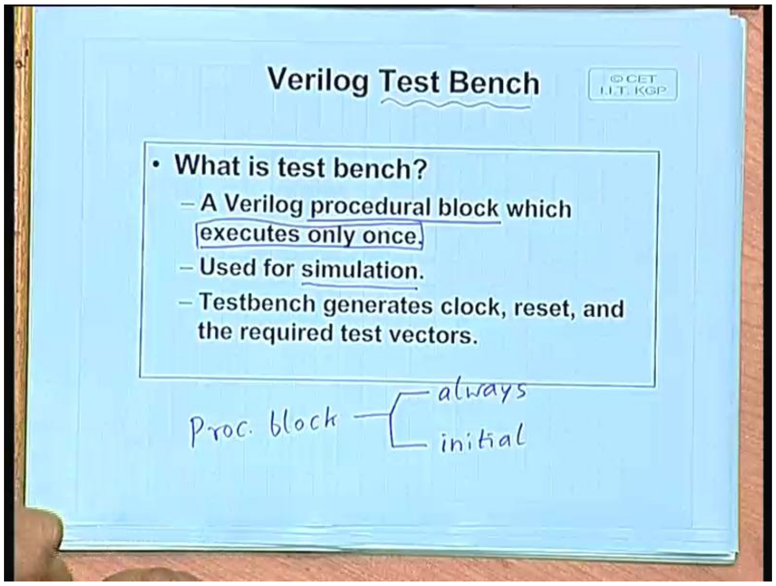
A Specific Example :: ROM/EPROM

```
module  rom (addr, data, rd_en, cs)
        input  [2:0] addr;   input  rd_en, cs;
        output  [7:0] data;
        reg [7:0]  data;          (rd-en and cs)
        always  @ (addr or rd_en or cs)
            case (addr)
                0:  22;    data = 22
                1:  45;    data = 45
                ....................
                7:  12;    data = 12
            endcase
endmodule
```

Say here you have a ROM where you have a address data you have a read enable and chip select of course read enable is optional. But here assume that you have both read enable and chip. Select both has to be activated in order to read the data. So just for the sake of illustration I have assumed a very small memory module which consists of 8 locations the address is ranging from 0 up to 7. So number of address lines will be 3 read enable and cs will be inputs and output are 8 bits each word is of 8 bits. This we are defining as register type and in an always block we say that when ever address or read enable or chip select changes well not changes yeah. Whenever address I think see inside the block there is a case statement depending on the address and applied line I am saying that address 0 contains 22, one contains 45, 7 contains 12 and so on. So this will be the data that will be actually I have written this incompletely.

So actually we will have to write data equal to 22. Not simply 22. So this will be data equal to 22, this will be data equal to 45 and data equal to 12, Yes, yes, yes. So this condition is also not correct. I believe this will say if either the address changes or both. These 2 are active so read enable and c s something like this. Then only you read the data okay fine right. So we have seen that means how we can model memory for our, you can say basically say modeling purposes for most smallest designs, we can model memory like this. As I said again if your purpose is to simulate for simulation you can use this type of memory modeling. So at least up to the behavioral verification design simulation you can use this model. But after that while synthesis you can instantiate the blocks from the library and you can start it doing from there. But now 1 thing which we have been talking about. So after you have written the program you want to check it through simulation you need to write a so called test bench.

(Refer Slide Time: 22:57)



Verilog Test Bench

©CET I.I.T. KGP

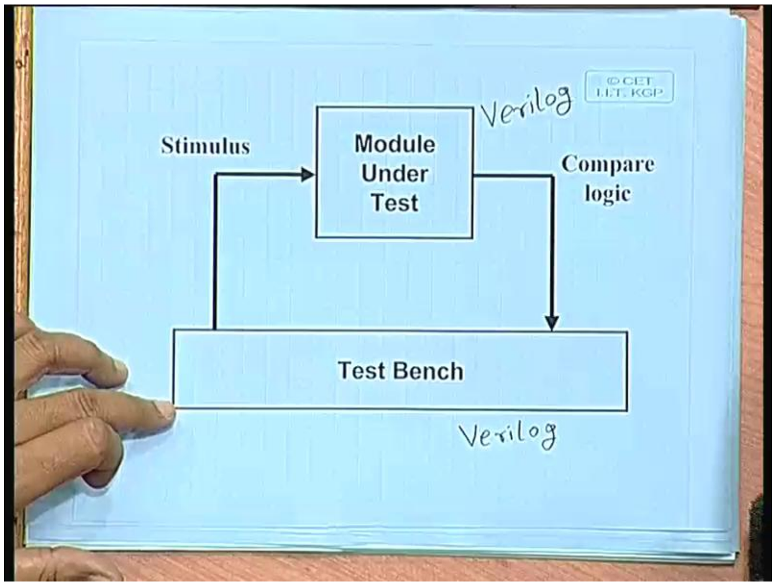
- What is test bench?
  - A Verilog procedural block which executes only once.
  - Used for simulation.
  - Testbench generates clock, reset, and the required test vectors.

Proc. block — always / initial

So now let us see what is a test bench and how it really is written see a verilog test bench is essentially a verilog program. Make this point it is a verilog program, it is a procedural block like the always statement. But the difference is that an always procedural block executes indefinitely. It executes in an infinite loop. But a procedural block that is used in a test bench is executed only once this is the main difference initial block. So I had said earlier that there are 2 coins of procedural blocks 1 is always which we use normally for design modeling and the other is called initial. Now it is the property of this initial block that it is executed only once and this is what we need when you are trying to run the test bench because what ever we have written in the test bench file that has to be executed once not repeatedly every time okay.

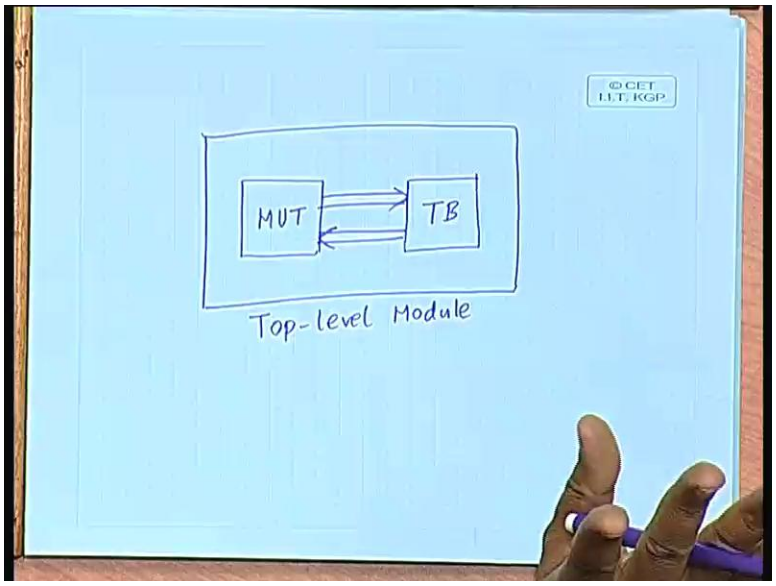
Test bench is obviously used for simulation we are not using it for synthesis test bench is something which is simply used to check whether another design you have made is correct or not. So a test bench can contain some constraints which cannot be synthesized like you can have a statement read from file you can output something on the screen. You can specify explicit delays using hash those things you can do in a test bench right and also the test bench will be generating what ever signal is required to test the other circuit. Well other than the test vectors it can have clock it can have reset and it can have any other control signals you need right. So we will see at some examples and look at some examples.

(Refer Slide Time: 25:08)



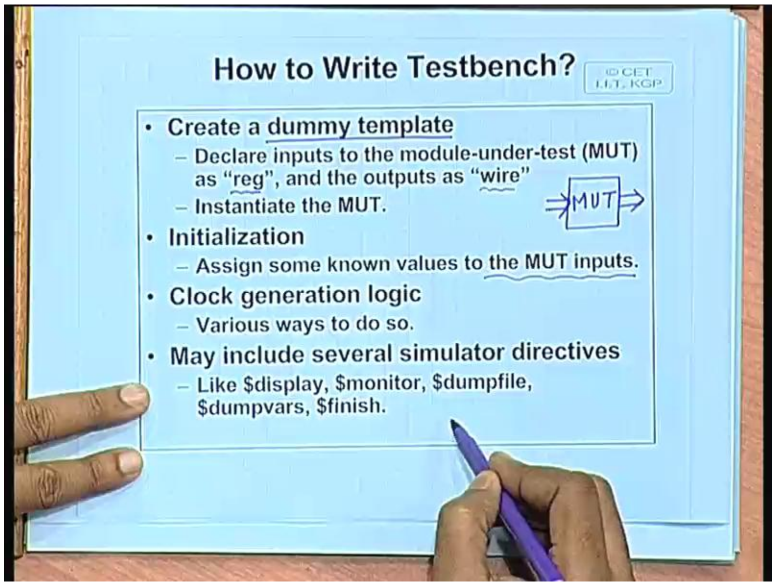
So broadly speaking this picture summarizes what is a test bench see you have a module on the test this is a verilog program you have the test bench this is also a Verilog program. Verilog program will be applying some stimulus to this module and the module on the test what ever output is coming it will be checked by the test bench. Now you see module on the test is also a verilog program test bench is again a Verilog program. Now whenever you are using a test bench for verifying the design of this module what actually we are doing is that you have your module under test.

(Refer Slide Time: 26:08)



You have your test bench. These are 2 independent Verilog modules now you wrap both of them into a top level module. Now in the top level module we use this MUT and TB as if they are already predesigned blocks you simply instantiate them. And you interconnect them as required as I had shown in the earlier diagram previous diagram that the MUT and the TB needs to be interconnected in this way. So you also specify how they are interconnected you simply instantiate these 2 blocks specify the interconnection. That is all now let the models execute and they will do whatever is I will give you an example how it is done fine. So now let us look at ways of writing test bench of course. We will not be talking of all the features available. We will be looking at some of the more popular features okay.

(Refer Slide Time: 27:24)



How to Write Testbench?

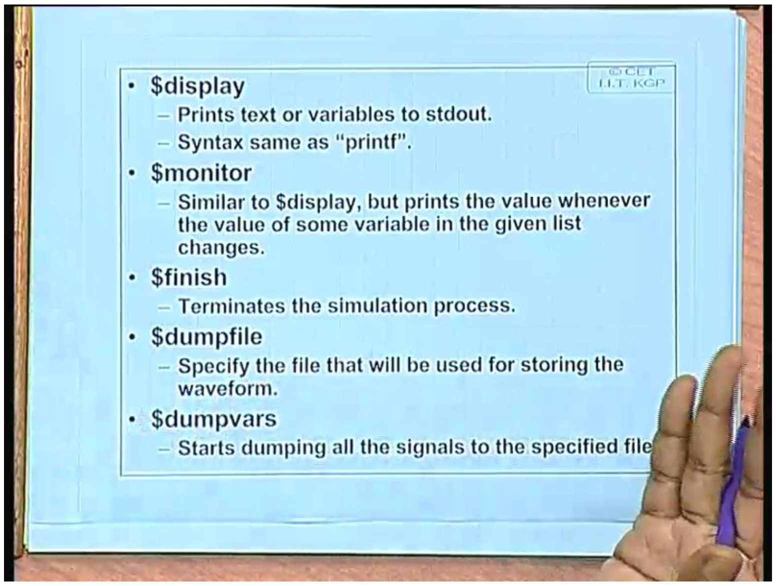
© CET I.I.T. KGP

- Create a dummy template
  - Declare inputs to the module-under-test (MUT) as "reg", and the outputs as "wire"
  - Instantiate the MUT.
- Initialization
  - Assign some known values to the MUT inputs.
- Clock generation logic
  - Various ways to do so.
- May include several simulator directives
  - Like $display, $monitor, $dumpfile, $dumpvars, $finish.

Now let us look at some of the general rules for writing the test benches first rule to be followed is that we start by creating a something called a dummy template dummy template. What it means is that so you have your module under test say this is your MUT, MUT will be having some inputs this will be having some outputs. Now in this dummy template all the input feature coming to MUT, you declared them as reg type variables and the outputs that will declare them as wide type variables right. Second you instantiate the MUT as I had just shown in the previous diagram you have to instantiate the MUT as well the TB. So you instantiate the MUT second is that before you start the actual testing process you may need to initialize some of values in the MUT. For example if your design is just a simple counter then you may need to reset or clear the counter to start it to the known state say all 0. So the next step will be we have to initialize.

Now in initialization you will have to assign some known values to some of the MUT inputs depending on what kind of MUT you are designing there can be some reset signal, some clear signal and so on. And if your MUT has a clock you also have to specify what is the frequency of the clock what is the duty cycle all this you have to specify then ways of doing, we will see how. So clock generation logic these are essential and in addition you can include several simulator directives will be just explaining this. There are in fact many others I am only explaining some of the more popular ones. See all this simulator directives start with the dollar sign they are the special keyword kind of things which has special meaning to the simulator display monitor dumpfile dump variables dumpvars and finish. So let us see what this mean. Display.

(Refer Slide Time: 29:47)



© CET
I.I.T. KGP

- **$display**
  - Prints text or variables to stdout.
  - Syntax same as "printf".
- **$monitor**
  - Similar to $display, but prints the value whenever the value of some variable in the given list changes.
- **$finish**
  - Terminates the simulation process.
- **$dumpfile**
  - Specify the file that will be used for storing the waveform.
- **$dumpvars**
  - Starts dumping all the signals to the specified file

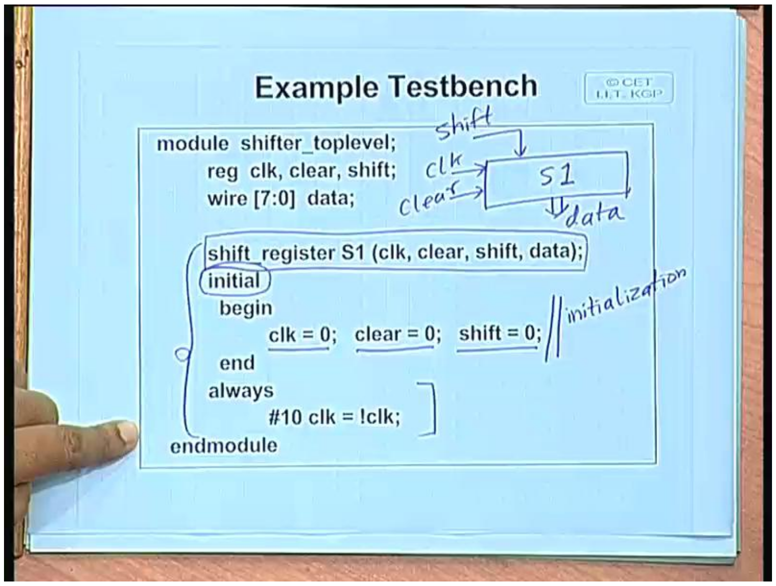
Display is just like a print statement. So whatever you give as an argument to display will be printed on the standard output which is the screen it is like whenever you want to output a string or a message at some point in your test bench file you give a display command. So that what ever you are giving that will get displayed the syntax is similar to print f in c. So syntax is same there is an alternate form of display that is called monitor. Monitor is also doing the same thing. This is also printing the specified string or the variables what ever you specify in the parameter. But there is 1 difference monitor is something like a trigger. Trigger means it remembers that well this is the thing which I have to print. But it will not print immediately. So every time anyone or more of the variables in the parameters changes value it will print automatically. So it is not a sequential statement like the display.

Display will be executed when it is encountered in sequence of code. But monitor is a code it is like say for example you are installing a interrupt service routing something like that that you are installing a interrupt service routing. But the routing will not be executed. Now the routing will be executed every time a variable changes. So every time you see a change in the variable that monitor command will be printing, whatever there in the parameter list to a file or to the screen where ever so that will give you the result of the simulation. You can exactly see that at what times the values are changing okay. So using the monitor command you can do this it prints the value when ever the value of some variables in the specified list changes at the end of the simulation. You give a finish command which tells the simulation to terminate and close all the files which it had been using okay.

Dump file and dump variables these are 2 commands which are often used by the simulator in conjunction with an auxiliary tool which is called the signal viewer or signal display. Now in the different simulators they are called by different names. But the idea is similar. See the first command dump file it simply specifies the name of the file nothing else dump file will specify the name of the file where some information will be outputted and dump vars will dump all the signals that are there in your specified module into that file when ever there is a change or after each instant of time. So after simulation if you look at this file which I have specified you will see a long stream of zeroes and one's which are written into it with the times.

If this time the value of the signal is 0 1 1 0, at this time the values is 0 1 1 something. Now you have another tool that I had mentioned signal viewer that tool will be reading the data from the file and we will be pictorially displaying it on the screen in the form of timing diagrams okay. So this is what I mean by saying here is that this file is used for storing the waveform. So actually we are not storing the physical waveform in the file. But the data for those. (( )) (33:50) This will be only with respect to that particular module the test bench module that we are using now. Yes. Not outside that. Scope of this will be only within that module right okay. So now let us look at some example test benches fine.

(Refer Slide Time: 34:23)



Example Testbench

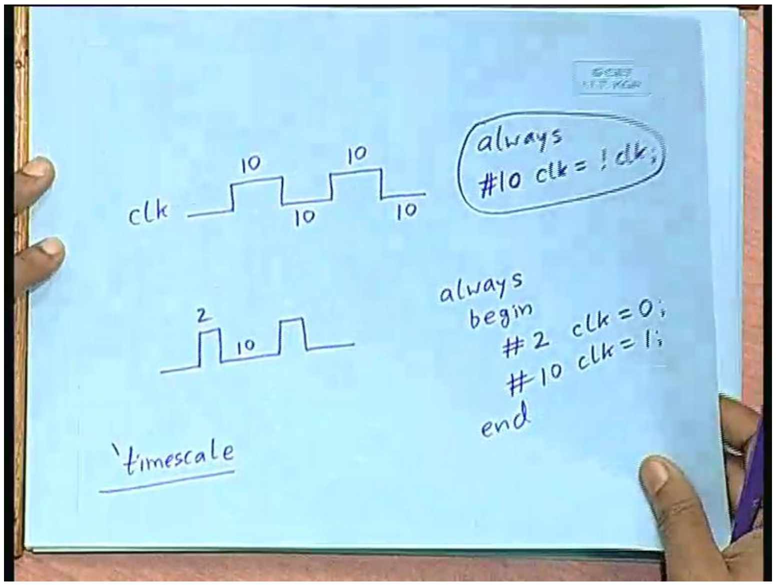
```
module shifter_toplevel;
      reg clk, clear, shift;
      wire [7:0] data;

      shift_register S1 (clk, clear, shift, data);
      initial
        begin
            clk = 0;   clear = 0;   shift = 0;
        end
      always
            #10 clk = !clk;
   endmodule
```

©CET I.I.T. KGP
shift
clk
clear
S1
data
//initialization

In this example we are considering a shift register. Now here we are only showing the test benches we are showing only part of the thing not the whole thing module shifter top level here we are instantiating the shift register module here.

This description will not show this had been defined else where in another module. So this is instantiated as a block, s1 in this design and the inputs are clock clear shift and data. So clock is there, clear is there, shift is there and data is the value which is coming out these are the input and the output signals that is 8 bits. Now see there is a initial statement here initial statement means that this is a test bench and initially you see what we are doing here first is that we are initializing something this part the first part of the initial block this is initialization. Here we are initially setting the clock signal to 0 clear signal to 0 and also shift signal to 0 we are assuming that all these are active high these are the initial values we are setting then you just look at this statement we are using always after 10 units of time clock equal to not clock. This is one way of specifying the clock signal.

(Refer Slide Time: 36:19)



```
always
#10 clk = !clk;
```

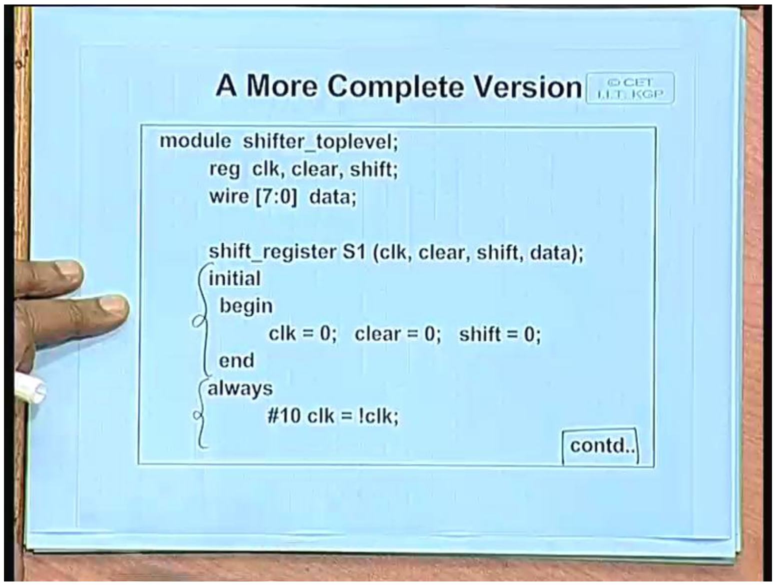
```
always
 begin
  #2 clk = 0;
  #10 clk = 1;
 end
```

This will say that the signal clock will be going high and low continuously and these delays will be 10 units of time each right. So this is 1 way of specifying always hash 10 clock equal to not clock this so just let me write. Okay so always here there is no parameter here as you see so it is triggered always that is no triggers no triggering condition in parenthesis after the always block. So it is an unconditional always block. So after hash 10 you say clock equal to not clock. So this kind of declaration you can use if your duty cycle is exactly 50 percent the waveform is symmetrical. But if you want a different kind of waveform for example the on period is less than the off period say we have 2 here and 10 here say. Then you can specify it like this always you have to give a begin here after time is changed 2 you make clock equal to 0 after time instance 10 make clock equal to 1 end this is the way of specifying an asymmetrical clock.

And if in some design you need to apply an arbitrary waveform delays are always different then you can write a sequence of statements after time 2 days, after time 10 days, after time 5 days, after time 8 days, so on. But since clock is continuously generating you will give it in a always block which is equal to indefinitely and you can specify the duty cycle or symmetrical whatever (( )) (38:30) See typical clock in case of simulation it really does not matter simulation. These numbers is specified those are some time units you specify that you can give some numerical values by giving the times just using the time scale command I talked about earlier. Now using the time scale command you can say that 1 unit of time is actually 1 nano second. But since we have not synthesized the design possibly that 1 nano second does not mean anything.

On the timing diagram the unit will be 1 nano second that is all (( ))(39:06) Digital circuits will as you know nowadays if you have a good design you can run a circuit above 1 giga hertz. So the clock frequency will be 1 the time period will be 1 nano second. (( ))(39:25) This as I said that this is incomplete just means I have shown it I will show more complete version. (( ))(39:37) Time scale if you do not specify it will take as a generic unit 1 unit of time. So absolute whether in nano second or Pico second or second it is not specified. (( ))(39:48) Yes the new simulation you may be only interested in checking the functional behavior of the circuit. Delays and other characteristics will be checking later after we have done synthesis okay fine. So now let us look at more complete version because the earlier version did not contain everything only a few of this statements.

(Refer Slide Time: 40:13)



A More Complete Version

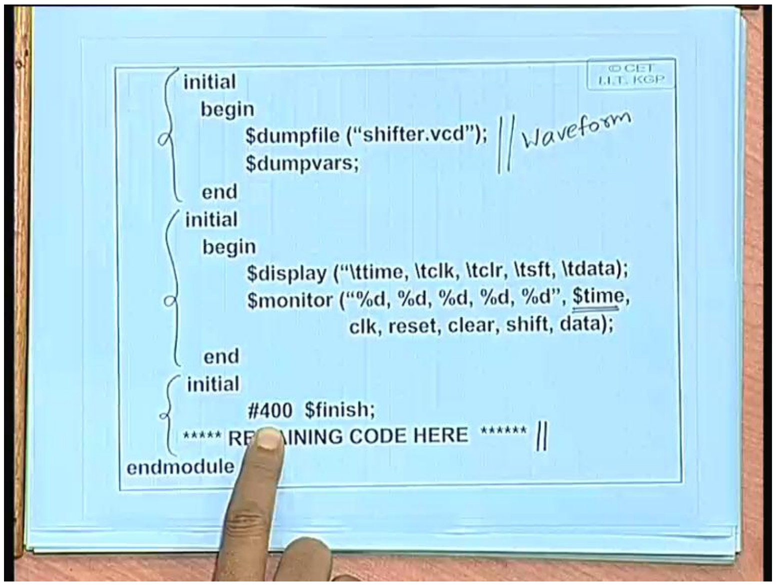
© CET I.I.T. KGP

```
module shifter_toplevel;
        reg clk, clear, shift;
        wire [7:0] data;

        shift_register S1 (clk, clear, shift, data);
        initial
          begin
                clk = 0;   clear = 0;   shift = 0;
          end
        always
                #10 clk = !clk;
```

contd..

So in this version the first part is the same. This continues on the next page. I will show the next page this first part is exactly same as we had earlier same shift register instantiated as s1. In this initial block first there is a begin int this initialization and this is clock generation. Now after this see there can be more than 1 initial blocks in the design. Now here for example this is 1 procedural block executed only once this is 1 procedural block we are executing continuously.

(Refer Slide Time: 40:52)



```
initial
  begin
      $dumpfile ("shifter.vcd");  || Waveform
      $dumpvars;
  end
initial
  begin
      $display ("\ttime, \tclk, \tclr, \tsft, \tdata);
      $monitor ("%d, %d, %d, %d, %d", $time,
                 clk, reset, clear, shift, data);
  end
initial
      #400 $finish;
***** RE  AINING CODE HERE ****** ||
endmodule
```

Then you can have as many such blocks you need. For example you have another initial block executed once another block here executed once we have another block here executed once. So you can as many number of initial and always blocks in a program if it is initial it will be executed once. If it is always it will be executed as many times specified and this blocks are all concurrent it is not that first this will be executed and then this okay. Now let us see what these initial blocks are doing the first initial block is specifying the name of the dump file where the data will be dumped far away from display.

Well some simulator use the extension dot v c d for this file. So I have also shown here as dot v c d and this is followed by a command dump vars dump vars means whenever I give this command then when ever some variable changes state all the values of the variables should be automatically dumped to this file. But if you do not give this command the dumping will not be automatic okay. And here in the next initial block we start with the display see display is like print f I told you (( ))(42:14) downwards by default take the file name specified by the previous dump file command.

Student: What if dump file is not there. dump file not there this will give an error? What is shifter there?

Just a name of a file.

Student: Sir dump file is for dumping the waveform.

Dumping the waveform. These 2 statements are only for waveform analysis all the signals that are defined because the waveform viewer you can specify which signal you want to see which you would not. So your file should contain information about all the signals okay. Now in the next initial block you are actually displaying the input output of everything on the screen of all the signals you want to see the first display command is like a header. You are printing header time clk clr sft data slash t is a tab using tab you properly want to display it in a tabular form this is followed by monitor command display is executed immediately. So the header is displayed monitor this is like a, printf statement taken you see there are 5 things you want to display time. This is the system time this is the simulation time dollar time simulation time is advancing 110, 20 whatever clock reset clear shift data. See one thing this dollar time does not require formatting stuff out here.

So you can see there are 5 other variables there are only 5 out here and of course here I have not given tab. You should have given this tab here. So that it is displayed properly in a tabular way okay. So slash t percentage d slash t percentage d that should have been the right way of looking at it anyway. So monitor says that whenever any 1 of these variables change state you print it on the screen. So on the screen you will be getting a nice print out of all the signals as they are changing end this is this block. And the last 1 says you continue whatever is being specified up to 400 units of time. So at 400 time units you finish okay. There can be some additional code which follow this also like you are applying some specific input pattern to the module end test checking the output that also you can do here. So you can add it as much as. (( ))(44:55) Sure the previous line (( ))(45:01) okay. Your question is how are we integrating this with the test bench. So that I have not yet mentioned. Let me now look at a complete example that you show exactly how you will normally use it. See whatever I have shown till now is just for the sake of illustration.

This is your main module these are the functions you write as test bench you integrate them. But normally as I have shown an earlier example that is how you normally do it you usually do it. Like you have a module under test you have a test bench. You instantiate both of them inside a top level module. That is how you normally do it I will give you an example of this like that okay. In this example this hash 400 finish means at time 400 this simulation should stop. So whatever I have mentioned clock everything everything will stop as soon as time reaches 400.

Student: Initial blocks they are executed only once.

Yeah these are executed only once these are delay.

Student: So whenever always input is first time all these processes also execute.

Yes.

Student: So always is in the next line so what happens to these blocks.

These blocks are executed only once. For example this hash 400 finish. This will make an entry to the simulator event queue simulator event queue that at time 400 there is an event that we have to process and as the simulation time advances that simulation queue is maintained as a priority queue. So that what ever time is earliest that event is processed earliest. So at some point of time this time 400 will be the first one. So you see that it is the finish event you close everything and stop the always will also be generating an event clock wise changed every 10 time unit. So clock will go up 10, 20, 30, 40, 50, 60, it will go on changing.

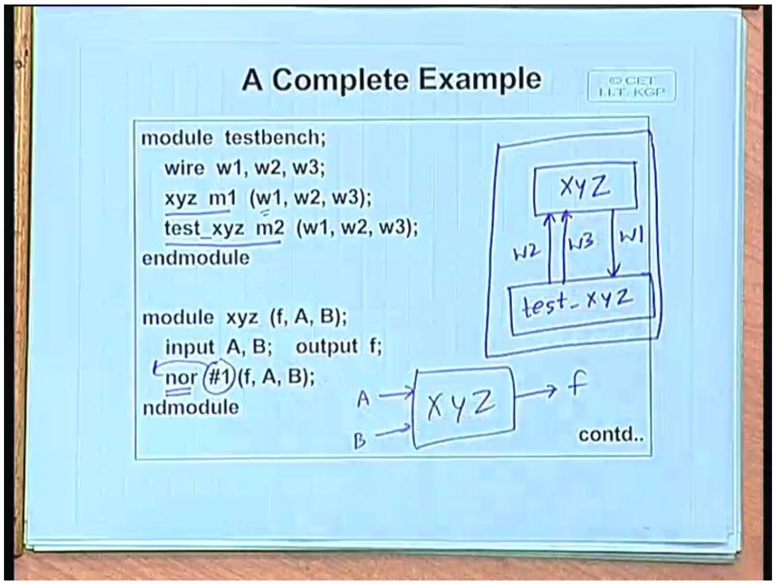
Student: Sir this dump vars used for dumping when ever changes take place.

Yes, yes.

Student: Sir in the previous slide I am not clear about variable change the variables of s 1 and the variables of s2.

Yeah I understand your question. I will show the next example. We will see that how it is situated because this example shows the complete picture okay.

(Refer Slide Time: 47:23)



A Complete Example

```
module testbench;
   wire w1, w2, w3;
   xyz m1 (w1, w2, w3);
   test_xyz m2 (w1, w2, w3);
endmodule

module xyz (f, A, B);
   input A, B;   output f;
   nor (#1)(f, A, B);
ndmodule
```

contd..

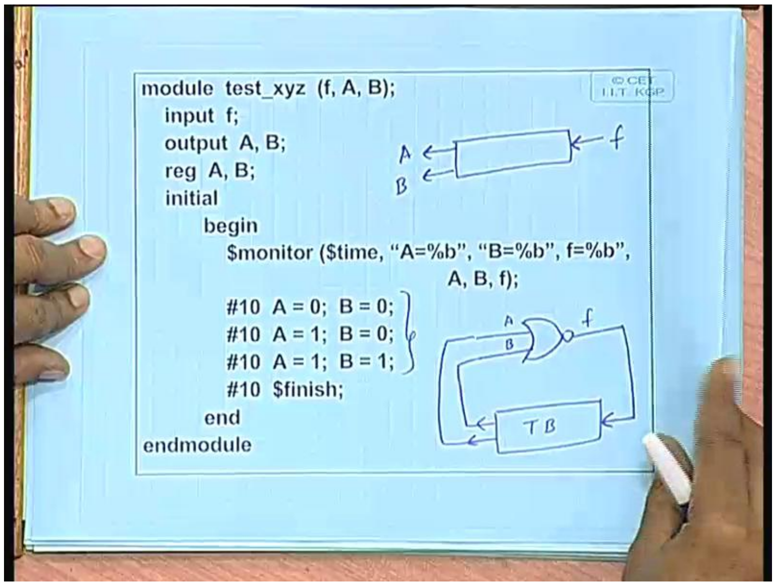
XYZ
W2
W3
W1
test_xyz
A
B
XYZ
f

Here I am showing the total thing there is a module under test this you are calling x, y, z. There is the corresponding test bench this we are calling test x y z and we are integrating both of them like this as part of a top level module. Let us see how well you see this module the first 1 here there are no parameters. Because this is the top level model top level model will just is like a black box nothing goes in nothing goes out. This simply encapsulates the module and the test bench nothing else. So there are no parameters and in this example there are 3 interconnecting signals. We call them this is w 1, w 2 and w 3, w 1, w 2, w 3 are defined as vars you instantiate 1 copy of x y z out here. You instantiate 1 copy of the test bench out here one you call the m 1 other you call m 2. So when you will be seeing the modules in the test bench you will see that they contain 3 parameters w 1, w 2, w 3, they are specified in the same order. So here you instantiate this and here we instantiate this. Now the way or in the order that you have specified the parameters this will define the interconnection okay directions will be defined by which you defined the input and output in the modules like in x y z you can see here x y z module. Here if A and B A and B are defined as inputs and f is the output well and inside it is defined as a simple NOR gate which has a delay of 1 time in it NOR gate takes the parameter NOR is a primitive. I told you earlier first 1 is the output second and third are the inputs. So you see for the test bench the first 1 is the output second and third are the inputs.

Student: Sir that hash 1 with the NOR gate is specified further.

That is specified anywhere before after anywhere you can specify. Because this is a predefined model you are instantiating this hash 1 you can give either before or after it makes the same thing same meaning. (( )) (50:11) No. That is optional because names also you draw because the signals which you are specifying here. They will define the interconnection and if you do not specify the names by default u1, u2, u3, some names will be assigned by the synthesizer when you synthesize it okay fine. All right here we have defined the test bench we have defined the x y z module, then let us see how we can define the test bench now.

(Refer Slide Time: 50:50)



```
module test_xyz (f, A, B);
  input f;
  output A, B;
  reg A, B;
  initial
     begin
       $monitor ($time, "A=%b", "B=%b", f=%b",
                                 A, B, f);
       #10  A = 0;  B = 0;
       #10  A = 1;  B = 0;
       #10  A = 1;  B = 1;
       #10  $finish;
     end
endmodule
```

© CET I.I.T. KGP
A
B
f
A
B
f
TB

The test bench will look like this test bench will also be having the signals f a and b f a and b. But now f will be the input a and b will be the outputs why because in the x y z module it was actually a NOR gate. So here whatever you are calling as a and b and what ever you are calling as f in this test bench this a and b has to be fed out okay into it. So for the t v these are the output signals and this has to be fed in for t v. This is the input signal. So now you look at this. So this a b is our output. That is why they are defined as edge there is a initial block. There is a pure combination circuit. So no problem no clock monitor monitor you show like this you display the time dollar time you can give anywhere or just before or after the string you display the time.

Then you display a equal to b means binary b equal to f equal to a b f is a b f a single bit variables they will display a equal to 0 b equal to 0 f equal to 1 something like that. Now here you are applying the test vectors this is 1 way mentioned. You specify the delay after 10 time units of time you apply a 0, b 0 after 10 units of time again we apply a 1, b 0 after 10, a 1, b 1, after 10 finish. So with just simple example I will illustrate, but this illustrates everything.

Student: sir that 10 ten occurs parallely or?

Sequentially, sequentially.

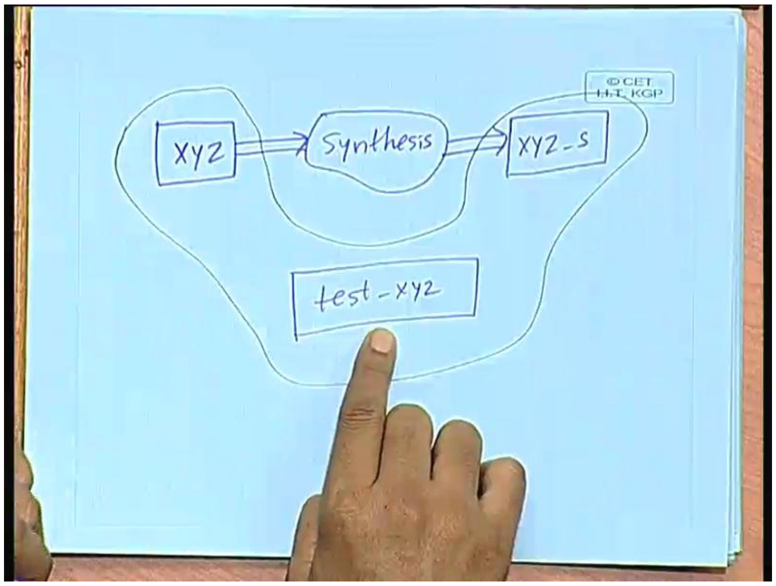
Student: 10 units after the first (( ))(52:54)

Only extra z is block test extra z is not (( ))(53:06)

Student: What is the need of the top level?

See now the question is what is the need of the top level module the need of the top level module is nothing. But flexibility because I am telling you right. Now the example I have shown you have a copy of the extra z you have the test bench. Now you can have another you can have another scenario which can happen after you have done the synthesis like you can have similarly like this.

(Refer Slide Time: 53:36)



You have this extra z okay. This you have written as that you give this specification to the synthesizer you carry out synthesis okay. You get you get another version of x y z. This you call x y z and you already have test x y z. But of course you will have to modify test x y z now. Now what you do you integrate these 3 in the top level module. Now as part of it test bench you can do that you apply this vector here and here after that you compare the output so that they are equal and not equal. If they are unequal you flag some error message that also you can write as part of the test bench. But just putting that together at the top level module is just for convenience and modulating nothing else.

Student: module x y z.

They are not (( )) (54:41). Since this tool will be ignoring this yes.

Yes true.

Student: Sir do we in the last slide can we give the parameters at the end?

Here f a and b yes you need to.

Student: sir how to do that?

No, no. It is a verilog module and these are the improper parameters. That is all.

Student: You can declare them as wire f a and b then the output will be the same.

Yes, no, no. What I am saying is that when ever you are doing this kind of instantiation
these are defined as individual verilog modules and when ever you define a when ever you define a verilog module just like a function you write in c you have to specify all the signals coming in and coming out that has to be there as part of the parameters. This is mandatory. But the top level module in contrast it does not take anything in or anything out that does not need any parameters.

Student: Actually what he is saying is in the test levels only you instantiate the desired module.

That is what I had shown in the earlier example. That is what I showed in the earlier example because how is it helping it is simply helping in case of modularity. Say yeah, yeah, because you can have several versions of the test bench also 1 is a simple test bench and the other is a more exhaustive test bench you can just include the 1 you need and you can rewrite okay. So from the next class onwards we will be starting our discussion on synthesis.