

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

Lecture - 31
Modeling Register Banks



So, in this lecture we shall see how we can model register banks or register arrays or register files. Now we have seen some examples earlier like we saw how we can design the data path and control path for certain designs like multiplied GCD computation etcetera. Those designs were pretty small designs, but even there we saw there was the requirement of several registers. Like for a multiplied you needed register AQM and so on, but you think of more complex systems. You think of a computer a processor. So, inside a processor there will be much larger number of registers. Well, a modern day processors can have 64 32 128 registers very large number of registers.

So, how to implement those registers? Registers you can implement as individual reg - reg 0 reg 1 reg 2 like that individual variables. Or you can also stored them similar to a memory model them like a memory, and just allow the synthesis tool to synthesize the register bank as an array of hardware registers. Because usually register banks or register files they are not implemented using memory technology, but that implemented using flip flop which are much faster, in a much faster static ram technology ok.

(Refer Slide Time: 02:04)

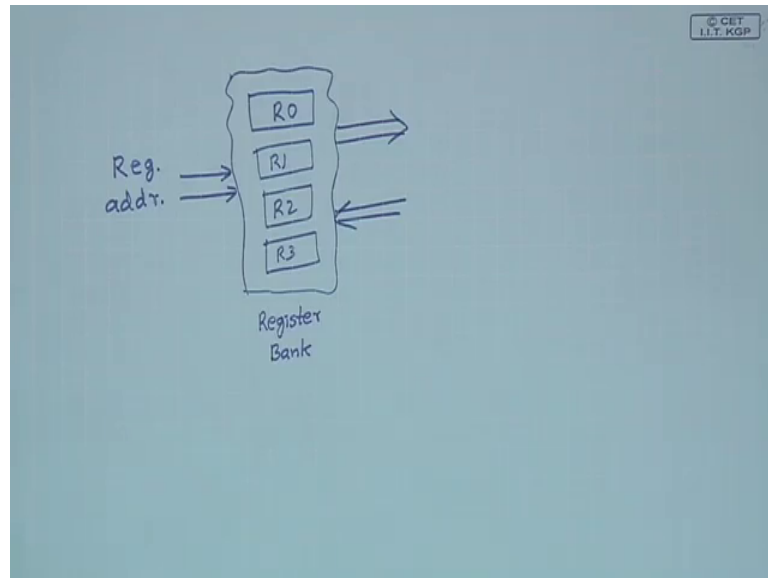
Introduction

- A register bank or register file is a group of registers, any of which can be randomly accessed.
 - Commonly used in computers to store the user-accessible registers.
 - For example, in the MIPS32 processor, there are 32 32-bit registers, referred to as *R0, R1, ..., R31*.
- Can be implemented in Verilog as independent registers, or as an array of registers similar to a memory.
- Registers banks often allow concurrent accesses.
 - MIPS32 allows 2 register reads and 1 register write every clock cycle.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, the topic of this lecture is modeling register banks. So, the first question is what is a register bank? A register bank which is sometimes also called a register file it is a group of registers.

(Refer Slide Time: 02:24)



Let us take an example. Suppose I have a register R 0, I have a register R 1, I have a register R 2 I have register R 3 let us say 4 registers. So, I can use them individually in a program R 0 R 1 R 2 or R 3, but suppose what I say is that well, let me group all the 4 registers together and let me call this a register bank right.

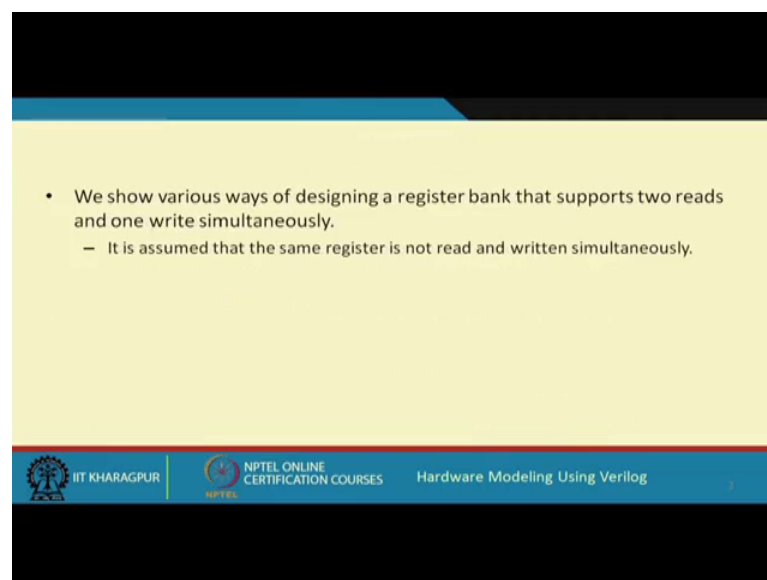
Now, in this register bank from outside I can specify a register address. A 2 bit register address because there are 4 registers in 2 bits I can specify R 0 R 1 R 2 or R 3. So, with this register address I will be selecting one of the registers and then I can either read or I can write into that register. This is the concept of a register bank. So, just to summarize I can use the registers individually as variables and read and write from them individually by the names, or I can treat them as a group just like a memory element and I can read and write from there fine.

Now, a register bank as I said is typically used in a processor in a computer to store the registers which are typically used for temporary storage of data. Now as an example there is a there is a common processor call MIPS 32 these an example of a reduced instruction set architecture. Here there are 32 registers and each of the registers are of 32 bit size. And these 32 registers are referred to as R 0 R 1 up to R 31.

Now, as I said in verilog you can either define 32 variables and implement them as independent registers, or you can implement them as an array of registers which will be very similar to modeling memory. Well, not only this a register bank typically supports some other features like concurrent access. As an example for this MIPS 32 processor that I was talking about, here the registered bank is so designed that you can carry out 2 register reads and one register write in every clock cycle. Which means 3 operations can be carried out concurrently on the register.

But of course, suppose you are writing into register R 5 and you are at the same time reading from R 5 there will be inconsistency.

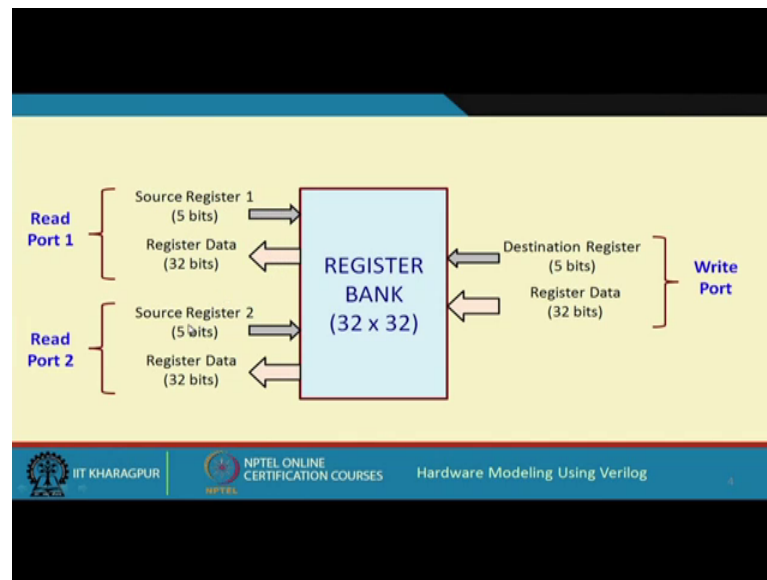
(Refer Slide Time: 05:47)



So, you should not read and write into the same register, but you can read from R 5 read from R 10 and write into R 8 that is possible and these 3 things can go on in parallel fine.

So, we shall be showing some alternate designs of register bank, which will support this kind of this can concurrent axis; that means, 2 reads and one writes at the same time. And of course, I made this assumption same register is not read and written together because if it is so there will be some conflict.

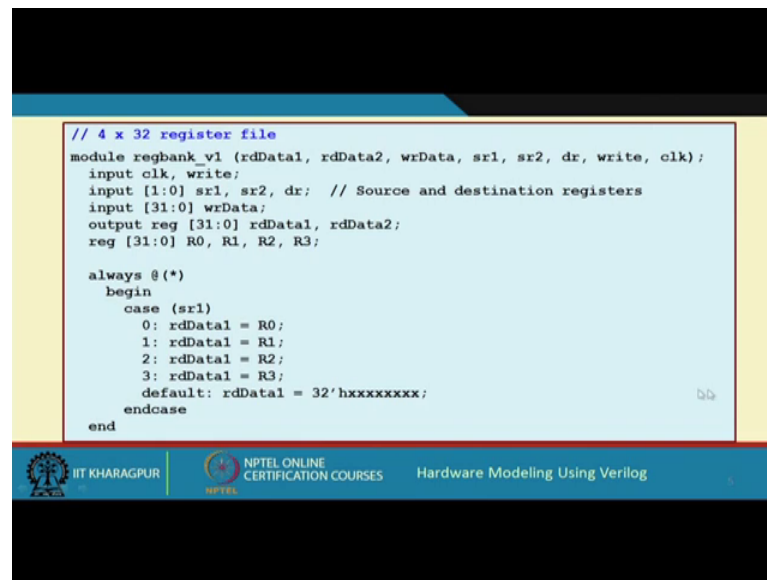
(Refer Slide Time: 06:13)



So, pictorially our register bank will be the MIPS 32 register bank 32 registers each of 32 bits will look like this. There will be 3 axis ports, 2 read and one right. In the read ports what do we have? We have a register number that which register I want to write, because there are 32 registers you need 5 bits here.

Similarly, for the second port there are 5 bits. This will actually be source register 2 not 1, this is 1, and this is 2. And so once we have specified the register number we can read them and the data will be available over these 32 bits, and on the other side when you are writing. So, again you have to specify which register you want to write, that is again 5 bits and the actual data you want to write 32 bits. So, there 2 read ports and one write port fine.

(Refer Slide Time: 07:27)



Let us take a simple example to start with. Let say we do not have 32 registers, but we have only 4 registers small number. So, because it is small we can declare the registers individually. This is what I am assuming. So, how is our declaration? Our declarations like this. So, in our register bank there are 2 read ports and one write port as I said, read data 1 read data 2 write data. So, what are these 3? This is read data 1 read data 2 and this is write data. And we have source register 1 source register 2 and destination register, this is here. Source register 1 source register 2 and destination register these are all. So, in this case because there are 4 registers this will be 2 bits each and of course, write and a clock, because read is by default.

Let us see the variable declarations clock and write will clearly be inputs. And source register 1 2 and the destination register they will be 2 bits because there are 4 registers. Write data read data 1 read data 2 all are 32 bit quantities and for read data since reading and assigning their defined as reg ok.

And the data you want to write that is coming from here and that is declared as input wire. And we define the 4 registers like this R 0 R 1 R 2 R 3. So, you will be using a setup always blocks very simple. This is always block for the port 1. So, you check always whenever something is changing case s R 1, source register 1 whatever is s R 1 is 0 1 2 or 3 because it is 2 bits if it is 0 in the value of R 0 will go to read data 1 if it is 1 then R 1. If it is 2 R 2 if it is 3 R 3, but if it is not if it is other than 0 1 2 3 because you

see these are verilog variables. So, the verilog variables can also contain values which are z or x, not necessarily only 0 and 1 ok.

(Refer Slide Time: 10:18)

```
always @(*)
begin
  case (sr2)
    0: rdData2 = R0;
    1: rdData2 = R1;
    2: rdData2 = R2;
    3: rdData2 = R3;
    default: rdData2 = 32'hxxxxxxxx;
  endcase
end
always @(posedge clk)
begin
  if (write)
    case (dr)
      0: R0 <= wrData;
      1: R1 <= wrData;
      2: R2 <= wrData;
      3: R3 <= wrData;
    endcase
end
endmodule
```

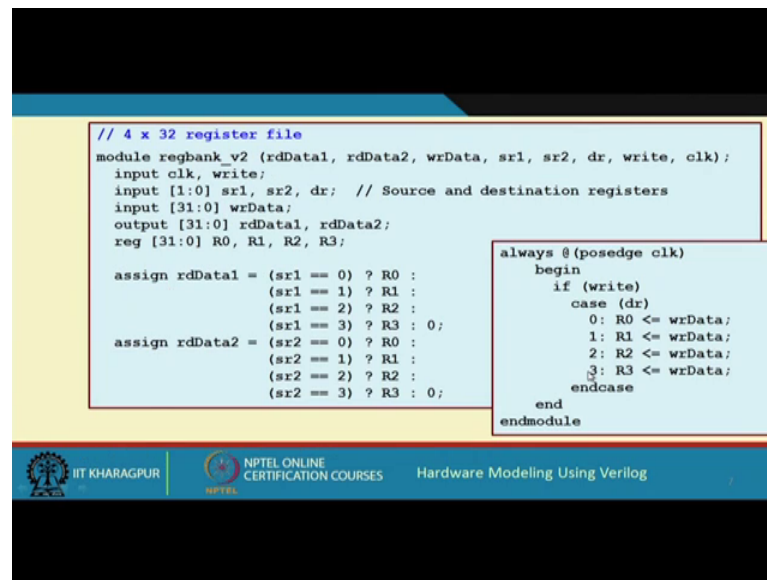
This way of modeling is feasible if the number of registers is small.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, if it is anything else then read data 1 will be assigned the undefined value x x x x. This is for the first read port. Similarly for the second read port same block, but here case s R 2 depending on s R 2 R 0 will be assigned to read data 2 or R 1 or R 2 or R 3. This is for the second port. And third one is for write and writing is in synchronism with a clock. So, whenever there is a clock then only we are writing. If write is active then you see what is your destination register. So, if d R is 0 you write into R 0 if d R is 1 you write into R 1 and so on.

So, these is a very simple description of a register file or register bank containing 4 registers, where the registers are defined individually independently. Let us see some other ways in which we can define or declare such register bank definitions. So, this as I said this way in numerating the values is feasible if the number of registers is small. Just imagine if there are 32 registers will be 2 tedious we have to write 32 lines here 32 lines here and so on.

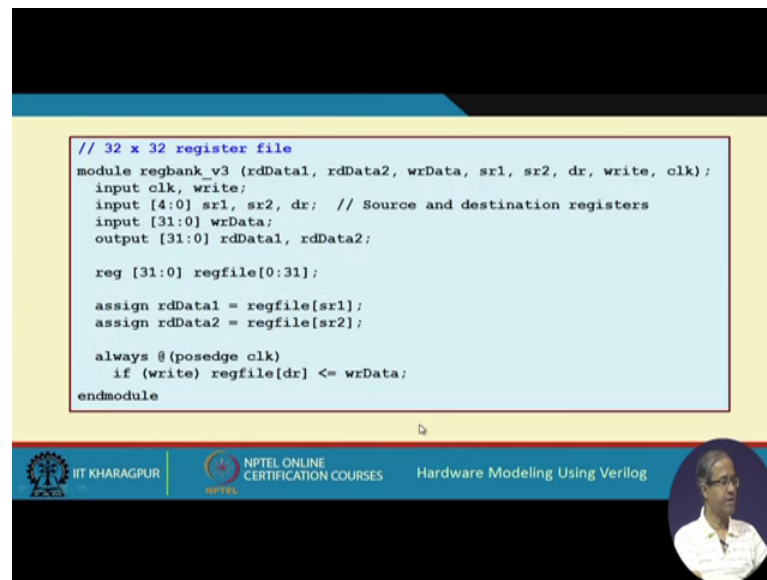
(Refer Slide Time: 11:36)



Now, let us again keep this number as 4, registers define independently, but we are using an alternate way of implementing the read ports. Like in the earlier case you see the read ports were implemented using an always block and a case. Here we are implementing them using assign statements. The first part is identical, there is no change here. Just only think because here read data 1 you are not assigning inside and always block. So, we have not given reg here only output right.

Here this is like an if then else kind of a statement using the conditional operator what is what does mean? It means if s R 1 equal to equal to 0 if this condition is true then R 0 R 0 is assigned. If not otherwise s R 1 equal to equal to 1, if this is true then R 1 assigned. Otherwise if it is 2 then R 2 otherwise if it is 3 then R 3 if nothing matches here given 0 or we can give undefined also if you want. Similarly read data is identical with s R 2. And the third block is identical again with a clock for writing. So, this is how you can implement a register file and you can do reading and writing like this.

(Refer Slide Time: 13:13)



```
// 32 x 32 register file
module regbank_v3 (rdData1, rdData2, wrData, sr1, sr2, dr, write, clk);
  input clk, write;
  input [4:0] sr1, sr2, dr; // Source and destination registers
  input [31:0] wrData;
  output [31:0] rdData1, rdData2;

  reg [31:0] regfile[0:31];

  assign rdData1 = regfile[sr1];
  assign rdData2 = regfile[sr2];

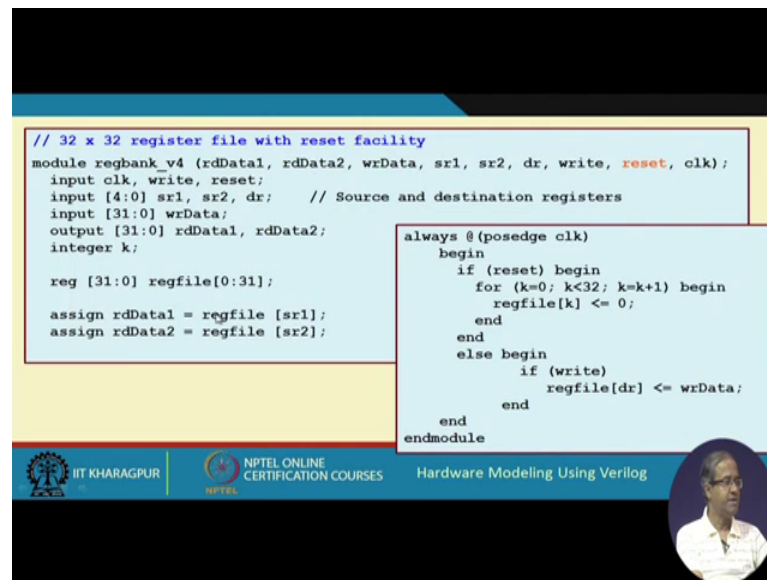
  always @(posedge clk)
    if (write) regfile[dr] <= wrData;
endmodule
```

Let us now come to a complete register file like MIPS 32 by 32. Here let see how we can modeled here we are using an array like a memory. So, the arguments are the same read data 1 read data 2 write data, source register 1 2 destination register write at clock. The difference is here s R 1 s R 2 and d R will be 5 bit quantities, because there are 32 registers and 2 to the power 5 is 32. That is why we will be needing 5 bits to represent a particular register right.

So, this is how we are declaring our register file just like a memory. So, each word contains 32 bits and there are 32 locations 0 up to 31. So, for reading we can simply give assign statements read data 1 equal to reg file this s R 1 as the index for the second port reg file s R 2 as the index. And for write always a posedge clock if write is active write data goes to reg file d R simple right. So, so in a behavioral wait is very compact and simple to specify a register file like this.

Now, let us in addition let us add another facility, like you see when you have a register file means of there is a facility using which you can reset the registers from outside. Resetting means let say there will be a reset signal control signal. So, if we activate reset from outside, the all the registers they will be initialized to 0s, this is what you want.

(Refer Slide Time: 15:21)



The image shows a Verilog code snippet for a 32x32 register file with a reset facility. The code is presented in a light blue box with a yellow border. The code is as follows:

```
// 32 x 32 register file with reset facility
module regbank_v4 (rdData1, rdData2, wrData, srl, sr2, dr, write, reset, clk);
  input clk, write, reset;
  input [4:0] srl, sr2, dr; // Source and destination registers
  input [31:0] wrData;
  output [31:0] rdData1, rdData2;
  integer k;

  reg [31:0] regfile[0:31];

  assign rdData1 = regfile [srl];
  assign rdData2 = regfile [sr2];

  always @(posedge clk)
  begin
    if (reset) begin
      for (k=0; k<32; k=k+1) begin
        regfile[k] <= 0;
      end
    end
    else begin
      if (write)
        regfile[dr] <= wrData;
    end
  end
endmodule
```

The code is part of a presentation slide. At the bottom of the slide, there is a blue banner with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the title "Hardware Modeling Using Verilog". A small circular inset image of a man is visible in the bottom right corner of the slide.

So, in the next version is just a modification of this. Here we are saying this is a 32 by 32 register file, but with reset facility. So, what is the change? The change is there is a variable additionally included here called reset, which is also an input variable and we have declared an integer k, rest is the same. The way you read from register file there is no change just in the previous 1 exactly the same, but while writing there is a little change, this is synchronous.

So, whenever there is a clock you first check whether reset is active or not. If reset is active then in a for loop you go from k equal to 0 up to k equal 31, and initialize all reg file k is to 0s. So, all the registers are initialized to 0. But if reset is not active you come to the else part, you see if write is active or not, if write is active then you write this data into reg file d R right. So, this is how it works.

(Refer Slide Time: 16:39)

```
module regfile_test;

reg [4:0] sr1, sr2, dr;
reg[31:0] wrData;
reg write, reset, clk;
wire [31:0] rdData1, rdData2;
integer k;



regbank_v4 REG (rdData1, rdData2, wrData, sr1, sr2, dr, write, reset, clk);

initial clk = 0;

always #5 clk = !clk;

initial
begin
    $dumpfile ("regfile.vcd"); $dumpvars (0, regfile_test);
    #1 reset = 1; write = 0;
    #5 reset = 0;
end
```

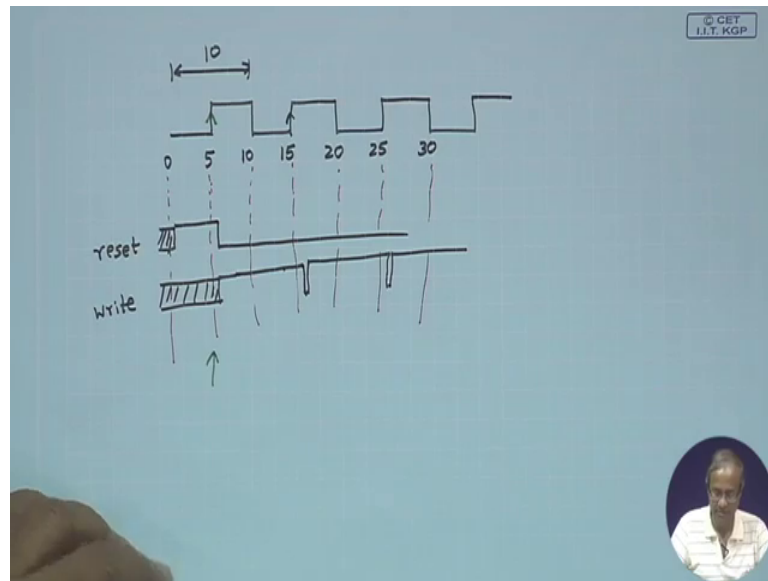
A test bench to verify operation of the register file

IIT KHARAGPURNPTEL ONLINE
CERTIFICATION COURSESHardware Modeling Using Verilog10

Now, let us look at a test bench to verify the reg bank v 4, which means we are just using the register bank with the reset facility the last example that we took this one right this one let us write a test bench for it. So, what you have done? Let us see. So, we have instantiated the register bank we have given a name reg. So, we have not change the names we can change the names if you want. So, read data 1 read data 2 wire inputs now here their output they were actually coming out. So, in this case they will be wires, but wherever you are writing they will all be reg reg write data s R 1 s R 2 d R write reset clock this well all be reg right, and we are defining an integer k.

Let say what we are doing here we are initializing clock to 0 and always at 5 clock equal to not clock this means again we are using a clock signal like this.

(Refer Slide Time: 18:07)



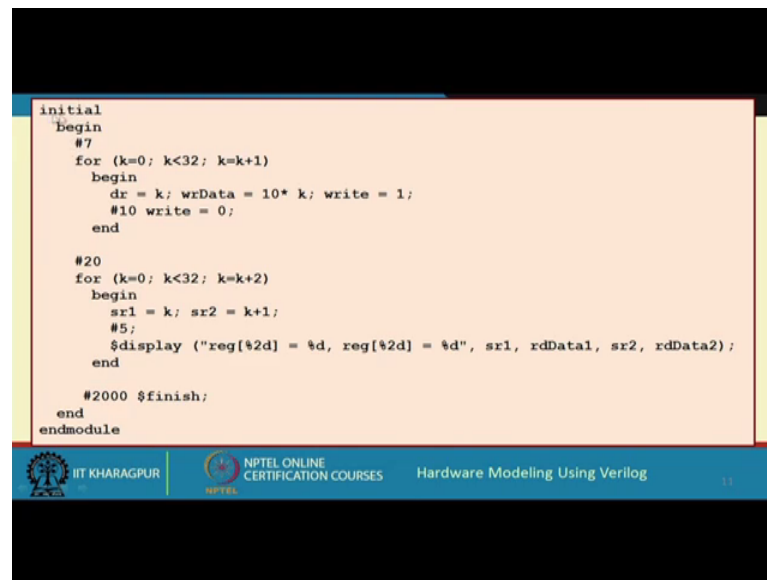
Which start with time 0 set 5 toggle again after 5 toggle again after 5 again after 5 like this. So, so after time 5 we are toggling the states. So, our time period will be 10, this will be our time period which is 10.

Now, let us look into the timing of this signal what we are actually doing. These are the edges right. Now what we have done? Now in this initial block we are actually creating the reset signal. So, here of course, we have specified a dump file and the variables to dump. We are see we are activating reset at time 1 write is 0. So, that write does not take place and after a gap of 5 we are setting reset back to 0. So, what will happen? This is time 0, right.

So, if you look into the reset signal, reset initially it was undefined. So, it was. So, at time 1, it is undefined let say this is one at time 1 you are setting reset to 1. And it continuous 1 up to time 6, at time 6 you set reset to 0. And after that you do not make it 1 again. So, what does this mean? This means that when the first clock reg comes here, out here your reset is 1; that means, initially all the registers will be reset to 0s right. This is what you do here.

And then there is another part of the test bench. You see here, from there is another initial blocks. So, that again it starts from time 0.

(Refer Slide Time: 20:20)



So, after a gap of 7 you see what you do? You go in a for loop 0 up to 31, you put k into destination register write data 10 into k, you write it write equal to 1. 10 into k means register number i you are storing the value 10 into i. So, register 0 will get 0, register 1 will get 10 register 2 will get 20 3 will get 30 and so on.

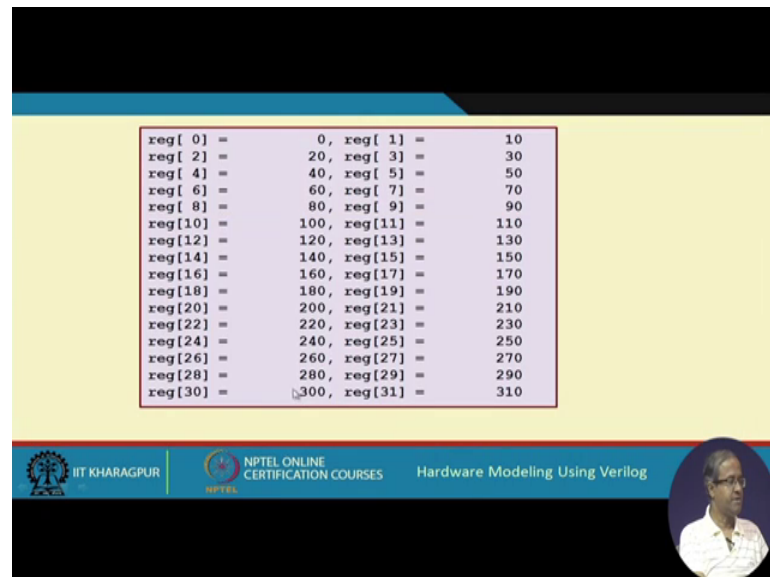
So, let us say we initialize registers like this. So, write data equal to 10 into k. And there is a delay of 10 after delay of 10, I set back write to 0 and repeat this again, this is repeated. So, what does this mean? You think of the write signal what happens write becomes 1 after a delay of 7 and after a delay of 10 it again becomes 0. So, what does this mean? So, how will the write signal look like. Write will become 1 at times 7 7 is so, this is 5. So, it is somewhere here before that it was not specified. So, write will be one here and it will remain one for a period of time 10; that means, it will remain one till here then it will again goes 0 right.

So, like this it will go on. So, when the next clock edge comes it will be high and write will take place. So, again when you go back to the loop; so again this will be one again after a time 10 it will be 0 again it will be 1 like this it will go on. So, it will be writing will be taking place with every clock edge one by one ok

So, in this loop the writings will take place. Well, after the writings are done here just as an example here we are just observing the contents of some of the registers. So, what were doing k equal to 0 up to 31, again? Here we are reading 2 registers at a time, and

here k equal to k plus 2 as skipping by 2, k equal to 0 3 2 4 6 8 like that. And source register 1 we are initializing k and s R 2 k plus 1. And we are displaying the values of a s R 1 read data 1 s R 2 read data whatever data is stored there. And we are printing it like this reg within bracket register number equal to data like that, and we finish here.

(Refer Slide Time: 23:29)

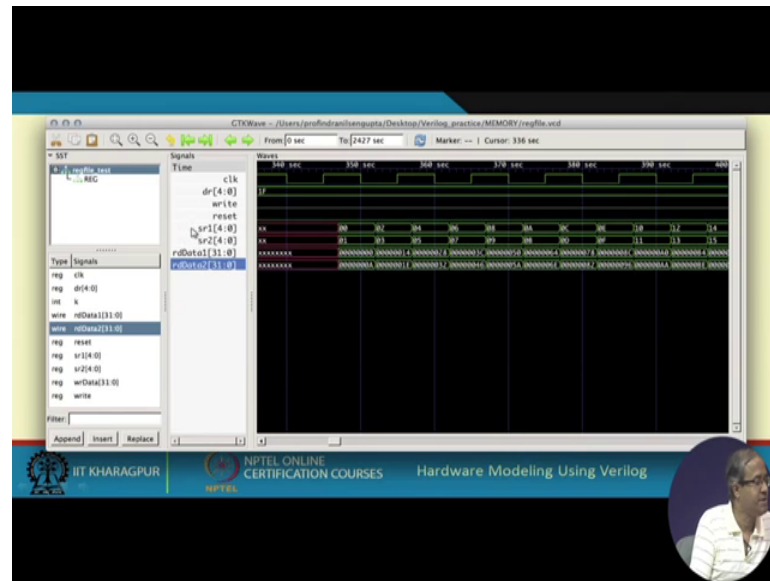


reg[0] =	0,	reg[1] =	10
reg[2] =	20,	reg[3] =	30
reg[4] =	40,	reg[5] =	50
reg[6] =	60,	reg[7] =	70
reg[8] =	80,	reg[9] =	90
reg[10] =	100,	reg[11] =	110
reg[12] =	120,	reg[13] =	130
reg[14] =	140,	reg[15] =	150
reg[16] =	160,	reg[17] =	170
reg[18] =	180,	reg[19] =	190
reg[20] =	200,	reg[21] =	210
reg[22] =	220,	reg[23] =	230
reg[24] =	240,	reg[25] =	250
reg[26] =	260,	reg[27] =	270
reg[28] =	280,	reg[29] =	290
reg[30] =	300,	reg[31] =	310

NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, when we just run this. Just if you look at the simulation output like whatever is getting printed by this display we say something like this. You see what you expected is actually that. So, each register is stored the twice the value register number 8 10 times the value sorry; so 8 contents 80, 13 contents 130, register 20, 3 contents 2 320 8 contents 280. So, it is actually 10 times it is stored, ok.

(Refer Slide Time: 24:00)



Now, if you also see the timing diagram. So, I am showing you 2 parts of the timing diagram. This is the first part where reset has been activated initially reset has been activated, and then you are writing the data 10 into k, you see the register number d R is changing 0 0 0 1 0 2 0 3. So, the right is continuously active, because it is going low and immediately going high. So, that you are not seeing here write is continuously active. So, 0 1 2 with the clock edge writing is going on ok.

And towards the end this is towards the end of the simulation. Here you are you are reading the values, you see s R 1 or s R 2 0 1 then 2 3 then 4 5 6 7 like that. And you see the read data 1 read data 2 for 0 1 the data are 0 0 0 and 0 0 0 a which means 0 and 10. 2 and 3 are 0 0 1 4 means 20 and this is one e means 30 for 4 and 5 it is 2 8 hexadecimal means 40 3 2 means 50. So, like that the data values are actually coming.

So, you see in this way you can model a register bank or register file, and you have seen through the test bench that actually you are able to read and write together at the same time. So, we will see some more examples of register banks later when we talk about more complex systems that involves something called pipelining, but this we shall be discussing later.

So, with this we come to the end of this lecture. So, in this lecture whatever we have basically seen we have summarized how we can define register banks or register files and how we can have multiple read and write ports you can read from the register banks

you can write into the register banks. And how we can write a test bench and simulate the design using the test bench.

So, we shall see later as I had said that will be using some more complex designs where we will be using this kind of register files as a basic building block. So, whatever description we have given something similar to that that will happen as part of the main model description therein.

So, with this we come to the end of this lecture.

Thank you.