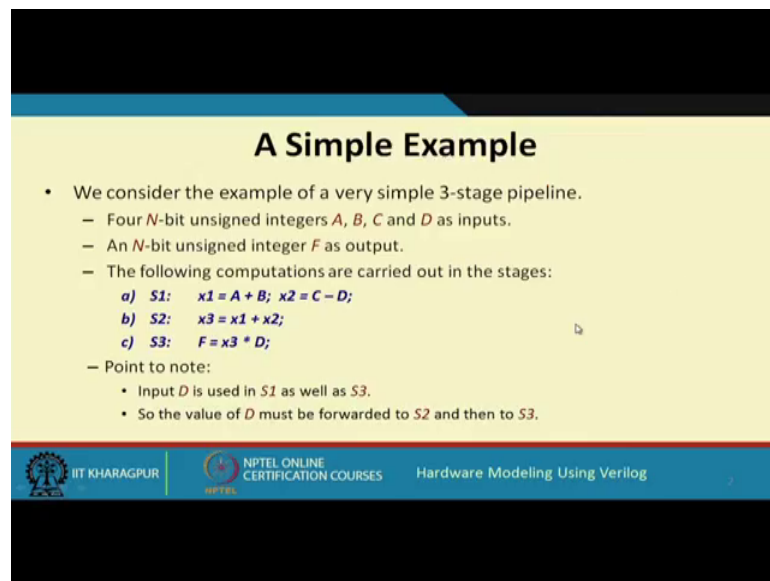


Hardware Modeling Using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 33
Pipeline Modeling (Part 1)



So, in the last lecture, we saw how a basic pipeline works and how it helps in getting a significant degree of speed up over an equivalent non pipelined implementation of a certain piece of hardware. Now, in this lecture, we shall give some illustrative example where you will be able to see given some computation that we want to implement in hardware, how we can implement it in the form of a pipeline, and how we can code the corresponding module in verilog. So, the title of the lecturer is pipeline modelling.

(Refer Slide Time: 01:01)



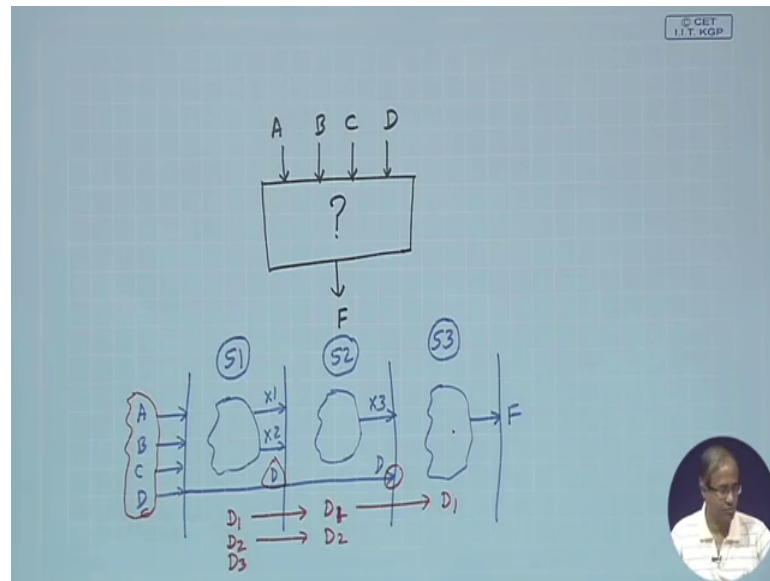
A Simple Example

- We consider the example of a very simple 3-stage pipeline.
 - Four N -bit unsigned integers A , B , C and D as inputs.
 - An N -bit unsigned integer F as output.
 - The following computations are carried out in the stages:
 - a) $S1$: $x1 = A + B$; $x2 = C - D$;
 - b) $S2$: $x3 = x1 + x2$;
 - c) $S3$: $F = x3 * D$;
 - Point to note:
 - Input D is used in $S1$ as well as $S3$.
 - So the value of D must be forwarded to $S2$ and then to $S3$.

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

So, here we shall be considering a very simple pipeline structure comprising of three stages. The pipeline computes some hypothetical computation, which is defined as follows. Let us suppose there are four numbers A , B , C , and D which are given as input to the computation and finally, we want to get F as output.

(Refer Slide Time: 01:34)



So, what we are saying is that we have a piece of computation. If you treat it as a black box there are four inputs A, B, C and D and there is one output the final output that is F. Let us see what is there inside this computation box. So, all these a B C and D are multi bit vectors let us in general call it N. So, we can define a parameter to fix some value of N later. Similarly, F is also an N-bit vector, which is as output. Now, the computation that is done is as follows.

First a and B are added the result is stored in another temporary variable x 1; C minus D, D subtracted from C we store the result in x 2; then this x 1 and x 2 are added to another variable x 3, finally, x 3 and D are multiplied to get the final result. Now, in order to have this in a pipeline, we observe the dependencies. See, there is an obvious dependency, the first line there are two statements, which are computing x 1 and x 2 which are used in the next statement. So, the next statement cannot be executed at the same time as you are executing the first two. So, this has to be in the next step.

Similarly, after x 3 is computed only then you can compute F. So, quite naturally we have divided this overall computation into three stages we call them S 1, S 2 and S 3 right now let us look at one thing see three stages S 1, S 2, S 3 we have divided computation into. So, let us represent that like this, this is our stage S 1, let us say this is our stage S 2, and this is our stage S 3. You see there are a few things you need to observe we shall be showing this using some diagrams later in the first stage A, B, C and

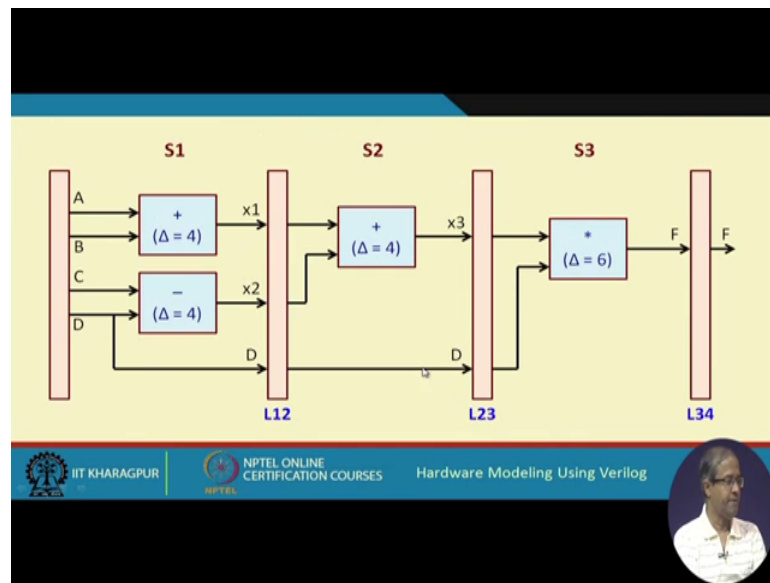
D are the inputs that are coming from outside. So, when you are starting competition in the first stage you must have the four inputs which are coming in A, B, C and D.

Then you see the first stage is computing the temporary values x_1 , x_2 , which will be forwarded all right to the next stage. So, next stage is computing x_3 . So, from here there will be some computation this x_1 and x_2 will be forwarded to the next stage. Look at the third stage; third stage is taking x_3 . So, what is x_3 ? x_3 is computed in stage 2; it takes x_1 and x_2 as input it computes another value x_3 . Now, in S 3 whenever you are doing the calculation, you are not only taking S 3, you see you are also taking the value of D. So, this value D which was here this also has to be forwarded up to this place. So, that S 3 can be computed and finally, and you can get the value F generated.

So, these intermediate buffers or the latches will be used to store this well. For example, D has to be forwarded it. So, you will also have to store D here, you will also have to store D here and then finally, F will be calculating the value. Because you just remember one thing, so when we are giving the first set of data let us call them D 1. So, D 1 will first be executing on stage S 1 then D 1 will be moving to S 2 when it is moving to S 2. So, S 1 will be starting with the next data D 2 sorry this is D 1. Similarly, D 1 will be moving finally to S 3, then D 2 will be moving to stage two, and the next data will be entering stage one.

Now, the point to notice that when the computation for D 1 is entering stage three, so the value of A, B, C, D which were provided for D 1 that should be available here that is why we have forwarded that D value up to this place. So, the same value will get forward because when D 1 is being this computation this is the D of D 1 in the meantime S 2 is calculating for D 2. So, whatever D is stored here, this is the D value for D 2; and whatever D is coming that is the D for D 3. So, this is how it goes on. So, here we have an example to illustrate diagrammatically. So, this is the computation that you want to do.

(Refer Slide Time: 07:01)



So, this is the pipeline diagram that we are just assuming you see in the first stage as I had said we are doing two calculations A plus B goes to as x 1, C minus D goes to x 2. You see in the first stage, the values that are coming are A, B, C and D. So, A plus B goes to x 1, C minus D goes to x 2. So, we are assuming that the delay of both addition and subtraction are four units. So, delta is the delay of this block and because D will be needed later as I had said, so the value of D is also forwarded here. In stage S 2 we are doing x 3 equal to x 1 plus x 2. So, there is another adder which takes x 1 and x 2, and it generates x 3. In the last stage, there is a multiplier, so we assume that the delay is six units it multiplies x 3 and D and generates F as the result.

Now, in this pipeline, what we are saying is that in order that you can execute this operation in a pipeline, because you see what is the essential advantage of a pipeline. When there are a number of data sets on which we want to perform the same kind of computation. Let us say for this example, let us assume, A, B, C, D are vectors, vectors in the sense that it is like an array. There are let say 1000 sets of data which are coming one by one A, B, C, D first value second value third value like that. So, as these values are coming like this computation in a pipeline becomes very efficient.

Now, in order to do it in a pipeline, you will have to insert as it said latches or registers in between the stages, because when the first calculation is over in stage one. And now you move to stage two these temporary results x 1, x 2 and also this D, they must be held

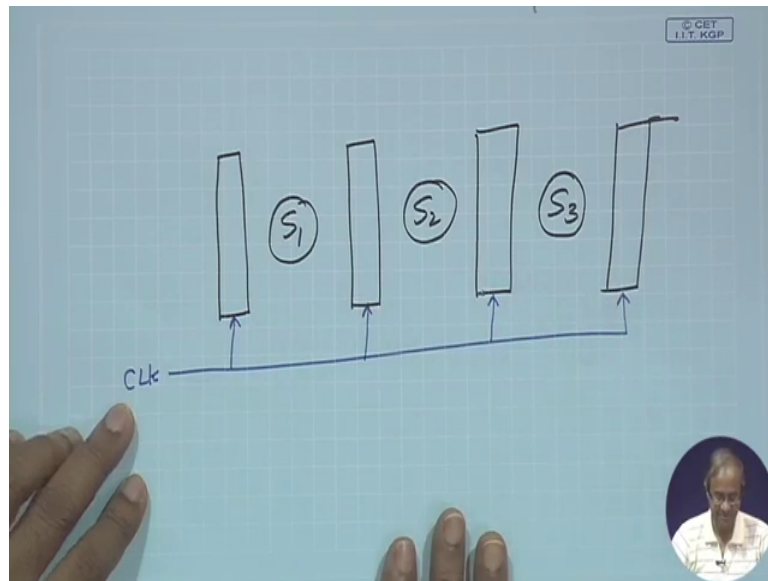
somewhere, so that stage two can start why they must be held somewhere because the next set of A, B, C, D values are already here. So, those values will make the x 1, x 2 and D values to change.

So, while S 2 is computing in order to ensure that this values do not change, we must have a storage element in between, similarly between every stages you need storage elements like this. So, these storage elements we refer to as this L 12, L 23 and L 34 and just for convenience this variables you see earlier we call them x 1 x 2 and D like. So, here we are renaming the variables by also mentioning which latch stage this is L 12 between 1 and 2. So, L 12 underscore x 1, L 12 underscore x 2, D; similarly L 23 underscore x 3 D. You see D is been forwarded. So, the value of D when it reaches here we call it L 12 underscore D and when that value reaches here we call it L 23 underscore D; and finally, when we store the value F here we call it L 34 F, because there will be a fourth stage emission after this.

Now, you see let us consider this L 12. So, in L 12, we need to store three values. So, it is not that x 1, x 2 and D are three registers that you need to have separately from the latch stage. Rather this latch is nothing but they will be allocated storage here itself for the variables x 1, x 2, D they will be allocated storage here itself, x 3 and D they will be allocated storage here, and F will be storage allocated here. Like it is something like this, these are the individual register elements let say or the latches, this latch stores L 1 to x 1 this stores L 1 to x 2 and so on

Now, these three latches taken together we call them as L 12. These two taken together we call it L 23, and this is L 34. So, this is our pipeline diagram, which we want to code in verilog. Now, when you code in verilog, we need to remember the name of this variables L 12 x 1, L 12 x 2, L 12 D because these are the variables we have to generate because all these latches are storage elements that need to be assigned values in synchronism with some clock.

(Refer Slide Time: 12:12)



Now, just one thing we are assuming here for the time being that there are some latches. I am showing the latches like this; there are three stages. There is a stage S₁ here, there is a stage S₂ here, there is a stage S₃ here. So, what we are assuming is that there is some kind of a clock which is connected to all the latch stages like this. Data moves through the pipeline stages in synchronism with the clock. Now, we shall see later that what kind of clocking mechanisms are better for this register stage we will see it later. For the time being, let us assume that these are common clock, which is feeding all the register stages like this.

(Refer Slide Time: 13:07)

```
module pipe_ex (F, A, B, C, D, clk);
  parameter N = 10;

  input [N-1:0] A, B, C, D;
  input clk;
  output [N-1:0] F;
  reg [N-1:0] L12_x1, L12_x2, L12_D, L23_x3, L23_D, L34_F;

  assign F = L34_F;

  always @(posedge clk)
  begin
    L12_x1 <= #4 A + B;
    L12_x2 <= #4 C - D;
    L12_D <= D;
    // ** STAGE 1 **
  end
endmodule
```

Pipeline Modeling

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


Let see. Now, we want to code this in verilog let us see how we do it. So, we define a module the name of the module is pipe underscore x and these are the parameters F is the output A, B, C, D are the four inputs and of course, a clock is there and parameter defining as 10, so that all the numbers are 10 bit values. So, A, B, C, D are declared as input, 0 up to n minus 1, clock is also an input, this F is an output of N-bit again. Now, this output F, I have not declared as reg because I am just assigning a value using the assigned statement, but all the temporary values like this L 12 x 1, L 12 x 2, L 12 D, L 23 x 3, L 23 D, L 34 F look at the previous diagram all these values one two three four five and six they are all declared of type register because you will be storing them in the latches they are all declared of type reg again of size N. And the last value this L 34 F that is nothing, but F right, so we are simply assigning it to F.

Now, regarding coding as I have said all computation will be done in a synchronous way always at posedge clock. So, what will happen when a positive edge of clock comes, for stage S 1 whenever a clock comes A and B has arrived, after delay of 4, this L 12 x 1 will get the value of the sum, we will have to store it here. Again after value of 4, C minus D will be stored here L 12 x 2; and D will be stored directly here we are assuming there is no delay. So, in terms of the declaration, we are using non blocking assignment we are writing L 12 x 1 assign with a delay of 4 A plus B; x 2 C minus D, and this gets D. And these three statements constitute our stage one of the pipeline.


(Refer Slide Time: 15:51)

```
L23_x3 <= #4 L12_x1 + L12_x2;
L23_D <= L12_D; // ** STAGE 2 **

L34_F <= #6 L23_x3 * L23_D; // ** STAGE 3 **
end
endmodule
```




IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog



Then comes the stage two. So, in stage two, what you are doing just go back once in stage two, we are taking the values of $L_{12 \times 1}$ and $L_{12 \times 2}$ and we are storing the sum into $L_{23 \times 3}$, and D we are moving straight. So, there are two things. This $L_{12 \times 1}$ and $L_{12 \times 2}$ are added after a delay of 4, it was assigned to $L_{23 \times 3}$ and the value of D is simply copied from L_{12} to L_{23} . And in stage L_3 , you are doing similarly a multiplication with the delay of 6; and the result and multiplication you are doing on $L_{23 \times 3}$ and this $L_{23} D$, and the result you are storing in $L_{23} L_{34} F$.

So, this is as simple as that you have the pipeline diagram and we have simply coded the pipeline specification in the form of the different stages inside an always blocked which are triggered with the clock. So, translate in a pipeline schematic diagram into a verilog code is fairly simple as we have seen. Once we have the diagram you can directly start writing the verilog code from there fine.

(Refer Slide Time: 16:51)

```

module pipe_ex (F, A, B, C, D, clk);
  parameter N = 10;

  input [N-1:0] A, B, C, D;
  input clk;
  output [N-1:0] F;
  reg [N-1:0] L12_x1, L12_x2, L12_D, L23_x3, L23_D, L34_F;

  assign F = L34_F;

  always @(posedge clk) // ** STAGE 1 **
  begin
    L12_x1 <= #4 A + B;
    L12_x2 <= #4 C - D;
    L12_D <= D;
  end

```

Alternate way of coding:

- One stage per "always" block.
- Code is more readable.

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

Now, there is another alternate style that we are showing here. So, you can do the same coding in a slightly different way. You see here what we did we used a single always block right. And inside the always block the first three statements were stage one of course, we wrote a comment here just to indicate that these three are stage one, these two are stage two, and this is stage three. But suppose if I do not write the comments then it would be a little confusing to the reader the person who sees the code to identify that which one is my stage one, which one is my stage two, and which one is stage three So,

the alternate style what we do we split or break the always block into three smaller always blocks, one for stage one, one for stage two, and one for stage three like this. The first part is identical now this is the always block for stage one you see there is a separate always block begin and end, the three statements for stage one are coming here.

Then there is another always block for stage two, and another always block for stage three, because there is a single statement, you do not need begin end. Now, this style is the equivalent of the previous one, but this code is more readable because you are showing the stages separately and the persons we seeing it will be very easily able to appreciate and understand the stages. And there is another reason that we will see later with another example we shall discuss that there is another need why we need to break it up into multiple stages.

(Refer Slide Time: 18:39)

```
module pipel_test;
    parameter N = 10;
    wire [N-1:0] F;
    reg [N-1:0] A, B, C, D;
    reg clk;

    pipe_ex MYPIPE (F, A, B, C, D, clk);

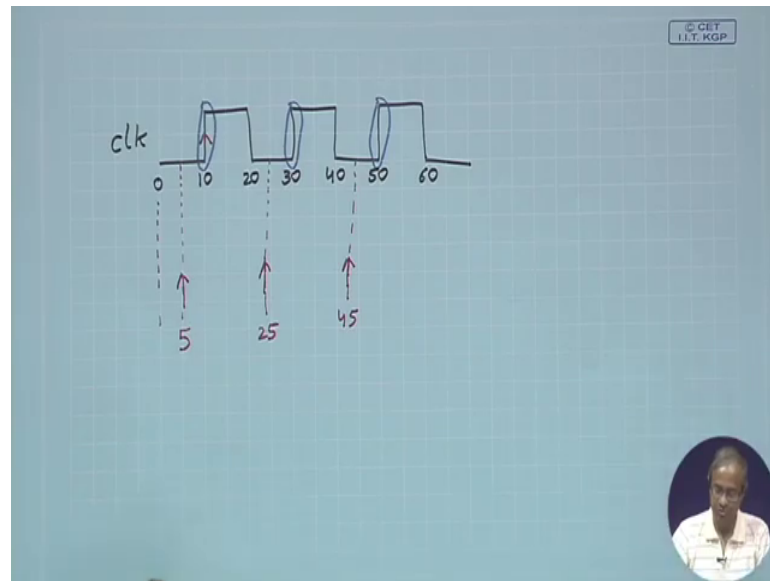
    initial clk = 0;
    always #10 clk = ~clk;
endmodule
```

Pipeline Test Bench

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

Now, let us try to write a test bench for this now in this test bench, what do we need we need to supply four values A, B, C and D and after sometime we will be getting the result F available with us. So, here we are instantiating this module these are the parameters. Now, this A, B, C, D are the inputs to the pipes. So, this will be declared as reg, because here will be assigning values, clock is also reg and F is the output which declared as wire. So, in the similar way these are all 10-bit values. And we are applying a clock you see the way we are applying clock is 0; and with the delay of 10, we are complimenting it.

(Refer Slide Time: 19:31)



So, the clock will be like this. If this is 0, this will be 10, this will be 20 this will be 30. So, after ten-ten intervals, the clock is tumbling changing state right, this will be our clock right.

(Refer Slide Time: 20:01)

```
initial
begin
#5 A = 10; B = 12; C = 6; D = 3; // F = 75 (4Bh)
#20 A = 10; B = 10; C = 5; D = 3; // F = 66 (42h)
#20 A = 20; B = 11; C = 1; D = 4; // F = 112 (70h)
#20 A = 15; B = 10; C = 8; D = 2; // F = 62 (3Eh)
#20 A = 8; B = 15; C = 5; D = 0; // F = 0 (00h)
#20 A = 10; B = 20; C = 5; D = 3; // F = 66 (42h)
#20 A = 10; B = 10; C = 30; D = 1; // F = 49 (31h)
#20 A = 30; B = 1; C = 2; D = 4; // F = 116 (74h)
end

initial
begin
$dumpfile ("pipel.vcd");
$dumpvars (0, pipel_test);
$monitor ("Time: %d, F = %d", $time, F);
#300 $finish;
end
endmodule
```

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

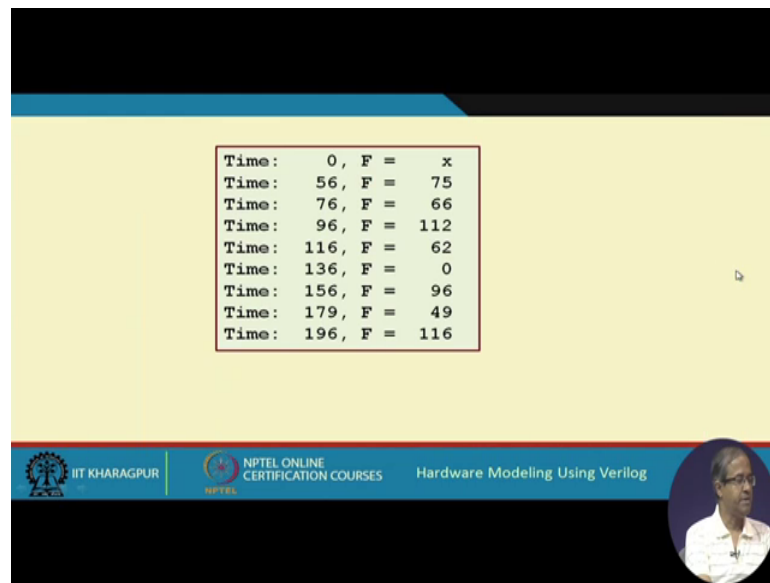
Next what we do, we apply the inputs. Now, let us see what we do the first set of inputs there is another initial block, so it starts with time 0 again; we give a delay of five and apply the first set of inputs. What does this mean? So, you see here is a timing diagram clock starts with 0. So, time five is somewhere right, this is 5. So, it is here we are

applying the first input. Why, because when the next clock edge comes this input is already ready. So, we can start processing this input from the next blockage and after this we are giving twenty-twenty gaps, because this was 5, next it will be 25, it is here. Next it will be 45, it is here. So, my first input is coming here second input will come here third input will come here and so on.

The reason is very clear that if I apply the first input here then the first clock edge will be able to capture this input. If I apply the second input here this second clock edge will be able to capture it similarly the third clock edge will be able to capture it. So, in this way we are applying the inputs. So, there are a set of eight sample inputs we have applied various values of A, B, C, D. So, initial delay of five and then twenty, twenty, twenty gaps. So, here also by the side I have shown the expected value of the result for this thing you see A plus B is 22, C minus D is 3. So, 22 and 3, what we are doing you see that computation was more you are doing the addition then subtraction then you are adding this $\times 1$ and $\times 2$, then you are multiplying it with D.

So, 22 and 3, you add them up it becomes 25, 25 you multiply with D, it become 75. Like that these are the values in decimal and side by side I am also showing the hexadecimal values because in the timing diagram that will be generated by gtk wave the values will be shown in hexadecimal. Now, in this initial block we are specifying the dump files and dump variables and will be monitoring the time and the values of F. So, time and F will be printed. So, we will be doing simulation up to certain maximum time let say 300, but here we do not need 300 because we have only how many three, four, five, six, seven, 140 by 150 should be over but anyway we are doing up to 300.

(Refer Slide Time: 22:52)

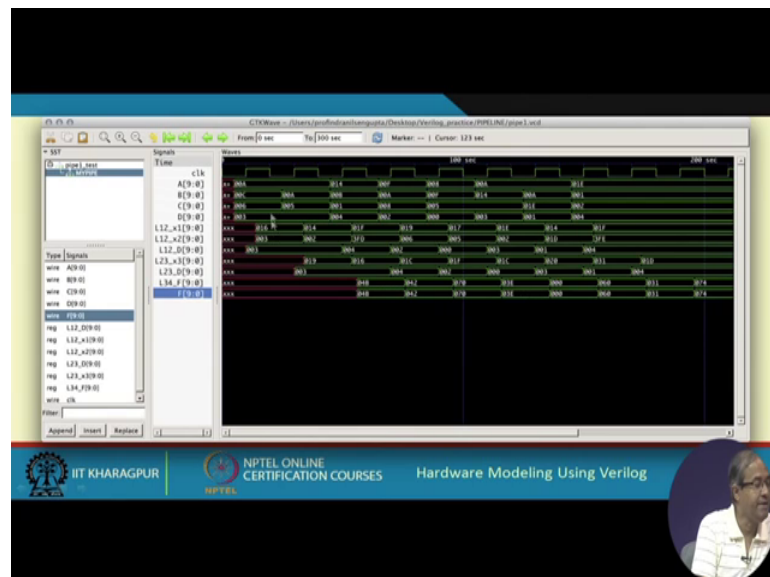


Time: 0, F = x
Time: 56, F = 75
Time: 76, F = 66
Time: 96, F = 112
Time: 116, F = 62
Time: 136, F = 0
Time: 156, F = 96
Time: 179, F = 49
Time: 196, F = 116

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Now, if you run the simulation the simulation result comes like this. The time is whenever F changes time is whenever F changes and you see the F value the 75, 66, 102 the same values are coming. So, whatever the expected values the same values are being generated. This is the final output.

(Refer Slide Time: 23:20)



The screenshot shows a Verilog simulation waveform. The signals are A, B, C, and F. The values of F are 00AC, 6, and 3. The values of x1 and x2 are 10, 12, 6, and 3.

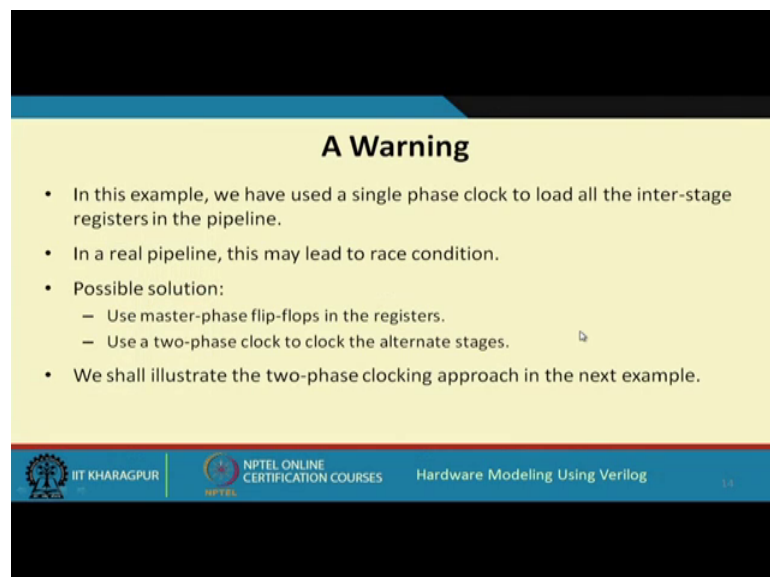
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

And if you see the output of the gtk wave the timing diagram here, you can see what is happening. You see the first row shows the clock this A, B, C and are being applied you see 00AC 6 and 3; that means, 10, 12, 6 and 3 and the values of x 1 and x 2 are getting

calculated. 10 plus 12 is 22 it is 16 in hexadecimal, it is 22. 6 minus 3 is 4, this is three these two are again added 22 and 3 is 25, it is 19 in hexadecimal, this is L 23 x 3, and this is multiplied by D. Finally, you get here 48; that means 16 for the 64, and B 75. So, this is your final result, and this is assigned to final F. So, you see four B four two seven zero three these are the actual values are coming four B four two seven zero three I have shown in hexadecimal, the same values are coming in the simulation output. And it is coming at every clock as is expected in a pipeline.

So, the advantage of the pipeline is that you are not waiting for one computation to finish before applying the next input. So, you are continuously applying input one after the other every clock and when the pipeline is full output will also be generated ones per clock cycle that is why the throughput will be improved to a great extent. This is illustrating by this example this diagram. This timing diagram as it shown, you can see from here that the output is also generated ones per first output, second output, third, fourth, fifth, sixth, but initially there is a delay for the pipe to fill up, but once the pipe gets filled up the outputs are generated one per cycle that is the advantage of pipeline.

(Refer Slide Time: 25:24)



A Warning

- In this example, we have used a single phase clock to load all the inter-stage registers in the pipeline.
- In a real pipeline, this may lead to race condition.
- Possible solution:
 - Use master-phase flip-flops in the registers.
 - Use a two-phase clock to clock the alternate stages.
- We shall illustrate the two-phase clocking approach in the next example.

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

Now, there is an important point to note that in the example that I have shown here we have used a single clock and that clock is used to activate all the latches. So, we have used a single statement always at the rate posedge of clock. So, it is actually a flip flop a

register we are implementing at the positive edge of the clock we are just activating the latch stage. That same thing is happening for all the stages right.

Now, the point to notice that this is actually a warning that in a real pipeline, this may lead to a race condition see what we are saying is that in order that correct operation is achieved in a pipeline all the register stages or latch stages, if you are supplying a common clock must be master-slave type, there must be a master stage, a slave stage and so that inputs and outputs are perfectly isolated. But if you do not use master-slave flip flops or latches a single stage then when the input changes, the output also changes which might affect the second stage that is the danger here. Of course, in this example, you could not see that, but in the next example we shall see in our next lecture, there we will do this isolation in a very properly.

So, what are the possible solution as I said first one is to use a master-phase flip flop, and the second one is to use a two-phase clock to clock the alternate stages. So, we shall be illustrating this in the next example which will be taking up in our next lecture. So, in this lecture, we have taken a simple example of a computation, and showed how we can map it to a pipeline structure, and how we can code it in verilog.

Thank you.