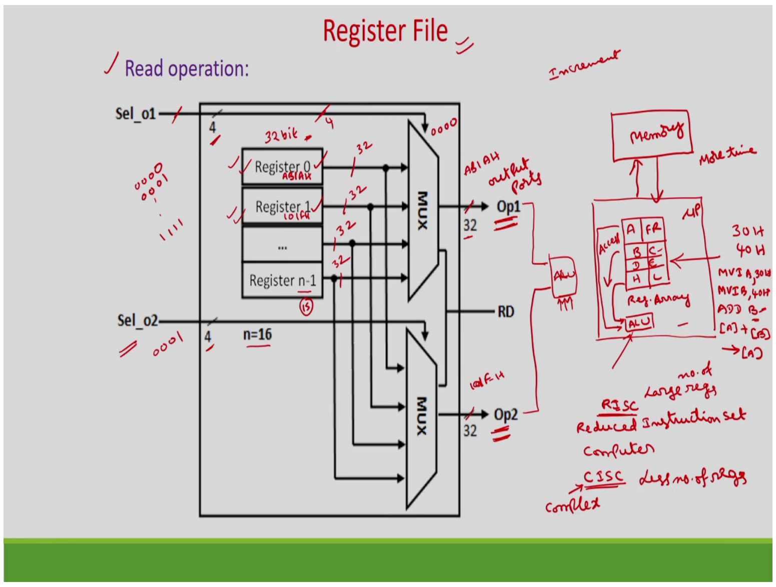
System Design Through VERILOG

Prof. Shaik Rafi Ahmed
Department of Electrical and Electronics Engineering
Indian Institute of Technology, Guwahati

Test Benches
Lecture - 23
Arithmetic and Logic Unit (ALU)

Ok. In the last lecture, we have discussed about the implementation of arithmetic logic unit using VERILOG. So, today we will discuss some more case studies.

(Refer Slide Time: 00:49)



So, the next one is a register file. Since, I have told in the earlier lectures also our main intention is to design a processor. So, one of the important block in the processor is a arithmetic logic unit, which we have discussed in the earlier lecture. Then, another important block is a register.

If we take this 8085 microprocessor architecture. So, this will be having 7 registers. One is called accumulator, then we have register B, register C, D, E, H, L. Then, we have flag register also. So, this is called register array. So, this a register array will act as a small memory, this will be present inside the microprocessor. So, here will be ALU also. So, this

ALU have discussed in the last lecture. Now, we will discuss the register array. These two are the main blocks in the arithmetic logic unit.

So, this number of registers inside the microprocessor depends upon the type of the processor that we are going to design. If you want to design, say, RISC processor; Reduced Instruction Set Computer is transfer Reduced Instruction Set Computer processor or you can have CISC complex instruction set; C stands for Complex remaining thing is same.

As the name implies this contains complex instruction set whereas, this will be having less instructions. So, in this case the register length size is more. This will be having large register; large number of registers, this will have less number of registers. Of course, we require this microprocessor, the outside this microprocessor we have the memory.

We can access the data from the memory also, but the memory access will take more time. Whereas, to access the data from the registers, this will take less time, because the registers are present inside the microprocessor itself this will take less time. In order to increase the processing speed of the computer, we can have more number of registers inside the microprocessor.

So, here I have considered a case study where I will consider register file, how to implement a register file using Verilog. So, here I have considered there are 16 registers. I have taken n is 16 this is n minus 1 is 0 to 15 this is register 0 1, this becomes n minus 1 becomes 15 for n is equal to 16. So, totally we have 16 registers and each register is capable of storing 32-bits of the information.

So, we are going to implement a register file or register array which contains 16 32-bit registers. So, we know that we can either read the data from the register or we can write the data into register. There are two operations in the microprocessor; one is called read operation, write operation. So, here, I will explain first the read operation.

So, already in this register some data is present I want to read the data. So, in order to read the data from particular register, first you have to select a particular register. For that we are using two output ports from where the data is available. Why do we require the two output forces as you have discussed in the last lecture also; so, we have this ALU, ALU will take two operands. In most of the operations, it requires two operands.

So, in the last lecture also, we have taken how this operation will be performed on the two operands addition, subtraction, multiplication and all. But there are some operations which requires only one operand like; increment operation, decrement, finding the 1's complement of a number. So, this requires only one operand. In that case, the second output port will be disabled.

But in general for addition, multiplication, subtraction, ANDing logical ANDing logical ORing; we require at least two operands. So, we require two output ports, this will give 1 operand, this will be 1 operand. So, normally in case of 8085 if you want to add two numbers say, 30 H plus 40 H. So, we will take 30 H into one register say normally, we will take one of the operands into the accumulator, using the instruction move immediate A comma 30 H move immediate B comma 40 H, then we will perform ADD B.

Because here, always one of the operand has to be taken into the accumulator, this add the contents of A plus this register whatever the register you can take this second operand to either B C D E H L; plus B and the result is stored into again accumulator. So, like that here we can store the one number in the one register second number in the second register we can perform the addition.

These two output this will be connected to these two will be connected to ALU. So, which perform the prescribed operation, depends upon the selection lines of this one the operation can be addition, multiplication, subtraction, as you have discussed in the last lecture.

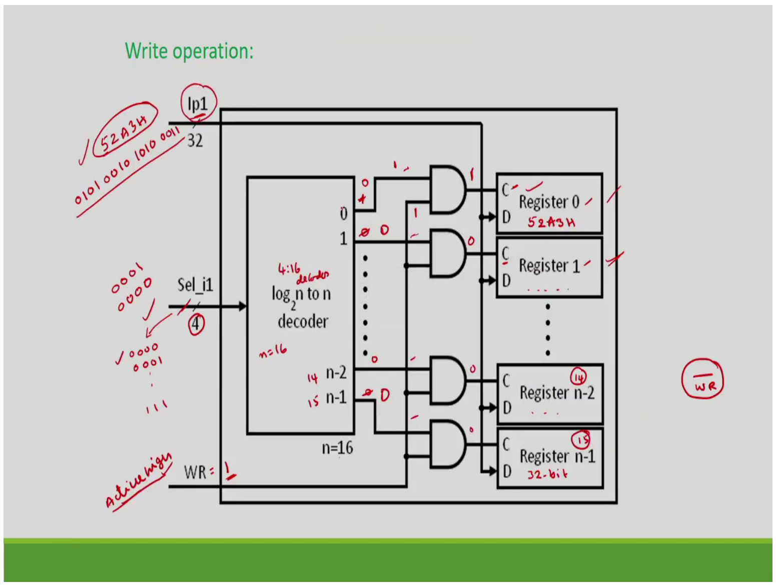
So, we require two operands. So, we require two output ports this is one output port this is another output port. So, I am using two multiplexers. So, because each data is 32-bit. So, these are all 32-bit lines at a time 32-bit will be fed to this multiplexer here also, this is 32-bit output of the multiplexer is 32-bit.

And out of this 16 we have to select 1. So, 16 inputs are there of course, each input is 32-bit number and I have to select at any time only one. So, we require four selection signals. If these four selection signals are 0 0 0 0, register 0 is selected 0 0 0 1, register 1 is selected. So, 1 up to if it is 1 1 1 1 these are four lines. Here, we have to show 4 instead of 4, I can write this as a 4. 1 1 1 1 register 15 will be selected.

Similarly, I have this is 4-bit number. This is called select output 1 select output 2, this is also another 4-bit. Suppose, if you want to select the contents of register 0 register 1 then. So, on this register, I will use 0 0 0 0 so that register 0 will be selected. So, the register 0 contents will be output here. So, 32-bit number if it is say for example, if I take in hexadecimal A B 1 A H. So, this A B 1 A H will be available here. And if I give here, 0 0 0 1 register 1 contents will be selected if this is a 101FH. So, this 101FH will be available here.

After that that will be given to the accumulator. This is how we can perform the read operation. We can read the data from any two registers simultaneously. If you want only one, then the second one will be disabled, but most of the applications we need two operands. So, we have to select two registers, using two selection signals. This is about read operation. Then how to perform the write operation?

(Refer Slide Time: 09:58)



Same 32-bit registers we are using here also. So, this is register 0 1 for n is equal to 16 this will be 14 this will be 15. So, totally we have 16 registers, each register is 32-bit. So, I want to write something. So, I can write a 32-bit number into either this register 32-bit number into this register or this register this register, depends upon this selection signal.

So, in order to select this, we require 32. So, this is for n is equal to 16, this is equal to sorry, 32-bit number but only we have 16 registers. So, this is 14 this is 15 this is basically decoder it is ok for n is equal to 16. So, this decoder becomes 4 to 16 decoder. Here, we have 4 inputs 16 outputs if all are 0's this inputs this 4-input is all are 0's, then 0 is selected; means, this is 1 remaining all are 0's.

This is positive logic, this I have already explained in the earlier lectures. If this is 0 0 0 1, this one is selected 1 will be selected this will be 1, remaining all will be 0's. If it is 1 1 1 1, the last register will be selected. So, like that we can select one of these registers through this logic, but instead of directly applying this output of the decoder to the register. So, you have to provide the right logic, because we have also read. So, we require a control signal right.

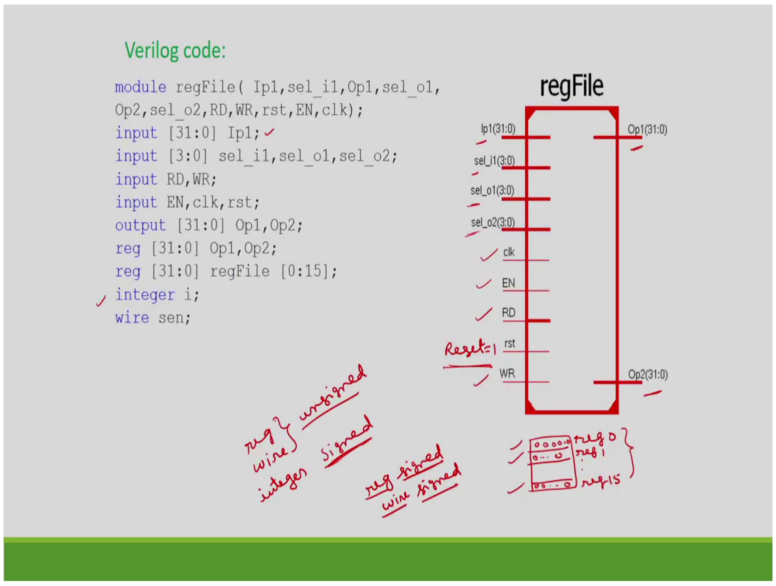
This is also I am calling as active high signal for the sake of simplicity, but normally in microprocessors, this will be active low signal that will be called as W bar signal write bar, but here for the sake of simplicity I am taking active high. So, whenever write is equal to 1 then only it will perform the write operation.

So, that is why I have given here 1. Suppose, if this is all 0's, this will be 1 this will be 1 then remaining all will be 0's and write bar is 1. So, the output of this 1 will be 1. So, remaining all will be 0's. So, this one will act as an enable signal for this register. So, this will be enable this register will be enabled remaining all will be disabled.

And we give whatever the data say 52A3H 32-bit data which is 5 nothing but 0 1 0 1 0 0 1 0 a is 10 1 0 1 0 3 is 0 0 1 1. This 32-bits of this information if I take here, and if I make right equal to 1 and here if I take this as 0 0 0 0 then this 32 52A3H will be written into this register.

If this is 0 0 0 1 the same data is there, because this data is common to all the registers, but at any time only one of the register is enabled through this output of the AND gate. So, that data will be feed into the corresponding register which is enabled. So, at any time, only one of the register is enabled, because at any time, one of the inputs of this area of and gates is 1. So, only one of the register is selected. This is how you can perform the write operation.

(Refer Slide Time: 14:01)



Verilog code:

```
module regFile( Ip1,sel_i1,Op1,sel_o1,
Op2,sel_o2,RD,WR,rst,EN,clk);
input [31:0] Ip1;
input [3:0] sel_i1,sel_o1,sel_o2;
input RD,WR;
input EN,clk,rst;
output [31:0] Op1,Op2;
reg [31:0] Op1,Op2;
reg [31:0] regFile [0:15];
integer i;
wire sen;
```

Then coming for this Verilog code. So, this is the overall register with input and output signals. So, this is the input data where we have to give 32-bit data. This is selection of the input 4 to 16 decoder. So, input lines are 4. So, this is selection of the output. So, you have a multiplexer which is a 16 to 1 multiplexer.

So, we require 4 selection signals and for the second 16 to 1 multiplexer 4 selection signals. Then, we require clock, because this is a sequential circuit. This is overall enable for this register file. And this is read signal this is write signal this is equal to 1. If this is equal to 1, read operation. If this is equal to 1, write operation.

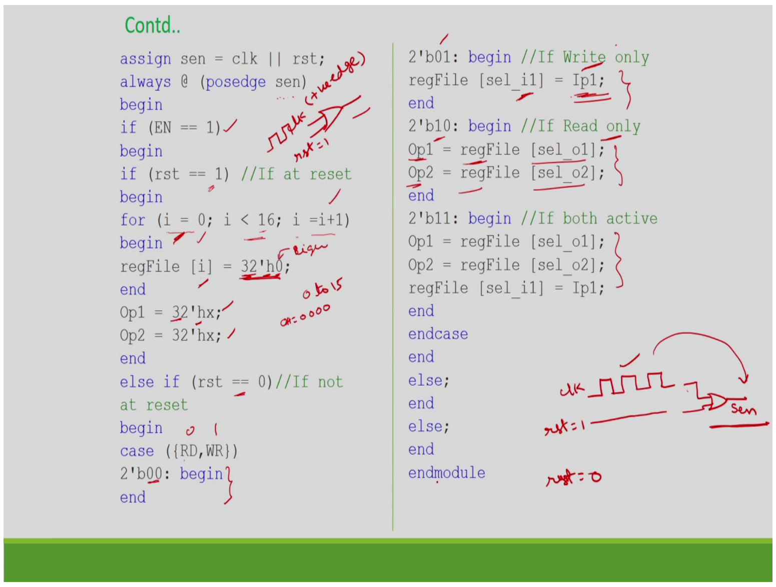
There is an additional signal which you have shown here is a Reset signal. Yes, we know that any digital circuit will have some Reset. If you want to clear all the 32 bit registers to 0 0 0 0, simply you make reset is equal to 1. So that all these 32-bit 16 registers. This is register 0 register 1 so on up to register 15. So, this will be 32-bits 0 0 0 0; all the 32-bits will be 0's, when Reset is equal to 1.

And these are the output ports from where you can read the data, as I have explained in the earlier slides. So, coming for this Verilog code. So, module register file is the name of the Verilog code. So, the inputs outputs I have defined. Then, the input is input port which is 32-bit, because we have to write the data which is 32-bit data, because the size of the registers is 32-bit. Similarly, input selection signals are 4-bit. So, this input read write for read operation read equal to 1, write operation write equal to 1.

Then, we have three more extra input signals; enable, clock and a reset. Output is we have to output port 32-bit and this we have to define as a registers also. In case of a non-blocking assignments or sequential circuits normally, we will define the output as registers, then integer i.

So, the difference between this register, wire and integer is. So, these are by default unsigned whereas, integer is signed. So, if you want to use this register for this signed also, we can write register signed, wire signed. If you do not mention signed keyword after this register or wire, by default they are unsigned whereas, integer is signed. That is the difference. So, here I have taken i as signed then, wire I am going to define sen; what is this wire I will discuss in the next slide.

(Refer Slide Time: 17:36)



Contd..

```
assign sen = clk || rst;
always @ (posedge sen)
begin
if (EN == 1)
begin
if (rst == 1) //If at reset
begin
for (i = 0; i < 16; i =i+1)
begin
regFile [i] = 32'h0;
end
Op1 = 32'hx;
Op2 = 32'hx;
end
else if (rst == 0)//If not
at reset
begin
case ({RD,WR})
2'b00: begin
end
2'b01: begin //If Write only
regFile [sel_i1] = Ip1;
end
2'b10: begin //If Read only
Op1 = regFile [sel_o1];
Op2 = regFile [sel_o2];
end
2'b11: begin //If both active
Op1 = regFile [sel_o1];
Op2 = regFile [sel_o2];
regFile [sel_i1] = Ip1;
end
endcase
end
else;
end
else;
end
endmodule
```

So, this basically I have given a OR gate here with one input as clock. This is positive edge clock other input is reset. So, when reset is equal to 1 it just neglect the clock. So, it will reset all the 16 32-bit register to 0 0 0 0. So, here even if you apply the clock signal here, if one input is the clock, another input is reset which is say high this is clk.

If you apply these two inputs to the OR gate what will be the output, because if one input is 1 for the OR gate output is always 1. So, it is independent of the clock. It just neglects the clock. The output becomes just simply high. When reset is equal to 0. If reset is equal to 0, then what will be output of this OR gate as it is the clock will be the output, because if one input is 0 output is the second input itself.

So, for that we have used output of this OR gate we have called as sen. So, we have defined this as a wire. So, always at the positive edge of sen. So, this positive edge happens only when reset is equal to 0. So, that is a positive edge of clock itself. When reset is equal to 1, so nothing to do with this positive edge will never occur.

So, then we are checking again for enable is equal to 1. If enable is equal to 1, then only I can perform the operation, whether read or write operation. If reset is equal to 1 as you have told, simply all the 16 registers will be made as 0's. So, this is a for loop i is equal to 0 it will increment by 1 every time.

So, if it is less than 16, up to 15 it will count. So, this register file 0 to register file this i varies from 0 to 15; this will be loaded with a 32-bit number this is 32-bit pin number this is hexadecimal 0 0 means all zeros 1 0 is hexadecimal is equal to 4-bits. This is hexadecimal is equal to 4-bits.

So, 32-bit means, totally we have eight 0's. But normally, because if all are 0's if I mention only one 0 it is understanding that there are eight 0's, because this is 32-bit. So, all the registers as you have told in the earlier slide; this all the registers will be loaded with all zeros, when reset is equal to 1 that will be explained by this line.

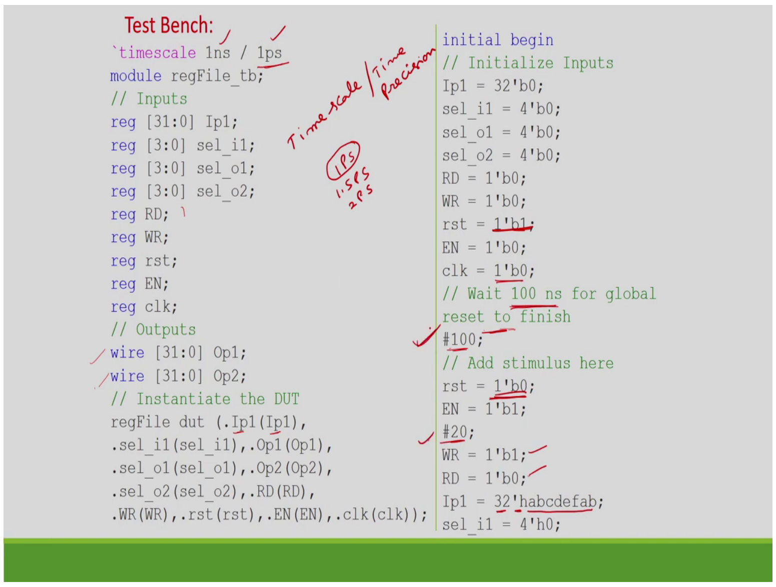
And what about the output ports, what will be available here? This will be do not care. So, these two are do not cares output ports will be do not cares 32-bit hexadecimal; x represents do not care. If reset is equal to 0, then the normal operation. So, begin then case, read, write. So, if both read write are 0 0, neither read nor write operation. So, 0 0 no operation, so begin end nothing ideal. So, in 0 1; 0 1 means write operation.

So, write operation what you have to do you have to write. So, this is your writing for write operation you have to take the data into Ip1. Here, you have to take the data and you have to write onto a particular register that is selected; that is what you have done. So, this selected register ok.

Now, this i we have taken from 0 to 15. That selected register will be loaded with the whatever the data that is available at input port. For 1 0, 1 0 is nothing, but this is write operation and this is read operation. So, we will read from the output ports; whatever the data available in a particular register which is selected. This is selected through these selection lines. And 1 1, then both the operation this as well as this, then this is the end.

This is about this VLSI code Verilog code.

(Refer Slide Time: 22:31)



Test Bench:

```
`timescale 1ns / 1ps
module regFile_tb;
// Inputs
reg [31:0] Ip1;
reg [3:0] sel_i1;
reg [3:0] sel_o1;
reg [3:0] sel_o2;
reg RD;
reg WR;
reg rst;
reg EN;
reg clk;
// Outputs
wire [31:0] Op1;
wire [31:0] Op2;
// Instantiate the DUT
regFile dut (.Ip1(Ip1),
.sel_i1(sel_i1),.Op1(Op1),
.sel_o1(sel_o1),.Op2(Op2),
.sel_o2(sel_o2),.RD(RD),
.WR(WR),.rst(rst),.EN(EN),.clk(clk));
```

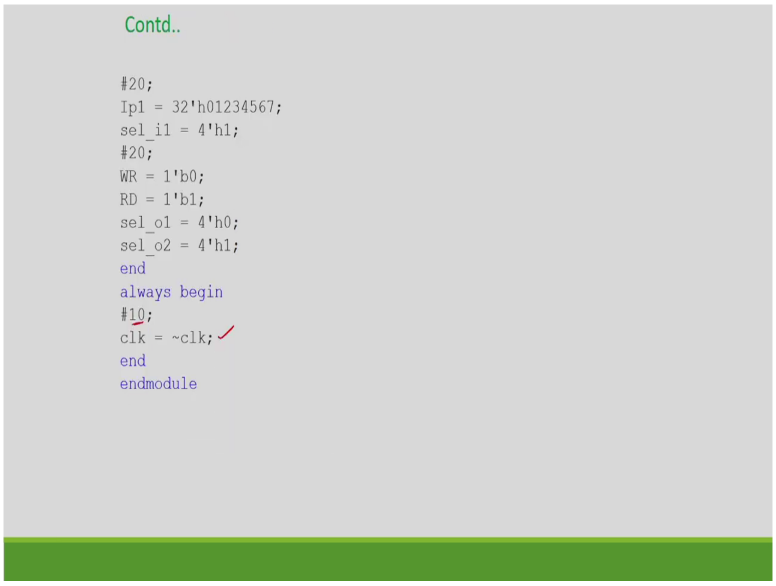
```
initial begin
// Initialize Inputs
Ip1 = 32'b0;
sel_i1 = 4'b0;
sel_o1 = 4'b0;
sel_o2 = 4'b0;
RD = 1'b0;
WR = 1'b0;
rst = 1'b1;
EN = 1'b0;
clk = 1'b0;
// Wait 100 ns for global
reset to finish
#100;
// Add stimulus here
rst = 1'b0;
EN = 1'b1;
#20;
WR = 1'b1;
RD = 1'b0;
Ip1 = 32'habcdefab;
sel_i1 = 4'h0;
```

Then a test bench. So, test bench, I have defined these signals as same signal for the test bench, as well as DUT. So, remaining registers will be this inputs and outputs and outputs will be wires, inputs will be registers and then I have given the initialization. So, this is actually self explanatory. So, I have kept the first reset signal for the 100 nanoseconds.

So, another important thing is here, 1 nanosecond 1 picosecond is there. The meaning of this one is the upper one represents time scale divided by, second one represent time precision. Means, if 1 picosecond change in the time is there, then only it will recognize a value of the time which is less than 1 picosecond is not recognized.

If I apply say for example, initially 1 picosecond 1.5 picoseconds we cannot give, we can give next 2 picoseconds only. So, a minimum of 1 picoseconds change is required to recognize the circuit. Here, 100 nanoseconds we are keeping this reset operation. After that we are making reset as 0; reset is 1 after 100 nanoseconds reset is 0. Then after 20 nanoseconds, we are doing this is write operation read is 0.

(Refer Slide Time: 24:18)



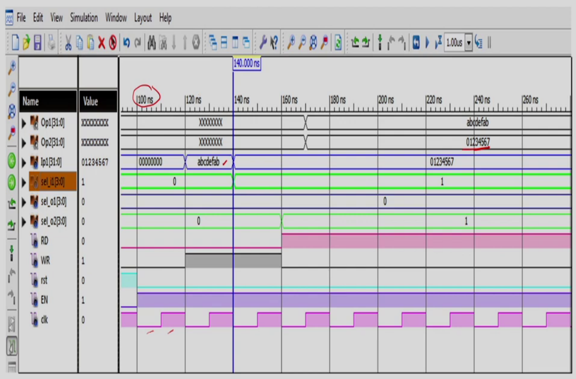
Contd..

```
#20;
Ip1 = 32'h01234567;
sel_i1 = 4'h1;
#20;
WR = 1'b0;
RD = 1'b1;
sel_o1 = 4'h0;
sel_o2 = 4'h1;
end
always begin
#10;
clk = ~clk;
end
endmodule
```

So, like that. So, we have completed this. And clock is for every 10 nanoseconds, clock becomes clock bar. This logic we have already explained in the earlier lecture also.

(Refer Slide Time: 24:30)

Simulation Results:



So, this is the corresponding waveforms. Once you have told this reset is initially high up to 100 nanoseconds after that reset will be low. So, that it will recognize the inputs and enable is high at 100 nanoseconds in the code that is written. Clock is for every 10 nanoseconds it is going to be complemented.

So, that 120 nanoseconds; 120 nanoseconds what you are feeling this 32-bit hexadecimal this h stands for hexadecimal a 2 a b c d e f a b. So, that we have stored at 120, this is 20 100; 120 a b c d e. At 140 another 20 we have 0 1 2 3 5 5 6 7, this is 0 1 2 3 4 5 6 7. Like that you can easily verify these are the simulation results which will implement the register file.

So, this is the second case study. Next, we will discuss some digital filter implementations as a case study in the next lectures.

Thank you.