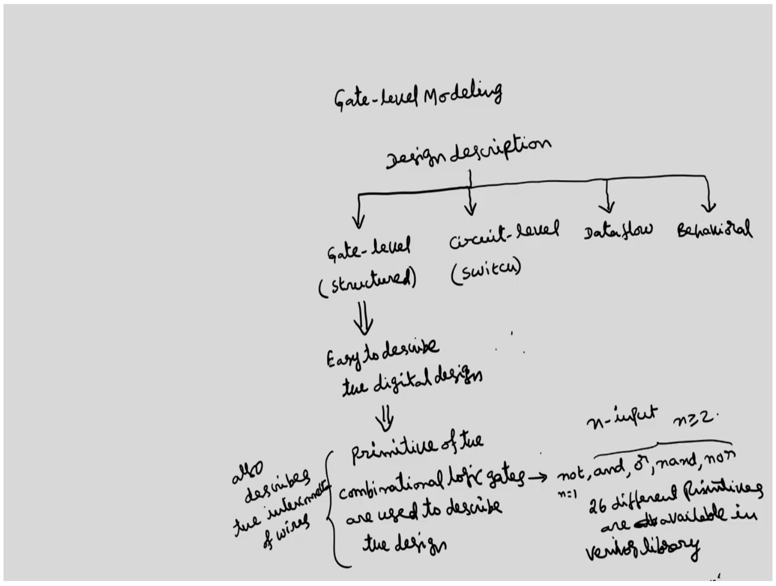
System Design Through VERILOG

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

Introduction to Verilog
Lecture - 03
Basics of gate level modeling

(Refer Slide Time: 00:34)


Gate-level Modeling
Design description
Gate-level (structured)
Circuit-level (switch)
Dataflow
Behavioral
Easy to describe the digital design
Primitive of the combinational logic gates are used to describe the design
also describes the interconnect of wires
n-input n≥2
not, and, or, nand, nor
n=1
26 different Primitives are available in verilog library

Ok. Today, we will discuss about the gate level modeling. As we know that there are four levels of design description, we take the design description. This can be done by using four ways. So, one is gate level – this also is called as a structured level. Then we have the circuit level – this also is called as switch level. And we have the data flow level, and behavioral level.

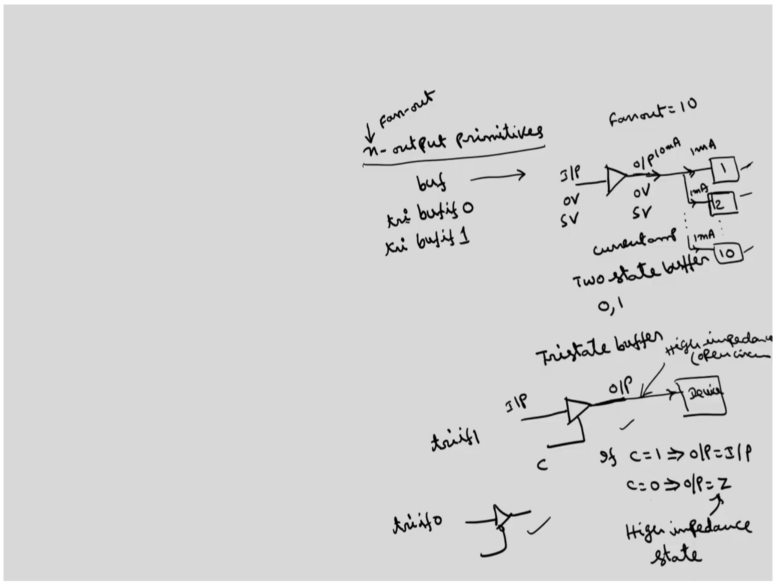
So, out of these four, so this gate level description is gradually easier to describe the design, basically all will be used to describe the digital designs. So, first we will start with this gate level modeling. So, in gate level modeling, so this basically uses the primitives of all the logic gates, primitives of all the combination logic gates are used to describe the design.

So, what are the basic combinational logic gates? Such as NOT gate we have primitive NOT. We have AND gate and, we have OR gate or, NAND gate nand, NOR gate nor. So, there are already available in the VLSI, Verilog library.

So, there are nearly 26 different primitives are available in Verilog library. So, this not only uses this logic gates, but also in addition to these this also describes the interconnections of the wires. So, if the different, I mean primitives of Verilog OR, NOT, AND, OR, NAND, these are all n input primitives.

So, here n varies from 1 to any value. If I take NOT gate, n is equal to 1, for AND gate, you can have any value of n ok for OR gate also, NAND gate, NOR gate, so n is n can be greater than or equal to 2. So, minimum two inputs are required, this is greater than or equal to 2.

(Refer Slide Time: 04:52)



So, there are some primitives which will be having multiple outputs, n output primitives such as you have buffer buf. And you have bufferif0, bufferif1. So, this can have the multiple outputs.

If you take a simple buffer, the symbol of buffer is same as the NOT gate or inverter except for the bubble at the output side. So, this will have some input, some output. In terms of the voltages output is equal to input. If you give the logic 0, 0 volts, output is also 0 volts. If you give 5 volts as the input, output is also 5 volts.

Then what is the need of the buffer? So, buffer is basically this will act as a current amplifier. So, if you want to drive the large current capacity loads, so normally we will use buffers. So, this is just two state buffers, you can call this one as a two-state buffer it can take only two states. So, states are 0 or 1.

So, there are some buffers which are called as tri-state. This is tribufferif 0, tribufferif 1. So, the difference between the tri state buffer and a two-state buffer is as the name implies in tri state buffer, we have three states. So, this is a buffer. In addition to this, we have one control signal. This is control signal; this is input, and output.

So, the operation is if c is equal to 1 implies output is input. If c is equal to 0, output will become state called high impedance state Z. Z is high impedance state that means if I want to connect this to some device, this will offer a high impedance path; the resistance of this wire will become high impedance.

As a result of that, this will act something like an open circuit, almost like open circuit, so that this does not draw any current. As a result of that, we can increase the current capability of the output of this particular gate. So, this is if c is equal to 1, output is input; c is equal to 0 output is Z.

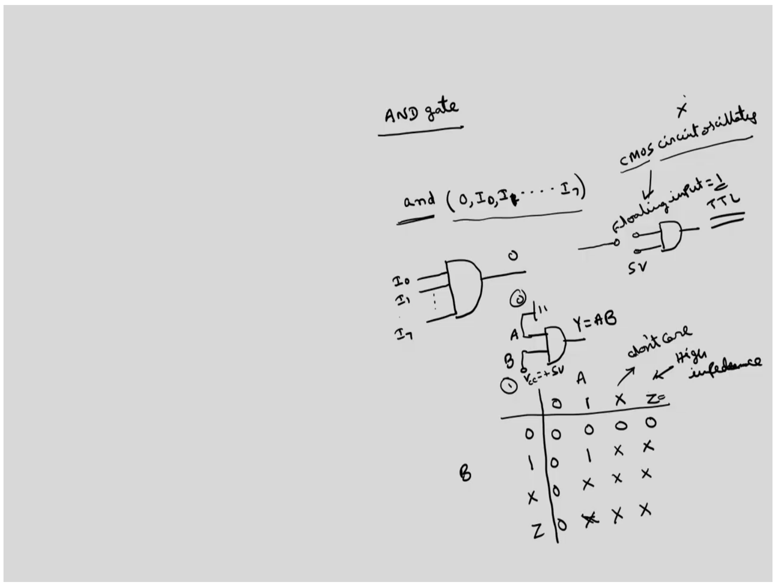
This type of buffer is called tri buffer, triif1. If I place a bubble here means the operation is reverse to this one. If c is equal to 1, high impedance state; c is equal to 0, output is equal to input. This type of buffer is called triif0 buffer ok. So, whether you take a two-state buffer or tri state buffer, they can have multiple outputs, an output primitive.

How they can control this multiple output is? So, we can connect output of this particular gate. Suppose, if I take this output, this can be connected to many devices. This is device 1, this is device 2, v is going to control these number of devices that can be connected to a single output.

Suppose, if you have this is 10th device, this is what is called fan-out you might have studied in your digital logic course. So, if the current required to drive this one is say 1 milliamp, this is also 1 milliamp. If I assume that all belong to same family, this is 1 milliamp. If this output is capable of giving 10 milliamps of the current, then I can connect 10 of these devices which draws 1 milliamp of the current, then fan-out is called as 10.

So, depends upon the current capacity of these devices and the current that is supplied by the output of a gate, we can connect multiple outputs ok. So, this is what is called n output primitive. This n is nothing but the fan-out. So, like that we have 26 different primitives in Verilog library.

(Refer Slide Time: 10:18)



B \ A	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

So, coming for our basic circuits if I take a simple AND primitive AND gate because any digital circuit can be implemented by using these basic gates such as NOT, AND, OR. Using these three gates, we can construct the remaining gates such as NAND, NOR, exclusive OR, exclusive NOR and so on ok. So, the basic the primitive of this one is you have to use lowercase letter, Verilog is a case sensitive language.

So, you have to write only lowercase letters, and so within the bracket you have to represent first the output. So, output will be normally only one for the AND gate we can have any number of the inputs. Suppose, if you have 8 inputs I 1, I 2, say I 0, I 1 so on up to I 7. So, this will represent an AND gate whose output is 0 whose inputs are I 0, I 1 so on up to I 7.

So, instead of writing this, you can simply write using this primitive, and with this output followed by input ok.

So, now, if you take the two input AND gate say A and B are the inputs, Y is the output, Y is given by AB. So, here each value A can take four different values here. This is A, this is B. So, A can take either 0 or 1 or it can be X which is a do not care it can be Z which is high impedance state.

This is do not care; this is high impedance state. As I have already discussed what is meant by high impedance state means it basically draws almost 0 current. Similarly, B can take 0, 1, X and Z ok. So, what is the output value Y for different combinations? So, we know the ordinary combinations like 0 0, output is 0. If any, one input is 0, output will be 0.

So, if B is 0 regardless of A whether this is 0 or 1 or X or w, output is 0 ok. Similarly, when A is equal to 0 regardless of B, B is 0, 1, X or Z, output will be 0. So, this row and this column both are 0s. Now, for 1 1 output is 1; 1 X output is X. 1 Z, here there is a logic, 1 Z is also X. The reason for this one is supposing if I have an input which is floating I can connect this input to either ground, I can connect this input to VCC which is plus 5 volts ok.

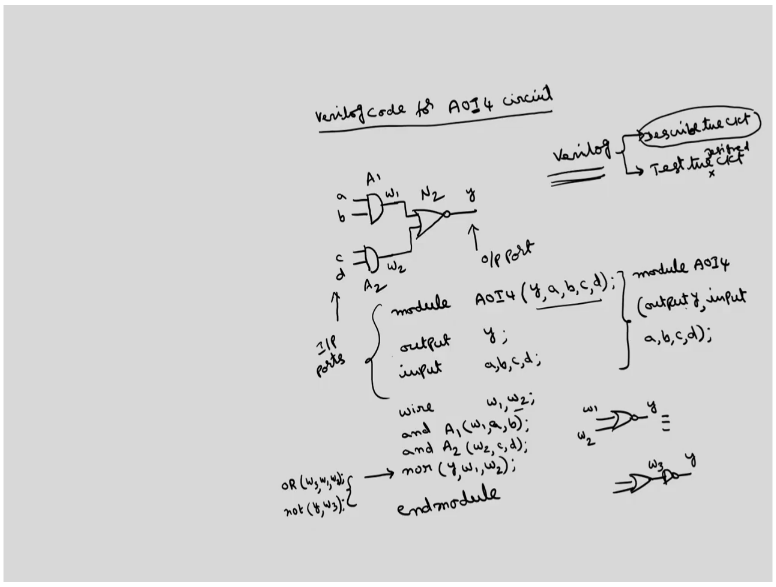
So, in this case, this is logic 0, this is logic 1. Suppose, if an input of the gate is floating, so I have connected here logic 1, 5 volts, this I have not neither I have connected this to be 5 volts nor I have connected this to ground this is floating. So, this is what the meaning of Z. Z is high impedance state; this is almost open circuit type of thing. So, it is coming from some other gate, but this is almost open circuit.

So, when a gate is floating, so normally NAND logic if input is floating, floating input will be taken as logic 1 in case of TTL gates – Transistor Transistor Logic. Whereas, the floating gate in CMOS logic is a do not care ok because this is going to I mean on both the NMOS and PMOS transistors thereby the circuit oscillates.

So, a floating input is there in CMOS, the CMOS causes CMOS circuit oscillates ok. In TTL it is really taken as 1, whereas in CMOS this will oscillate means do not care basically. So, if one input is 1, Z in case of this I mean TTL gate is 1, but normally it is CMOS gate because the circuit oscillates. So, it is fully taken as a do not care so that is why this will be do not care. If one input is 1, other input is Z, this will be taken as a do not care.

So, similarly, X 1 is X, X X is also X, X Z is also X. So, 0 Z 0 is 0, Z 1 is X as we have discussed earlier like here we have Z 1 is X. And then we have Z X is also X, Z Z is also X. So, this is I mean the logic operations of AND gate for all combinations of the inputs. Similarly, can you have for OR gate the other gates also NAND, NOR and remaining gates ok. So, now, using these basic gates, how to I mean write a Verilog code.

(Refer Slide Time: 16:08)


Verilog code for AOI4 circuit

```
module   AOI4 (y,a,b,c,d);
output   y;
input    a,b,c,d;
wire     w1,w2;
and A1(w1,a,b);
and A2 (w2,c,d);
nor (y,w1,w2);
endmodule
```

```
module AOI4
(output y, input
a,b,c,d);
```

```
OR (w3,w1,w2);
not (y,w3);
```

So, I will discuss a Verilog code for a simple circuit. For and or inverter 4 input circuit. This is and or inverter 4 input circuit. As we have discussed this Verilog has twofold operation, Verilog can be used for describing the hardware given a circuit you can describe by using the key words.

We can describe the circuit or we can test the designed circuit ok. First, I am going to explain with an example how this Verilog code can be used for describing a circuit ok. The circuit that I have taken is AOI4 means this is something like we have 4 inputs, 2 inputs for this AND gate, 2 input for this AND gate a b, c d, then we have OR gate and inverter. I am going to club these two together, so that this becomes an OR.

Basically, we have AND gate, OR gate, NOT gate inverter AOI. These 4 represents four inputs. Output let us call as Y. So, I want to describe this instead of drawing this diagram, I want to describe by using Verilog code ok. So, how to write the Verilog code? So, whatever the code that we have to write, we have to write between the module and end module.

So, the starting this one is module, we have to give some name. So, I am giving the name of the module as AOI4 itself. After reading the program, the last keyword is end module. So, in between this, so whatever the description that you have to write. So, AOI followed by here you have to specify the inputs and outputs of the circuit ok. So, we have one output and four inputs. First you have to specify output y, then the inputs a, b, c, d.

So, after that, we require a semicolon here. Then you describe what are these values output y semi colon input you have a, b, c, d semicolon. So, this instead of writing these three I mean key words, this can be described in a single keyword also like module is same, the name is also you can write the same name AOI4.

So, within this bracket, here itself you can write output y comma input. You have to give some tab. Here you have to give the gap after this output or after this input a comma b comma c comma d. So, this single keyword is equivalent to these three. So, you can write anyway ok. So, I am using this notation only. So, this definition of the input and output ports also you can give here itself ok.

Now, coming for this one. So, we have four input ports, these will be connected to the input port. And this will be connected to the output port. In between, if you want to interconnect this gate ok, we require some wires. This is similar to the physical connection that you are going to give in a digital laboratory.

So, I want to connect the output of this AND gate to the input of this NOR gate. So, I require a wire, I will call this one as w 1, similarly here also we require another wire w 2. So, we have to define this wire. Wires are basically used to connect the gates in the design ok, wire w 1 comma w 2 semicolon.

Now, I can give some name for this AND gates and OR gates ok. So, this I will call as A 1, this I will call as A 2, this I will call as N 2. We can directly give this as a NOR, or we can give OR and NOT also, this is up to you ok. So, I am writing this primitive of the AND gate is as we have discussed in the earlier slide is and – lowercase and. And in the parentheses, we have to first write the output which is w 1. And you have to give the name because there are two AND gates. To distinguish between these two, you can write a name.

So, I am defining this AND gate as A 1. For A 1, what is the output? w 1 comma the inputs are a comma b. Similarly, there is second AND gate A 2, output is w 2 the inputs are c and d. Now, the final output y will be the output of the NOR gate. So, you can write directly NOR whose output is y, the inputs are w 1 and w 2. These w 1 and w 2, we can write in any order ok, so that is all.

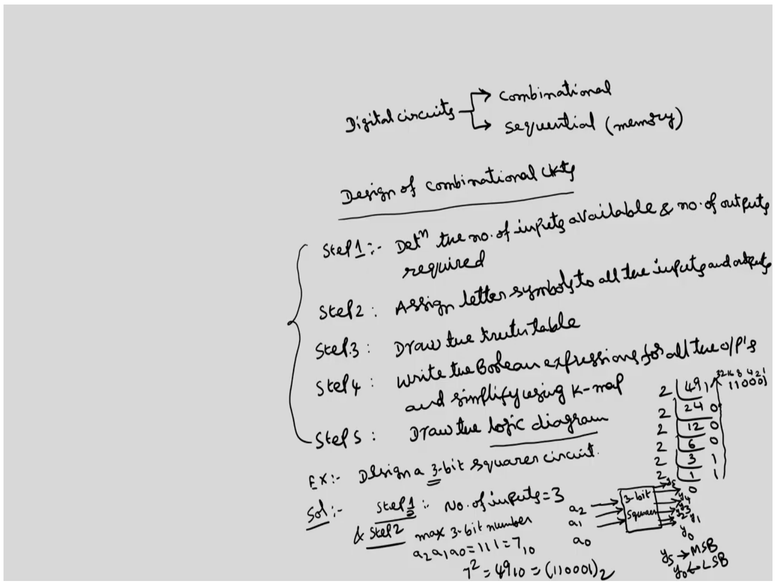
This is the description of this given circuit which is called as AND, OR and inverter circuit. So, we can also write this NOR as otherwise this you can write using two I mean keywords one is you can use OR. So, you have to define one more wire here ok. If want to right this NOR as the inputs are w 1, w 2, and this is y. This is equivalent to OR followed by you can define another w 3.

In that case, you have to define here another w 3, then NOT gate this is finally, is y. So, OR you have to write here w 3 in that case output is w 3 comma w 1, w 2 semicolon NOT y comma w 3. So, this is how if you write this program, this will give the description of the given circuit ok.

So, this is I mean Verilog is having two folded uses, one is it will describe the circuit. So, we can test the designed circuit. So, you have some design. So, normally, how to test the design? Normally, we will connect using the ICs and I will test the functionality.

So, whereas, using Verilog code, you can test the functionality of the design by writing the Verilog code ok. So, I will consider another example of how to I mean test the design. So, for that, I have to first discuss how to design the digital circuit.

(Refer Slide Time: 23:44)



Digital circuits → combinational, → sequential (memory)

Design of combinational ckts

Step 1 :- Det^n the no. of inputs available & no. of outputs required

Step 2 : Assign letter symbols to all the inputs and outputs

Step 3 : Draw the truth table

Step 4 : Write the Boolean expressions for all the o/p's and simplify using K-map

Step 5 : Draw the logic diagram

Ex :- Design a 3-bit squarer circuit.

Sol :- Step 1 & Step 2 : No. of inputs = 3

max 3-bit number $a_2a_1a_0 = 111 = 7_{10}$

$7^2 = 49_{10} = (110001)_2$

We know that digital circuits can be broadly classified as combinational circuits and sequential circuits. So, combinational circuit is a logic circuit, where the output at any time depends upon only the present input without regard to the previous outputs ok. Whereas, in case of sequential logic circuits, the output not only depends upon the present input, but also on the previous outputs.

So, you need some memory in case of sequential to remember the previous output. So, this requires some memory. And there will be some feedback connection from output to input ok. So, first I will discuss the design of combinational circuits and the corresponding Verilog codes. CKT is the symbol for the abbreviation of circuit ok. So, there are certain rules for design of combinational circuits ok.

So, step 1 for the design of the combinational circuit is, so we have to define the problem. From the specifications determine, so you will be given the specifications, determine the number of inputs available and the number of outputs required. Then step 2 is assign letter symbols, to all the inputs and outputs.

3rd step is drawing the truth table which describes the relation between the input and output. Then next step is writing the Boolean expressions for all the outputs and simplify by using

K-map. And next step is this is the last step, draw the logic diagram. Once the logic diagram is obtained, to check the functionality, normally we will connect using the ICs ok.

For simple circuits, which contains a smaller number of gates, we can test manually, but in case of VLSI circuits where millions of the gates are there. Then we have to go for the computer aided designs. So, here in order to test this logic diagram that we have obtained from the design, so you can write a Verilog code and you can run the simulation. You can give the different combinations of the inputs, and you can check the outputs ok.

So, in order to I mean explain this design rules or design steps, I will take an example of design a 3-bit squarer. So, the problem statement is given how to obtain the logic diagram which I mean squares the 3-bit input value ok. So, the first step is we have to determine the number of inputs available; it is already given that 3-bit. So, number of inputs are 3.

Then how to find out the number of outputs required. So, basically this circuit has to square the input number, we have 3-bits say you have a 2, a 1; a 0 is the 3-bit number. This is 3-bit squarer. So, how many outputs are required? How to find out this? So, what is the maximum 3-bit number ok? So, the maximum 3-bit number is all the a 2, a 1, a 0 should be 111. This is in binary.

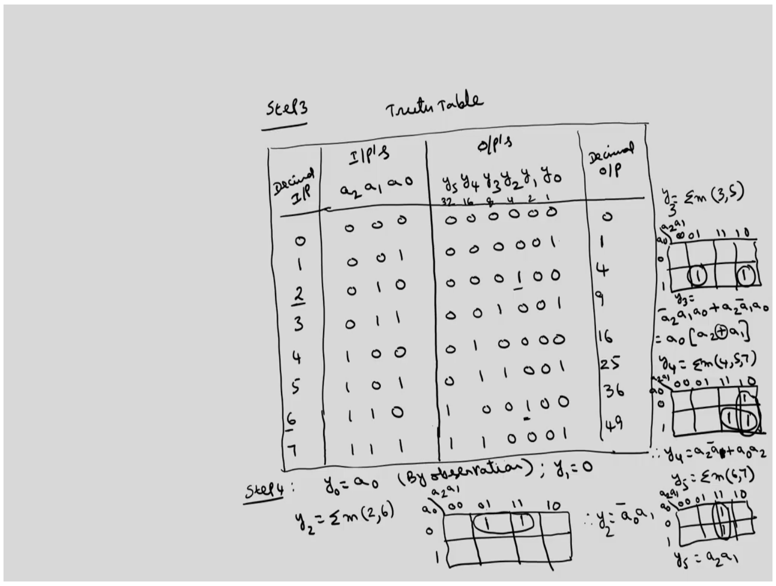
This is 7 in decimal. If I square 7 in decimal, what is the value 49 is the decimal value, sorry, this is decimal 10 stands for the decimal base. So, what is the binary equivalent of 49? So many bits are required at the output because you have to accommodate the maximum possible value also.

So, 49 if you convert into binary 24 are 1, 2 12 are 0, 2 6s are 0, 2 3s are 0, 2 1s are 0 1, 2 0s 1. Until the quotient is 0, you have to stop here and you have to read this value from bottom to top. So, what is that value is 11001. If you want to verify, you can verify also. The weight of this one is 1, in decimal this is 2, this is 4 powers of 2, this is 8, this is 16, this is 32.

So, basically you are adding 32 because 1, 32 plus 16 which is equal to 48 plus 1 is 49. So, this in binary form is 110001 in binary. So, we require 6 outputs. So, 6 outputs are required. So, starting with y 5, y 4, y 3, y 2, y 1, y 0 ok. I am assembling letter symbols, and finding this one. The second one is I am assigning the letter symbols I am assuming that a 2, a 1, a 0 is the, this is second step also I have combined this. Step 1 and step 2, I have combined together.

So, I have assigned the letter symbols for the inputs as a 2, a 1, a 0, and outputs as y 5 to y 0, y 5 is the MSB; y 0 is the LSB, y 5 I have called as MSB. You have to clearly define this y 0 is LSB. MSB stands for most significant bit; LSB stands for least significant bit.

(Refer Slide Time: 31:56)



Step3 Truth Table

Decimal I/P	I/P's $a_2\ a_1\ a_0$	O/P's $y_5\ y_4\ y_3\ y_2\ y_1\ y_0$ (32 16 8 4 2 1)	Decimal O/P
0	0 0 0	0 0 0 0 0 0	0
1	0 0 1	0 0 0 0 0 1	1
2	0 1 0	0 0 0 1 0 0	4
3	0 1 1	0 0 1 0 0 1	9
4	1 0 0	0 1 0 0 0 0	16
5	1 0 1	0 1 1 0 0 1	25
6	1 1 0	1 0 0 1 0 0	36
7	1 1 1	1 1 0 0 0 1	49

Step4: $y_0 = a_0$ (By observation); $y_1 = 0$

$y_2 = \Sigma m(2,6)$ $\therefore y_2 = \bar{a}_0 a_1$

$y_3 = \Sigma m(3,5)$

$y_3 = \bar{a}_2 a_1 a_0 + a_2 \bar{a}_1 a_0$

$= a_0 [a_2 \oplus a_1]$

$y_4 = \Sigma m(4,5,7)$

$\therefore y_4 = a_2 \bar{a}_1 + a_0 a_2$

$y_5 = \Sigma m(6,7)$

$y_5 = a_2 a_1$

In the third step, you have to draw the truth table. For all the combinations, how many combinations are there? Because inputs are 3 a 2, a 1, a 0, so we have 8 combinations; outputs are 6 right, so y 5, y 4, y 3, y 2, y 1, y 0. So, we take all the eight combinations 0 0 0. So, if you want you can write decimal value also here, decimal input value decimal output value, so that we can easily understand this problem decimal output decimal input.

So, 0 means 0, 0 square what is the decimal equivalent value 0 only. So, what is 6 bit equivalent of 0? All 0s. Decimal 1, if the input is decimal 1, the representation in input binary is 001. 1 square is decimal is also 1. How to represent 1 using six bits 000001. If the input is 2, the binary representation of 2 is 010.

So, the decimal square of this one is 4. So, how to represent 4? 000100. If the decimal value is 3, the binary representation is 011. 3 squares are 9. How to represent 9? 001001. These are basically the weights you can add. This weight is 1, 2, 4, 8, 16, 32.

So, to get 9, you have to add 8 plus 1 ok, 4 100. So, this is basically 16, 16 is directly there, so 010000. 5, 101; 5 square is 25. To get the 25, it is 16 plus 8 is 24 plus 1. So, 011001. 6 is 110; 6 squares are 36 – 32 plus 4, so 100100. 7, 111; 7 is 49. We have obtained this 32 plus 16 plus 1. So, you see the truth table so which will describe the given problem for all the combinations of the inputs it will give the output square value.

So, what is next step? Step 4 is you have to obtain the Boolean expressions for the all the outputs and we have to simplify by using K-map ok. We can see that one even if you simplify also you will get same thing y 0 is nothing but this is 01010101 a 0 also 01010101. So, simply y 0 is equal to see by observation you can do it by using k map also you will get same thing. This is by observation.

And y 1 is always 0. And y 2 is what are the min terms? y 2 is sigma m. So, wherever Boolean expression you have 1, this 1 is correspond to this min term 2, 2 this min term is 6. You can use the k map basically the inputs are a 2, a 1, this is up to you can write in any way I am writing a 2, a 1, a 2 is MSB, a 0 is LSB, or you can write a 2, a 1 here, and a 0 here.

So, in any case, you will get the same final expression. So, we know that we are going to use gray code to remember the rows and columns 0001. If it is binary, 1011; if it is a gray, 1110. So, this box number is 2, 2 means 0 10, 010 is this is 2. And 6 is 110; 110 is this ok. This is two box combination.

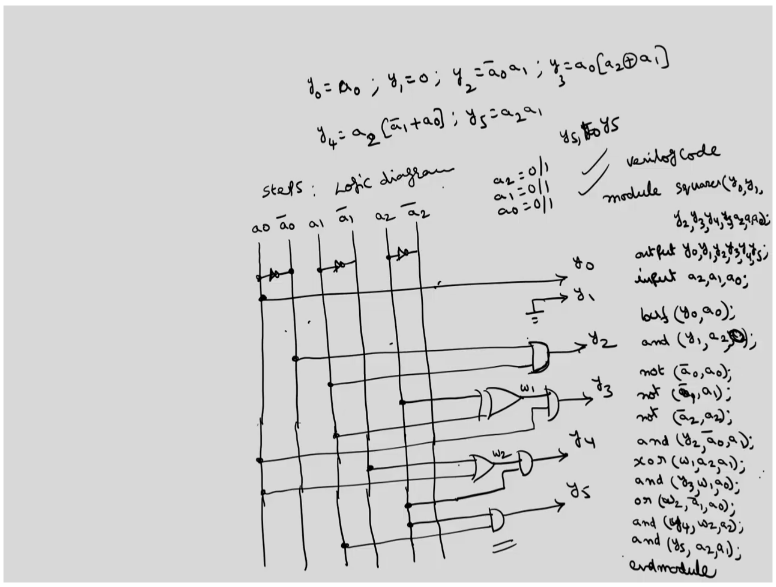
So, therefore, y 2 is equal to. So, here the variable that is changing will get cancelled a 2 is 0 here, a 2 is 11 here. So, a 2 will get cancelled, a 1 is constant at 1. So, you have to take without complement, whereas here a 0 is 0. So, this is a 0 bar a 1 ok.

Similarly, you can get y 3 as sigma m y 3 is the min term is 3, 5. So, this is a 2, a 1, a 0, 00, 01, 11, 10, 01, 3, 5. 011 is 3; 5 is 101. So, this is nothing but just there is no I mean combination at all. This is single box combination, this is single box combination. So, you will have y 3 is equal to this expression is a 2 bar a 1 a 0 plus this one is a 2 a 1 bar a 0.

So, a 0 is common. You can write this one as this is y 3. This is equal to a, a 0 is common, then it is exclusive OR between a 2 and a 1. So, a0 times a 2 exclusive OR with a 1; y 4 sigma m min terms are 4, 5, 7. So, 00, 01, 11, 10, 01, a 2, a 1, a 0. So, 4, 5, 7, 100, 5 is 11 101; 7 is 111. So, this is two box combination; this is another two-box combination.

So, therefore, y 4 is this expression is a 0 will changes. So, this is a 2 a 1 bar, and this one is a 0 and here a 2, say it is common. And similarly, y 5 is equal to sigma m 6, 7. So, if you simplify this, you will get 00 01 11 10 a 2, a 1, a 0, 6, 7, 110, 111. This is two box combination. This is simply y 5 is a 2 a 1. So, we have the final expression. Then we have to draw the Boolean the logic diagram ok.

(Refer Slide Time: 40:07)



$y_0 = a_0$; $y_1 = 0$; $y_2 = \bar{a}_0 a_1$; $y_3 = a_0[a_2 \oplus a_1]$
$y_4 = a_2[\bar{a}_1 + a_0]$; $y_5 = a_2 a_1$
y_5, ~~y_0~~ y_5
steps : Logic diagram
$a_2 = 0|1$
$a_1 = 0|1$
$a_0 = 0|1$
a_0 $\bar{a}_0$ a_1 $\bar{a}_1$ a_2 $\bar{a}_2$
y_0
y_1
y_2
w_1
y_3
w_2
y_4
y_5
verilog code

```
module squarer(y0,y1,
        y2,y3,y4,y5,a2,a1,a0);
output y0,y1,y2,y3,y4,y5;
input a2,a1,a0;
buf (y0,a0);
and (y1,a2,[illegible]);
not (ā0,a0);
not (ā1,a1);
not (ā2,a2);
and (y2,ā0,a1);
xor (w1,a2,a1);
and (y3,w1,a0);
or (w2,ā1,a0);
and (y4,w2,a2);
and (y5,a2,a1);
endmodule
```

So, I will draw here the expressions for y 0, y 1. y 0 is a 0, y 1 is 0, y 2 is y 2 we have got this y 2 as a 0 bar a 1; y 3 is a 0 a 2 exclusive OR a 1; y 4 is a 2 into a 1 bar plus a 0 a 2 into a 1 bar a 0; and y 5 is a 2 a 1 right, a 2 a 1. So, last step is we have to draw the logic diagram. So, you can draw this logic diagram using gates.

Now, see not I mean write the Verilog code correspond to this one module using these Boolean expressions I have directly I am writing the Verilog code ok, or otherwise you can draw the logic diagram, so that it will be easier. So, this will be y 0 is a 0, you have three inputs a 0, a 1, a 2, and the corresponding complements. This is how you can draw the logic diagram easily. You have a 0. And if I connect through the NOT gate a 0bar.

This is a 1. If you connect the NOT gate between this, you will get a 1 bar, this is a 2, this is a 2 bar. So, we have a NOT gate. So, to get this is a 0 directly you have to connect this from the a 0, this your y 0. So, y 1 is basically ground; y 2 is a 0 bar a 1 a 0 bar line is this, a 1 is this, this is y 2. y 3 is a 0 into a 2 exclusive OR a 1. a 2 line is this, a 1 line is this.

This is exclusive OR, and this you have end with a 0, this is your y 3. And y 4 is a 2 into a 1 bar plus a 0; a 1 bar is this plus a 0 is this, OR operation, and you have to end with a 2. This is a 2, and y 5 a 2 to a 1. This is the complete design of this one. If you want to verify this, normally we will in manual mode, we will connect the ICs correspond to all the logic gates and we will verify.

Whereas, in Verilog code, we want to write Verilog code, module I will give the name as squarer output is y 0 so on up to y 1, y 2, y 3, y 4, y 5 comma a 2, a 1, a 0 semicolon output y 0 comma y 1 comma y 2 comma y 3 comma y 4 comma y 5 semi colon input a 2 comma a 1 comma a 0 semi colon.

Then how to get output y? y 0 is same as a1. For that, we can write buffer. Buffer because buffer will give the same output is equal to input. Buffer with output as y 0 input as a 0. And y 1 is 0, you can directly give assign y 1 is equal to 0, or we can give if want to use a gate, then you can use something like AND y 1 inputs. If you give any input like a 2 comma a 2 bar if I use 0 say other input.

This you have to you can design in many ways, you can assign directly in case of behavioral modeling ok. I will discuss that in the coming classes. This is 0, a 2 comma 0. Then y 2 is nothing but so y 2 if want a 0, first you obtain this a 0 bar, a 1 bar, a 2 bar using three not gates not. So, we call this one output as a 0 bar, a 0, not a 0 bar sorry a 1 bar, a 1, not a 2 bar a 2.

Then y 2 is AND, and y 2 comma inputs are a 0 bar a 1, a 0 bar we have already obtained ok. And y 3 is we have to first we have to define the wire here, we have to define wire y 1 w 1.

So, XOR output is w 1 comma the inputs are a 2 and a 1. Then we have to use AND operation, AND output is y 3 the inputs are w 1 and then a 0.

Then y 4 first you have to write OR between a 1 bar and a 0 this output you call as y w 2. So, w 2 comma the inputs are a 1 bar comma a 0, then AND between output is y 4 input is w 2 comma the other input is a 2. AND y 5 is and between a 2 and a 1, output is y 5 a 2 comma a 1 end module. So, this is the complete design using Verilog ok.

So, if you run this program and if you simulate this a 0, a 1, a 2, a 0, this you can give 0 or 1, this will give 0 or 1 using the simulation tools. And you can observe these outputs y 5 to y 0, y 5 to y 0, it has to satisfy the truth table that I have written in the previous slide ok. This is how we can check using this Verilog code ok. This is all about this first design. So, in the coming classes, we will discuss some more examples of the combinational circuits.

Thank you.