Hardware Modeling using Verilog

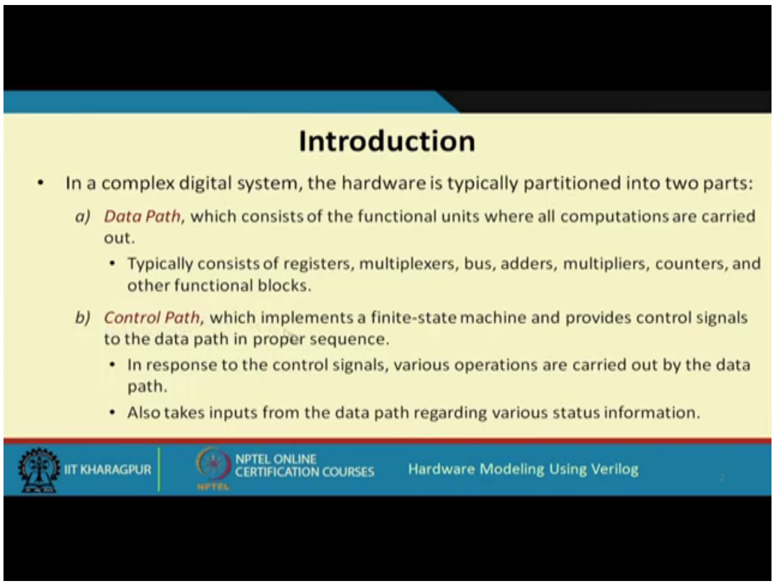
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 25
Datapath And Controller Design (Part 1)

So, in the earlier lectures we have seen how we can design both combinational and sequential circuits. Specifically we talked about finite state machines both Moore and mealy type machines. So, we looked at the various ways in which we can model those sequential machines in verilog, but you think of a complex system. So, whenever we are trying to build a complex system the complex system will consist of a mix of everything. There will be some combinational parts; there will also be some sequential parts. So, in this lecture we shall try to talk about such systems which are called data path and controller design.

So, in such a system we shall see that there are 2 parts called data path and controller part, and how they are related and how they can be designed ok.

(Refer Slide Time: 01:29)



Introduction

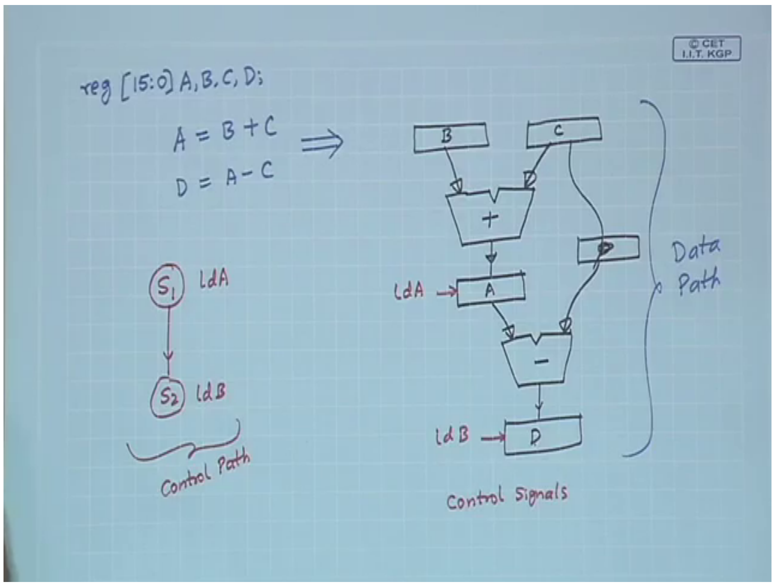
- In a complex digital system, the hardware is typically partitioned into two parts:
  - a) *Data Path*, which consists of the functional units where all computations are carried out.
    - Typically consists of registers, multiplexers, bus, adders, multipliers, counters, and other functional blocks.
  - b) *Control Path*, which implements a finite-state machine and provides control signals to the data path in proper sequence.
    - In response to the control signals, various operations are carried out by the data path.
    - Also takes inputs from the data path regarding various status information.

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

So, let us see the basic idea. So, as I have said that for complex digital systems we normally do not design the whole hardware in one piece. They are typically partitioned in to 2 parts; the first part is called the data path. What is the data path? It consists of the functional units, where all the computations are carried out. So, what are there in the data path. They will consist of typically some registers to store some data, multiplexers, bus, the adders, subtractors, multipliers, counters and similar functional blocks. So, in a data path there are a lot of hardware and things which are there, but we are not specifying or telling exactly what to do with those hardware. Let us say I can tell that there are 3 registers there is one adder one subtractor and one counter, but I am not specifying within the data path that exactly how I am going to use them, so for that there will be a second path. There will be a control path, control path is nothing but a finite state machine.

This will be generating or providing some control signals for the data path in a particular sequence. And by doing that the data path will be activating or it will get activated accordingly, and the operations will be carried out as per the intended requirement. The control path can also take some input from the data path to get various status information. Let us take a very simple example.

(Refer Slide Time: 03:23)

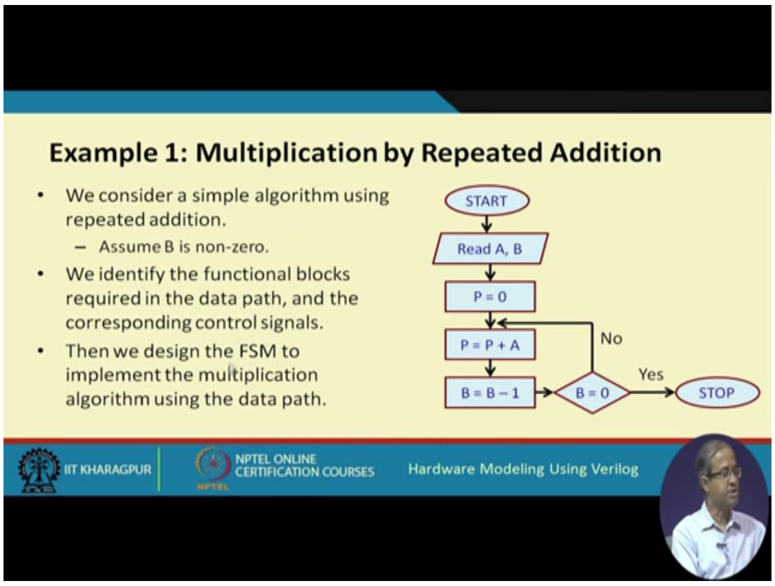


Suppose we have 2 statements A equal to B plus C, followed by D equal to A minus C. Now A B C D, let us say these are all registers. Let us say we have defined registers. Let us say these are all 16 bit registers A B C and D. Now when we are actually trying to build a hardware for this. First thing is that intuitively speaking you look at this A equal to B plus C D equal to A minus C, what should be my hardware look like. My hardware intuitively you can say that well there will be a register to hold the value B, there will be another register to hold the value of C, there will be an adder. This adder will take the inputs from B and C, and it will generate the result in to another register A, this is A, and then will be having a subtractor. It will take a as the first input.

There will be another register D, it will come from here, no sorry this will be is from C from C. And the output will be generated in D right. So, whatever I have shown here this will be my data path. This is what is meant by data path. Some hardware blocks and the way they are interconnected. Now you see in some of these blocks there will be some controls like in this a there can be a load control, let us say load A, in D there can be a load B control right. So, let us assume that this is a fixed circuit. So, these load A and load B these are called control signals.

Now, this is a very simple example. So, in terms of the FSM, I will say that well I have 2 states s 1 followed by s 2. In state s 1 I am simply activating load A, why? You see if I activate load A then B and C whatever is getting added that will get stored in A. In s 2 I am activating load B, which means whatever is in a and whatever is in C they will be subtracted and it will be loaded in D. So, this is my data path and this will be my control path. Control path of course, will be some implementation of this they were you implement. So, this is what I mean by data path and control path.

(Refer Slide Time: 06:38)



Example 1: Multiplication by Repeated Addition

- We consider a simple algorithm using repeated addition.
  - Assume B is non-zero.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the multiplication algorithm using the data path.

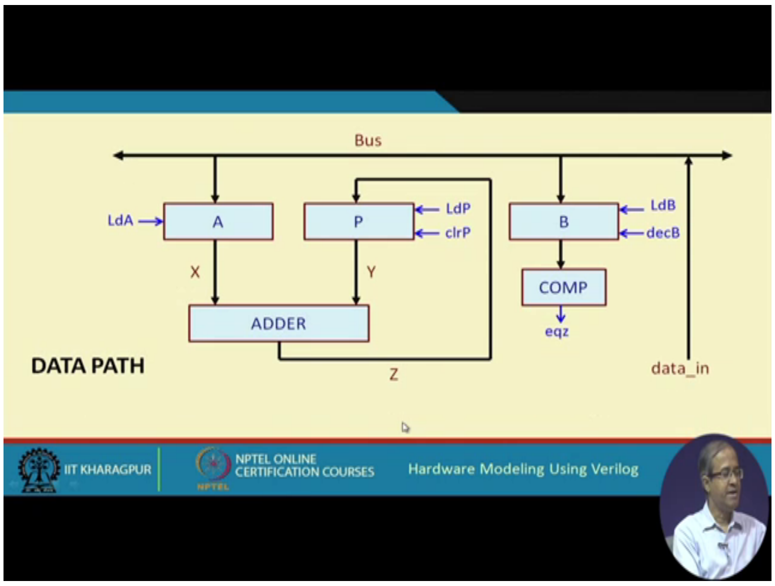
START
Read A, B
P = 0
P = P + A
B = B − 1
B = 0
No
Yes
STOP
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Let us see through some examples. The first example we take in this regard is a very simple example, where we are trying to multiply 2 integers by repeated addition. So, the algorithm is like this we are let us say reading the 2 numbers A and B the product where initializes in to 0. And you are repeatedly we are adding A to the product in this loop, and decrementing B every time till the time B reaches 0. So, if B is 5 this loop will be repeated 5 times and 5 times will be adding a to 0. So, it will be 5 in to A. So, the product is done. So, here of course, we have made an assumption in this flow chart that we assume B is nonzero, because if B starts with a 0 this algorithm will not work, because we are first decrementing and then checking ok.

So, for this example let us try to illustrate how we can design the data path and control path. So, in the first step we shall be identifying the data path, and also now in the data path what are the control signals that need to be activated. Then we shall be designing the finite state machine which will correspond to the control path right. So, for this example let us try to see; what are the requirements for the data path. Like one thing I can immediately see that I need 3 registers A B and P.

I need an adder, the B register should also be a down counter because I have to decrement it by 1. There has to be some kind of a comparator which checks for 0. Right these are the few things I need. So, from this requirement we can directly come up with a data path.

(Refer Slide Time: 08:41)



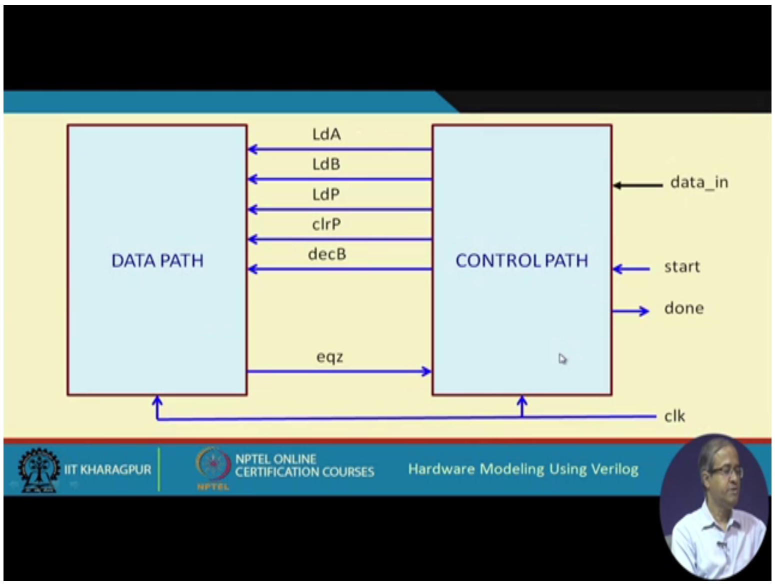
So, this is a data path which we have arrived at from this flow chart specification, let us see this one by one. First we have to read A B from outside. So, the values of A and B I am assuming these are supplied from outside through an input called data in. This is my A register, this is my B register. So, from this data in the data can go to either to input of A or to the input of B. There are 2 control signals load A and load B which can be used to load this data either in A or to B. So, let us say first we apply the value for a activate load A gets loaded then apply the second data, then activate load B it gets loaded in B right. Now let us see the other requirements. So, we are initializing P to 0, right.

So, P is another register which has a special control called clear P. So, if this control signal is activated P will be initialized to 0, right. And another thing, in every step of the iteration I am we are doing P equal to P plus A P and A are added result is going back to P. So, you see P and A we are adding them output of the adder is going back to P and going back to P means when the value will be loaded. It will load it only when this load P signal is activated. So, there will be another control signal for loading this P, and regarding this B register, so B has to be decremented.

So, B is actually a counter which has another control input decrement B. So, if it is activated it will be decremented by 1, and we have to check B for 0. So, there will be a comparator circuit, which will be checking whether B is 0 or not. So, how you can check something for 0? So, it will be nothing but a nor gate, multi input nor gate where all the inputs or the nor gate you connect to the different bits of B. So, the output one means the number is 0.

So, this is a signal which is generated by the data path, and this will be required as input by the control path. So, you see this is my data path. So, from outside it is taking the data in and the control signals are load A load B load P clear P and decrement B.

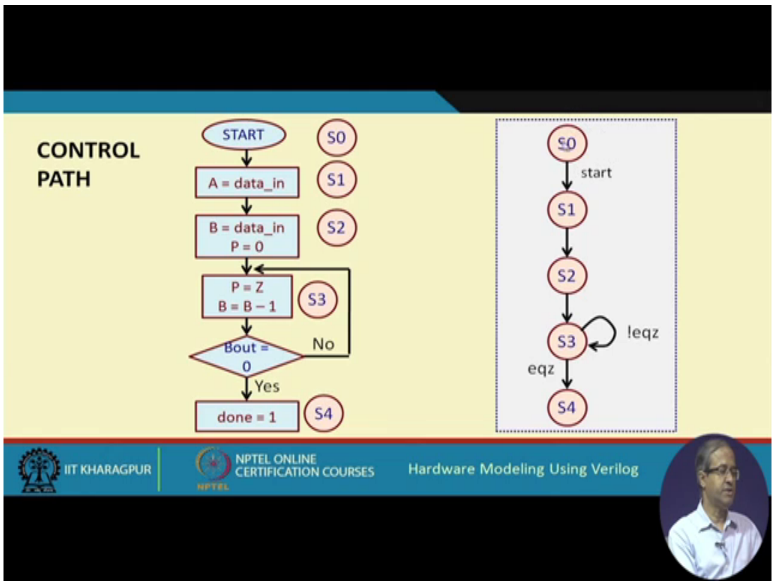
(Refer Slide Time: 11:43)



And it is generating a status signal equal to Z. So, the overall the picture will be like this I have my data path, which will be having this 5 control signals as I mentioned. And it is generating one status signal. Now I will have to generate or design a control path. The control path will be generating this control signals, and it will be taking the status as input. So, clock will be connected to both.

Well, data in I am showing your actual data is also coming to data path because in data path also data is there. And there are 2 additional signals in the control path, start means now you start the multiplication. And after the multiplications finished done will be activated so that from outside you will be knowing that the multiplication is done and the result is available. Now result where is available it will be available in P right product fine.

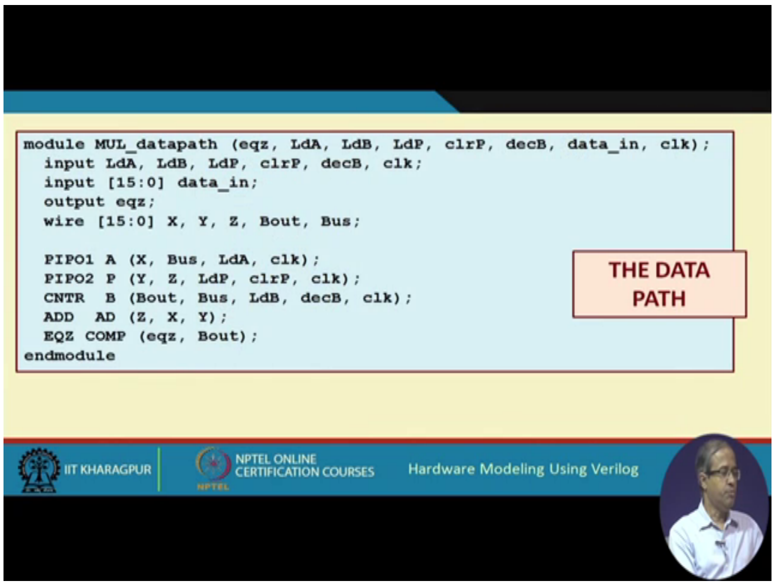
(Refer Slide Time: 12:52)



Now, let us see how we can design the control path. That same flow chart we are showing it in a slightly different way, step wise we are trying to break. See we have to read the values of A and B right, in the first step we are saying data in we are loading in a second step data in you are loading in B and parallely we can clear P because these are not related you can do it together. And what is Z just go back to the diagram? Z is actually the output of the adder. So, the output of the adder is going to P and your decrementing B. So, as long as B out which is B out B out is the output of this B, B out as long as this is not 0 you are repeating this. And when it is 0 you are done you activate the signal done equal to 1.

So, you see here you are defining the different steps and also identify some states side by side. So, the initial state I am calling it s 0. So, whenever start is activated I go to state 1, then s 2 then I go to s 3, and I remain in s 3 till B out is equal to 0. So, when B out equal to 0 only then I go to state 4 s 4. So, this state transition diagram for this example will look like this is my FSM. I start with s 0 if start is 1 I go to s 1. So, whenever start is activated then only I start I go to s 1. Then from s 1 when the next clock comes I go to s 2, from s 2 whenever the next clock comes I go to s 3, but in s 3 I check this equal to Z signal. If it is not equal to 0 I remain in s 3 at every clock I just do a you see P equal to Z means what A plus B is Z. We are just adding storing in P next time again adding a storing in P, again adding storing in P, this goes on. And as soon as equal to Z is active I go to s 4 and stop right.

(Refer Slide Time: 15:27)



```
module MUL_datapath (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  input LdA, LdB, LdP, clrP, decB, clk;
  input [15:0] data_in;
  output eqz;
  wire [15:0] X, Y, Z, Bout, Bus;

  PIPO1 A (X, Bus, LdA, clk);
  PIPO2 P (Y, Z, LdP, clrP, clk);
  CNTR  B (Bout, Bus, LdB, decB, clk);
  ADD   AD (Z, X, Y);
  EQZ COMP (eqz, Bout);
endmodule
```

THE DATA PATH

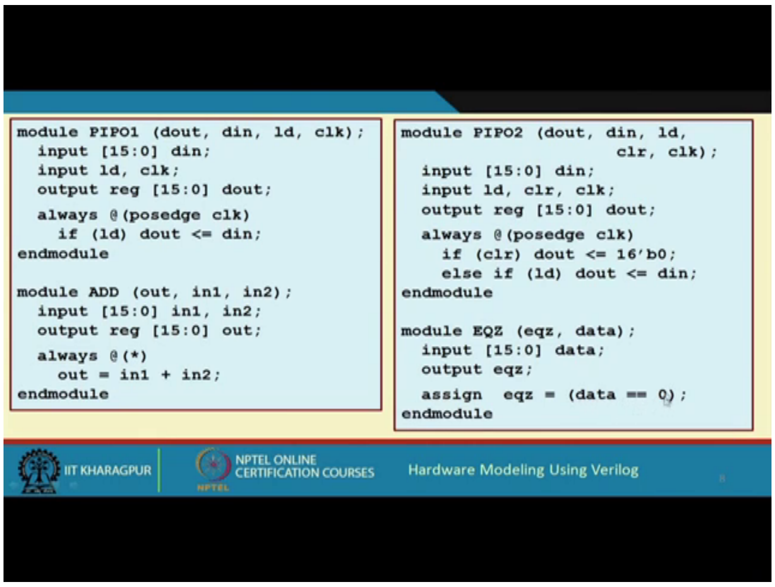
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, let us look at the verilog coding of the designs. Just let us have a look once more here. This is my data path you remember this there are there is one register like this with only load. There is one register with load and clear, one register with load And decrement, this is actually a counter. So, here at the top level you are instantiating the modules. So, in the multiplier data path you see you can correlate with that previous example, previous diagram; that these are the signals or the ports that this multiplier data path requires. There is only one output equal to Z others are all inputs. These are the control signals, and this is the clock. And I am assuming data in is a 16 bit data. And all these X Y Z B out bus, what are these? Let us go back to the diagram once more. This is X output of a I am calling as X output of P m calling, it Y this is Z and this I am calling as bus fine.

So, we are instantiating the registers one by one, this is a this is P this is B. So, there are 2 kinds of parallel in parallel (Refer Time: 16:57) one is a register without a clear I am calling it pipo 1. A is the instantiated name and these are the parameters output is X input is bus; the control cells load control and clock. Pipo 2 is a parallel in parallel out register with load and clear control. These units for the P register. So, Y is the output, Z is the input to this register and this is load this is clear and took, B is a counter at the output of the counter I call it B out input is bus you can load it you can decrement it and clock.

And of course, you have an adder where the inputs are X and Y output is Z. And you have a comparator we call it equal to Z, this is the name. This is the output it takes B out as the input and generates the output. So, whatever I have mentioned here, this is exactly the data path diagram that I have shown. From the data path diagram whatever components are there I directly instantiate those blocks in my top level data path module. Now I will have to specify all these modules. Now here for simplicity I am considering all of these at the behavior level, but in reality some of them you can also code in structure or gate level mode. Pipo 1, pipo 1 has a output, input, load And clock.

(Refer Slide Time: 18:31)



```
module PIPO1 (dout, din, ld, clk);
  input [15:0] din;
  input ld, clk;
  output reg [15:0] dout;
  always @(posedge clk)
    if (ld) dout <= din;
endmodule

module ADD (out, in1, in2);
  input [15:0] in1, in2;
  output reg [15:0] out;
  always @(*)
    out = in1 + in2;
endmodule
```

```
module PIPO2 (dout, din, ld,
                        clr, clk);
  input [15:0] din;
  input ld, clr, clk;
  output reg [15:0] dout;
  always @(posedge clk)
    if (clr) dout <= 16'b0;
    else if (ld) dout <= din;
endmodule

module EQZ (eqz, data);
  input [15:0] data;
  output eqz;
  assign  eqz = (data == 0);
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

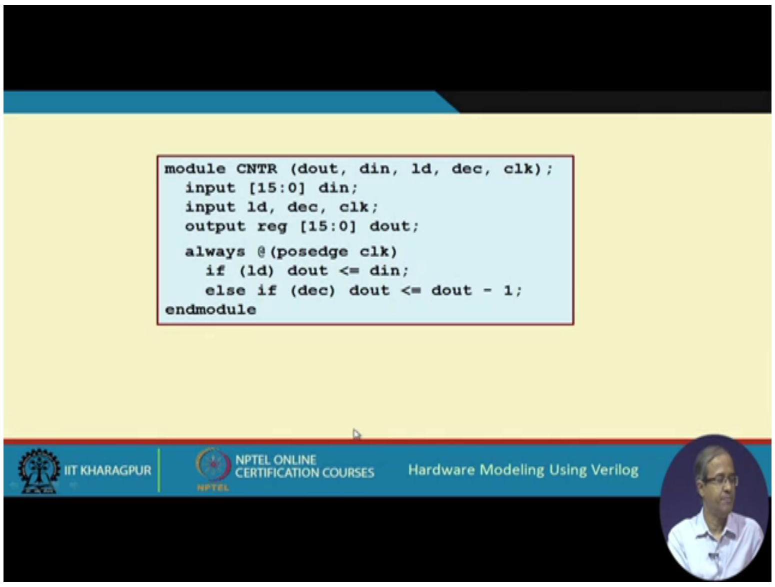
So, input is 16 bit output is also 16 bit. So, it very simple always at posedge clock; if load is active the input gets loaded right. Let us look at this was pipo 2 first pipo 2 is an extension of this where you also have a clear input in addition to load. So, here there will be an additional input clear. So, again always at posedge clock if clear the counter is cleared then D out equal to 16 0s.

16 bit 0, else if load same D out equal to D n. You say here is because we are using h triggered means event we are using non blocking assignment. Now the adder, this is just a behavioral specification out in one in 2 these are all 16 bit quantities, it is always at star this out equal to n 1 plus n 2. Then the comparator this equals at the input is the 16 bit data output is a this value great. So, you just write like this assign eqz equal to data equal

0; so you can either do this or you can also use the reduction operator nor basically you are using a nor operation, right.

So, you can write this assign eqz equal to reduction nor on data n data. So, this is also a way to do it. Data equal to equal to 0 is a conditional it will generate 2 or 4 0 or 1, 0 means naught 0, 1 means equal to 0. So, accordingly this output eqz will be affected. So, this completes our specification for the data path.

(Refer Slide Time: 20:39)

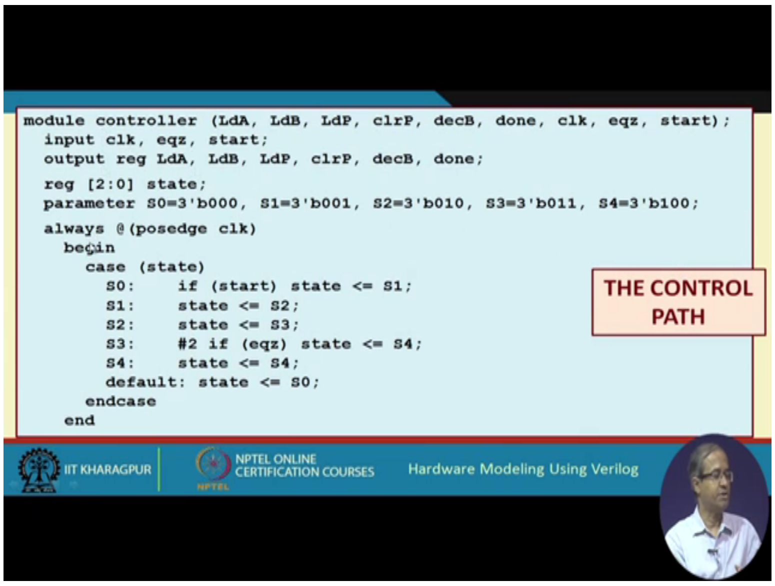


```
module CNTR (dout, din, ld, dec, clk);
  input [15:0] din;
  input ld, dec, clk;
  output reg [15:0] dout;
  always @(posedge clk)
    if (ld) dout <= din;
    else if (dec) dout <= dout - 1;
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

And of course, one thing is left the counter the counter is output input load decrement and clock same way declaration. So, whenever there is a posedge clock if load is active you are loading it else if decrement is active your decrementing the counter by 1. So, you have defined all the blocks of the data path and you have instantiated them in the top level module; so one pipo 1 1 pipo 2 a counter an adder and a comparator. Now you see this was my finest state machine you remember this.

(Refer Slide Time: 21:23)



```
module controller (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);
  input clk, eqz, start;
  output reg LdA, LdB, LdP, clrP, decB, done;
  reg [2:0] state;
  parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
  always @(posedge clk)
    begin
      case (state)
        S0:     if (start) state <= S1;
        S1:     state <= S2;
        S2:     state <= S3;
        S3:     #2 if (eqz) state <= S4;
        S4:     state <= S4;
        default: state <= S0;
      endcase
    end
```

THE CONTROL PATH
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

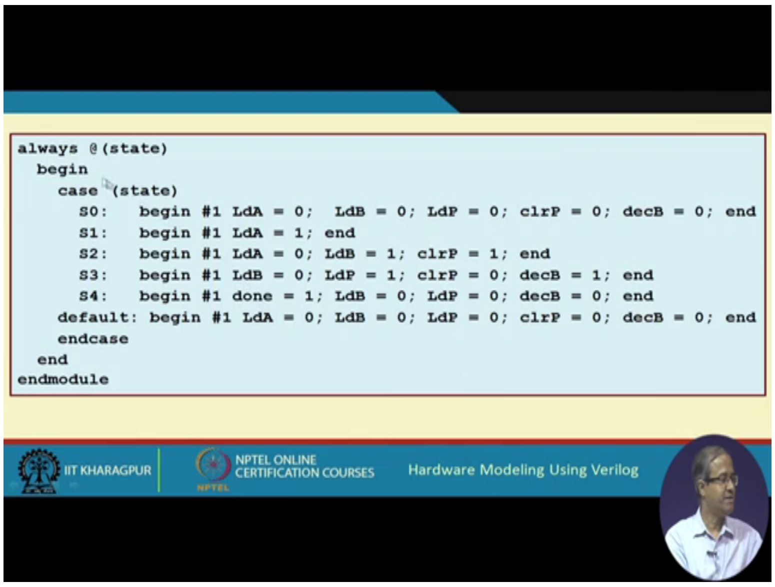
So now will be coding the FSM the controller; so you have seen already earlier how to code an FSM we had just taken examples of both Moore and mealy type FSM. So, here the FSM is very simple let us see. So, here the controller will be generating these signals load A load B load P and clear P decrement B and of course, done. So, these are the output signals of the controller. And the inputs to the controller are of course, clock this equal to 0 signal which is coming from the data path, and the start which is coming from outside.

Now, in this example there were 5 states. So, you need 3 bits to encode this state. So, I am using a parameter statement to define the states and give them names s 0 s 1 s 2 s 3 and s 4. Well, the use of parameter makes it much easier for this code to understand otherwise you have to write 0 0 0 here 0 0 1 here and so on. Now this always block is exactly the state transition diagram which you saw earlier, it says there is a case statement. So, if my present state is s 0. So, you remain in s 0 until start is activated. So, if start is true then state becomes s 1 now if it is in s 1 if there is a clock you will always go to s 2.

Now if you are in s 2 an clock you will always go to s 3. Now if you will in s 3 then we check if equal to 0 is activated or not. So, if it is true; that means, you are done then you go to state s 4. Or else you remain in s 3 no change. And once you are in s 4 you do not change you remain in s 4 and by default if it starts with any undefined it will be initialized to s 0.

Now this delay we have given just for the purpose of simulus just for the purpose of synchronization and to get the correct output from simulation, because this little delay is required whenever you are going to be final state right. Now next part of this controller will be to generate this signals load A load B load P clear B decrement.

(Refer Slide Time: 23:59)



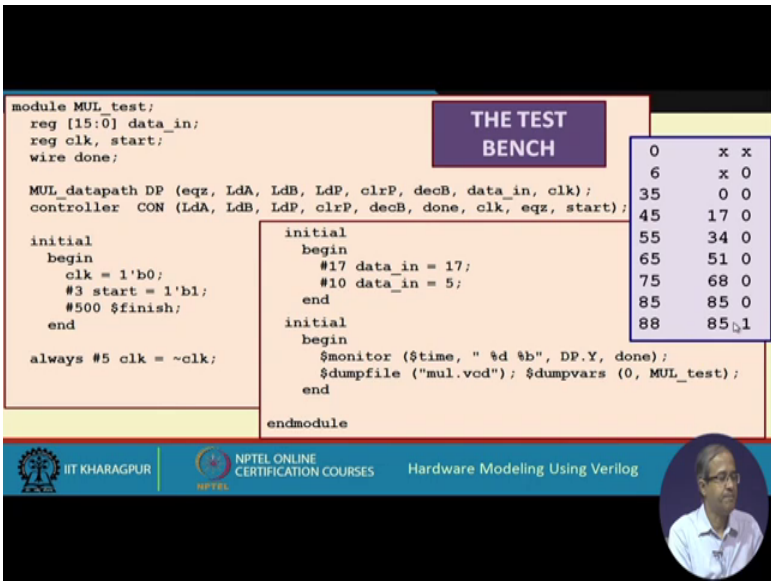
```
always @(state)
  begin
    case (state)
      S0:    begin #1 LdA = 0;  LdB = 0; LdP = 0; clrP = 0; decB = 0; end
      S1:    begin #1 LdA = 1; end
      S2:    begin #1 LdA = 0; LdB = 1; clrP = 1; end
      S3:    begin #1 LdB = 0; LdP = 1; clrP = 0; decB = 1; end
      S4:    begin #1 done = 1; LdB = 0; LdP = 0; decB = 0; end
    default: begin #1 LdA = 0; LdB = 0; LdP = 0; clrP = 0; decB = 0; end
    endcase
  end
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

This is in a block where we using blocking assignments. So, again in the s 0 state we are not activating any signals. So, all the signals are 0s. In state s 1 you are loading the value of a right A equal to data A. So, load A is activated in s 2 you are loading B. So, this load A is deactivated and load B is made 1. And in parallel we are initializing P 2 0. So, clear P is also initialized to 1, in s 3 we are actually doing the computation. So, you need not have to load B anymore.

So, load P will have to be done in this loop, clear P is not required anymore. So, decrement B. So, P equal to X plus Z and B equal to B minus 1 these 2 things you are doing here. And s 3 and here you will be remaining in s 3 until eqz is true. So, this steps will be repeated clock after clock. And finally, when you are in state s 4 you activate done equal to 1, and again deactivate all the signals. So, if it is default again you deactivate all the signals to 0 right. So, this is how you can specify the FSM this is exactly like the FSM design which you saw earlier, given a state transition diagram how to map it to a verilog behavioral finite state machine description.

(Refer Slide Time: 25:44)



THE TEST BENCH

```
module MUL_test;
  reg [15:0] data_in;
  reg clk, start;
  wire done;

  MUL_datapath DP (eqz, LdA, LdB, LdP, clrP, decB, data_in, clk);
  controller  CON (LdA, LdB, LdP, clrP, decB, done, clk, eqz, start);

  initial
    begin
      clk = 1'b0;
      #3 start = 1'b1;
      #500 $finish;
    end

  always #5 clk = ~clk;

  initial
    begin
      #17 data_in = 17;
      #10 data_in = 5;
    end

  initial
    begin
      $monitor ($time, " %d %b", DP.Y, done);
      $dumpfile ("mul.vcd"); $dumpvars (0, MUL_test);
    end

endmodule
```

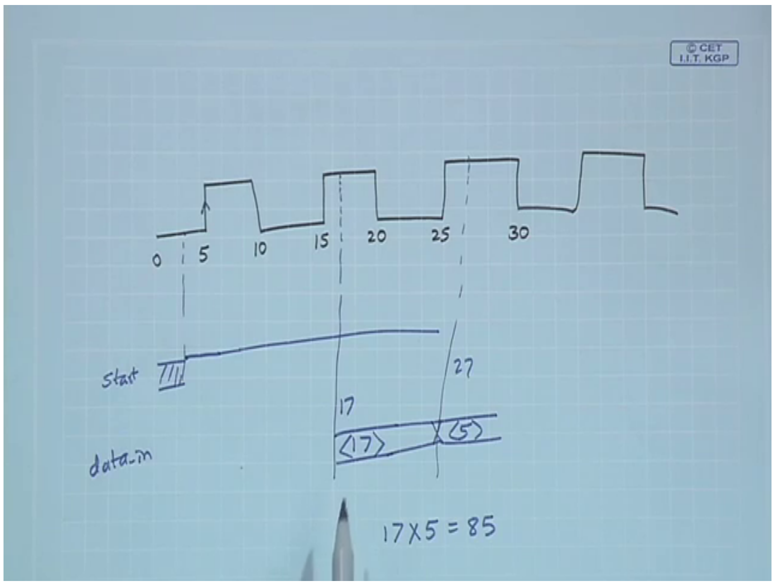
```
 0       x x
 6       x 0
35       0 0
45      17 0
55      34 0
65      51 0
75      68 0
85      85 0
88      85 1
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, here we have done the same thing. Now the test bench we have written a very simple test bench to verify our design. So, you see what we have done, we called it mul test. Data in this is coming from outside. So, this will be generated by the test bench somehow we have initialized data in clock and start and done is the output signal. So, we have instantiated the data path and the controller with all the signals are specified right.

So, in the initial block we are just initializing the clock to 0, and in this always block when generating continuous clock with a delay of 5 clock equal to not clock. And at add at time 3 we are setting start to 1; that means, we say that we are starting the multiplication here, and from the next clock you should start. And we are repeating till time 500 where we finish. Now in this initial block you see we are applying the data at time 17 and 5. So, you just see one thing, this is how we are generating the clock.

(Refer Slide Time: 27:04)



So, we start with time 0, this is 5, 10, 15, 20, 25 30 and so on.

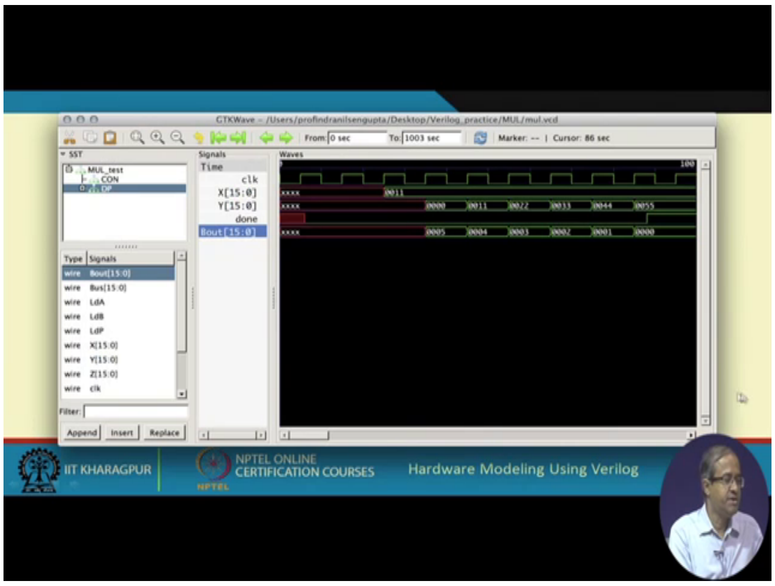
Now, if you think of the start signal start you are activating at time 3 start. So, initially start was undefined. So, it was tri state. So, start is activated at time 3. So, multiplication will start from here. So, the next edge that will be obtained is here. So, start will remain like this. So, what I doing data in the first data we are providing at time 17. At time 17 we are giving the first data, which is 17 the data is 17 also. And after a gap of time ten; that means, 27, at time 27 we are giving a the next data which is 5. So, we are trying to multiply 20, 17 and 5 17 in to 5 should be 85.

So, we are actually giving it here. So, the load Also will be activated here, after that let us see what happens. And here in the initial block we are monitoring the value of Y, and what is Y? Y if we just go back once. Y is the output of the product; that means the final value. You see this output of P is not directly available as a signal in this model right, but you can access some signal from the other model from D P, you say Y is not available in the parameter, but you can write D P dot Y and access that Y and print the value this is possible ok.

So, we are printing that value and also the status of done. And you are dumping them in to a file. So, if you actually simulate this is the kind of output you are getting. So, you see the result finally, that you are getting is actually 85 these are the numbers in because here you are printing in decimal right percentage D value of Y you are printing decimal.

So, it initially it was 17 and 5. So, 17 then 34 then 50 or. So, 17 it was added 0 plus 17 plus 17 plus 17 plus 17 plus 17 at the end done is 1. So, it is done ok.

(Refer Slide Time: 30:12)



So, if you look at the timing diagram the same thing is shown here. You see just exactly what I have said that this first data 17 in hexadecimal 0 0 1 1, this is available in time 17 10 and the second data was available here. So, the loading I am not shown here. So, I have shown the just the time steps the output of the B the counter.

So, it was initially it was 5 then it goes to 4 3 2 1 and then 0. And the just output of the Y it gets just added it was initially 0, then 1 1 means 17 1 1 in hexadecimal means 17; then 2 2 means 34, 3 3 means 51, 4 4 means 68. 5 5 means 85, and here develops done becomes 1; that means, it is done. So, in this example what we have seen is that given any reasonably complex design we can separate out the data part containing the hardware blocks. And the control part which will be activating the hardware blocks in a proper sequence. This is the standard way of designing complex systems.

And, we shall be looking at some more examples also over the next couple of lectures.

Thank you.