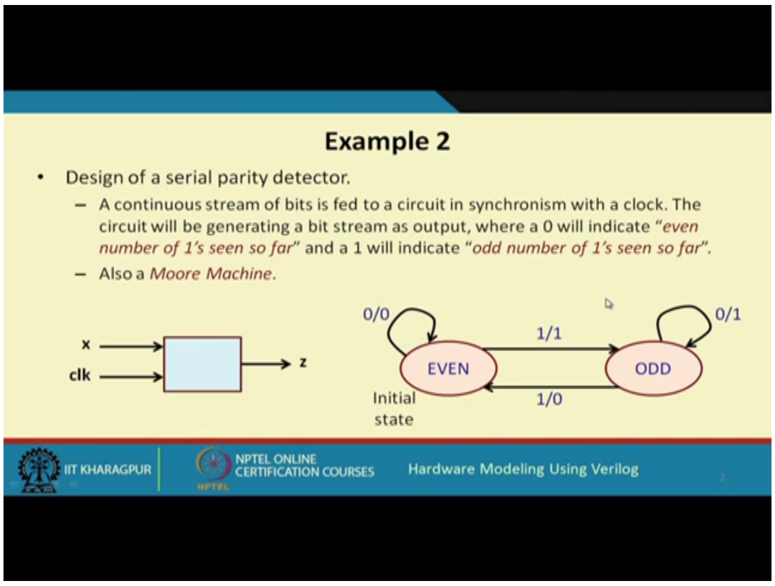
Hardware Modeling using Verilog

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

Lecture - 24
Modeling Finite State Machines (Contd.)

So, we continue with a discussion on modeling finite state machines. So, if you recall in our last lecture we talked about finite state machines the different types and how to model finite state machines in verilog we considered one example there. So, continue with our discussion and you look at some more examples of modeling finite state machines. So, our lecture is modeling finite state machines the second part ok.

(Refer Slide Time: 00:50)



Example 2

- Design of a serial parity detector.
  - A continuous stream of bits is fed to a circuit in synchronism with a clock. The circuit will be generating a bit stream as output, where a 0 will indicate *"even number of 1's seen so far"* and a 1 will indicate *"odd number of 1's seen so far"*.
  - Also a *Moore Machine*.

x
clk
z
0/0
EVEN
1/1
ODD
0/1
1/0
Initial state
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, the example we take now this is also a Moore machine, this is the design of a serial parity detector. Let us first try to understand what is parity. Suppose I have an n bit word parity actually indicates whether the number of once in that word is odd or even. If it is odd we say it is odd parity if it is even we say it is even parity. Now the parity can be represented by a single bit let us say one can indicate odd 0 can indicate even or the vice versa, ok.

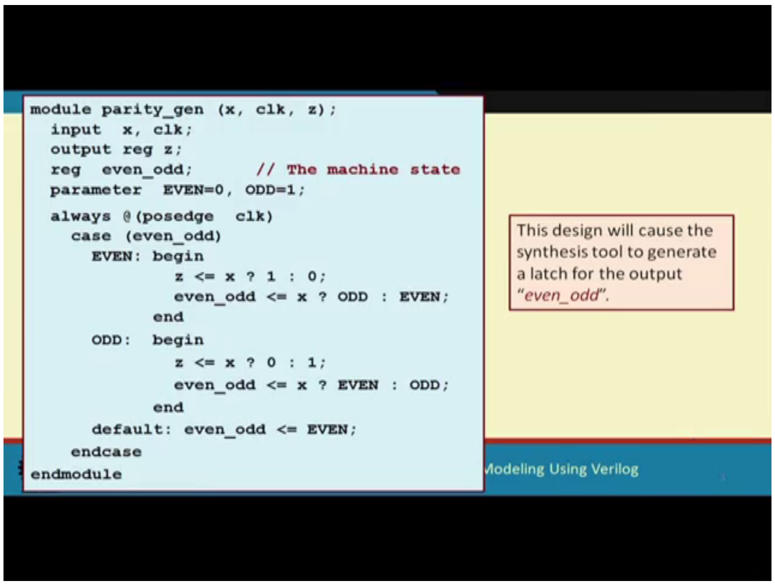
So, here we are trying to design a circuit where the input number that we are talking about this is not available in parallel, but coming serially bit by bit. So, as it is coming bit by bit we are continuously computing the parity, and we are continuously outputting a bit which indicates the parity of this stream that we have seen so far. So, what will be our logic here? We need to store only one piece of information, whether the parity of the bits that we have seen so far is odd or even.

Now if my next bit coming is 0 then I will not change it will remain, but if my next bit is one then it will change. If it is odd it will become even if it was even it will become odd, this is our logic. So let us see, how we specify our design. So, here as it said a continuous stream of bits is fed to a circuit let us say x is that bit stream in synchronism with the clock. This circuit will be generating a bit stream in the output which is here z, and in the output stream a 0 will indicate even parity; that means, so far we have seen even number of ones in x, and output z will indicate that so far we have seen odd number of ones in x, ok.

Now, the output z see here we need to maintain 2 states. Clearly one indicating that whether the number of bits we have seen so far is even and the other number of bits so far is odd. Now you see the output z can directly be generated from this state. So, it really does not depend on the input. So, this is also this is an example of a Moore machine. So, this state transition diagram we can show like this. This even can be your initial state initially nothing has come let us say it is initial. So, if a 0 comes the output is 0 and you remain in the even state. Now if a one comes you go to the odd state and in the odd state the output is out to be 1.

Now while in the odd state if a 0 comes. So, it will remain odd. So, the output will remain 1, but if a one comes this odd will become even. So, the output will again go back to 0.

(Refer Slide Time: 04:24)



```
module parity_gen (x, clk, z);
  input  x, clk;
  output reg z;
  reg  even_odd;       // The machine state
  parameter  EVEN=0, ODD=1;

  always @(posedge  clk)
    case (even_odd)
      EVEN: begin
                z <= x ? 1 : 0;
                even_odd <= x ? ODD : EVEN;
            end
      ODD:  begin
                z <= x ? 0 : 1;
                even_odd <= x ? EVEN : ODD;
            end
      default: even_odd <= EVEN;
    endcase
endmodule
```

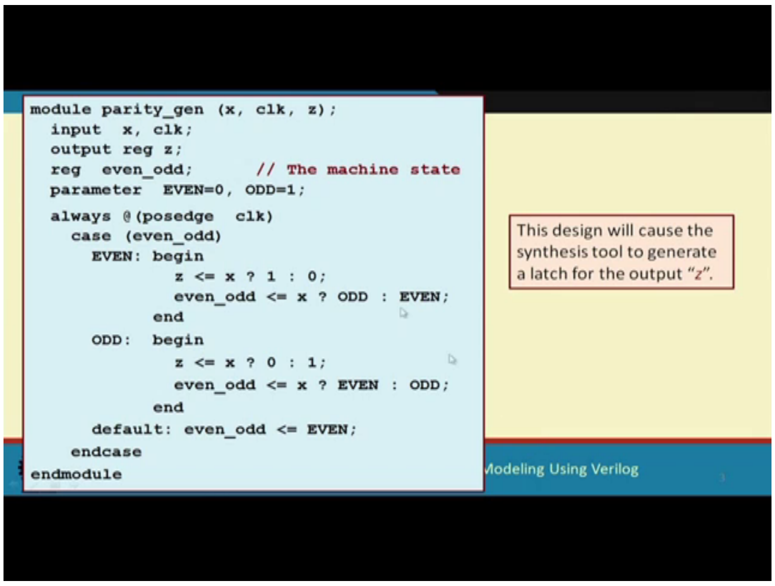
This design will cause the synthesis tool to generate a latch for the output *"even_odd"*.

So, much state transition diagram is fairly simple with 2 states. So, here this is the first version of our design, where it is the same kind of a design, but there is an input x. You see in the early example which we took there were no inputs separately. There was only 3 states red green and yellow they were cyclic in wherever the clock is coming, but now there is also an input x. Not only the clock I will have to decide my next course of action depending on what is my next bit x if it is 0, I will do something, if it is one I will do something else. So, so in the first version again I have used a single always statement you see my x and clock are the inputs and z is an output because it is coming on the left hand side as reg.

And this machine requires a single state variable because there are 2 states odd and even I need one flip flop one bit. So, I define a single state variable even odd that can be either 0 or 1 and I am defining by parameter 0 means even, e v e n and one means o d, odd. This is my procedural block. So, whenever the positive edge of the clock comes, I check this state even odd. If it is even I do this if it is odd I do this. So, what I do I update both my output and also my state depending on the value of x.

So, I use a conditional assignment statement here. What does this mean? If x is 1, then assign one if x is 0 assign 0. And here it means if x is one make this state odd if x is 0 make this state even. So, this is exactly as far this state transition diagram. Similarly if my state is odd then I do like this, if x is one I assign 0 to the output if x is 0 I remain in the odd state assign 1; and here if it is 0 even if it is 1 odd. And by default because say initially if you do not initialize the state variable it can the outputs will be all indeterminate. So, here we have included a default statement. So, initially this even odd will start with even if nothing matches. So, here again because I am using a clock triggered always block, and you are using non-blocking assignment, there will be one flip flop generated for z and one flip flop for this state, even odd this even odd is the state.

(Refer Slide Time: 07:27)

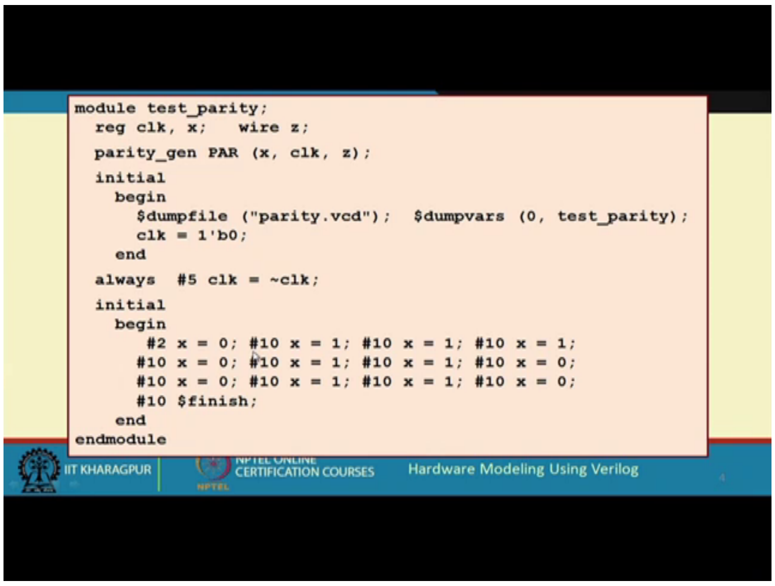


```
module parity_gen (x, clk, z);
  input  x, clk;
  output reg z;
  reg  even_odd;       // The machine state
  parameter  EVEN=0, ODD=1;
  always @(posedge  clk)
    case (even_odd)
      EVEN: begin
                z <= x ? 1 : 0;
                even_odd <= x ? ODD : EVEN;
              end
      ODD:  begin
                z <= x ? 0 : 1;
                even_odd <= x ? EVEN : ODD;
              end
      default: even_odd <= EVEN;
    endcase
endmodule
```

This design will cause the synthesis tool to generate a latch for the output "z".

So, this design will cause the synthesis tool to generate a latch for the output z, but again you try to understand the design, just by knowing the state I can tell what the output is. I really do not need a latch to store the output, if it is even my output should be 0 if it is odd my output should be 1 simple fine.

(Refer Slide Time: 07:56)

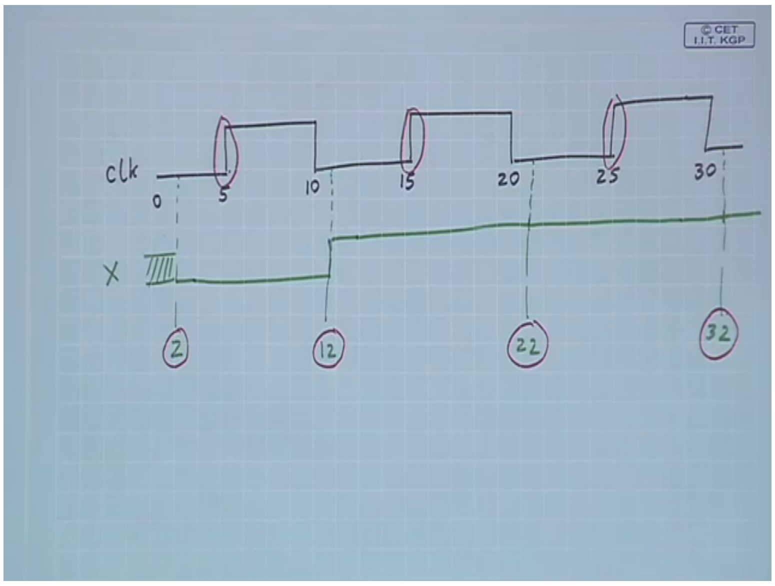


```
module test_parity;
  reg clk, x;   wire z;
  parity_gen PAR (x, clk, z);
  initial
    begin
      $dumpfile ("parity.vcd");  $dumpvars (0, test_parity);
      clk = 1'b0;
    end
  always  #5 clk = ~clk;
  initial
    begin
       #2 x = 0; #10 x = 1; #10 x = 1; #10 x = 1;
      #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
      #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
      #10 $finish;
    end
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, this will be the corresponding test bench for this I am showing one example. So, we have instantiated this parity generator x clock and z, these are the inputs you see x clock z same set of inputs. And in this initial block I am doing 2 things I am specifying the dump file, I am specifying the variables to dump all variables, and initializing the clock to 0. Because I have not given a delay this will be done at time t equal to 0, and again this is the clock generation with a delay of 5 I am complementing the clock.

(Refer Slide Time: 08:43)



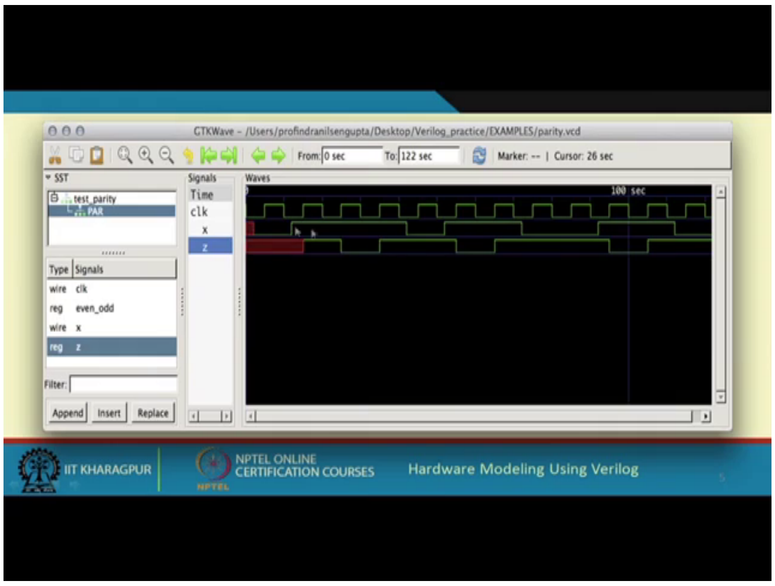
Now, here I am applying the bits. So, you see again now what has happened the clock will be coming like this. So, we have initialized clock at 0 at time 5 we are complementing. This will be time 10 this will be 15 20. So, after a gap of 5 we are complimenting the clock right. 5 again at 30 it will go back to 0 and so on. Now you see what I am doing to x. You say we are giving a delay of 2, such that when the clock h comes by that time this input should be stable. So, how we are applying x? We are applying x at a time equal to 2.

So, we are applying 0. So, initially x can be anything I do not know it will be something n value, but at time 2 I am making it 0. Now again I am applying the consecutive bits with delays of 10, say 0 1 1 1 and so on, I applied many bits. But initial bit I am applying at 2 then successive bits and applying after delays of 10, 10, 10 like that. So, the first bit 0 will be applied at 2 the next bit one will be applied at a delay of 2; that means, at 12; so here if this time is 12.

So, at 12 it will be one next bit is 1 the next bit is again 1. So, at time 22 if this time is 22 here I am applying again a 1. So, this one remains one then again at time 32, I am because here also I am applying 1. So, this will remain 1. So, like this I am applying the consecutive inputs at times 2 12 22 32 and so on. So, what I am trying to achieve? I am trying to achieve is that whenever the active edge of the clock comes the rising edge well before that the input should be stable, because if I change the input right when the clock edge is coming there will be a risk condition.

So, the output may not change properly. That is why we have given a small delay this delay only 2 or 3 or 4 no problem it should be before 5. So, like this we have done it. Before the clock edge comes the input should be changed and be stable fine.

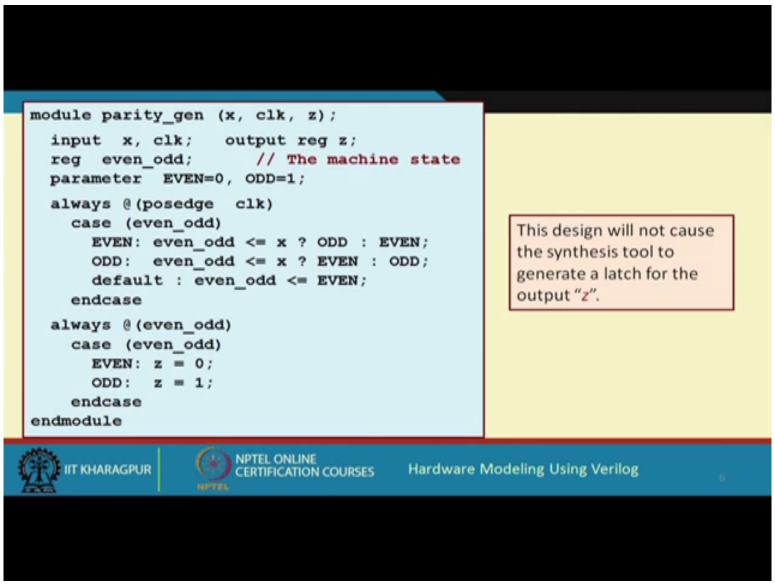
(Refer Slide Time: 11:18)


GTKWave - /Users/profindranilsengupta/Desktop/Verilog_practice/EXAMPLES/parity.vcd
From: 0 sec
To: 122 sec
Marker: -- | Cursor: 26 sec
SST
test_parity
PAR
Type Signals
wire clk
reg even_odd
wire x
reg z
Filter:
Append
Insert
Replace
Signals
Time
clk
x
z
Waves
100 sec
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, here you see the timing diagram of the simulation output. This is the clock signal which is coming, and this is my x you see. Initially it was undefined at time 2 I am making it 0. You see what I have said here you look at here. So, initially it was undefined at time 2 it was made 0. So, for up to time 12 it will remain 0 then it will go to 1. So, exactly the same thing happening you see it too at time 12 it is going to 1, at time 22 it remains 1, at time 32 it remains 1, but at time forty 2 it goes to 0 because next bit is 0 you see next bit I have applied 0 1 1 1 next bit is 0.

So, like this the bits are coming. And the outputs the outputs that will be generated it will be coming only after the clock the first clock is coming here. The first clock is coming here. So, the output is getting initialized here. And the output will be generated accordingly, then this one means odd then even odd odd even odd odd odd even like that it goes on changing right. So, this is just an example.

(Refer Slide Time: 12:43)



```
module parity_gen (x, clk, z);
  input  x, clk;    output reg z;
  reg  even_odd;       // The machine state
  parameter  EVEN=0, ODD=1;
  always @(posedge  clk)
    case (even_odd)
      EVEN: even_odd <= x ? ODD  : EVEN;
      ODD:  even_odd <= x ? EVEN : ODD;
      default : even_odd <= EVEN;
    endcase
  always @(even_odd)
    case (even_odd)
      EVEN: z = 0;
      ODD:  z = 1;
    endcase
endmodule
```

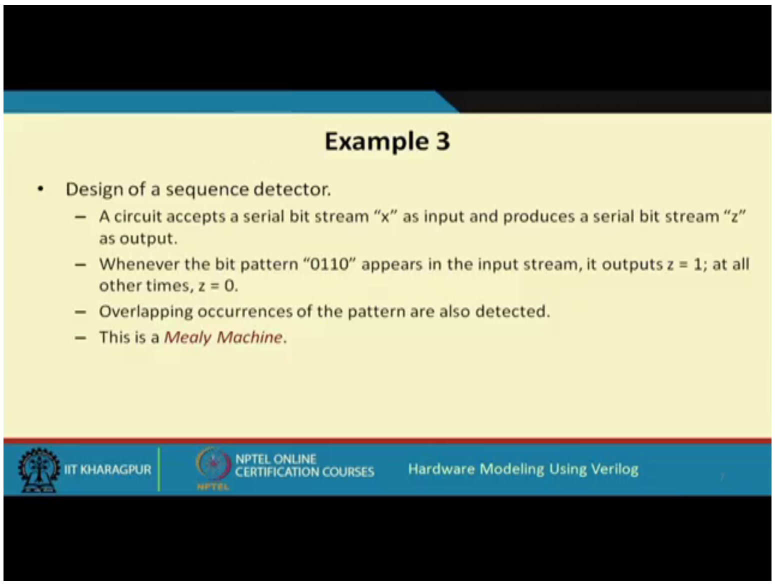
This design will not cause the synthesis tool to generate a latch for the output "z".

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So now we can improve this design to avoid the flip flop on the output. So, how you do just like the example we considered in the last lecture, we split the always block in to 2. So, in the first always block we are only updating the state with respect to the clock edge. So, whenever there is a positive edge of the clock, we check whether this state is even or odd if it is even depending on the input x if x is one we change the state to odd if x is 0 will leave it as even, but if this state is odd. Depending on x if x is 1, you make it even if x is 0 you make it odd. And default is even and in the other always block you trigger it whenever the state changes you do a case. So, if even is if the state is even the output is set to 0 if it is odd the output will be set to one just that ok.

So, here z can be directly generated from this state where. In fact, you do not need any circuit at all. This state directly will give z you can just connect a wire connecting even odd to z, z and even odd will be the same value ok.

(Refer Slide Time: 14:16)



Example 3

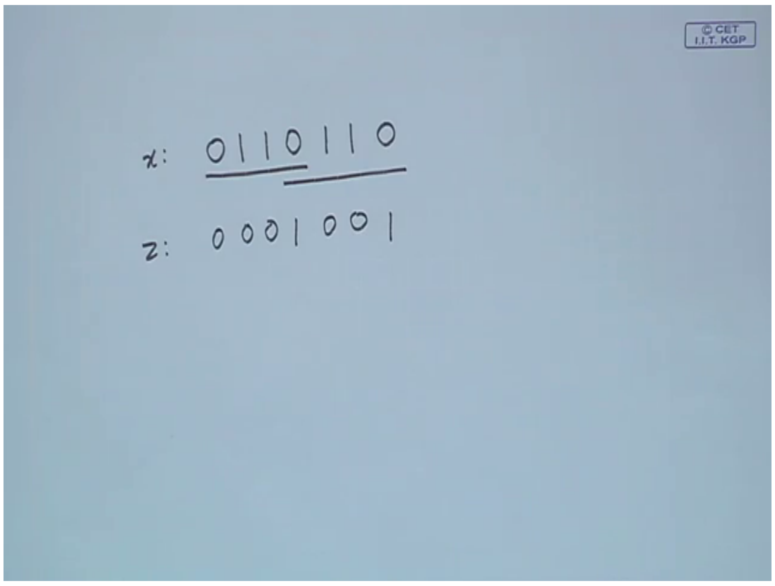
- Design of a sequence detector.
  - A circuit accepts a serial bit stream "x" as input and produces a serial bit stream "z" as output.
  - Whenever the bit pattern "0110" appears in the input stream, it outputs z = 1; at all other times, z = 0.
  - Overlapping occurrences of the pattern are also detected.
  - This is a *Mealy Machine*.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

So, no latch is required here fine. Now let us take the example of a mealy machine. Now in a mealy machine you recall the output value will depend not only on the state of the machine, but also on the present input right. So, here let us see. So, the example that we are taking is that of a sequence detector. So, what is the sequence detector? Sequence detector is the circuit that again there is a serial bit stream x coming. Let us say as input and again it will be generating a serial bit stream as output let us call it z.

So, our design specification says that whenever we encounter the bit stream 0 1 1 0 in the input stream, this output z will immediately go to 1, and at all other times it will remain as 0. And overlapping occurrences will also be detected for.

(Refer Slide Time: 15:22)

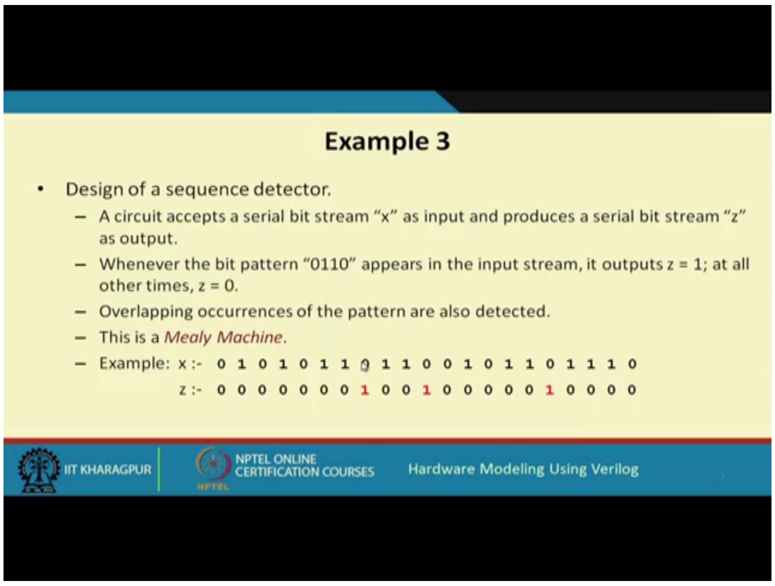


x: 0110110

z: 0001001

For example If I have 0 1 1 0 then again 1 1 0, then this will be considered as 0 1 1 0 sequence. And this will be considered as another overlapping 0 1 1 0 sequence. So, if this is your x your output z will be here it will become 1, and here also again it will become 1, at all other times it will remain 0 right. This is an example of a mealy machine because the output also depends on the input. So, this is an example bigger example you see this is my x.

(Refer Slide Time: 16:08)



Example 3

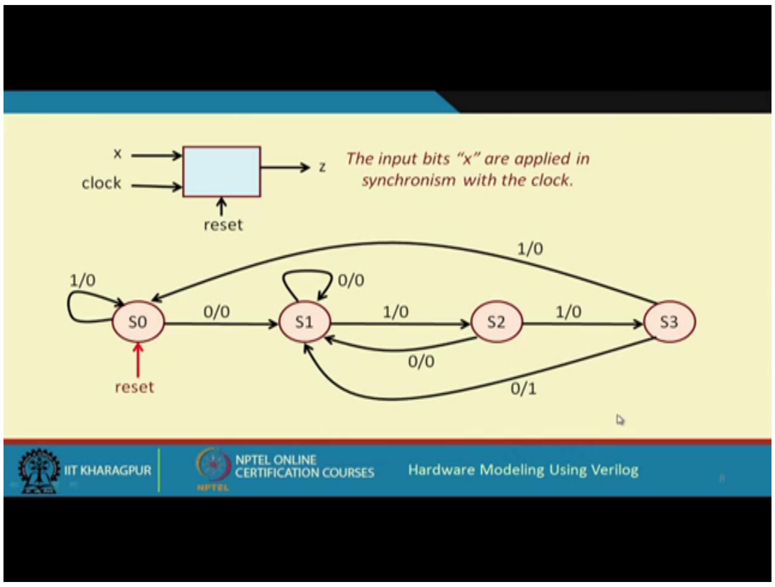
- Design of a sequence detector.
  - A circuit accepts a serial bit stream "x" as input and produces a serial bit stream "z" as output.
  - Whenever the bit pattern "0110" appears in the input stream, it outputs z = 1; at all other times, z = 0.
  - Overlapping occurrences of the pattern are also detected.
  - This is a *Mealy Machine*.
  - Example: x :- 0 1 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 0

    z :- 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0

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

And here I get my first 0 1 1 0. So, this is my first one generated, again overlapping 0 1 1 0 ago 1, generated and again you have one here 0 1 1 0 again a 1 generated this will be your output stream, right.

(Refer Slide Time: 16:25)


x
clock
z
reset
The input bits "x" are applied in synchronism with the clock.
1/0
0/0
S0
S1
S2
S3
1/0
1/0
1/0
0/0
0/1
reset
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

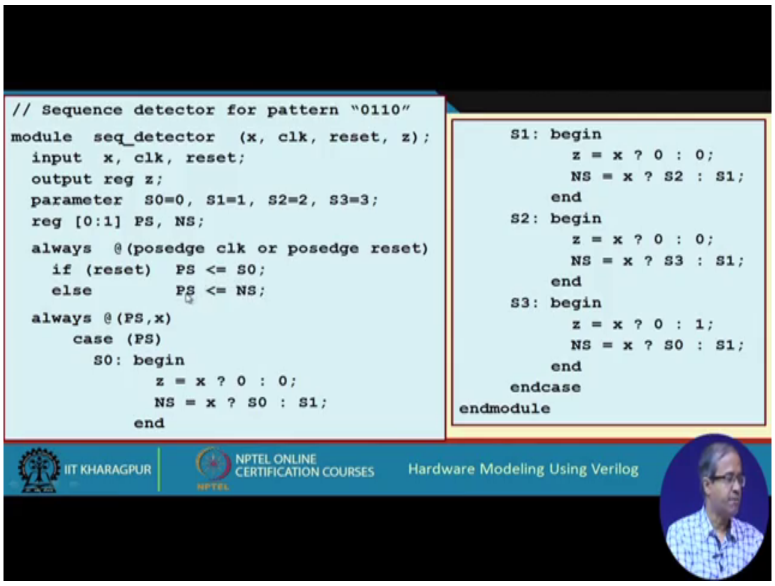
Now, what will be my state transition diagram, let us try to understand. You see intuitively here I want to detect 0 1 1 0. So, you may say that well I start with some initial state, 0 I go to one state, 1 I go to another state, 1 again 0 there should be 5 states. But when you start drawing the state transition diagram you can find out whether you really need 5 states or we can do it less in this example, you can specify the description using 4 states only. How it is like this? So, I am assuming my circuit is as follows where there is one input x there is of course, a reset input reset time showing separately, clock and the output is z. The input bits they are applied in synchronism with the clock. Now here whenever the reset signal is active I start in state s 0.

Let us see. Here we are trying to detect this string 0 1 1 0. And whenever it is that output should be 1. So, let us identify the states, s 0 means the initial state. So, whenever I get the first 0 I go to s 1, what does s 1 mean? That I have seen the first 0 of the string; then if I get a 1 I go to s 2 what does s 2 mean that I have got 0 followed by a 1. Then I go to s 3, this s 3 means s 3 whenever 1. So, 0 1 1 s 3 means I have seen 0 1 1.

Now if I get another 0 what I could do I could have brought it back to s 0, you start with another string, but now because I am allowing overlapping of the strings 0 1 1 0 I am moving it back to s 1 not is 0 y is one because this last 0 can also be the first 0 in the string. So, as I had given that overlapping example earlier the last 0 of 0 1 1 0 that 0 can also indicate the beginning of the next 0 1 1 0. So, whenever a 0 comes which means I have encountered a 0 1 1 0 the output goes to one and my next state goes to s 1 because I am looking for the overlapping string if there is another 1 1 0. So, again a one will be generated right.

But now let us look at the other edges. So, in s 0 if I see that a one is coming which means where it is not starting with 0. I remain in s 0 I wait for the first one. Now while in s 1 I am expecting a one to come, but if it is 0. So, I remain in state s 1; that means, a 0 is still there may be a one is coming. Now in state s 2 I have received 0 1 I am expecting a 1, but if a 0 comes I go back to s 1 because I have seen a 0 maybe I am again starting to look at this string. Similarly that s 3 I have seen 0 1 1, but again a one comes. So, I have to start from the very beginning I have to again look for a 0 and like this. So, this will be my state transition diagram for this example right.

(Refer Slide Time: 20:05)



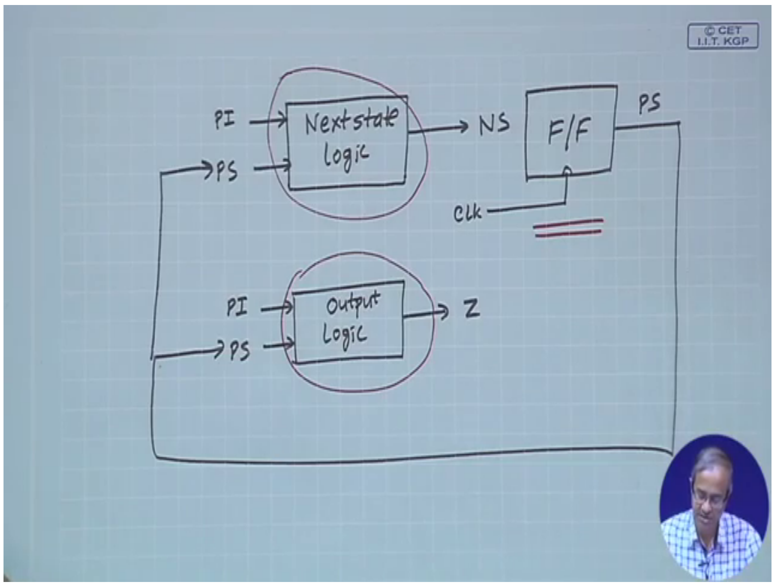
```
// Sequence detector for pattern "0110"
module  seq_detector  (x, clk, reset, z);
  input  x, clk, reset;
  output reg z;
  parameter  S0=0, S1=1, S2=2, S3=3;
  reg [0:1] PS, NS;
  always  @(posedge clk or posedge reset)
    if (reset)  PS <= S0;
    else        PS <= NS;
  always @(PS,x)
      case (PS)
        S0: begin
              z = x ? 0 : 0;
              NS = x ? S0 : S1;
            end
        S1: begin
              z = x ? 0 : 0;
              NS = x ? S2 : S1;
            end
        S2: begin
              z = x ? 0 : 0;
              NS = x ? S3 : S1;
            end
        S3: begin
              z = x ? 0 : 1;
              NS = x ? S0 : S1;
            end
        endcase
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
Hardware Modeling Using Verilog

So, this is my verilog description. So, you see here we have already separated out 2 always blocks. So, in the one always block we have just only talked about the state changes you see the description how we have done it. X clock and reset are the inputs z is the output reg. And because there are 4 states I need 2 state variables reg 0 to one. So, I call or I define 2 variables one is the present state and the next state. And using parameter for convenience I call them the 4th state s 0 s 1 s 2 and s 3. In the first always block which is triggered by the positive edge of the clock or the positive edge of the reset what do you do, if the reset is active I initialize the present state to s 0, or if the reset is not active and a clock has come.

So, whatever is my next state that will now become my present state; so as the machine progresses clocks are coming. So, I have some inputs I have generated the next state when the next clock comes that next state will become my present state again some input will be coming I generate the next state. Next time that next state will again become my present state and this will continue. So, here with the clock my next state is becoming my present state. And in this always block we are actually generating the next state and z.

(Refer Slide Time: 21:54)

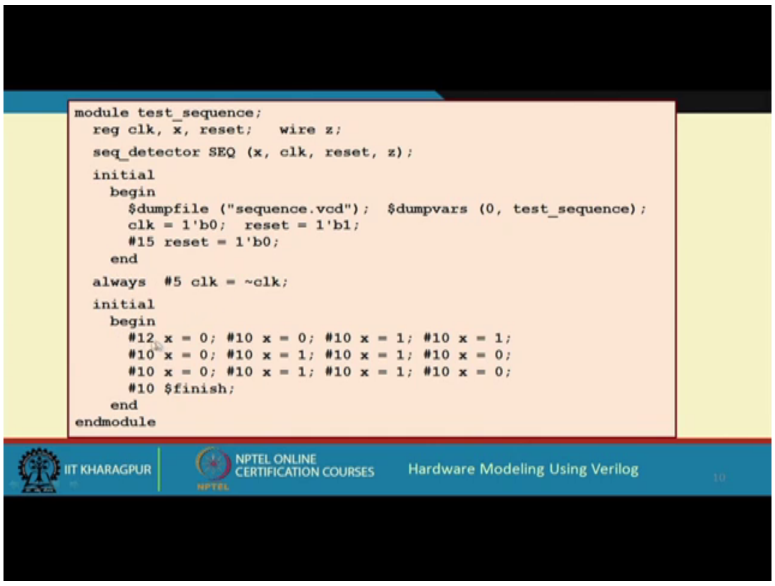


You see you just recall for the that model of the sequential circuit we have given. We talked about something called next state logic. And we talked about another block called output logic. So, what does the next state logic contain it will contain the primary inputs and my present state, it will generate the next state. And the output logic it will also take as input the primary input and the present state it will generate the output let us say z. Now both of these are combinational circuits.

Now this next state this is being fed to some flip flops. This flip flop is fed with a clock, and the output of the flip flop I am calling as the present state. So, whatever is my present state this is being fed here as well as here right. So, this is how it works. So, the description of this flip flop I have kept in one always block for basically n s is going to p s. And description of these 2 combinational blocks I have kept separate, both of these will be combination. So, that I am doing in this second always block. So, I am just activating it or triggering it whenever p s or x changes if the state is s 0 you see what happens in s 0 if the input is 0 I go to state s 1 if the input is 1, I remain in state s 0.

So, if the let the next state if the input is 1, I remain in state s 0 if the input is 0 I go to state s 1. And the output is 0 under both conditions. So, here you can either write z equal to if x then 0 8 0 or you can simply write z equal to 0, because your z will be equal to 0 irrespective of x. So, this also and this also you can simply write z equal to 0. So, some s 1 if x is one I go to s 2 if x is 0 I remain in s 1. Like that just exactly following the state transition diagram I code these if then else statements right, and my description is complete.

(Refer Slide Time: 24:29)



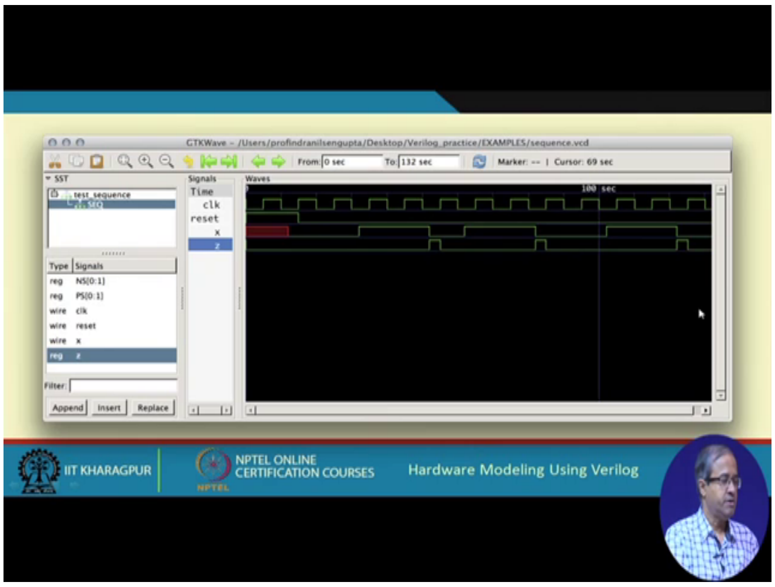
```
module test_sequence;
  reg clk, x, reset;   wire z;
  seq_detector SEQ (x, clk, reset, z);
  initial
    begin
      $dumpfile ("sequence.vcd");  $dumpvars (0, test_sequence);
      clk = 1'b0;  reset = 1'b1;
      #15 reset = 1'b0;
    end
  always  #5 clk = ~clk;
  initial
    begin
      #12 x = 0; #10 x = 0; #10 x = 1; #10 x = 1;
      #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
      #10 x = 0; #10 x = 1; #10 x = 1; #10 x = 0;
      #10 $finish;
    end
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, similarly I can create a test bench. So, I instantiate my sequence detector x clock reset and z. X clock and reset are the inputs there is a declared as reg z will be wire. So, again I am creating the dump files. I am initializing the clock to one and the reset clock to 0 and reset to 1. So, I am initially resetting and at time 15 I am changing the reset to 0. So, from time 15 onwards my actual operation will start, and clock again with time 5 I am toggling.

So, again I am giving a little delay 12 because the first operation should start at time 15 right. Because till 15 reset is active, after 15 actually operation start. So, just before that at time 12 I am setting or I am sending the first bit. Then I am sending the consecutive bits after gaps of 10 10 10 because clock period is 10. So, 0 0 1 1 0 1 1 0 0 1 1 0 like this we are applying right.

(Refer Slide Time: 25:49)

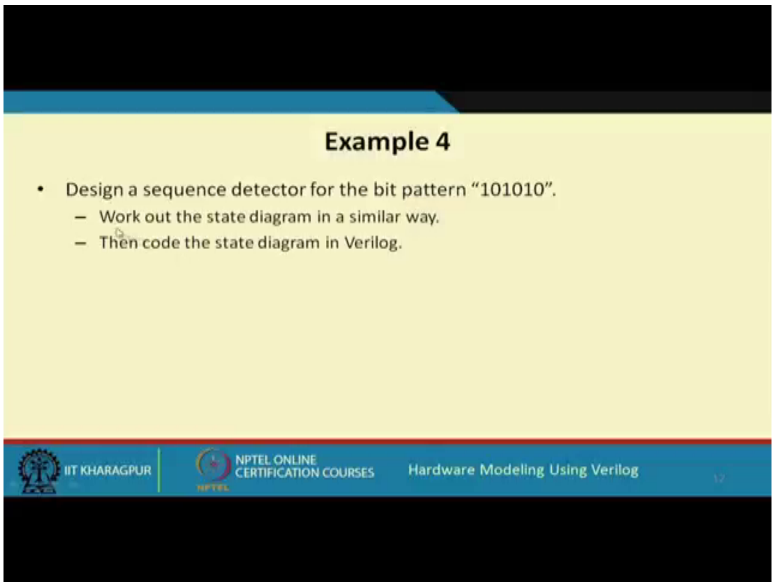


So, and then out delay we finish. So, the simulation output comes like this. So, this is I am not showing the whole of it a part of it, the clock is coming the reset you see the reset was activated initially to 1, and it remained till 15 it becomes 0. So, reset was one this is time 15 at time 15 it becomes 0 and after that it remains 0. So, my circuit starts operating from that point onward.

And what about x I have set x equal to 0 at time t equal to 12. So, c at time t equal to 12 I have before that it was undefined this red block red block means undefined. It becomes 0 here then the successive inputs I am applying 0 0 1 1 I am applying like that 0 0 1 1 then 0 1 1 0. So, 0 1 1 0 like that; so wherever I get a 0 1 you see the first 0 1 1 0 comes as 0 1 1 0 the first one comes here then there is an overlapping 0 1 0 1 1 0 ok.

Then at the end there is again a 0 1 1 0. So, there will be 3 detections z will become one here 0 0 1 1 first detection then overlapping 0 0 1 1, second time, then again a 0 0 1 1 third time. So, there will 3 outputs that equal to 1. So, we have seen how we can actually model both mealy and Moore machines I means using verilog. You can tell other examples also similarly. So, the process of coding them in verilog is exactly similar. So, you can take some other example, like for example, talking about the talking about the sequence detector again.

(Refer Slide Time: 27:50)



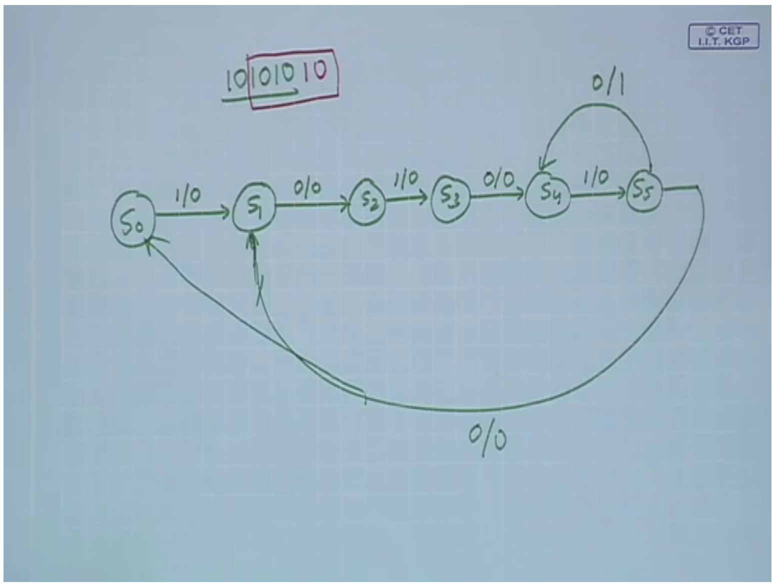
Example 4

- Design a sequence detector for the bit pattern "101010".
  - Work out the state diagram in a similar way.
  - Then code the state diagram in Verilog.

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

Suppose I want to detect this sequence 1 0 1 0 1 0. So, how will my state diagram look like?

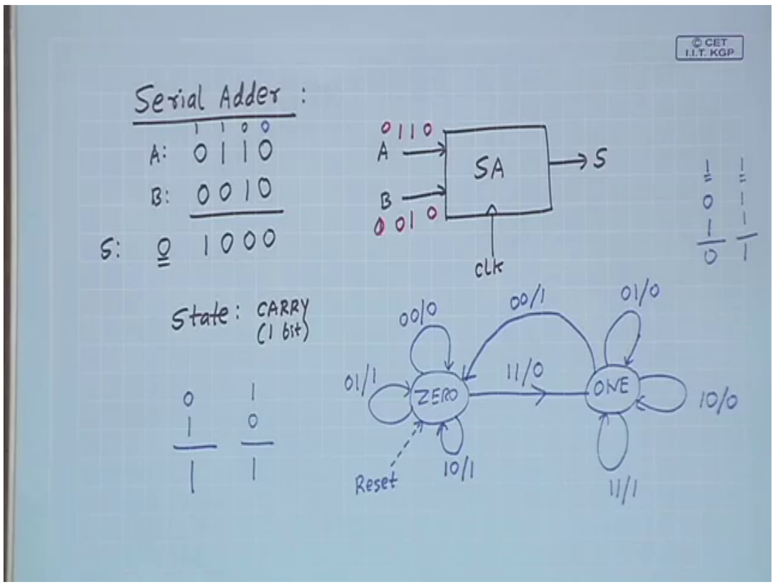
(Refer Slide Time: 28:05)



So, it will look something like this, I mean state s 0. So, I am trying to detect the sequence 1 0 1 0 1 0. So, when the first one comes I go to state s 1. This is my first one comes, then a 0 comes I go to s 2. Then again a one comes, I go to s 3. Then again a one comes I go to s 4, 0 1 0 1 0 0 comes. Then the fifth one I go to s 5. Then if a 0 comes, where do I go now again if I look at overlapping patterns, see 1 0 1 0 1 0 if there is again a one and a 0 I can consider this as overlapping pattern right. So, if a one comes after this.

So, I should actually send it to a state here where I am expecting a 0 again. So, if it is one I have to send it to s 1, not really s 1 I can actually send it to I need a one and a 0 after that you can send it to s 4, if there is a; 1 0 1 0 1 0 with an output of 1. So, if there is another one I go to s 5 it is another 0 I again go to here, but if my next bit is let us say instead of one if it is 0, then not here I have to send it back to this s 0 again if it is 0. So, the other edges you can define like that ok.

(Refer Slide Time: 30:08)



Let us also talk about another example, suppose we want to design a serial adder. Well, you know about binary addition suppose I want to add these 2 numbers I do a bit by bit addition 0 plus 0 is 0 with a carry of 0 0 1 1 is 0 with a carry of 1 1 1 0 is 0 with a carry of one this is one with a final carry out of 0. We do it like this normally here we have seen various kinds of adders like ripple carry adder or the carry look adder boat are both both these designs we have seen earlier. But now let us assume that we want to do this addition bit by bit serially, how? Suppose we have an adder this is my serial adder. So, my 2 inputs are let us say A and B. So, there is a single bit of a and single bit of b I am applying at one time. And this is my sum and I am generating one bit of sum at a time. So, if these are my inputs let us say 0 and 0 first I will apply 0 here I will apply 0 here. Then I will apply one then I will apply one I will apply one then I apply one I apply 0 then I apply 0 I apply 0.

So, this I will be doing in synchronism with a clock, there will be a clock signal. And you see internally if you know talk about this state, what do we have to remember? Here for every stage of addition we only need to remember the previous carry. So, the only state we need to remember is a carry, and that carry will be 1 bit. So now, you can construct a state transition diagram directly by considering the state of the carry let us say the carry was 0 the carry was 1. So, if you are resetting the circuit it will start with carry of 0. So, initially there is no carrying right. So, you start with 0 carry.

Now, let us say. So, if your input is 0 0 what will happen, if the input is 0 0 there will be no carry. So, it will remain in the 0 state and the output will be 0, because if you add 0 and 0 output is 0, but carry is still 0. Now if we apply 0 and 1 what will happen? Say if you are apply 0 and 1, the sum will be 1, but no carry. So, carry will still be 0. So, there will be another h if it is 0 1, but now sum will be one similar will be the case if the inputs are 1 and 0 again sum will be one, but no carry; so 1 and 0 1, but if it is 1 1 like this then sum will be 0 and carry will be 1.

So now, the carry state changes if it is 1 1 the sum will be 0, but carry has become one, but once carry has become 1, let us say my inputs are say 0 1 or 1 0 or 1 1 then there will be a carry you see. 0 1 means what? I have applied 0 and 1, but there is also a carry of 1. So, this 3 bits are added sum will be 0 and again carry will be 1. So, it will remain in the one state sum will be 0. Similar is for one same thing, but for 1 1 if there is also a carry 1. So, I have a sum of one and also carry of 1. So, the output will also be 1. Now the only way to go back to 0 is when the input 0 0 is applied. 0 0 with a carry of 1 this sum will be 1, but next carry will be 0. So, this will be my state transition diagram for a serial adder.

So, in this way you can actually create the state transition diagram of any f s m you can think of, and once you have created this state transition diagram of the state table. Translating it to a verilog code in the way we have seen is absolutely straightforward.

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

Thank you.