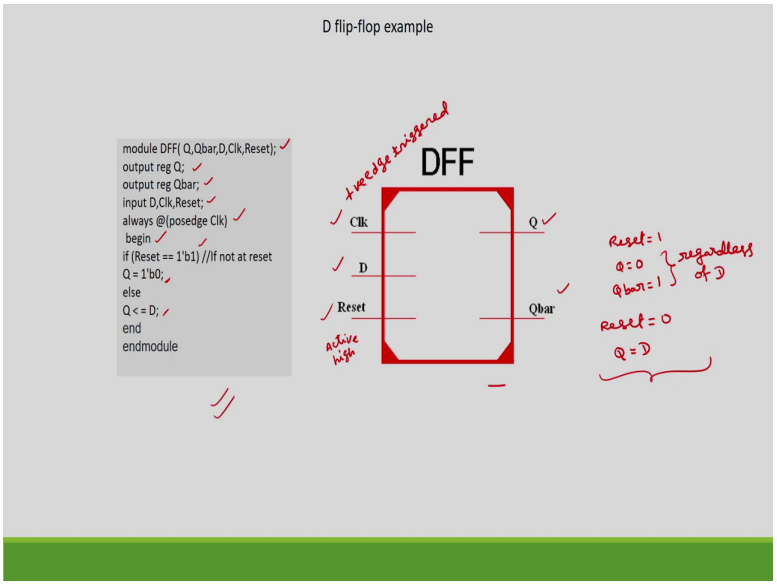
System Design Through VERILOG
Prof. Shaik Rafi Ahmed
Department of Electrical and Electronics Engineering
Indian Institute of Technology, Guwahati

Test benches
Lecture - 22
Sequential circuit examples

(Refer Slide Time: 00:31)



In the last lecture, we have discussed about the test benches of two different combinational circuits. So, I have given a simple AND gate and then, multiplexer. Now, we will discuss about the test benches of sequential circuits, first I will start with D type flip flop.

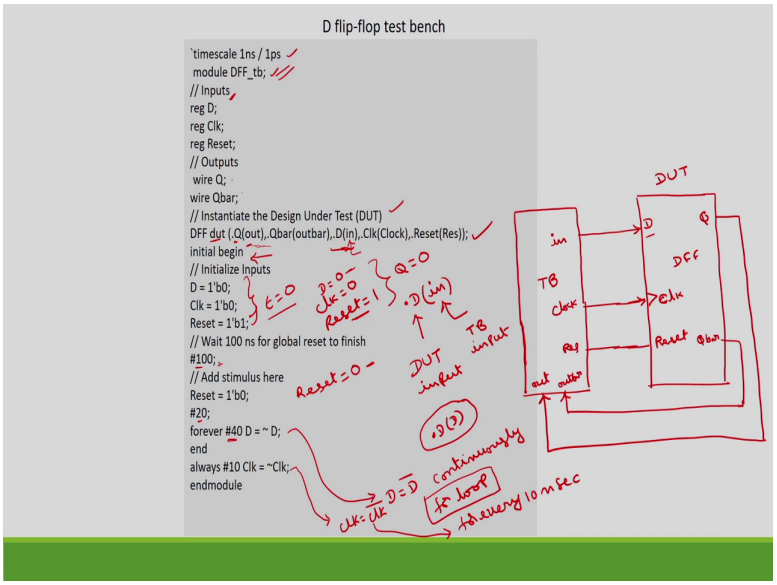
So, I am considering this D type flip flop. So, this is having three inputs, D is the input, Clock and then Reset also then, we have two outputs Q and Qbar. So, I am using this Reset as an active high signal, but in most of the practical ICs, active low signals will be there, but here for the sake of simplicity, I assume that this is active high signal.

So, the operation is something like if Reset is equal to 1 regardless of the D, output of Q becomes 0 and Qbar becomes 1, this is regardless of D. To recognize the D, Reset you have to make as 0. If Reset is 0, then output Q will be simply equivalent to D because we know that for D flip flop, output is simply equal to the input D. And we are assuming this clock as positive edge triggered.

So, if you consider the Verilog code for this D flip flop, starting with this module D flip flop, you have given the inputs and outputs Q, Qbar, D, Clock, Reset, then we have defined this output and as well as register. So, as a Q, as I have told for the sequential circuits normally, the output will be defined as a register.

Similarly, Qbar, if we want to store Q and Qbar, similarly the inputs are D, Clock, Reset, then always at positive edge of Clock begin if Reset is equal to 1. Then what happens? Q will make as 0 else Q is equal to D and end module. So, whatever the operation that I have explained here, same thing is written here in Verilog code.

(Refer Slide Time: 03:18)



Now, how to write the test bench for this D flip flop? This is the test bench for the D flip flop. So, we will define the time scale as 1 nanosecond or 1 picosecond. So, module name we have given D flip flop tb and we know that this test bench does not have any port declarations, then the inputs will be represented by the registers. So, the inputs are D, Clock and Reset, three will be defined as registers and outputs by wires so, we have two outputs Q and Qbar, this is the first step.

So, first step is actually we have to do this module declaration and then followed by registers and wires, the next step is instantiation. We have to instantiate the device under test. So, we are using the test bench here I am using the different notations for the test bench and design

under test. If I assume that this is the test bench, this is design under test which is D flip flop; simply D flip flop.

So, we have one input D, one output Q, Qbar, Clock, I have defined the clock as Clk, then we have reset we have defined as Reset. Now, I can give the same names in the test bench also or we can give different names also. So, in the previous examples, I have explained with the same name. So, here, I will use different names so that you can be familiar with this process also.

So, what I will do is using test bench, I am going to generate some input I will call this as in, this I will apply to D and then, that is why here this you can see here. So, this dot D represents DUT input, because within the parentheses this in represents the test bench input.

If I use the same notation for this also instead of in if I use D here, then you can write dot D as dot D dot D. So, this type of notation we have used in the earlier examples, here I will use different thing. So, the meaning of this one is; so, inside this in is the input of tb which is connected to the D input of the flip flop. Similarly, I have used this Q for out. So, this Q will be connected to the out of this signal and this I will call as out bar; out bar is Qbar.

Then, Clock I have connected to the clock, this I am calling as Clock whereas, this is Clk so I will connect two together. And then Reset, I am calling as here only Res whereas, for this design under test we call as Reset. This is how the connections, this connections will be explained by this instantiate the device under test format.

So, this is name I have given for this D flip flop DUT, and then I have given the connections between test bench and DUT, then initial begin. So, we will initialize the input D initial I will make as 0, Clock I will make a 0. So, this initial after this initial whatever this statement this will be executed, this will be executed at t is equal to 0. At t is equal to 0, we are making D is equal to 0, Clock is equal to 0, Reset is equal to 1.

After hundred nanoseconds, then we are making Reset is equal to 0, after another 20 seconds forever we are making D is equal to D bar, this is complement. So, this forever, so as the name implies this will assign this value D is equal to D bar continuously. Whereas, in case of for loop and all so until a condition is satisfied, the value will be assigned whereas, in case of

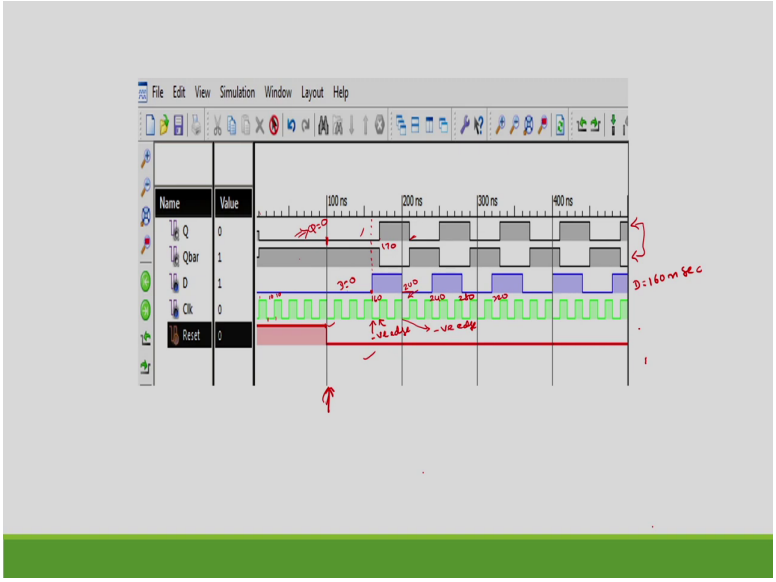
forever, it will assign continuously, it will go up to infinity. In for loop, this assign statement will be over only a finite time whereas, in case of forever, it will be up to infinite.

So, initially t is equal to 0, these are the values after 100 nanoseconds, we are making reset is equal to 0 so that it will recognize the D input. Here for this, whatever this D, Q becomes simply 0. Here Q depends upon D. So, what is the D we have given?

Initial value of D was 0 and for every 20 plus 40 total 60 nanoseconds and here is 100 is there; total 140 nanoseconds so, we will get D is equal to D bar. Then, always for every 10 nanoseconds, the Clock will be complemented, meaning of this one is clock becomes Clock bar for every 10 nanoseconds.

These things you can see in the test bench waveforms. You can see that the Reset, Reset was initially at t is equal to 0, Reset was 1. After 100 nanoseconds, Reset was 0 that you can easily see here.

(Refer Slide Time: 10:45)



So, up to 100 nanoseconds, Reset was high, then Reset is 0. When reset is high, what will be the output? As I have told output is equal to 0 regardless of the D, Q is equal to 0 mean Qbar is equal to 1. So, these two are complements to one and the other, this is Q and Qbar. So, it is about the reset.

Coming for the next signals. So, what are the next signals? So, we have initially D is equal to 0, initially D is equal to 0. After 100 nanoseconds and then again, this is 100 plus 20 plus 40 total 160 nanoseconds, D becomes D bar. You can see here that this is 160, this is 100, this is 150, 160. At 160, you can see that D is equal to 1 because D is equal to D bar, initial value of D is 0 so that it will becomes 1 and that complement operation will take place at a every 40 nanoseconds.

So, initially, this D was 0, after 160 nanoseconds, D becomes 1 and this will change at every 40 seconds. So, this is 160; 160, this is 200, this is 240, 280 for every 40 nanoseconds, the D value will be complemented 320 and so on. That is about the D waveform.

Now, coming for this Clock. Clock is basically for every 10 nanoseconds; Clock becomes Clock bar. So, initial value of clock is 0. So, after 10 nanoseconds, the Clock becomes Clock bar. You can see that initial value is 0. At 10 nanoseconds, this Clock is 1 and then, every 10 nanoseconds this becomes this duration is 10 nanoseconds, this duration is 10 nanoseconds. For every 10 nanoseconds, the Clock will change.

Now, coming for this Q and Qbar; I will just give only the Q, Qbar is simply the complement of Q. So, as long as this Reset is equal to high, Q becomes 0; Q becomes 0 here. Then, when Reset is equal to 0, it depends upon the D and Clock. So, D was 0 from here to here so, output Q is also 0 because simply it will follow this at every positive edge of the Clock.

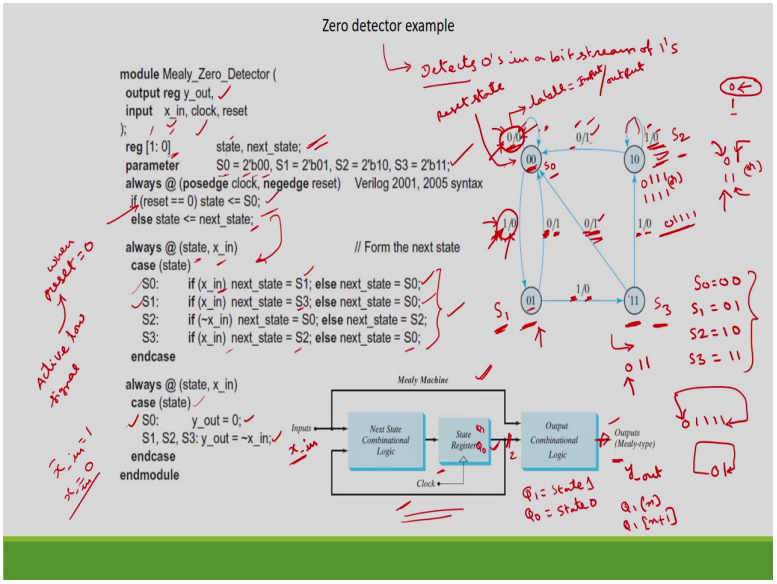
Now, actually here this D becomes 1, D is equal to 1 at 160 nanoseconds. At 160 nanoseconds actually, this is the negative edge of Clock, but the flip flop that we have considered is positive edge.

So, at negative edge even though D is equal to 1 that change will not be recognized, the D is equal to 1 will be recognized by the flip flop at the next positive edge of that one, that is the next positive edge is here which is at 170. At 170 onwards this is 170, this is 170, so this Q becomes high.

So, Q becomes high. So, here again at 200 nanoseconds, even though D becomes 0, but because this is also another negative edge, this change will not be affected, this change will be affected in the next coming positive edge here that is why from here onwards, output Q

becomes 0. So, this is how we can explain this timing waveforms. This is about the test bench for the D flip flop.

(Refer Slide Time: 15:27)



Now, we will consider the test bench of a sequential circuit which is called as zero detector. As the name implies, this detects the 0's in a stream of 1's. This will detect 0's in a bit stream of 1's. We can easily design the circuit corresponding to this 0 detector, but here we are not considering the circuit diagram because we know that we can directly write the Verilog code if I know the state diagram.

From this state diagram, I can obtain the logic diagram also, we can write the Verilog code from the logic diagram as well as from the state diagram also. So, here, I will write using state diagram because this is a simple process.

So, what is the state diagram for the circuit which detects 0's in a bit stream? So, here I will take four states and I will take Mealy model. So, in case of Mealy model, output depends upon the input also in addition to the states; output is function of the input as well as the states of the state register. So initially, I am assuming that S0 state is 00, S1 state is 01, S2 state is 10, S3 state is 11. So, I will start with this S0 state.

So, the first bit that will be received can be either 0 or 1. If the first bit received is 0 so it simply stays, so this is the label, this label is actually input by output, the format of this one is input by output. The numerator value is the input, denominator value is the output.

So, whenever it receives the input of 0, output also becomes 0 because this problem has to detect the 0's in a bit stream of 1's. The first bit itself is 0 so, there is no 1 to find out the 0's in the bit stream of 1's.

So, that is why what happens is it will stay in the same state until it receives 1, this will keep on going into the same S0 state until it receives the first bit as 1. Once if it receives a first bit as 1; this is the state where it has received the input bit as 1 so then it will go to the next state. Because it has not detected the first bit is itself is 1; so I am going to have a bit stream.

So, first bit can be 0 or 1. In case of 0, the problem is not at all started so, it will keep on staying in the same state. When the first bit is 1 so, there is the series has been started so bit stream of 1's it has begun. So, then I will check for the second bit. But the output is 0 because it has not detected 0's because the first bit itself is 1.

Now, at this stage second bit, second bit can be 0 or 1; first bit is 1, second bit can be either 0 or 1. So, either this possibility or this possibility. This is second bit that we are going to check here. So, if the second bit is 0, what happens? Second bit is 0 this means in a bit stream, already 1 has been started, second bits 1 has been started, second bit is 0 means 0 has been detected in the bit stream.

So, this will go to this S0 state because this is already detected, I want a bit stream of all 1's only. So, whenever any 0 comes, I will again go to the starting of this; I will go on counting from the starting.

So, here in this case after 1, the 0 has been detected so, it will go to the starting. So, 0 has been detected, it will go to the first state is 0, but because 0 has been detected, output is 1. If the second bit is also 1; second bit is also 1, output is 0 because 0 has not been detected, then it will check for the third bit in the next state, ok. This is S2 is 10; so, it will check in the next state.

Here, it can receive 0 or 1. If it receives 0 means already at this stage, already the first two bits are 11. If the third bit is 0, this is detection of the 0 so, it will go to the initial state, but it will give the output as 1 because already a 0 has been detected in a bit stream of 1's. If not, if third bit is also 1, then it will go to the next state here, all the three bits are 1's.

So, at this stage, the fourth bit can be 0 or fourth bit can be 1, there are two possibilities here. If fourth bit is 0, then it will go to the initial state, S0 state and it will produce the output as 1 otherwise, it will stay in the same loop because all are 1's only. So, it will stay in this loop until a 0 is detected as the input.

This is the state diagram of a zero detector. We can design the circuit also, but here I will write the Verilog code; here I have written the code using the states itself, state diagram.

So, what is the code corresponding to this one? So, module I am giving the name as Mealy Zero Detector and output I am calling as this output finally, output here we have input and output; four states, one input and one output. Here, this is one input, we are calling this input as x in, this output we are calling as y out, this is y out. So, we have defined as output as well as register, this input is x in is applied here and clock is applied here so, clock.

And we can have a reset signal also, here I have not shown here, which will reset all the state values to 0000. See here we have a four the states of this flip flop will be 0000. So, here, this will be having 2-bits because corresponding to this only 2-bits are there. If I call this one as Q 1, Q 0 this is a 2-bit value, this is 2-bit signal.

Then, the state because 2-bits as I have told this is Q 1, Q 0 type of thing, but I am calling as state, the present state and next state. So, this is state 1, state 0 like Q 1, Q0. Q 1 you are calling as state 1, Q 0 we are calling as state 0. This is vector; so, state 1 and state 0, this is similar to the Q 2, Q 1 if I design the circuit using flip flops.

Similarly, we will be having the next stage also same Q 1, Q 0, the only difference is normally we will call as Q 1 of n is the present state, next state is Q 1 of n plus 1. Similarly, Q 0 of n and Q 0 of n plus 1, ok. So, the next state also we have 2-bits because each state you are representing with 2-bits. Then, parameter I have explained this keyword, parameter can

be used for assigning some binary values to the states or any variable. So, S0 this is 2-bit binary 00, this is what I have defined here, this can be done by using this parameter keyword.

Now, initially we are checking for the reset. So, if reset similar to the previous D flip flop where if reset is equal to 1, regardless of the input the flip flop states will be 01. Similarly, here also we will check so, if reset is equal to 0, if this is true, then the reset state is this S0 state this S0 state we are defining as a reset state.

This I have explained while explaining about the sequential circuits behavioral modeling. So, whenever reset is equal to 0, state becomes reset is equal to 0 is true, this is active low signal. When reset is equal to 0, if this is true, then state will be S0 this reset is actually in fact this is active low signal. But in the previous D flip flop example I have assumed active high, this is actually low signal because when reset is equal to 0, it is resetting; else state is equal to next state, it will remain in the same state.

Now, when does that state changes? Now, it will depend upon the previous state and input, ok. Once it come out of the loop as long as reset is equal to 0, then it will be S0 only. So, when does it change from the S0 to other states? That depends upon the present state and the input that is applied, ok.

So, there is a case statement, here you can see that here, when x in, when this is true x in is equal to 1, what is the next state for S0? For S0, this is S0, when the input x in is equal to 1, x in is equal to 1 this, next state is S1, this is S1 state. This is S0, this is S1, this is S2, this is S3. So, when input is equal to 1 next state is S1 this, else 0 it will stay in the S0 itself.

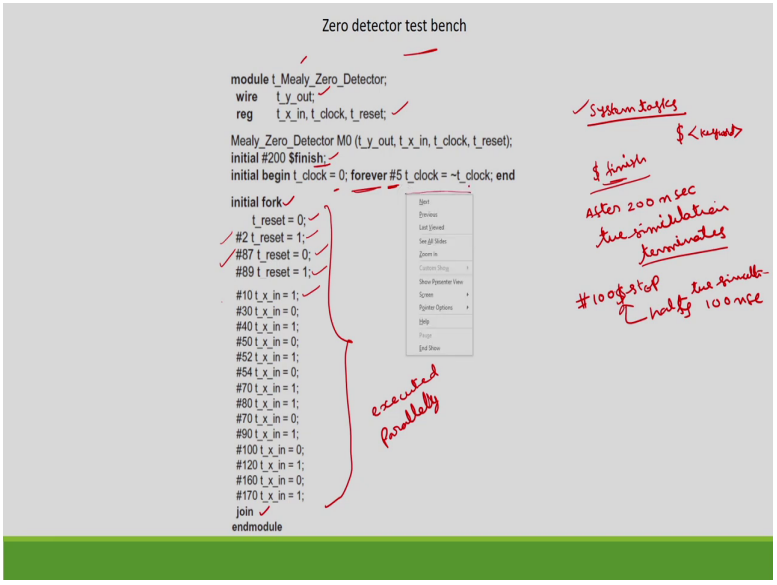
Similarly, for S1 state, if input is 1, it will go to S3 state. If input is 0, it will go to S0 state. Similarly, for S2? For S2 input is 0 it will go to S0 state, but here I have taken as x bar so when x bar is true means; when x bar of in is true means 1 means x is 0, x in is 0. So, when x in is 0, if this is true, S0 for S2. For S2, if x in is 0; 0 it will go to S0. False it will stay in the S2 itself.

For S3 state, for S3 state is 0 means it will go to 1; if this is true, 1 means this will go to S2 state, 0 means it will go to S0 state. So, whatever the information present in this state table, we can explain using these four statements, then end case.

Then, always what about the output? This is regarding the states. So, for this particular Mealy model, we have the states as well as the output. Output is of course only single output. So, what about the output states we have defined? Output is again we will check the state, for S0 we can see that this S0 y out is always 0. Whether the input is 0 or 1, you can see that for S0 if input is 0 output is also 0, if input is 1 output is also 0. So, in any case in S0 state output is equal to 0.

Whereas in the other states you can see that in S1, when input is 0, output is 1. When input is 1, output is 0. Similarly, for S2, when input is 1, output is 0. When input is 0, output is 1. Similarly, for S3, when input is 1, output is 0. When input is 0, output is 1; so, simply the complement of x in. So, end case, end module. So, this is about the Verilog code for the zero detector circuit.

(Refer Slide Time: 29:31)



Then, we have the test bench corresponding to this. So, this is the test bench. So, here, we have taken the outputs as wire, inputs as the register as usual. Then here, you have used some extra commands like keywords like finish. So, there are some system tasks in Verilog out of which one is this. System task, the general format of the system task is dollar followed by keyword.

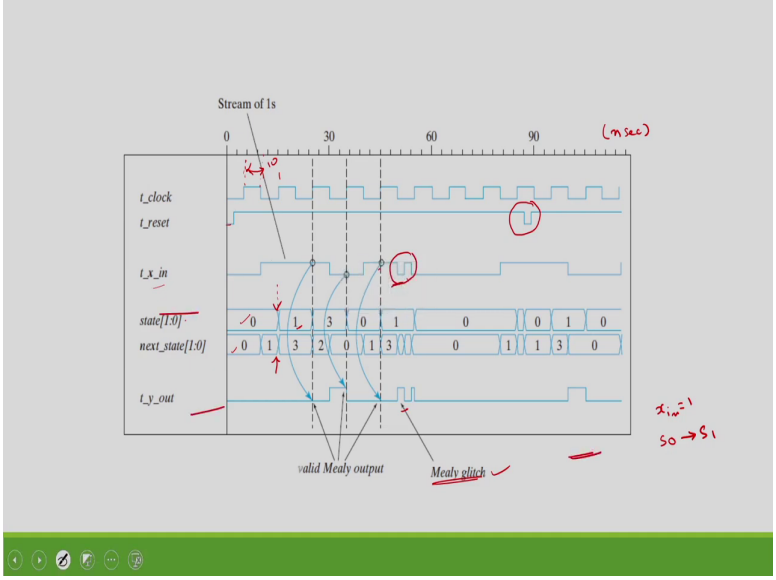
So, one of such system task is one is finish. As the name implies, this will finish the simulation after 200 nanoseconds. So that we can define here the nanoseconds or seconds whatever it may be the value; so, after so many nanoseconds or seconds, it will finish the simulation. So, means simulation will run for only 200 time scales. So, after 200 say nanoseconds, the simulation will terminate.

Termination is completion of this one, sometimes you can halt the program after this, after some time. Suppose if I write something like 100 stop; dollar stop so, this stop is also another system task. So, the difference between this finish and stop is finish will terminate this one whereas, this one will halt after 100 nanoseconds; halts the simulation after that if you want you can continue.

And then again we have fork and join. So, whatever the statements between this fork and join, they will be executed parallelly. So, all these timing of definitions so, we are defining this reset, x in, x out; so, how does this change with respect to time? So, this all this initialization will be executed parallelly; then forever also I have explained. So, there are some more system tags are there, we will discuss this later.

So, coming for this again this test bench of this zero detector. So, reset is initially 0, then reset after 2 milliseconds is 1, after 87 milliseconds is 0, 87 plus 2 is 89 and 89 plus 89 this is 1 ok. So, after 10 nanoseconds, t input is 1. So, like this we can easily see this in the waveforms.

(Refer Slide Time: 33:18)



So, you can see that this is the clock, I have defined the clock as this clock is 0 and for every 5 milliseconds or nanoseconds, t clock is equal to complement of that one. This I have used in the previous example also the same logic, here also I have not given the any time scale, this can be nanoseconds or seconds ok.

So, for every 10 nanoseconds, this gap is 10 nanoseconds the clock will flops; clock will change the state. This again 10, this is again 10 and so on. We have given reset at different times. I would have given this high reset because low reset is corresponding to output is 0 regardless of the states are 0, it will stay in the 0 state regardless of the inputs.

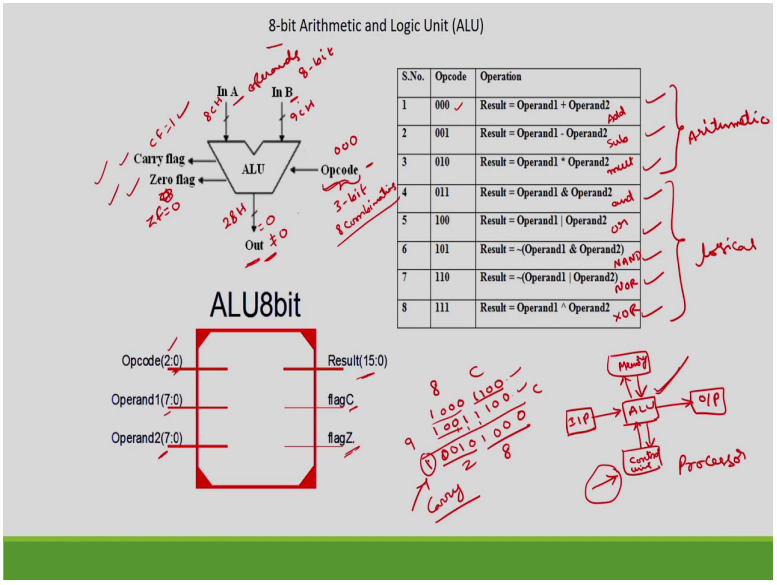
So, when reset is equal to 1, then the output depends upon the present state and the input. And then, input also I have defined in this manner; so, this input is different scales we have given, g 1, 0, 1, 0, 1, 0 values, this is 1, 0, 1, 0, 1, 0. So, this is the present state, next state. At this particular point, x in is equal to 1 and present state is changing from 0 to 1, S0 to S1.

So, what about the next state? Next state will change from 1 to 3 that is you can easily see from here. When x is equal to 1, S0 changes from S0 to S1, you can easily see that here. So, whenever x is equal to 1, S0 to S1 change. S is equal to 1, S0 to S1 when x is equal to 1, S0, when x is equal to 1, this is going to S1; this is going to S1. So, the output is actually this depends upon the input and states.

So, the problem with this type of Mealy model is; the problem with this one is the output also depends upon the present input. So, if any change occurs in the input, that change will be reflected in the output that is what we have explained here. So, input there are some glitches here so, because of that output also will change. This is undesired one, actual signal is this; this is undesired input changes due to some disturbances.

So, whatever the disturbance that causes in the input that will be reflected in the output, this is what is called the glitch; this is undesired effect. This is the drawback of the Mealy model. So, normally, you will prefer Moore model, but the advantage of the Mealy model is the circuit complexity will be less. So, you can easily verify these waveforms with respect to the code that is we have given in the previous slide.

(Refer Slide Time: 36:20)



So, the next example is; so, with this introduction to this combinational sequential circuits we have written the test benches for AND gate, multiplexer and then, for D flip flop and then, zero detector. Now, I will take some case studies. So, the first case study is ALU. So, in the starting of this course itself as I have told so, after this course, you should be able to design a processor, ok. One of the main block in the processor is an arithmetic logic unit.

So, we know that there is input block in the processor, if I take any microprocessor or DSP processor, there will be input block, arithmetic logic unit, control unit, then output block, then

there will be some memory. So, this ALU is the main important block of any processor. Here, I will discuss design of a simple ALU and control unit basically this consisting of the flip flops, decoders, multiplexers, this also we have already discussed in the earlier lectures.

So, here I will take a simple ALU, so this ALU varies from the processor to processor. So, to explain this Verilog coding, I will start with a very simple ALU with limited functions ok. So, if we take nowadays, RISC and CISC processors we have lot of functions but here I will consider only a simple ALU 8-bit ALU with eight functions, so we have eight different functions. As the name implies, this will perform arithmetic logical operations. Here, I have taken three arithmetic operations and five logical operations. So, these are three mathematic operations, and these are all logical operations.

So, what are these arithmetic operations? First one is addition, this is subtraction, and this is multiplication. And what are the logical operations? This is AND, this is OR, this is NAND, this is NOR and this is exclusive OR. So, I am designing a 8-bit ALU which can perform these eight operations. Among the eight operations, three are arithmetic operations, five are logical operations. Even you can incorporate division also, if we want to design a complex ALU.

Like in case of 8085 microprocessor, we do not have the divider or multipliers inside whereas, in 8086 multiply dividers are there. So, like that depends upon the complexity of the CPU, you can have more functions in the ALU.

Here, there are two operands, this data input data circuit operands also, they are 8-bit operands. So, we have to perform some function that will be decided by this opcode. So, we know that if you write this microprocessor program, there will be opcode which is going to decide the operation.

We have the eight options of this one, this is 3-bit opcode so that we can have 8 combinations. If we want to increase the functionality of this ALU's, you will have to use more number of the bits for the opcode so, 8-bits. Then, we are using two flags also, carry flag and zero flag. So, carry flag will be set whenever there is a carry out of the MSB bit.

Zero flag will be set if the result of operation is 0. If this result is 0, this zero flag will be set. If this result is not equal to 0, this will be reset.

Carry flag is if you want to add say for example, 10001100 plus 10011100 so, two 8-bit numbers if you want to add, this is 00 1 0 11 10 1010 so, there is a carry. If the result is more than 8-bits, then in that case carry flag will be set otherwise, it will be reset. So, the result will be here. So, you give the two inputs here and you find out which operation is to be performed, then accordingly, you will get this.

Say for example, if you want to take this same thing; so, what are these two numbers? This is 8 in hexadecimal, this is C 12, and this is 9, this is C so, 8CH you give here; there are total 8-bits, I am representing hexadecimal form and 9CH, you are applying here. Suppose if I give you these 3-bits as 000, I want to perform the addition operation, then what will be the result? This is 2, this is 8. So, you will get here the result has 28H, but the carry flag will be set, zero flag will be reset.

So, like that depends upon this operation, the output and the status of the flags will be different. So, same thing I have explained in this block. So, opcode is 3-bit 2 to 0, vector 3-bit and operand is 8-bit, operand 2 is also 8-bit result is 16-bit because if I multiply two 8-bit numbers, result can be 16-bit. Then this is a single bit flag, single bit carry and zero flag.

(Refer Slide Time: 42:55)

```
module ALU8bit( Opcode,
Operand1,Operand2,Result,
flagC,flagZ);
input [2:0] Opcode;
input [7:0] Operand1,
Operand2;
output reg [15:0] Result =
16'b0;
output reg flagC = 1'b0,
flagZ = 1'b0;
parameter [2:0] ADD = 3'b000,
SUB = 3'b001,
MUL = 3'b010,
AND = 3'b011,
OR = 3'b100,
NAND = 3'b101,
NOR = 3'b110,
XOR = 3'b111;
```

```
always @ (Opcode or Operand1 or
Operand2)
begin
case (Opcode)
ADD: begin
Result = Operand1 + Operand2;
flagC = Result[8];
flagZ = (Result == 16'b0);
end
SUB: begin
Result = Operand1 - Operand2;
flagC = Result[8];
flagZ = (Result == 16'b0);
end
MUL: begin
Result = Operand1 * Operand2;
flagZ = (Result == 16'b0);
end
AND: begin
Result = Operand1 & Operand2;
flagZ = (Result == 16'b0);
end
```

So, this is the Verilog code corresponding to this ALU. So, even though this is lengthy code, but the operations are straight forward. So, here we have defined this opcode which is a 3-bit number, two operands, result is the output, then flag, flag two, these are the three outputs.

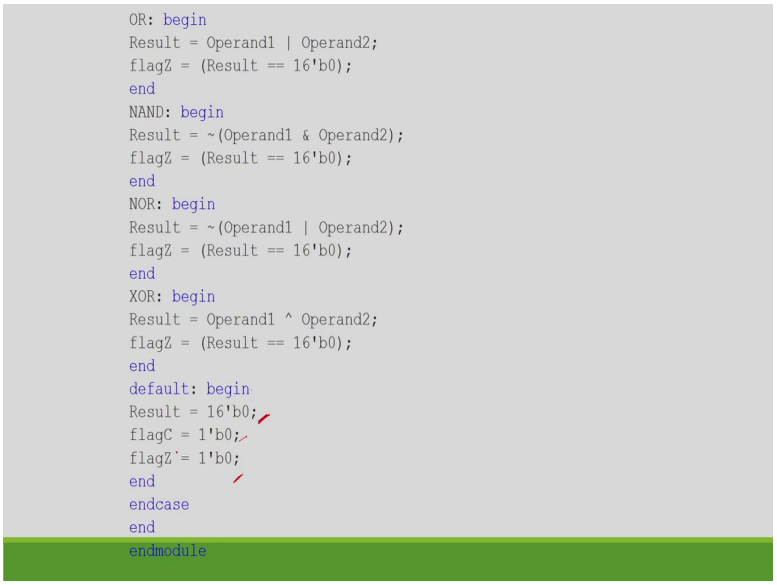
So, the input is opcode, operand1, operand2 are the inputs, this is 3-bit, these two are 8-bits and output and register are result is 16-bit and flag is single bit initially I am setting this flag to 0, initially I am setting zero flag also to the 0.

Then, parameter ADD, for ADD actually, this is 000 in the table, I have explained. 000 is corresponding to add operation, 001 is correspond to subtraction like that. So, subtraction is 001, multiplication is 010 and so on. Then, always at opcode or operand 1 or 2, so we will define the case. If opcode is 001, then ADD operation. For the second option which is 001, subtraction otherwise multiplication, then AND operation, all those things. So, the result is what here? This only addition.

So, and what happens to this result of this one? Result is 8th bit. As I have told this is if I add this is A 7, A 6 so on up to A 1, A 0 is the first operand, B 7, B 6 so on up to B 1, B 0 is the second operand, if I add this one, this is 7th bit and if you get 8th bit which is the carry, then that result is set to carry.

If the 8th bit is 0, carry flag is reset, 8th bit is 1, carry flag is set. And flag 0 is nothing but if the result is all 0's only 16-bit because the output is 16-bit result, if all 0's, then only the flag will. If this is true; if this is true, zero flag is set otherwise, zero flag is reset.

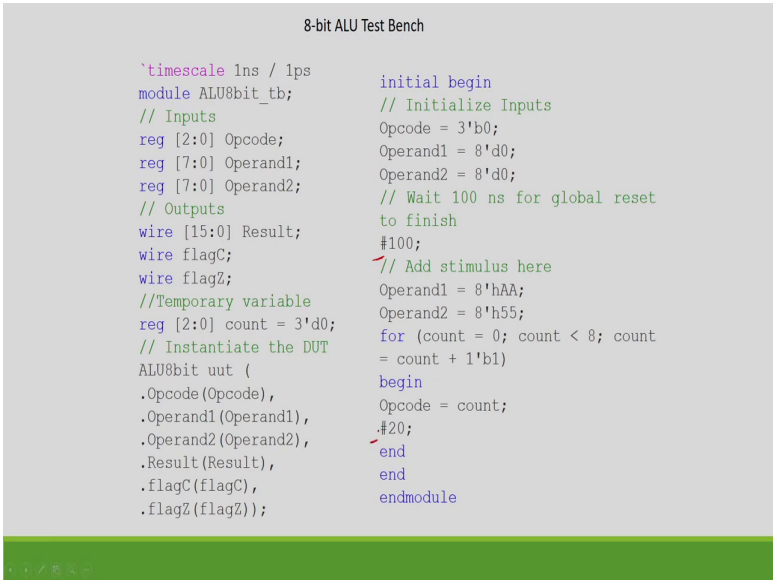
(Refer Slide Time: 45:28)



```
OR: begin
Result = Operand1 | Operand2;
flagZ = (Result == 16'b0);
end
NAND: begin
Result = ~(Operand1 & Operand2);
flagZ = (Result == 16'b0);
end
NOR: begin
Result = ~(Operand1 | Operand2);
flagZ = (Result == 16'b0);
end
XOR: begin
Result = Operand1 ^ Operand2;
flagZ = (Result == 16'b0);
end
default: begin
Result = 16'b0;
flagC = 1'b0;
flagZ = 1'b0;
end
endcase
end
endmodule
```

So, similar type of explanation is therefore, subtraction, multiplication, AND operation, then we have NAND operation, NOR, exclusive OR. So, finally, we will have result as initially, I will begin with by default, this result will be 0, this sets I mean both carry and zero flags will be reset.

(Refer Slide Time: 45:50)

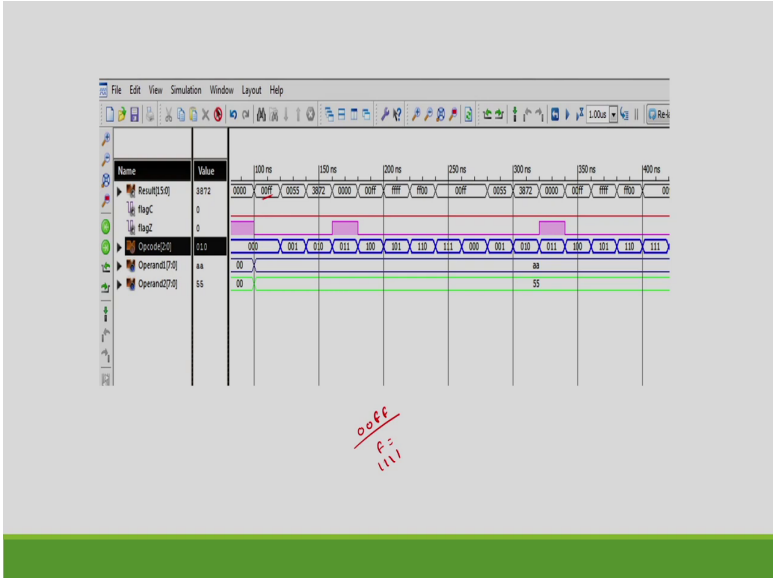


8-bit ALU Test Bench

```
`timescale 1ns / 1ps
module ALU8bit_tb;
// Inputs
reg [2:0] Opcode;
reg [7:0] Operand1;
reg [7:0] Operand2;
// Outputs
wire [15:0] Result;
wire flagC;
wire flagZ;
//Temporary variable
reg [2:0] count = 3'd0;
// Instantiate the DUT
ALU8bit uut (
.Opcode(Opcode),
.Operand1(Operand1),
.Operand2(Operand2),
.Result(Result),
.flagC(flagC),
.flagZ(flagZ));
initial begin
// Initialize Inputs
Opcode = 3'b0;
Operand1 = 8'd0;
Operand2 = 8'd0;
// Wait 100 ns for global reset
to finish
#100;
// Add stimulus here
Operand1 = 8'hAA;
Operand2 = 8'h55;
for (count = 0; count < 8; count
= count + 1'b1)
begin
Opcode = count;
#20;
end
end
endmodule
```

And corresponding test bench is also written, this is also self-explanatory so, you can go through this. Here, I have given the different time scales.

(Refer Slide Time: 46:05)



And if you run this simulation; so, this is the simulation, you can check the results also for different operands and different opcodes, what is the 16-bit result? This 16-bit really stored in the hexadecimal form. So, 00FF means F is 15, 1111. So, here all these values are given in hexadecimal so we can easily verify this results.

So, this is about one of this the case study. So, I have given a simple ALU so that based on this ALU using some control logic, you can directly design the CPU also. So, in the next lectures, we will take some more case studies.

Thank you.