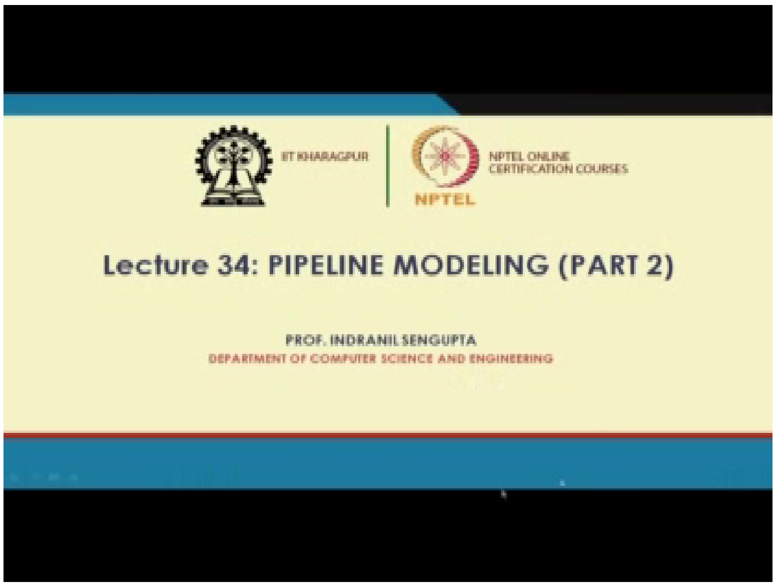
Hardware Modeling using Verilog

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

Lecture - 34
Pipeline Modeling (Part 2)

So, in our last lecture, we had seen an example where we can map a given computation into a pipeline and implemented in Verilog. In this lecture, we shall be taking up another example slightly more complex and look at a more conservative and a more proper way of clocking that will lead to correct operation without any race condition possibilities.

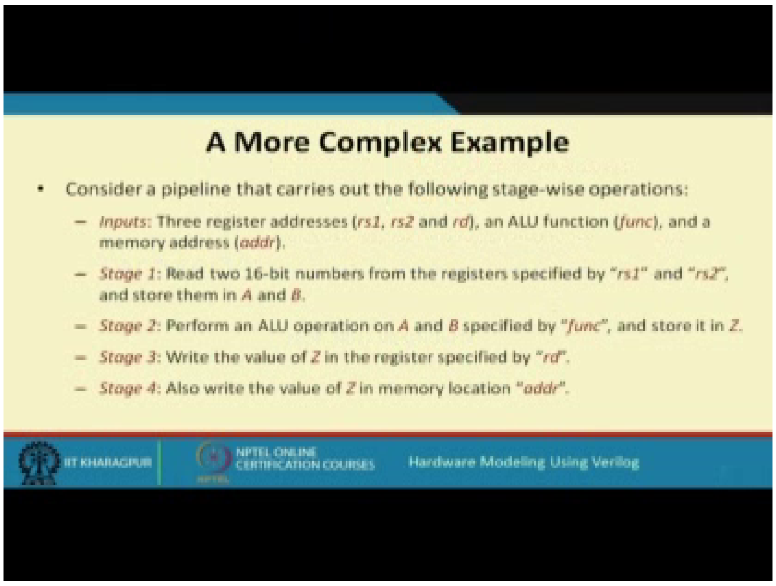
(Refer Slide Time: 00:51)


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES NPTEL

Lecture 34: PIPELINE MODELING (PART 2)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Refer Slide Time: 00:56)



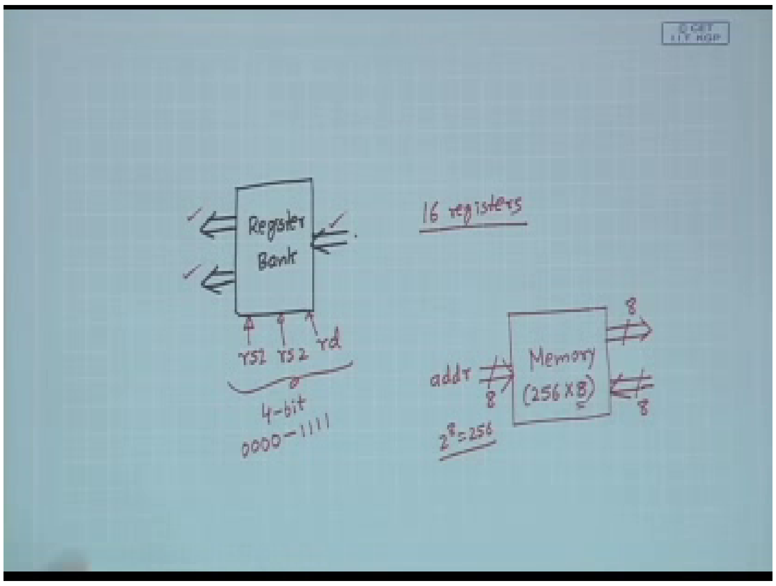
A More Complex Example

- Consider a pipeline that carries out the following stage-wise operations:
  - *Inputs*: Three register addresses (*rs1*, *rs2* and *rd*), an ALU function (*func*), and a memory address (*addr*).
  - *Stage 1*: Read two 16-bit numbers from the registers specified by "*rs1*" and "*rs2*", and store them in *A* and *B*.
  - *Stage 2*: Perform an ALU operation on *A* and *B* specified by "*func*", and store it in *Z*.
  - *Stage 3*: Write the value of *Z* in the register specified by "*rd*".
  - *Stage 4*: Also write the value of *Z* in memory location "*addr*".

So, this is the second part of our lecture in pipeline modeling. Now, here the example that we take is actually a 4 stage pipeline example. Now here, you do not try to understand that why we are doing this. It is just some example stages and the reason we have taken this example is that later on; we shall be looking at the complete design of a processor. Processor means a central processing unit or a CPU; how it can be implemented in Verilog; in a pipeline fashion, there some of the concepts that I will show in this example will be used. So, just to get a feel; so, we have incorporated a few of the complex blocks in our design. Here, let us see what we have done.

Here we are assuming that there is a register bank and ALU. So, I will explain this first thing to notice that in this problem.

(Refer Slide Time: 02:01)



I am assuming that there is a register bank register bank we have seen that you can have multiple registers stored here and you are assuming that there are 2 read ports and 1; sorry, 2 read ports and 1 write port, yes.

Now, in order to activate them, I am assuming that you have the control signals; 1 is read source 1, read source 2 and register source 1, register source 2 and register distribution. So, depending on rs 1 and rs 2; the corresponding registers will be read here and here and depending on rd the values applied here will be written into the corresponding register

Now, we shall see how this register bank, let us say if there are 16 registers in this register bank, then all this rs 1, rs 2 and rd will be 4 bit numbers because you can specify one out of 16 registers in 4 bits 0, 0, 0, 0 to 1, 1, 1, 1 register number 0 up to register number 15.

Similarly, we are assuming that we have a memory. This is a very simple example of a memory, we are assuming that there are 256 words in the memory and each word is of 8 bits. So, since there are 256 bits in the memory, there will be an address, this will be 8 bits in size because we have 8 bits of data. So, if there are separate data out and data in lines; they will be 8 bits and 8 bits also this is 8 bits because 2 to the power 8 is 256 that is why you need to 8 bits of address and this is 8 because data size is 8 and addition you have the control signals; read, write, all those are there.

So, with this, we are stating the problem; what we are trying to do? So, the inputs to our pipeline will be 3 register addresses that will be corresponding to the register bank, there is also an arithmetic logic unit an arithmetic functional block where we specify some function; what kind of operation you want to do and for the memory we supply a memory address.

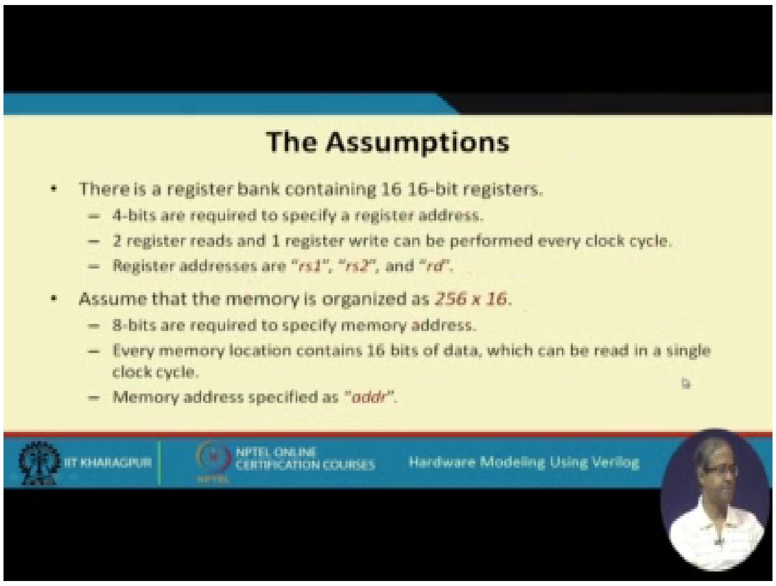
So, you see here; there are 3 functional units, we have talked about a register, we have talked about a memory and also there is an arithmetic logic unit arithmetic and logic unit. So, ALU will be taking 2 input data, it will carry out some computation and what computation that is determined by a control word called function, right.

So, these 3 kind of blocks are required in this pipeline implementation. So, the inputs will be what are the register addresses; what is the ALU function; we want to do and what is the memory address and what are the computations that we have to do; let us see in stage one; we read 2 numbers from the register bank, we assume that all numbers are 16 bit numbers in the registers; registers are 16 bits depending on whatever you have given in rs 1 and rs 2 we read 2 16 bit numbers and store them in 2 temporary registers a and b.

Stage 2; we perform the ALU operation, we operate on a and B and what operation it depends on this func; what function it can be addition, subtraction, whatever I have specified and after the ALU operation is done the result is sorted again in a temporary register Z.

In stage 3, what we do the result Z, we are writing back into the register bank depending on whatever we have given in rd destination register and in stage 4, the same data Z; we are also writing into memory at address addr, right.

(Refer Slide Time: 07:16)



The Assumptions

- There is a register bank containing 16 16-bit registers.
  - 4-bits are required to specify a register address.
  - 2 register reads and 1 register write can be performed every clock cycle.
  - Register addresses are *"rs1"*, *"rs2"*, and *"rd"*.
- Assume that the memory is organized as *256 x 16*.
  - 8-bits are required to specify memory address.
  - Every memory location contains 16 bits of data, which can be read in a single clock cycle.
  - Memory address specified as *"addr"*.

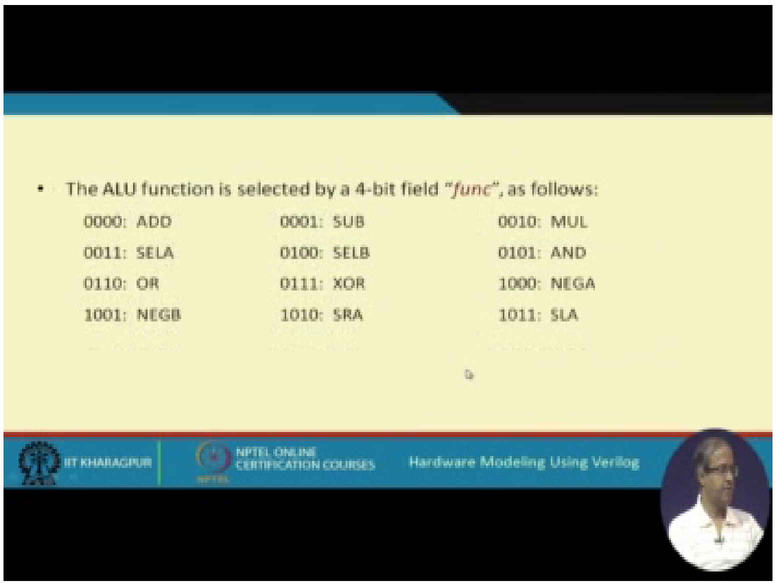
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, this is just an assumption we are making these are the stages of computation; let us see the assumption that we have made is that the register bank, we have said it consist of 16-16 bit registers because there are 16 registers, we need 4 bits to specify the register address. So, rs 1 and rs 2 and rd will be 4 bits and we assume that 2 register reads and one register writes are possible every cycle; there are 2 read ports and one write port.

Well and because the data size is 16 bits; we have assumed and the memory also has to be 16 bit word; here we have said that the memory is 8 bits, but actually if to make it compatible; this has to be 16 bits. So, let us consider these to be 16, right; data bus size will be 16.

So, for this 256 word memory; this 8 bits will be required to specify the memory address and this memory address is specified by this addr. So, addr is an 8 bit quantity and the data will be 16 bits in size.

(Refer Slide Time: 08:29)



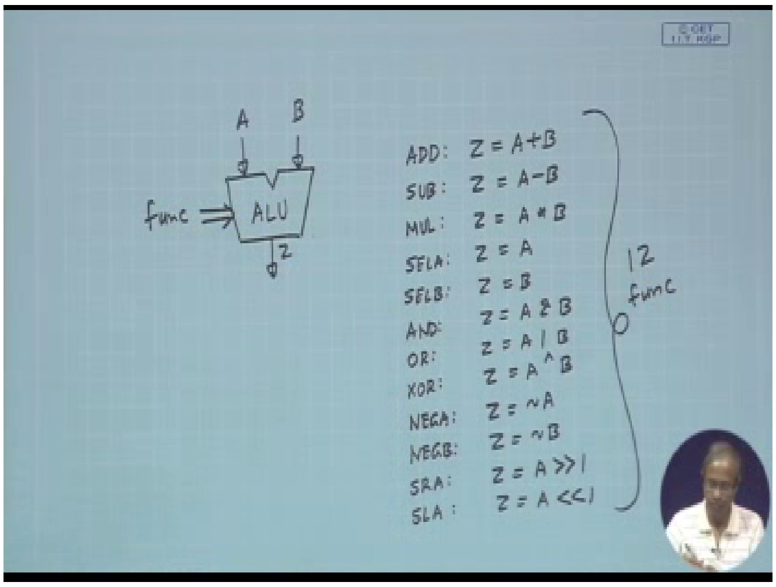
- The ALU function is selected by a 4-bit field *"func"*, as follows:

0000: ADD	0001: SUB	0010: MUL
0011: SELA	0100: SELB	0101: AND
0110: OR	0111: XOR	1000: NEGA
1001: NEGB	1010: SRA	1011: SLA

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Now, the ALU functions; we assume are as follows because func is a 4 bit field and this 4 bit; well, we have not specified for all possibilities there are 000 up to 1011 means there are 12 functions which are defined; let us see what functions are there.

(Refer Slide Time: 08:57)

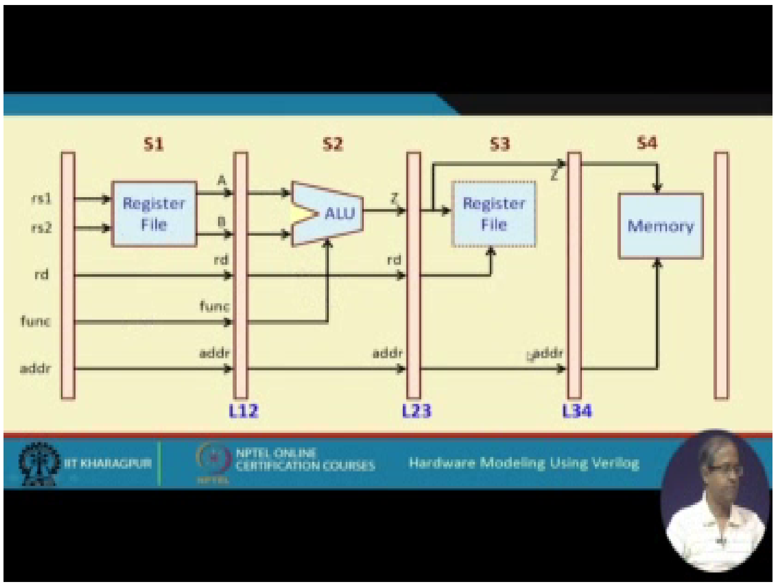


ADD means let us look at it like this; suppose, this is my ALU, the 2 inputs are A and B and the output is C. So, let us see one by one ADD 000 means ADD; ADD means Z equal to A plus B 0001 means SUB; SUB means Z equal to A minus B 10 means MUL; MUL means multiply; MUL means Z equal to A multiply B, then fourth is select a cell a means if you select cell A, then the value of A will be going to the output; you are selecting A.

Similarly, there is an option for selecting B, if you use cell B, then output will be B, then you have an option AND and OR and XOR. So, AND, OR and XOR; so, for and Z will be bit by bit; A and B or it will be A or B and for XOR; it will be again bit by bit exclusive or A XOR B, then you have a function NEG A and NEG B; NEG A means it is just not bit by bit negation of A and NEG B means bit by bit negation of B, right and the last 2 functions are shift right A and shift left A.

So, if you use S r a; this will mean you take A and shift it right by one position and if it is shift left A Z, it will be a shift left by 1 position. So, these are the 12 functions that the ALU supports and this is controlled by the input func 4 bit value, fine.

(Refer Slide Time: 11:42)



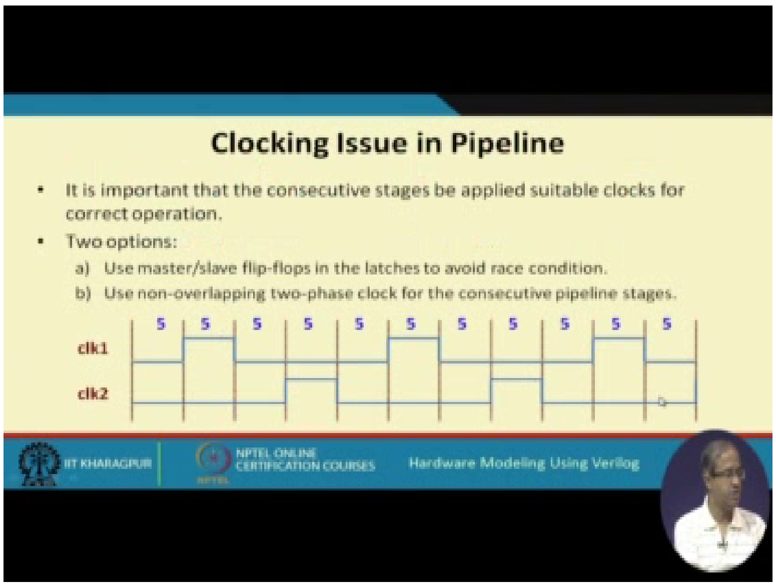
Now, this is our overall pipeline diagram based on whatever we have said let us see this rs 1, rs 2 rd func and addr; these are our inputs. So, we have a register file in stage one based on whatever you have given in rs 1 and rs 2, we are accessing the registers and values are read into A and B nothing else is has been done in S 1.

In S 2; we are doing some functions on these 2 values which have read from the register bank register file A and B which are coming; they will be the inputs of the ALU and the function that we have given here that will be forwarded and that function will be controlling the ALU operation and the result will be Z.

In stage S 3, we are writing in the register file, we are showing it as dotted because this is not a separate register file. It is the same register file; I am just showing it that it is being written here that is why dotted.

So, what we are writing, we are writing the value of Z and while we are writing that is rd register destination. So, this has to be forwarded up to here and then last stage we have the memory write where what we want to write the value of Z. So, Z has to be forwarded here, from here to here and address has to be forwarded all the way from here; down to here. So, it is a 4 stage pipeline where these operations are going on, right.

(Refer Slide Time: 13:22)



Clocking Issue in Pipeline

- It is important that the consecutive stages be applied suitable clocks for correct operation.
- Two options:
  - a) Use master/slave flip-flops in the latches to avoid race condition.
  - b) Use non-overlapping two-phase clock for the consecutive pipeline stages.

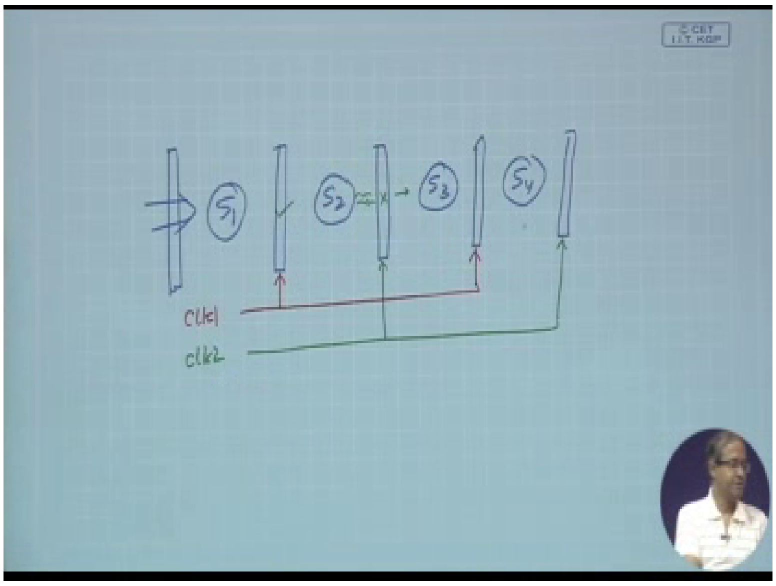
5 5 5 5 5 5 5 5 5 5 5
clk1
clk2
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, let us see how we can code it in Verilog, but before that there is an important clocking issue that I have said during the last lecture that we have to apply clock in a proper way, we mentioned that the 2 of the safe way is used to either use master slave flip flops or use non overlapping clocks for the consecutive stages. So, if we use non overlapping clocks, then you can also use latches rather than clocks which are triggered by edges.

Non overlapping; an example is shown here clock 1 and clock 2; you see clock 1 is high here, again it is high here, again it is high here. So, the clock period is 20 from here to here 20 and clock 2 is high when clock 1 is not high and there is a period in between where both the clocks are in active both are 0. So, there are 2 clocks sometimes clock 1 is high sometimes clock 2 is high, but they are not overlapping, they are non overlapped and in order to take care of variable delays like clocks queue, there is also a gap in between; this is a very safe kind of a clocking scheme where we have non overlapping clock with the safe margin in between this is what is meant by 2 phase clock.

Now, in a pipeline; what we are intending to do is as follows.

(Refer Slide Time: 14:58)

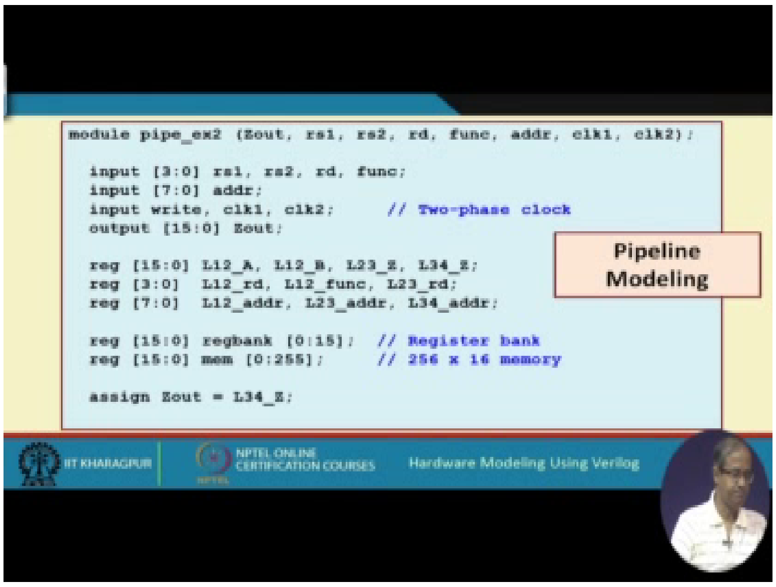


In this example, we have 4 stages, right, this is stage S 1 then stage S 2, then stage S 3, then stage S 4, this we are ignoring; here, we are just supplying the inputs. now out of this 4 latches; what we are saying is that latches of flip flops; whatever we say that we will be clocking, these 2 by clock 1 and we will be clocking these 2 by clock 2 which means we are alternately clocking the latches using the 2 phases.

Because we are doing that there is no scope on an overlap because when this latch is on it is guaranteed that this latch is not on. So, whatever S 2 is computing, it can never reach here because this latch is off because of this non overlap in nature, this kind of complete isolation of one stage from the other can be achieved.

So, in our Verilog test bench that we see or in the implementation also, we shall be assuming that we are having this kind of 2 phase clocking let us look at the Verilog code now.

(Refer Slide Time: 16:30)



```
module pipe_ex2 (Zout, rs1, rs2, rd, func, addr, clk1, clk2);

  input [3:0] rs1, rs2, rd, func;
  input [7:0] addr;
  input write, clk1, clk2;     // Two-phase clock
  output [15:0] Zout;

  reg [15:0] L12_A, L12_B, L23_Z, L34_Z;
  reg [3:0]  L12_rd, L12_func, L23_rd;
  reg [7:0]  L12_addr, L23_addr, L34_addr;

  reg [15:0] regbank [0:15];  // Register bank
  reg [15:0] mem [0:255];     // 256 x 16 memory

  assign Zout = L34_Z;
```

Pipeline Modeling

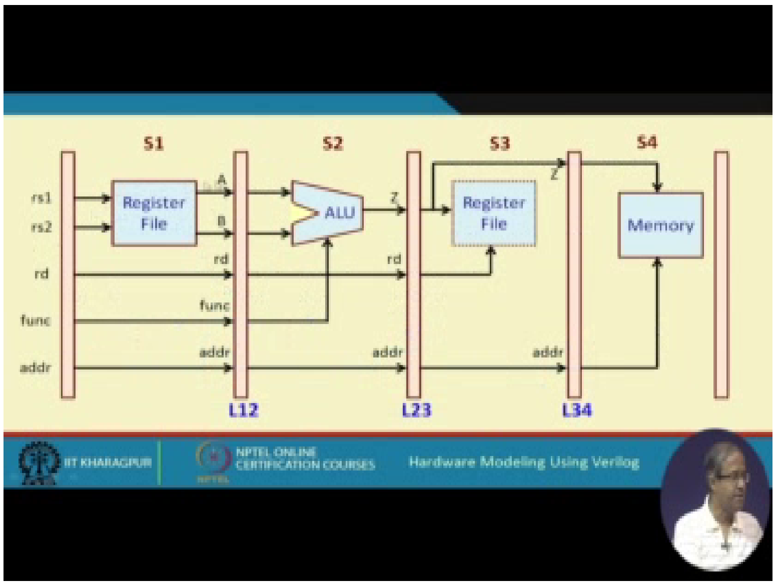
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

This is our pipeline description. So, we have the final output Z out; Z out again, this is a 16 bit quantity, we are assigning it from the final value of Z and rs 1, rs 2, rd func addr; there are all inputs and of course, there are 2 clocks; clock 1, clock 2.

Register select and also the function select these are all 4 bit quantities and memory address because there are 256 words, it is 8 bits and write of course, it is not required actually write can be there clock 1, clock 2.

Now, these are the intermediate register variables, we are defining you see this A and Z these are holding the results these are 16 bit quantities, but rd and func; these are 4 bit and address is 8 bits you see here you just go back to that diagram.

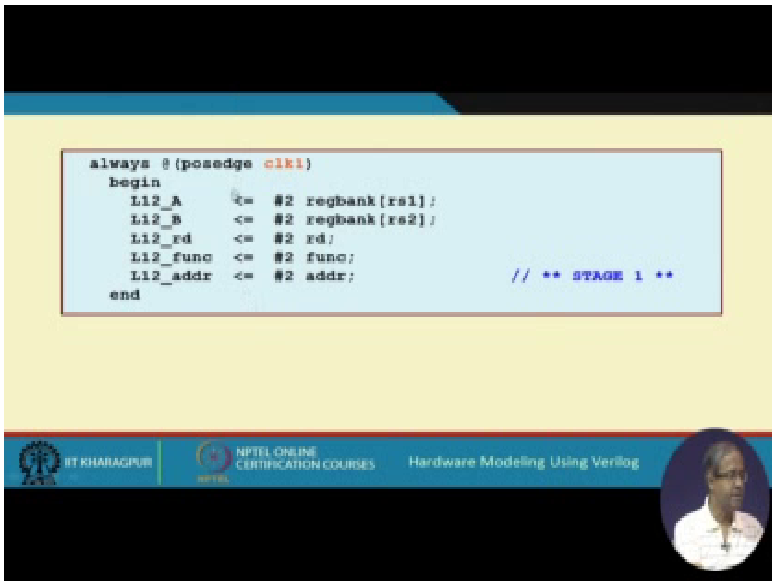
(Refer Slide Time: 17:42)



A B rd func and address are to be stored in L12 Z rd and addr in the L23 and Z and addr in L34. So, we are we have defined so many variables and the convention is as same as we followed in the last lecture, it will be L12; underscore A, L12 underscore B, L12 underscore rd and so on, right. So, same convention we have followed.

And we have defined a register bank and we have defined a memory separately both declarations are similar both are defined as a 2 dimensional array of registers. In fact, so in the first case, we have defined that there are 16 registers each of 16 bits. In the second, we have declared a memory of 256 words each of 16 bits and the final L34 Z which is been computed that is assigned to the final output Z out.

(Refer Slide Time: 18:50)



```
always @(posedge clk1)
  begin
    L12_A     <=  #2 regbank[rs1];
    L12_B     <=  #2 regbank[rs2];
    L12_rd    <=  #2 rd;
    L12_func  <=  #2 func;
    L12_addr  <=  #2 addr;                  // ** STAGE 1 **
  end
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

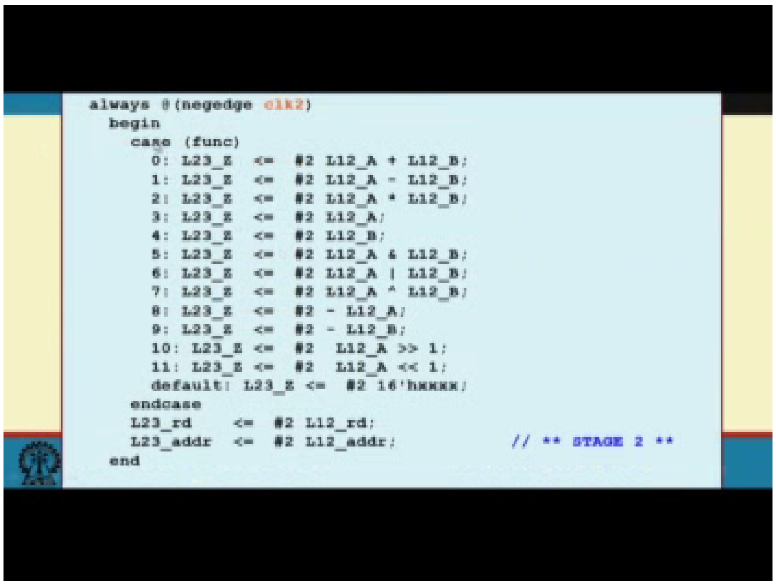
Now, you see; now how we have done; just I am showing the overall thing you see; this is the first stage, this is activated by clock 1; this is the second stage; stage 2 activated by clock 2; stage 3 activated by clock 1, stage 4 activated by clock 2. So, we are alternately applying clock 1 and clock 2 to the successive stages clock 1, clock 2, clock 1, clock 2; like that this is the first thing we have done here.

And if we look at the stage definitions, it is straight away it follows from whatever we have done like you see at the diagram; once more first stage, what we are doing? This A, B, we are reading from the register file from address rs 1 and rs 2 rd; we are forwarding func, we are forwarding addr; we are forwarding; now see I have done exactly that.

This L12 A is reg bank rs 1 reading from the register bank L12 B, reg bank rs 2 read, we are forwarding func; we are forwarding addr, we are forwarding and here we are assuming that everything takes a uniform delay. So, all 2, 2, 2; we have shown here.

Second stage second stage we have an ALU and some signals are forwarded like you go back; once more you see rd is forwarded addr is forwarded remaining thing is the ALU function it takes A B and func and generates Z.

(Refer Slide Time: 20:27)

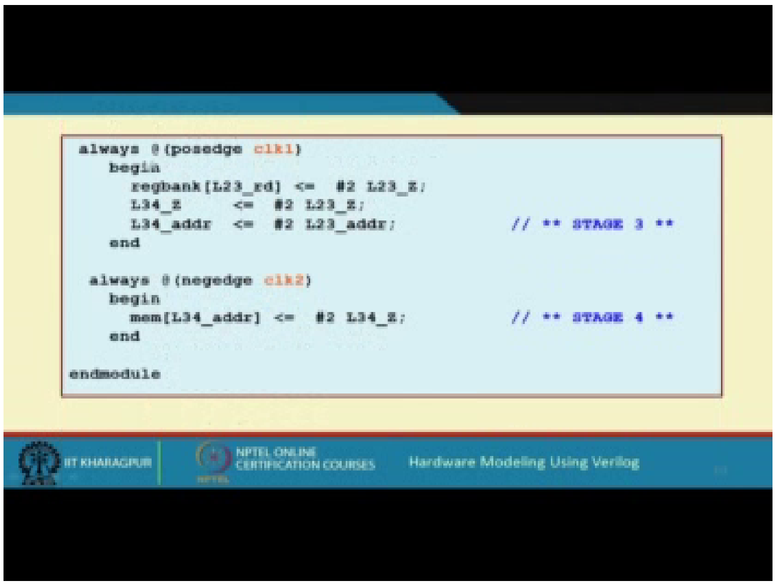


```
always @(negedge clk2)
  begin
    case (func)
      0: L23_Z  <=  #2 L12_A + L12_B;
      1: L23_Z  <=  #2 L12_A - L12_B;
      2: L23_Z  <=  #2 L12_A * L12_B;
      3: L23_Z  <=  #2 L12_A;
      4: L23_Z  <=  #2 L12_B;
      5: L23_Z  <=  #2 L12_A & L12_B;
      6: L23_Z  <=  #2 L12_A | L12_B;
      7: L23_Z  <=  #2 L12_A ^ L12_B;
      8: L23_Z  <=  #2 ~ L12_A;
      9: L23_Z  <=  #2 ~ L12_B;
      10: L23_Z <=  #2  L12_A >> 1;
      11: L23_Z <=  #2  L12_A << 1;
      default: L23_Z <=  #2 16'hxxxx;
    endcase
    L23_rd    <=  #2 L12_rd;
    L23_addr  <=  #2 L12_addr;                 // ** STAGE 2 **
  end
```

So, the ALU function is here case func. So, if it is func is 0; that means, 0, 0, 0, 0, then you do addition; you add store in Z, if it is 1, you do subtraction, multiplication, select A, select B AND or XOR negation of A negation of B, shift right A, shift left A and if it is mean any other value, these are undefined, then Z will be x x x x and the other 2 values we are forwarding.

This is the definition of state 2, stage 3 is very similar in stage 3; what we are doing again, let us go back in stage 3, we are writing into the register file and in stage 4, we are writing into the memory.

(Refer Slide Time: 21:24)



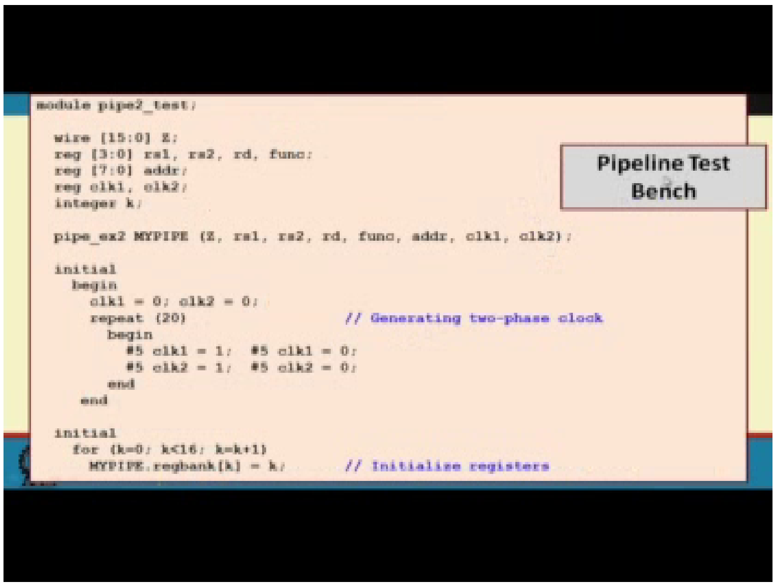
```
always @(posedge clk1)
  begin
    regbank[L23_rd] <=  #2 L23_Z;
    L34_Z       <=  #2 L23_Z;
    L34_addr    <=  #2 L23_addr;              // ** STAGE 3 **
  end

 always @(negedge clk2)
  begin
    mem[L34_addr] <=  #2 L34_Z;               // ** STAGE 4 **
  end

endmodule
```

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

So, we are doing exactly that here in stage 3. So, whatever is the value of Z you have computed we are writing into reg bank where the address is L23 rd and Z, we are forwarding addr, we are forwarding this will be requiring for the memory write and in the memory lastly the value of Z which has been forwarded that is been written into memory L34 addr.

Just one thing here we have mentioned an input write, but actually write is not required, this write you can emit this here; there is no write, fine, there is no write.

(Refer Slide Time: 22:18)



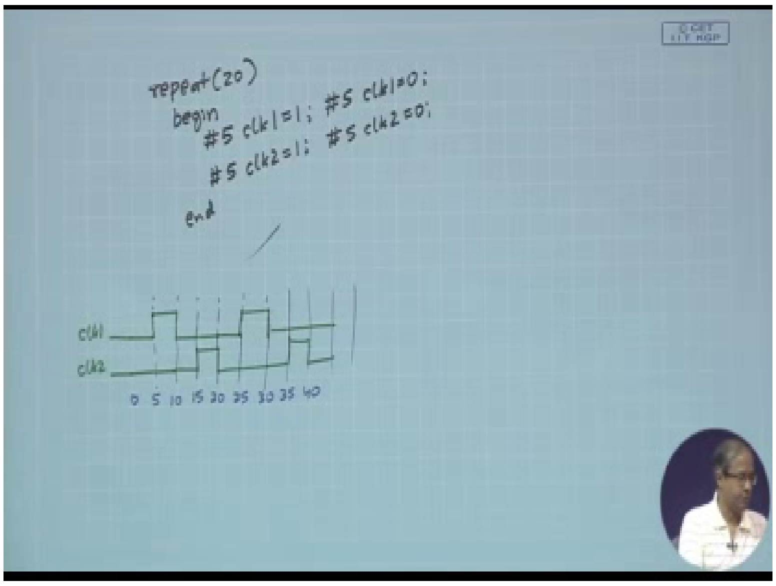
Pipeline Test Bench

```
module pipe2_test;

  wire [15:0] Z;
  reg [3:0] rs1, rs2, rd, func;
  reg [7:0] addr;
  reg clk1, clk2;
  integer k;

  pipe_ex2 MYPIPE (Z, rs1, rs2, rd, func, addr, clk1, clk2);

  initial
    begin
      clk1 = 0; clk2 = 0;
      repeat (20)                    // Generating two-phase clock
        begin
          #5 clk1 = 1;  #5 clk1 = 0;
          #5 clk2 = 1;  #5 clk2 = 0;
        end
     end

  initial
    for (k=0; k<16; k=k+1)
      MYPIPE.regbank[k] = k;         // Initialize registers
```

Now for this pipeline, let us try to write the test bench. So, here we have instantiated our pipe this is the output rs 1,, rs 2, rd func addr are the inputs and the 2 clocks. So, these are declared as reg because we will be initializing them 4 bits. This addr is also 8 bits reg clock 1, clock 2 is also reg, but Z is the output let us say it is a wire 16 bits and we have declared an integer k for the purpose we will see.

First thing is that let us see how we are initializing the clock you see clock is 0; clock 1, clock 2, both are initialized 0 at time 0 and we are generating 20 because 20 is enough for this example, we have given repeat 20. So, what you are doing? So, I am just writing the code and explaining; what is happening?

(Refer Slide Time: 23:17)



```
repeat(20)
  begin
    #5 clk1=1;  #5 clk1=0;
    #5 clk2=1;  #5 clk2=0;
  end
```

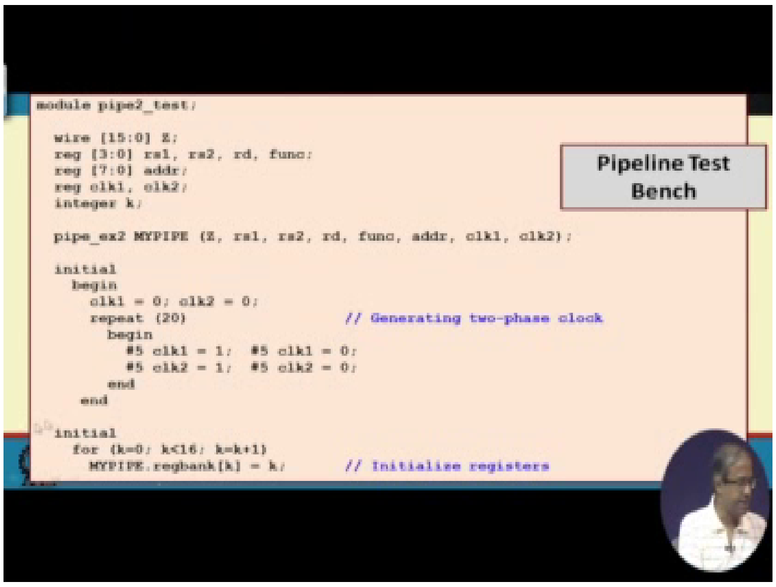
So, the Verilog code; we have written is repeat 20 and inside this, we have given a gap of 5 clock 1 equal to 1. Again a gap of 5 clock 1 equal to 0, then again a gap of 5 clock 2 equal to 1, again a gap of 5 clock 2 equal to 0, let us say what happens here.

Let us assume these are our time scale of 5, these are our time scale of 5, 5, 5, 5, these are gaps of 5; this is 5, this is 10, this is 15, this is 20, this is 25, 30, 35, 40 and so on, right.

Now, initially and this is 0 of course, this is 0, it starts with 0, right initially both clock 1 and clock 2 are 0s, this is clock 1 and this is clock 2; both are 0s, this is also 0, this is also 0. So, at time 5, we are making clock 1 equal to 1. So, at time 5, we are making clock 1 equal to 1. So, again after a delay of 5; we are making clock 1 equal to 0.

Clock 2 as 0 so long; so, again after a gap of 5; we are making clock 2 equal to 1; after gap of 5 clock 2 equal to 1 after a gap of 5 clock 2 equal to 0 this. So, clock 1 is 0 in the meantime. So, this we repeat. So, again after a gap of 5; we make clock 1 equal to 1 even after gap of 5 clock 1 equal to 1; after a gap of 5; 0 again after a gap of 5 after this make this one make this one you see; we have generated a perfect 2 phase clock like this right just the earlier diagram we showed. So, we are generating a 2 phase clock here.

(Refer Slide Time: 25:41)



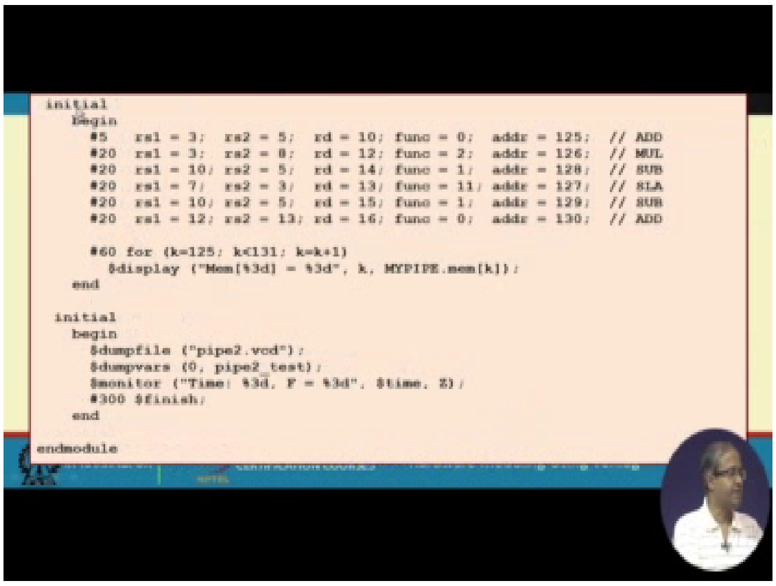
Pipeline Test Bench

```
module pipe2_test;

  wire [15:0] Z;
  reg [3:0] rs1, rs2, rd, func;
  reg [7:0] addr;
  reg clk1, clk2;
  integer k;

  pipe_ex2 MYPIPE (Z, rs1, rs2, rd, func, addr, clk1, clk2);

  initial
    begin
      clk1 = 0; clk2 = 0;
      repeat (20)                     // Generating two-phase clock
        begin
          #5 clk1 = 1;  #5 clk1 = 0;
          #5 clk2 = 1;  #5 clk2 = 0;
        end
     end

  initial
    for (k=0; k<16; k=k+1)
      MYPIPE.regbank[k] = k;          // Initialize registers
```

And in this initial block, we are initializing the register bank you see register bank is not accessible directly as a parameter. So, we are accessing it like this my pipe is the instantiated module dot reg bank was the variable which was declared inside it, right.

You see inside it there is a reg bank, right. So, we are referring it like this my pipe dot reg bank, in this for loop k goes from 0 up to 15 reg bank k equal to k which means register 0; get 0 register 1 gets 1 2 gets 2 3, gets 3, like that you are initializing register 15 gets 15, this is how we are initializing registers.

(Refer Slide Time: 26:34)



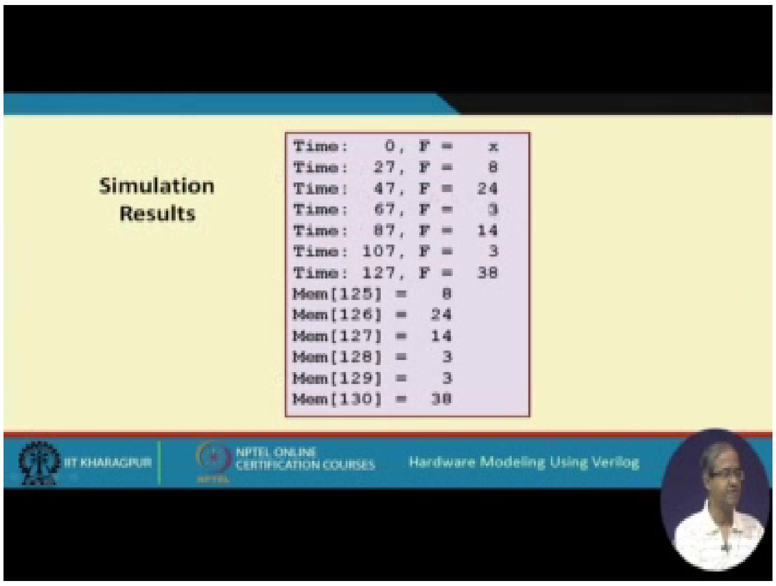
```
initial
  begin
    #5   rs1 = 3;  rs2 = 5;  rd = 10; func = 0;  addr = 125;  // ADD
    #20  rs1 = 3;  rs2 = 8;  rd = 12; func = 2;  addr = 126;  // MUL
    #20  rs1 = 10; rs2 = 5;  rd = 14; func = 1;  addr = 128;  // SUB
    #20  rs1 = 7;  rs2 = 3;  rd = 13; func = 11; addr = 127;  // SLA
    #20  rs1 = 10; rs2 = 5;  rd = 15; func = 1;  addr = 129;  // SUB
    #20  rs1 = 12; rs2 = 13; rd = 16; func = 0;  addr = 130;  // ADD

    #60 for (k=125; k<131; k=k+1)
      $display ("Mem[%3d] = %3d", k, MYPIPE.mem[k]);
  end

 initial
  begin
    $dumpfile ("pipe2.vcd");
    $dumpvars (0, pipe2_test);
    $monitor ("Time: %3d, F = %3d", $time, Z);
    #300 $finish;
  end

endmodule
```

Then in this initial block, we are applying some sample inputs. So, we are giving a delay of 5 and after the delay of 5, we are giving 20, 20, 20 gaps. So that next clocks are applied. So, what I am doing; just giving rs 1 equal to 3 and rs 2 equal to 5; that means, we are reading from registers 3 and 5. So, what we expect here register 3 contains 3 register 5 contains 5 that is all we have initialized, right and func 0 means add. So, 3 and 5 result will be 8 and this result 8 will be storing in register number 5 as well as memory address 125.

Similarly, next one is 3 and 8. So, register number 3 is still contains 3 8 still contains 8 and the function is multiplication 2. So, 8 into 3 is 24; this 24; you are storing in register number 12 as well as here 126. So, in this way you have some examples and at the end after everything is over here in a for loop after some delay you are displaying the memory contents that what is there in memory starting from 125 address onwards, right and also we are monitoring the value of f time and f.

So, if you run this simulation, you will see the output coming like this you see first output, I had said the result will be 8 second one will be 24; this we can verify because you see you have already modified register number 10 rd equal to 10, it becomes 8. So, in the third one when you are accessing register 10, it is not 10, it has become this has become 8. So, 8 and 5 it is subtraction 8 minus 5 you see result is 3 and you can verify the memory contents also they also contain the same values, right.

(Refer Slide Time: 28:42)

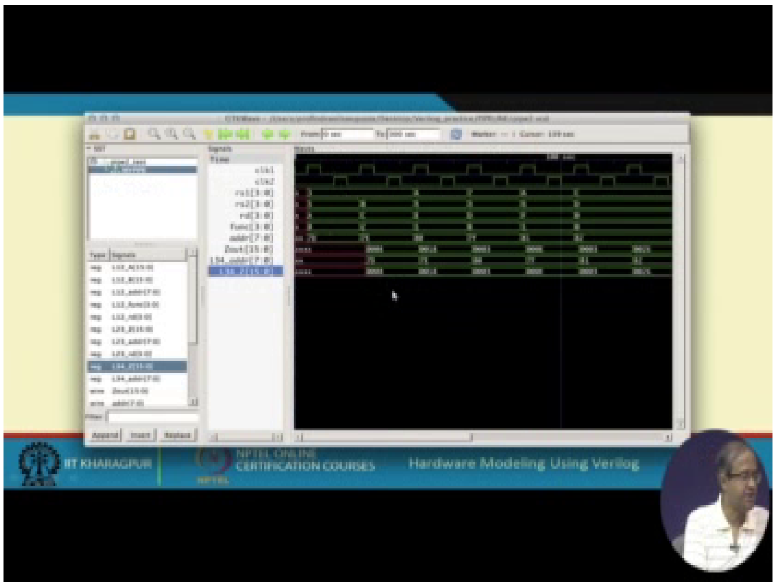


Simulation Results

```
Time:   0, F =  x
Time:  27, F =  8
Time:  47, F = 24
Time:  67, F =  3
Time:  87, F = 14
Time: 107, F =  3
Time: 127, F = 38
Mem[125] =  8
Mem[126] = 24
Mem[127] = 14
Mem[128] =  3
Mem[129] =  3
Mem[130] = 38
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

(Refer Slide Time: 28:57)



IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, if you look at the timing diagram using the same thing you can observe. So, you can see the 2 phase clock very nicely, this is 2 phase clock, this is the values of rs 1, rs 2 and rd you have given 3, 5 and 10 function; you have given 0 address, you have given something you see the test bench the address is 125; first 1, 3, 5, 10, function 0 address 125. So, 3, 5; A is 10 0 7; D is 125. So, this result will be final 0 0 0 8. So, you need some delay for that. So, that pipeline delay will required and at the end of the third clock you will be generating this result and after that result will be generated once every clock, right. So, this is exactly what is happening here.

So, what we have seen in this lecture is that this is the more recommended way to implement a pipeline that you use a 2 phase clock, but in the latches you can either use edge triggered in this Verilog code; we have written we have used pose edge triggered flip flops, but you can also use level triggered latches there as long as they took clock phases are non-overlapping and they sufficient gap between them even if we use non overlapping latches means those are called transparent latches not edge triggered there is no problem it will still work very fine.

So, we shall see later as it said with the more compressive example the design of a processor; there we will see how we can implement the instruction execution hardware data path of a processor in a pipeline fashion, there we will be following some of the lessons that we have learnt over the last 2 lectures.

Thank you.