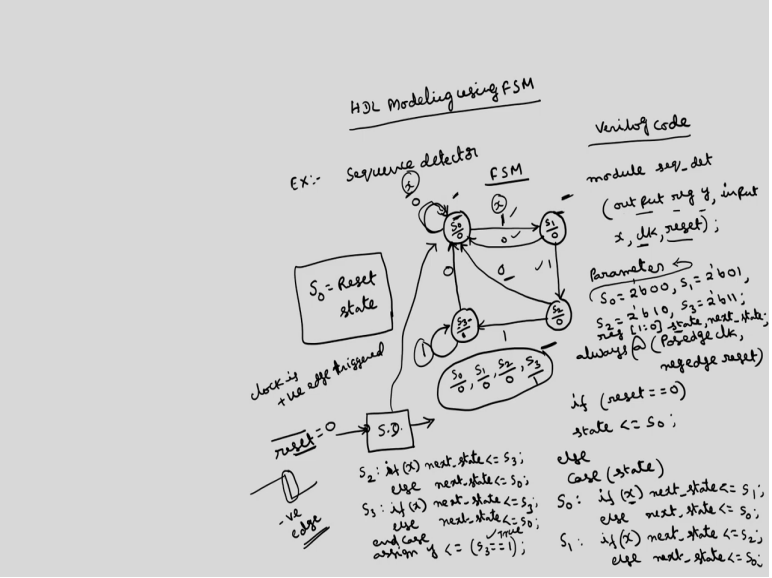
System Design Through VERILOG
Prof. Shaik Rafi Ahmed
Department of Electrical and Electronics Engineering
Indian Institute of Technology, Guwahati

Behavioral modelling of sequential circuits
Lecture - 20
Verilog modelling FSMs and shift registers

In the last lecture, we have discussed about the sequential circuit design and then its Behavioral modelling. So, we have discussed the circuit diagram of a sequence detector, which can detect three or more ones in a bit stream. So, we have written the Verilog code using behavioral modelling.

So, because of this circuit complexity, sometimes it becomes difficult to write the Verilog code. So, we can directly write the Verilog code from the state diagram; for the complex systems whose state diagrams are complicated, so we can directly draw the Verilog code using state diagrams. Today we will discuss how to write the Verilog code using the state diagram or FSM.

(Refer Slide Time: 01:31)



I will call as HDL modelling using FSM or state diagram. I will take the same example of the sequence detector, which can detect three or more consecutive ones in a bit stream. So, we

know the state diagram of this one is we have four states; s 0 output is 0, s 1 output is 0, s 2 output is 0, s 3 output is 1. So, if it receives 0, it will stay here itself, this label represent input; if the input is 1 it will go to the next state, because it has to detect three or more consecutive ones.

Similarly, here if a 0 is received, so it will go to again the reset state which is s 0; if a 1 is received it will go to the next state here; if 0 is received it will go to s 0; 1 is received it will go to the next state; here 0 is received it will go to the s 0 state; 1 is received it will stay itself, because already detected 3. So, this circuit can detect three or more consecutive ones.

So, this is the state diagram or FSM we have discussed in the last lecture. But in the last lecture using this FSM, we have designed the sequential circuit using flip flops and then we modelled the flip flop-based circuit using Verilog. Whereas, here we can directly model this FSM, so that the Verilog code becomes somewhat simpler. So, how to write the Verilog code in terms of the FSM?

So, module sequence detector, output and also register we will call as y; I will write all inside this one also, this is also another way to represent input output, instead of writing again separately this input output, so I will write here itself. So, there will be only one output which is called y and one input x, another input is clock, I am going to use another input called reset. Here no need to again define this register the output and input.

Now, here I am assuming that this s 0 is reset state, that is whenever this sequence detector circuit, receives a reset which is active low signal. So, reset bar means negative edge, whenever this becomes 0; if it is high, then it goes to low, this is at negative edge. So, this sequence detector automatically will go to the state s 0, this is what is called the reset state; whenever reset bar is 0, so this state diagram we will assume the state s 0, that is what is called the reset state.

That is why I am using another signal reset. I am assuming the clock is positive edge triggered and reset is negative edge triggered. So, we are going to define these states with binary values, using a statement called parameter; parameter is the key word used to assign

some binary values to the states. So, here I am using s 0 is equal to 2 ‘b 0 0, s 1 is equal to 2 ‘b 0 1, s 2 is equal to 2 ‘b 1 0, s 3 is equal to 2 ‘b 1 1 semicolon.

This is we have to write after the parameter. So, we are going to define some constant binary values to the states using this parameter keyword. Now, always @ (posedge of clock, negedge reset), we have to define state current state and next state, so we can have a register here, register [1:0] state, next state.

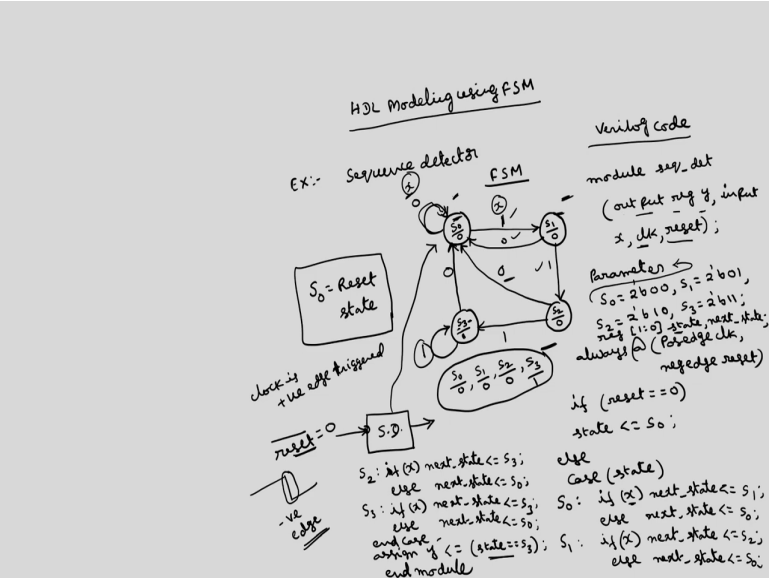
So, always @ (posedge of clock, negedge reset), if (reset = 0); the initial steady state is equal to s 0; else case now we have four different values of this state. So, this state is 2 bit value, because this is 2 bit, 1 0; because state 1, state 0, state 0 is LSB, state 1 is MSB. So, we have already defined s 0, these are x, this is representing x.

So, when the state diagram is in state s 0, when the input is 0, it will go to the s 0 itself and input is equal to 1, it will go to s 1 state. If this is true, then the next state= s 1; else next state = s 0 itself. If x is equal to 1, if this is true, next state is 1; if x is equal to 1, the next state for this s 0 is s 1; if false, it will remain in the s 0 state only.

Similarly, for s 1, if x for s 1; 0 it will go to s 0 state, 1 it will go to s 2 state. So, this is true, next state is equal to s 2; else next state is equal to s 0. Similarly, s 2 if (x) next state at s 2, 0 means next state is s 0 itself, if true, next state is s 3; else next state s 0. s 3 if (x) next state is s 3 itself; in s 3 if the input is 1, it will go to s 3 itself, else next state is s 0.

So, this is about the description of the state diagram; then whatever the output; in s 3 state only, output is equal to 0, you can see that s 0 we have output is 0, s 1 output is 0, s 2 also output is 0, only for s 3 we have output is equal to 1. So, for that you can write end case, this case is over, assign output y; we have called the output as y is equal to s 3 equal to 1, means whenever s 3 is equal to 1, this is true, y is we will get y, y is equal to 1 or state is equal to you can write.

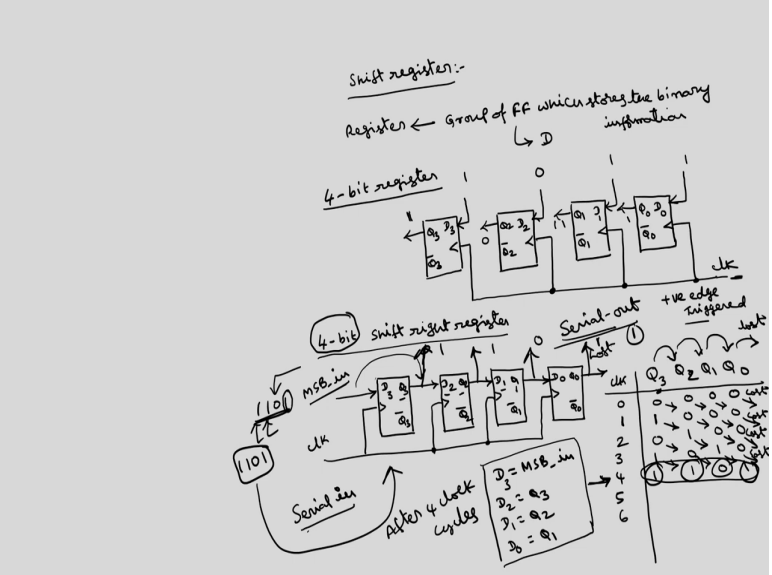
(Refer Slide Time: 12:56)



So, y is equal to state, if the state is s 3, if this is true; y is equal to 1, otherwise y is equal to 0 end module. So, this is about the Verilog code using FSM. We can see that this code is very simple when compared with the code which you have written for the sequential circuit, which we have implemented using JK flip flops.

So, you see about this sequence detector, in a similar way you can design any sequential circuits, all these synchronous counters; say for that matter any synchronous sequential circuit can be designed in the similar manner, so using the procedure that we have described in the last lecture. This is one of the important sequential circuit. So, basically sequential circuits means normally we will deal with the counters and shift registers.

(Refer Slide Time: 14:12)



So, the next sequential circuit is shift register. As the name implies, so upon the application of this clock; the contents of register will be either shifted to the left or right, depends upon whether the content shift to left or right, it is called as shift left register or shift right register.

So, before going for the shift register, if you see basically a register first of all; what is a register? It is a group of flip flops, which stores the binary information. So, normally this flip flop that we are going to use is D; because in case of shift registers, we want to transfer the data from input to output. So, D flip flop is very much suitable for the shift registers.

So, we know that each flip flop can store 1 bit of the information. So, if we want to store n bits of information, we require n flip flops. So, if you connect all these n flip flops together, it can store n bits of the data. So, in general a n bit register consisting of n flip flops.

If I consider a 4-bit register, it will have 4 flip flops; this is D 0, LSB flip flop, this is Q 0 output of the LSB flip flop, D 1, Q 1, Q 1 bar, D 2, Q 2, Q 2 bar; D 3, Q 3, Q 3 bar. And all these flip flop will be having some clock, either it can be positive edge triggered or negative edge triggered; even it can be level triggered also, because in D flip flop there is no problem with race around.

So, if we want to store any data, you can directly load the data at this point. Suppose if you want to feed a data of 1 0 1 1, so we take here 1 0 1 1. We just apply the clock signal, then

this data will be fed here; means this will be 1 will be available here, 0 will be available here, 1 will be available here, another 1 will be available here.

So, this will continue until the next clock signal is applied; whether it can be positive edge triggered or negative edge triggered, let us assume that all are positive edge triggered. So, till the next positive edge is applied, so this data will be stored. If I do not apply this positive edge, this will be present as long as the power supply is there and that is how we can store the data.

But in many applications, it is required to shift the data either to the left or right. If we take a binary number if I shift the data to the right, it will divide by 2; if you shift to the left, it will multiply by 2, select that multiplication division operations also can be performed. So, in addition to that there are a lot of applications where we require to shift the data of the register. So, this is a simple 4 bit register without any options of shifting the data to the right or left.

Now, I will slowly move on to the shift register, say first I will take 4 bits shift right register. If you want to shift the data to the right what we have to do is, circuit diagram is same; you have to use 4 D flip flops, but the connections are different. This is D 3, D 2, D 1, D 0 correspondingly the outputs are Q 0, Q 0 bar, Q 1, Q 1 compliment, Q 2, Q 2 complement, Q 3, Q 3 complement here the data that has to be feed has to be given here, this is called MSB_in.

And the data from Q 3 will shift to D 2, Q 2 to D 1, Q 1 to D 0 and this will be lost, this data will be lost. So, you see the circuit diagram of 4 bit and this will be having common clock; if we assume that this is positive edge. This is the circuit diagram of 4 bit shift right register.

If you want to explain the operation, so you can write Q 3, Q 2, Q 1, Q 0. So, here if the data that is to be feed here is say 1 is the first bit 0, second bit 1, 1 and so on. So, initially let us assume that when clock is equal to 0, initial state is 0 0 0 0. How to establish this initial state? Using preset and clear inputs.

Now if I apply the first clock signal and if the first bit of MSB is 1, whatever this 1 will be transferred to the Q 3. So, this 1 will be transferred here, whereas this is shift right as the name implies; this Q 3 will shift to Q 2, Q 2 to Q 1, Q 1 to Q 0, and Q 0 will be lost.

So, this 0 will shift here, this 0 will shift here, this 0 will shift here, this 0 will be lost. At the 2nd clock signal, the second bit that is applied as MSB_in will be inputted into Q 3; 0 this 1 will be shifted here, this is 0 will be shifted here, this is 0 will be shifted here, this will be lost.

At 3rd clock signal, the third bit is say 1. So, this 1 and this 0 will be shifted here, this 1 will be shifted here, this 0 will be shifted here and this will be lost. At 4th clock signal, if I assume that the fourth bit is 1; this 1 will be shifted here, this 0 will be shifted here, this 1 will be shifted here. So, you can see here this will be lost.

Now we can see here this 1 0; 1 1 0 1 whatever you have taken here this, this bit is 1 1 0 1; this will be feed into the register after 4 clock cycles. Then if we want to take this data, we are feeding this data serially; this is called serial in, we want to take the serial data out, serial out, if you apply 4 more clock signals.

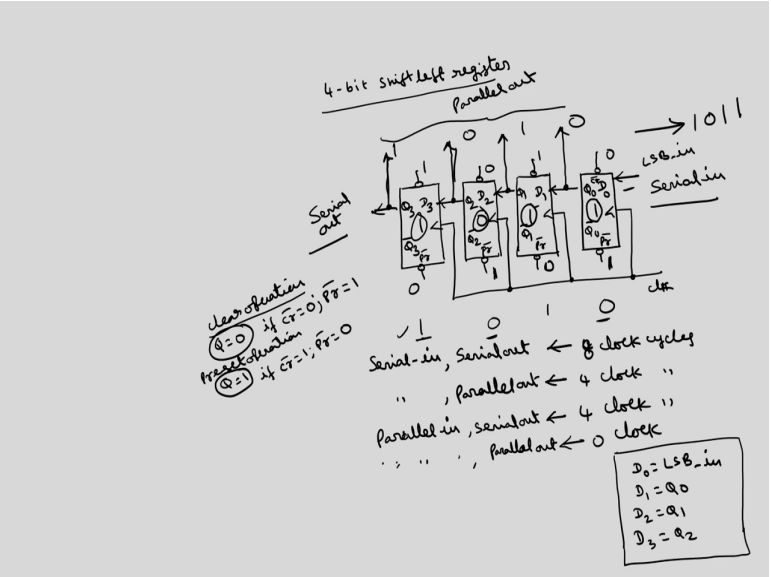
If I apply 5th clock signal, this 1 will be outputted here; at 6th clock signal, this 0 will be outputted; at 7th this 1 will be outputted; 8th this 1 will be outputted. So, after 8 clock signals what happens; whatever this data we have feeded serially, that will be outputted serially.

So, this type of register can also be called as serial in serial out register. If you want the serial in parallel out, then 4 clock cycles are required; after 4 clock cycles, if I read the data from here. So, what will be the data available is, whatever that you have feeded serially is 1 1 0 1.

So, if we want to convert the serial in data into parallel out, we require 4 clock cycles in case of 4-bit shift register. In general, for n bit shift right register, to convert the serial in data into parallel out, we require n clock cycles; whereas for serial in serial out, we require 2 n clock cycles.

So, this is about the shift right register. So, in shift right register, so what is D 3? D 3 is MSB_in, D 2 is Q 3, D 1 is Q 2; we can see here, D 2 is Q 3, D 1 is Q 2, D 0 is Q 1. So, these are the binary expressions for 4-bit shift right register. In a similar manner, you can implement shift left register also, 4-bit shift left register.

(Refer Slide Time: 24:58)



Now, here the serial in will be at LSB. So, here this is D 0, at D 0 we have to feed the external data which is called as LSB_in or we can also call as serial in. Then this is Q 0, Q 0 compliment, this Q 0 is connected to D 1; this is Q 1, Q 1 complement, this Q 1 is connected to D 2; this is Q 2, Q 2 complement and this Q 2 is connected to D 3; this is Q 3, Q 3 bar, we can take the data serial out from Q 3. We can explain the operation in a similar manner.

Similarly, if we want to take the parallel data, you can take from these four points, this is parallel out. So, we can perform three operations here. Similarly, in previous case also we can perform three operations, one is serial in serial out. So, how many clock cycles are required as we have discussed in the earlier circuit? So, for serial in serial out, we require 8 clock cycles; in general, for n bit, n clock cycles, whereas for serial in, parallel out 4 clock cycles.

Now, other two options are parallel in serial out; parallel in parallel out. How to input the data parallel? For that you can use the preset and clear signals also or we can directly feed onto the D's; here I will show using preset and clear. If I make all preset values to 1; this is preset values, this is clear values.

So, we know that preset operation, when Q is equal to 0, this is called clear operation. This can be obtained, if clear bar is equal to 0, preset bar is equal to 1. And preset operation, Q is

equal to 1; if clear bar is equal to 1, preset bar is equal to 0. Suppose if I want to feed the data parallelly say 1 0 1 0.

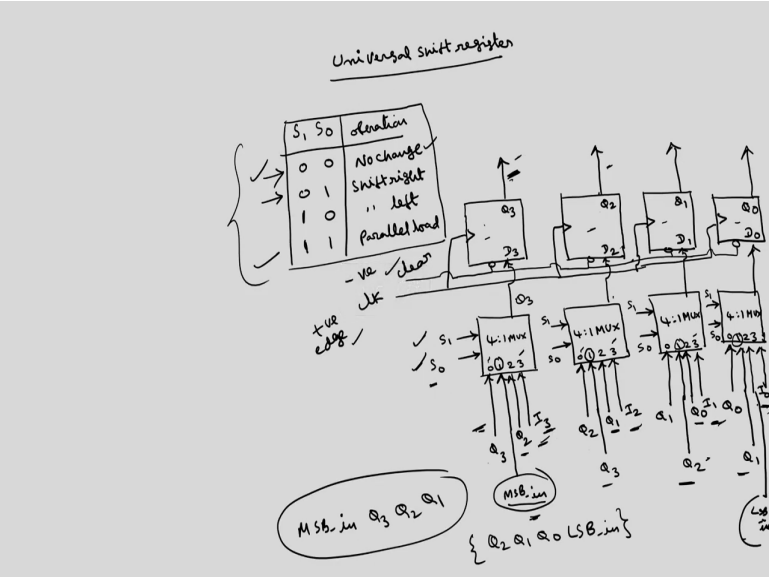
So, these two are 1s means, this preset is 0, this preset is 0, so that 1 will be transferred. This preset is 1, this preset is 1; whereas the clears these two has to be 0, this is 0, this is 0 and these two are 1s, then automatically this data will be feed. So, for parallel in to input the data parallelly we require only 1 clock cycle; to get the serial data, we have to apply 4 clock cycles, so that this 1 0 1 0 whatever you have feeded here, 1 0 1 0 to take this from here out, we require 4 clock cycles.

After first clock cycle, this will be out; second clock cycle, this will be out; third clock cycle, this will be out; fourth clock cycle this will be out. So, we require total 4 clock cycles; because this preset and clear are asynchronous inputs, they do not require any clock cycle. We can feed this 1 0 1 0 simultaneously without clock by making corresponding preset and clears 0 or 1.

So, parallel in, if want to input the data parallelly, we do not require any clock cycle. To take serial out, we require 4 clock cycles; for parallel in parallel out, this does not require any clock cycle. Once if you feed this, if you read from this Q 3 Q 2 Q 1 Q 0 you will get; so, whatever you have feed 1 0 1 0 will be available here.

This way you can perform four operations; one is serial in serial out, serial in parallel out, parallel in serial out, parallel in parallel out. This is about shift left and shift right registers and with parallel load. Now, if I want to combine all those things, shift right operation, shift left operation, parallel load operation in a single shift register, that is called universal shift register.

(Refer Slide Time: 31:56)



Here I am going to perform four operations; four operations mean we require two selection signals. So, I will take the selection signals as s 1, s 0, then the operation. You can design in anyway, so I am designing this circuit such that 0, 0; no change; whatever the state of the register remains in the previous stage only, the register remains in the previous state only; 0, 1; I want to shift right, 1, 0; I want to shift left, 1, 1; I want to apply parallel load.

So, what will be circuit diagram for this. So, because we have four different operations and two control signals, I can use a 4 by 1 multiplexer. So, basically here I am going to use four 4 by 1 multiplexers; this is another 4 by 1 mux, this is another 4 by 1 mux, this is another 4 by 1 mux. So, we know that this is 0, 1, 2, 3, we have four inputs 0, 1, 2, 3; 0, 1, 2, 3; 0, 1, 2, 3 and 1 output; there are two selection signals, this I will apply s1, s0, these are common to all the four multiplexers.

Now, what are the signals that has to be applied at 0th input of the first multiplexer; 1st, 2nd and 3rd? Similarly, here what are these four inputs? Similarly, for the third multiplexer, what are the four inputs? Fourth multiplexer what are the four inputs? Who is going to decide this depends upon whether the operation is shift left or shift right, parallel load or it remains in the same state.

So, basically these four multiplexers are going to drive 4 D flip flops. These are 4 D flip flops with common clock signal and you can also provide the clear signal also; similar to the previous counter, you can have clear input, reset inputs are permanently tied to 1. So, if I connect this clear signal, which is active low signal; this is clear, this is positive edge and this is say negative edge.

So, whenever 0 is applied, what happens all the four flip-flops will be reset. So, we know that we have four flip flops with the different D's; say this is D 3, D 2, D 1, D 0. So, this I am going to connect to outputs of the corresponding multiplexers. This output I am going to connect to D 2, D 3, this to D 2, this to D 1, this to D0; because the multiplexer will be having only one output. So, the corresponding outputs I will call as Q 3, Q 2, Q 1, Q 0.

Now to satisfy this table, so 0, 0 no change. So, what should we this 0th input, 0th input, 0th input, this 0th input, such that so these four flip flops will remain in the previous state only. So, far that you have to apply here Q 3, here Q 2, here Q 1, here Q 0. So, whenever s 1, s 0 is 0 0; so this 0th will be selected, so the output will be simply Q3, Q2, Q1, Q0.

So, when input is Q 3, output is also Q 3. So, it will remain in the same state, this is how we can satisfy this condition; 0, 0 no change means, so just feed this output, there is a connection from this output to this input, but I am not showing, directly I am writing Q 3, Q 3, because there is a these two are shorted.

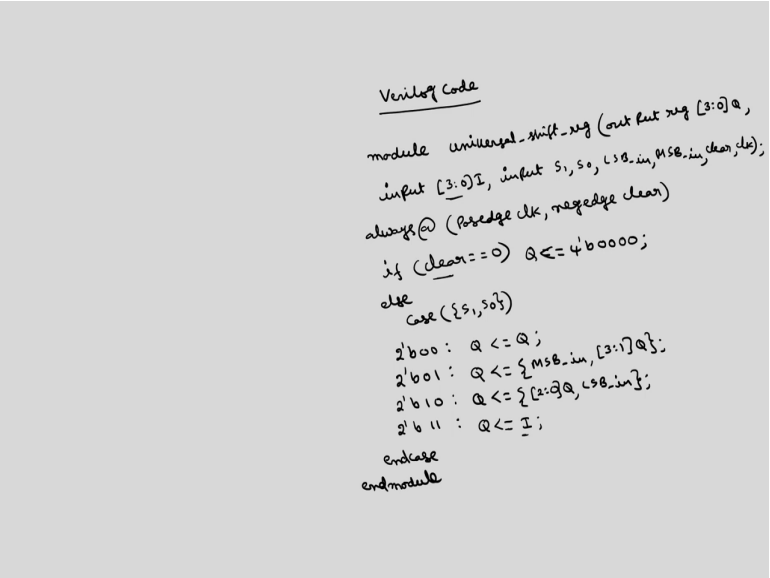
Now, we can think of what will be the first input. When s 1, s 0 is 0, 1; then this one inputs of all the four multiplexers will be selected. Now, what will be the first input? So, for shift right I have already derived the relation; for shift right D 3 will be MSB_in and D 2 is Q 3 Q 2 Q 1. So, D 1 is Q 2, D 0 is Q 1, so MSB_in, Q 3, Q 2, Q 1. So, this is the point where we have to apply MSB in and this is the point Q 3, this is Q 2, this is Q 1.

So, this Q 3 is this, this Q 2 is this and so on; there is a connection, so I am not showing those connections. Now, for the shift left, so I have drawn the circuit; but I have not given the expression, we can write in a similar way. So, what is D 0 here? D 0 is equal to LSB_in, D 1 is equal to Q 0, D 2 is equal to Q 1, D 3 is equal to Q 2 these are the expressions for shift left register.

So, D 0 is LSB_in; then Q0, Q 1, Q 2. So, this is LSB_in, this bit is LSB_in and this is Q0, Q 1, Q 2. Now, only thing left is 1 1; s 1, s 0, 1 1 this three inputs will be enabled. So, I want to use parallel load. So, whatever the data that I want to feed parallel, I will take here; say I will take the input as I 3 bit I will apply here, I 3 can be 0 or 1, in general I will take I 3, I 2, I 1, I 0.

So, whatever the data given it I 3, I 2, I 1, I 0and if s 1 this is equal to s 0 is equal to 1, 1; this data will be feed into the register, that is called parallel load. So, this is about the universal shift register, this can perform these four operations. Now, to how to write the Verilog code corresponding to this universal shift register?

(Refer Slide Time: 41:35)



Module universal shift register. So, what are the inputs and outputs? Inputs are s1, s0, I 3, I 2, I 1, I 0, then we have MSB_in, LSB_in, then clear, clock this many inputs are there. Here two inputs, here four inputs, here two inputs, here two inputs. So, totally I have ten inputs.

And how many outputs? Outputs are finally, this Q3, Q2, Q1, Q0 only four outputs. First I will write output register [3:0] because vector Q, input [3:0] I input and we have a s 1, 0, LSB_in, MSB_in, clear, clock; you have defined inside the parentheses itself.

Now always @ (posedge clock, negedge clear) if clear is equal to 0. So, clear is equal to 0; simply all the Q 3, Q 2, Q 1, Q 0 has to stay in 0 0 0 0 state. Q is equal to 4'b 0 0 0 0; so, 4 bit

0 0 0 0, else we have to follow the case. Case, so who is going to decide? s 1, s 0. So, what are the possibilities? 2'b 0 0. If s 1 s 0 is 0 0, so what will be the outputs? No change means outputs will be Q 3, Q 2, Q 1, Q 0, Q only.

2'b 0 1, what will be Q? This is MSB_in; if I follow in the MSB to LSB, what is the order? MSB_in; then the second bit is Q 3, Q 2, Q 1, this is the order we have to give. So, I will give in the same order MSB_in Q 3 Q 2 Q 1; MSB_in, [3:1] Q, this we will Q 2 MS_ in Q 3 Q2 Q 1; 2'b 1 0 Q, this is Q 2 Q 1 Q 0 LSB_in.

So, we have to be in this order Q 2 Q 1 Q 0 is [2:0] Q, Q 2 Q 1 Q 0 , LSB_in. 2'b 1 1 is parallel load, Q has to be loaded with I; I 3 I2 I1, this is I 3, I 2, I 1, I 0, this I have already defined as vector here, we can write this as simply, so end case, end module.

This is the Verilog code for universal shift register, which can perform the four operations; one is it remains in the same state, shift left operation, shift right operation or we can load the parallel data.

With an additional thing, whenever clear is equal to 0; the flip flop will be clear to 0 0 0 0. With this, this is about the shift registers. So, I will discuss about this test bench how this can be run through this simulator; how to write the test benches and all, we will discuss in the next class.

Thank you.