Digital VLSI System Design

Prof. S. Srinivasan

Dept of Electrical Engineering

Indian Institute of Technology Madras

Lecture – 2

Combinational Circuit Design

Slide – Summary of contents covered in the previous lecture.

(Refer Slide Time: 01:45)

In the earlier course on digital circuits, we would have seen the design of combinational logic; there the approach would have been to minimize the number of gates used in the design. There are conventional techniques such as, karnaugh maps to reduce the number of gates to be used in implementing a given function. These conventional techniques are good, provided the circuits are simple. In today's technology we design circuits, which are, very large, extensive and of course complex. Complexities can always be implemented with whatever gates available. It is the size of the circuit, which is of concern here.

If you have a huge circuit to be implemented using gates and karnaugh maps, the design becomes extremely difficult. Another thing you would have learnt in the karnaugh map approach is, even though the circuits rely and use AND, OR invertors, sometimes you are asked to manipulate the circuit to use specific type of gates, such as NAND gates for example. In that case, you take the simplified function using karnaugh maps and manipulate it using a given type of a gate like the NAND gate. These techniques are as I said, good to be used for small circuits.

When the circuits become larger, as in today's technology, we need it to look at other methods. These other methods revolve around the available components; you would decide on the hardware that you would use to implement the circuit and try to map your circuit functions on the available hardware. For example, in today's technology like, field programmable gate arrays, you have the capability of 100,000 gates or 10,000 gates or 4000 gates etc. So, when you look at the function, I will simplify it such that it can be fitted into that available gate.

Before going to that level, the scope of this entire course is on IC design and digital VLSI design, we shall go one step further than the gate level implementation. Mapping of the given function onto an available hardware or given hardware is the theme of the VLSI design; that is the emphasis of VLSI design. In order to get there, we shall take it one step at a time. Let us look at some of the smaller, less complex components of hardware and see how our circuits can be mapped on to this.

As an example of this approach we will talk about multiplexers; let us take multiplexer designs as an example of fitting a design onto an available hardware. Let me say, a

Multiplexer based design (Refer Slide Time: 05:49). All of us know what a multiplexer is. A multiplexer is a component, with several inputs and 1 output; let us say 4 inputs and 1 output; and which of these inputs gets connected to the output depends on the inputs known as selector inputs. In 4 to 1 multiplexers, where there are 4 inputs and 1 output, we need to have 2 selector inputs so that, one of these 4 can be selected at a time to the output. Let us call these inputs; I_0, I_1, I_2 and I_3, for inputs 1, 2, and 3. The selectors are S_1 and S_0 and the output as F. We can write the function F as: F is equal to I_0 if both the S_1 and S_0 are low, zero, S_1 bar S_0 bar I_0; if it is 0 1, that is S_1bar S_0, I_1 which is the output; if input is 10 then S_1 is 0, it is S_1S_0 bar output is I_2 and finally when both are 1, output is ... I_3

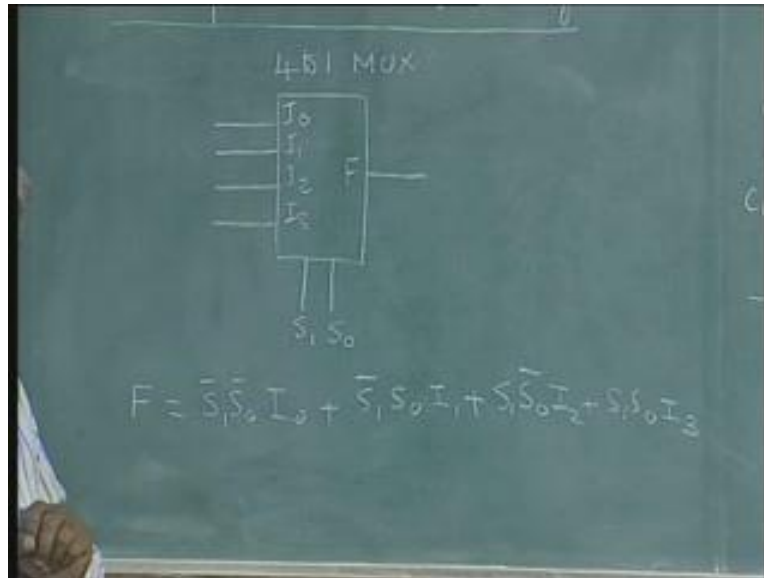
This is a multiplexer, which selects from a given number of inputs, one output, one of the given number of inputs, one of the inputs as outputs and this output that is selected depends on the selector combination. This component can be used to design a combinational logic. Now this is not a gate approach, but it uses gate because we know that a multiplexer cannot be built without gates. If you break into the multiplexer you will still find AND gates and OR gates. Like other combinational logic, everything can be reduced to a basic set of 'AND' 'OR' inverters or a universal gate such as a NAND gate or a NOR gate combination. It is not necessary for the designer to know the inside of this component. If the input output specifications of the component are given, the designer should be able to fit the required design into these pieces of hardware. This is what is known as a level higher than the gate level. A level in which the hardware is given to you and that hardware is used to fit in the given design. Sometimes it may not fit, because the design requirement is larger than the available component; that case we will have to see how to do that. That we will see later on.

Let us continue with the example by trying to implement a one bit full adder using multiplexers. Let us look at a one bit full adder whose inputs are, a, b and carry. We shall call this: a_i, b_i and c_i minus 1, (Refer Slide Time: 09:39) which was carried from the previous stage, and call the output as S_i and C_i. The truth table for this would be:

a_i	b_i	c_i-1	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

This is a well known truth table, seen it numerous books. This table can be implemented using gates. This is done by drawing one karnaugh map with a_i, b_i and c_i minus1 as inputs and S_i as output, an 8 cell karnaugh map and another karnaugh map with $a_{i,}$ $b_{i,}$ c_i minus 1 as inputs and C_i as output, again an 8 cell karnaugh map. Then you simplify them and implement these two simplified functions for S_i and $C_{i.}$ It can also be implemented using gates. If we are asked to use only NAND and NOR gates, then we have no further manipulation. But right now we are going to see if it can be implemented using an available component. Now if we were told to use only multiplexers by the person who has given the problem to us and we had the choice of multiplexer type, we would choose a multiplexer with 8 inputs and 1 output.

(Refer Slide Time: 12:19)



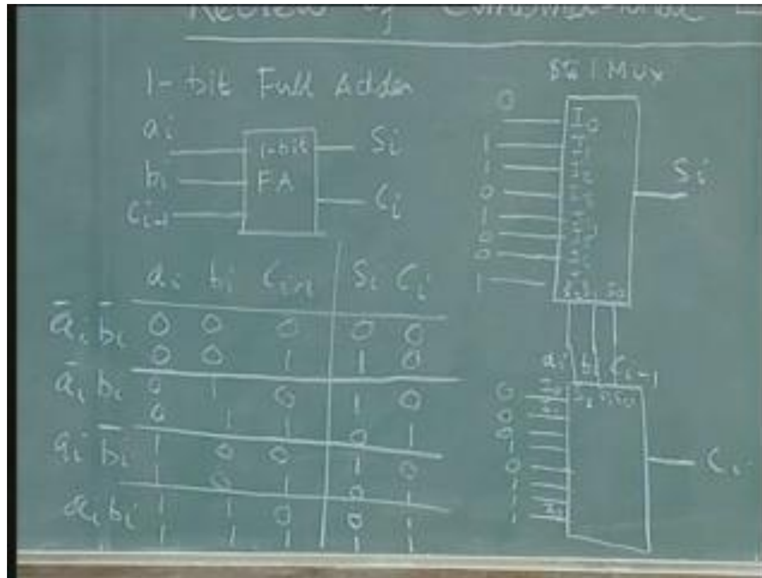
If we had 8 inputs, I_0 to I_7, then we need 3 selectors S_2, S_1, and S_0. That will be the simplest possible implementation. So we are going use a multiplexer with 8 inputs and 1 output; 8 to 1 MUX where output F is nothing but S_i. We need to connect our selectors, a_i, b_i, $c_{i\ minus1}$ and connect it to 8 inputs I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, with corresponding values: 0,1,1,0,1,0,0 and 1. Whatever the combination is, the corresponding input will be connected to the output.

For example, if you take the combination 1, 0, 1 where a_i is 1, b_i is 0 and $c_{i\ minus1}$ is 1, then the output should be 0. 1, 0, 1 corresponds to the I_5, which is connected to 0, and then output will be 0. This is the simplest implementation. We have not reduced the karnaugh map nor have we drawn the karnaugh map, we have simply mapped on the available hardware, namely, the multiplexer, by our given design specification. This is straightforward and simple. Another multiplexer will be required in order to finish the job and to do C_i. So we will use the same inputs as selectors, S_2, S_1 and S_0 and the inputs from I_0 to I_7. In this case we will put 0, 0, 0, 1, 0, 1, 1, 1 corresponding to the inputs. This is similar to the previous multiplexer and it gives us our C_i output carry. This means that 2 outputs S_i and C_i corresponding to the 2 outputs of the full adder with 8 inputs can be implemented using 2, 8 to 1 multiplexers. This fits the available hardware into the design specs.

Today that is gaining momentum because these are simple examples of full adder. Usually to do this, we would probably do exclusive 'OR' gates and simple 'AND' gates, as we are used to in the previous course. Some may think that this is simpler, but think of this example growing, with larger number of inputs and more and more complex functions. The mapping of these available components, especially we are talking today of circuits which have functional complexities that are equal to tens of thousands of gates, we would have gone with this type of readymade component rather than writing a karnaugh map or using other techniques by which you can simplify, to say so many gates are being used. It is not gate optimization that is important today, it is the fitting into your available hardware to the given design. In an integrated circuit design the essence is to fit into the available or the given hardware as much functional complexity as possible.

Optimization of course; this is a waste of 2, 8 to 1 multiplexers (Refer Slide Time: 16:29) This is the approach I want to show; this might not be the best way to do it. Can I optimize it? Instead of using 2, 8 to 1 multiplexers I would say is an overkill for this simple problem. There is a simpler way to optimize this. Instead of having 3 variables as selectors and 8 inputs, if I have only 4 inputs and 2 selectors; suppose I put a restriction. The person who gave this problem said "use a multiplexer but do not use 8 inputs, use only 4 inputs". Similar to the karnaugh map case, where, instead of using 'AND' gate and 'OR' gate invertors we were told to use NAND gates for further manipulation, the same concept applies here (Refer Slide Time: 17:57). Instead of having 3 variables as selectors and 8 inputs, we are made to use a 4 input multiplexer (I_0 to I_3) with 2 selectors (S_0 and S_1). Let us assume we use a_i and b_i as selector inputs out of the 3 selectors. Instead of going to the truth table and designing line by line, we shall take a_i and b_i to be 0. There are 4 combinations possible here, let us identify these combinations. The first combination: a_i and b_i are 0, second combination is a_i is 0 and b_i is 1, third combination is a_i is 1 and b_i is 0 and the fourth combination is a_i and b_i are 1 (Refer Slide Time: 19:54). Now let us see how to do this, earlier we had one row corresponding to each input. We looked at the first row; found the input as 0 and connected to 0; second row found the input 1 and connected it to 1. That was easy, but now for the mapping we have to do more exercises here.

(Refer Slide Time: 20:11)



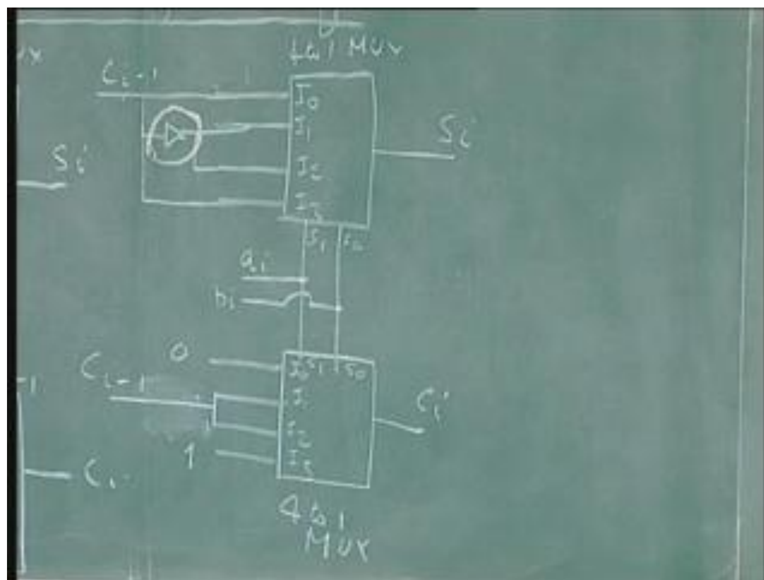
Now these two rows together correspond to the first input here, where ai and bi both are 0. That means, in order to get S_i, Si is 0 when a_i and b_i are both 0. S_i is 1 when a_i and b_i are both 0. What distinguishes this row from the previous row is the c_i value, if the $c_{i \text{ minus1}}$ is 0, then the output is 0. When the $c_{i \text{ minus } 1}$ is 1, then the output is 1. This means the output Si is the same as $c_{i \text{ minus } 1}$ (Refer Slide Time: 20:39). So instead of connecting a 0 or 1, we are going to connect $c_{i \text{ minus } 1}$. Let us go to second combination if $c_{i \text{ minus } 1}$ is 0 then we get output 1 and when $c_{i \text{ minus } 1}$ is 1 we get the output 0, so this would be $c_{i \text{ minus } 1}$ bar. Likewise in the third case it is $c_{i \text{ minus } 1}$ bar and finally when both a_i and b_i are one, then $c_{i \text{ minus } 1}$ is 0, the output is 0, Si is 0 and $c_{i \text{ minus } 1}$ is 1, then the output is 1; this will be $c_{i.}$ So now we have added to the extra restriction of using only 4 to 1 multiplexer, rather than 8 to 1 multiplexer and arrived at this.

We can repeat this for the next case. In the first combination both are 0 if C_i is 0. In the second combination c_i is 1 if ci minus 1 is 1. Third combination c_i is 1 if $c_{i \text{ minus } 1}$ is 1. Finally, if c_i is 0 or if c_i minus 1 is 0 in both cases, output is 1. In this case, the design has 4 inputs and 1output and we are able to get a S_i function and a c_i function. There are two significant things we have to see in this design; one is that we have reduced the size of the multiplexer from 8 to 1, to 4 to 1. We are not using any extra logic here, the only is that thing c_i minus 1 is an input. We can use that input straight away, but an inverter is required here. $c_{i \text{ minus } 1}$ requires an inverter here, in the true form and in the

complementary form, true form complementary form and true form. So we need an inventor here, maybe we can replace the $c_{i \text{ minus } 1}$ by an inverter. In the second 4 to 1 multiplexer there is no inverter required. By adding just one extra inverter (Refer Slide Time: 23:17) we can reduce the design from 2, 8 to 1 multiplexer to 2, 4 to 1 multiplexer.

The other significant feature of the design is that normally, these multiplexers come in dual packages, we cannot buy a single 4 to 1 multiplexer in the market, we have to buy a twin package. In 1 IC we will get 2 of these 4 to 1 multiplexers. We can get 1 IC package and use both the 4 to 1 multiplexers for the entire design, instead of 2 different IC packages.

(Refer Slide Time: 23:55)



The above concepts, by using multiplexers and full adders as examples, help us understand clearly how to proceed using an available component. These concepts are simple, for as the problems get bigger, we can get more and more complex circuits using multiplexers. It can be proved, just as we had proved in our earlier combinational logic, all logic functions are implemented using ‘AND’ ‘OR’ inverters to start with and can further be simplified using NAND gates or NOR gates. So it can be proved that all the combinational logic can be derived using multiplexers. Today there are ICs available with huge number of multiplexers inside them and we can program those multiplexers and interconnect them any way, in order to realize any complex function that we want to

implement. Any complex function you implement today, VLSI you call it; you take an array of multiplexers, some of them in one package, you can interconnect them and implement various functions automatically. We will see some of them later on in this course.

Multiplexers are not the only component; there are other components like the decoder, which is used frequently. This component is also seen in the course 'Digital design'. The decoder is a circuit, which has several inputs and several outputs and the outputs are related to the input by a code. For example, let us take a simple case where we have 2 inputs and 4 outputs, this is called a 2 to 4 decoder (Refer Slide Time: 26:21). Let us call these inputs a and b and the output as O_0, O_1, O_2, O_3, meaning, output 0, output 1, output 2 and output 3. O_0 is 0 when both and b are 0 this output is high and less than low. So output will be 'a bar, b bar'. If 'a' is 0 and 'b' is 1, then it is 'a bar b', here the output 1 is active. Output 2 is active when, a is 1 and b is 0, which is 'a b bar' and finally, when both are 1 output 3 is active (ab). This is an example of a decoder. This is again a packaged IC that is available in the market, which we can use to implement a given logic function, rather than going to the gate level and unifying and simplifying it.

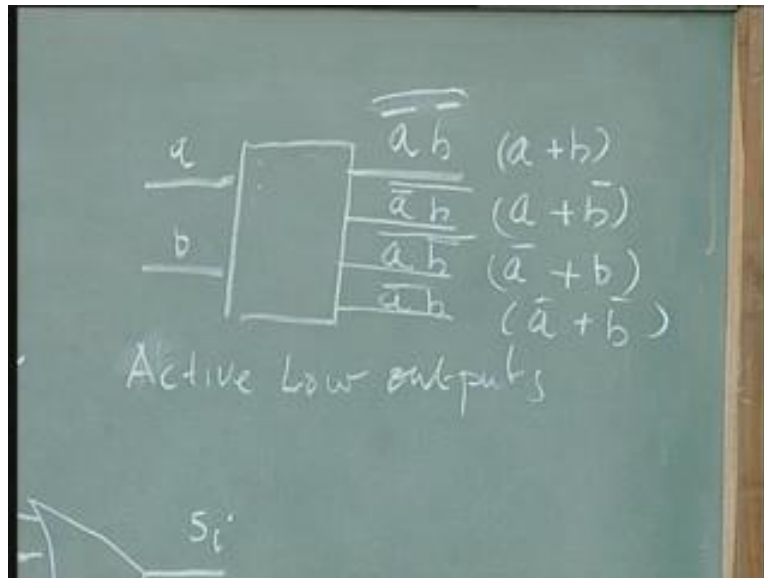
Now let us see, how to do a problem, similar to the full adder using a decoder. We could use a 3 to 8 decoder, where we have 3 inputs - let us call them a_i, b_i and $c_{i \text{ minus } 1}$ and 8 outputs O_0 to O_7. This is a simpler design, compared to the 8 to 1 multiplexer. The decoder gives us outputs corresponding to the combination of the inputs. Based on the combination of the inputs, the corresponding output will be high and the rest of the outputs will be low. Now for a sum S_i, we need the outputs 1, 2, 4 and 7(Refer Slide Time: 28:46). So the output will be high, whenever these combinations are occurring, these outputs will be high and the sum is nothing but the 'OR' of these. The sum is this or this or this or this (Refer Slide Time: 29:11).

All we need to do is put an 'OR' gate to the output and tie them together and call it S_i. Now for the carry we want the outputs 3, 5, 6 and 7. Put these outputs together with another 'or' gate and we get c_i. Now with one simple 3 to 8 decoder, with 2 'OR' gates we can implement the full adder circuit, as given in the truth table. There is one catch though; the decoders generally do not come with active high outputs. In the decoders,

which are generally available in market, the commercial decoders, the outputs are active low, which means when we choose a combination the corresponding output will be low and the rest of the outputs will be high. For example; if this is 0, 0, 0, O_0 will be 0 and all others are high; if we take 0, 1, 1, which corresponds to the combination, O_3; then O_3 will be low and all others will be high. This is called active low combinations.

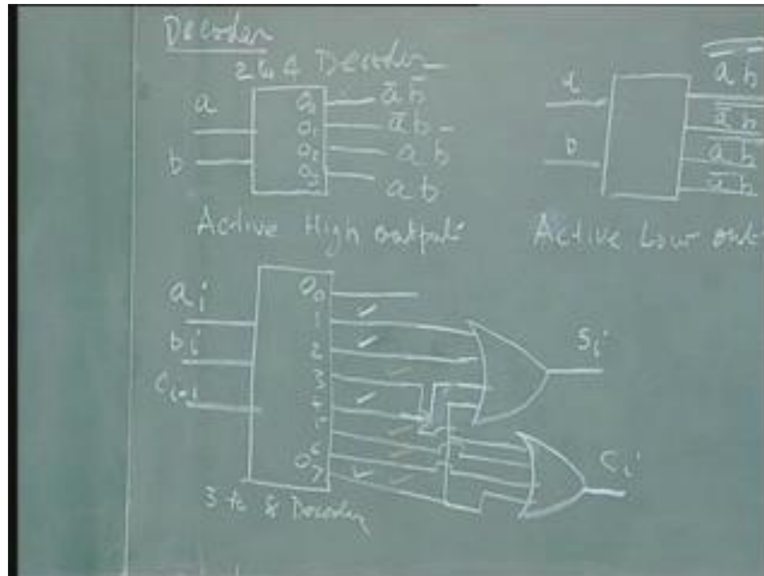
Usually in the decoders the outputs are selected based on the input combination. In this case, for example, if all are 0, 0, 0, then the output will be high and rest of them will be 0. This is called active high, where the output is high. In active low outputs with inputs a and b, output will be will be: a bar b bar, bar, next one will be: a bar b, bar then, a, b bar, bar and ab bar.

(Refer Slide Time: 33:35)



Which means, applying Demorgan’s theorem, this will be a or b, this will be a or b bar, this will be a bar or b and this will be a bar or b bar. So what we get is not this, but this (Refer Slide Time: 33:32); that has to be factored into the design of the full adder. When we use that logic here, our gate combination will be changed. Using my active low outputs, if I use my input as a_i, b_i, $c_{i\ minus\ 1}$, the outputs will not be 0, but O_0 bar, O_1 bar, O_2 bar, O_3 bar, O_4 bar, O_5 bar, O_6 bar and O_7 bar.

(Refer Slide Time: 33:40)

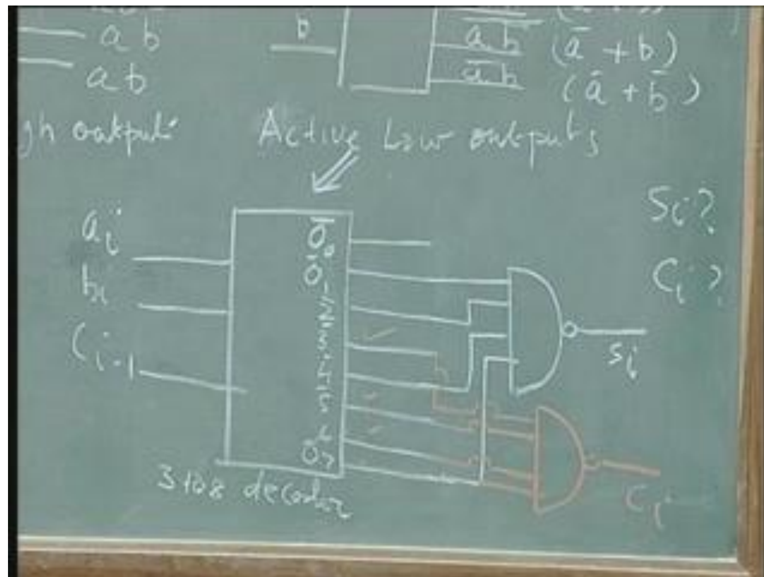


To get out output functions S_i and c_i, using the 3 to 8 decoder, with active low outputs and inputs a_i, b_i and $c_{i\ minus\ 1}$, we have to do some manipulation. What I want is the sum is the output will be 1 or 2 or 4 or 7. Our output requirement from the truth table of the full adder is: S_i is equal to 1 plus 2 plus 4 plus 7. It can also be written as 1bar.2bar.4bar.7bar, if we use the Demorgon's theorem (Refer Slide Time: 36:29). Since we have 1 bar, 2 bar, 4 bar, 7 bar, all we have to do is to take these take this put them into a NAND gate.

(Refer Slide Time: 37:20)

Likewise my carry (C_i) would be: 3 or 5 or 6 or 7 which are the same as 3 bar and 5 bar and 6 bar and 7 bar, whole bar. Now we take these and add a NAND gate.

(Refer Slide Time: 37:55)



Now we got our 1 bit full adder, using a 3 to 8 decoder, with a_i, b_i and c_i minus 1 as our inputs and S_i and C_i as our outputs, with an active low output for the decoder. We have seen an example using the decoder that is another way in which hardware can be implemented; wherein, you fit into the design specification in the available hardware. Multiplexer was the first example we saw and decoder was the second example. We used the decoder example, to show that multiplexers are not the only component that can fit in a design using the available hardware. But let us switch back to the multiplexers again, to see some other interesting features of the multiplexers based design.

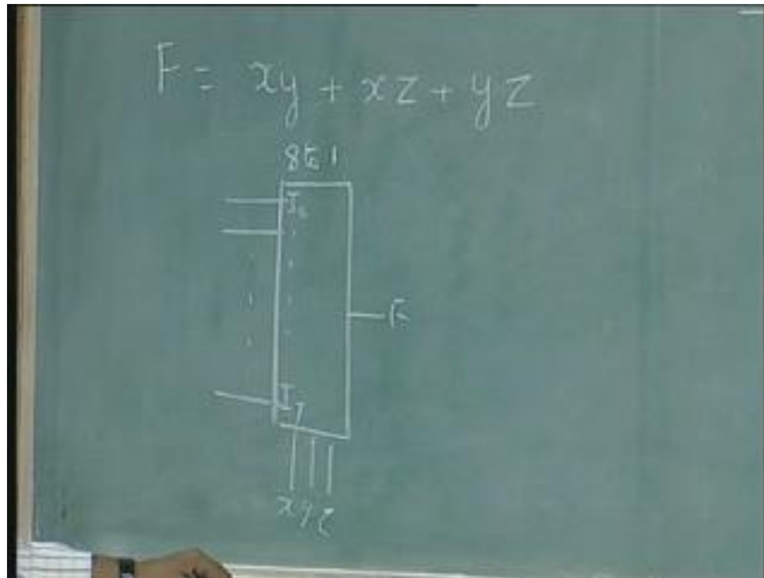
In FPGA based designs, where FPGA stands for Field Programmable Gate Arrays, We saw this term in the introduction and will see more in these lectures. A lot of the functions are available on a single chip. We have to map the designs to these functional blocks within the chip. The multiplexers are a major component or major functional block, available in these FPGAs. For this, reason let us go back to the multiplexer and see more interesting things about it.

The example we took in the multiplexer case was a 1 bit full adder, first we used 2, 8 to 1 multiplexers, then using the same example we implemented 2, 4 to 1 multiplexers, this

reasonably reduced the size and a single package can accommodate 2, 4 to 1 multiplexers. In a given environment of a FPGA or a large IC - VLSI design, you may not have multiplexers of different sizes available to you, whereas functional requirement may be different. There may be several functional requirements, requiring multiplexers of different sizes, but within a single chip of the FPGA, only one type of multiplexer may be available, that means the given number of inputs and given number of outputs will have the same type of function. Several of them, in fact hundreds and thousands of them might be available, but all of them will be the same type. Functional requirement would be different in some cases and may have a fewer inputs and one output or in some cases, may have large number of inputs and an output. So to tailor or reduce the input/output specification to the type of multiplexers available on the chip, we shall see a few examples.

Let us take a function, which would normally require an 8 to 1 multiplexer (with 3 inputs and 1 output), but in case we are told to use 2 to 1 multiplexers (with 2 inputs and 1 output), with no restriction on the number of multiplexers to be used, but I will have a restriction on the type of multiplexer used. Let us take an example of the function F which would be; F equals xy plus xz plus yz. In this function there are 3 variables; we would normally use 8 to 1 multiplexers with I_0 to I_7 with inputs x, y, and z and output F (Refer Slide Time: 42:17). But, in this case we will implement it using only 2 to 1 multiplexers. Even though there is no restriction on the number of 2 to 1 multiplexers, we shall try and keep the number to a minimum of 2 to 1 multiplexers.

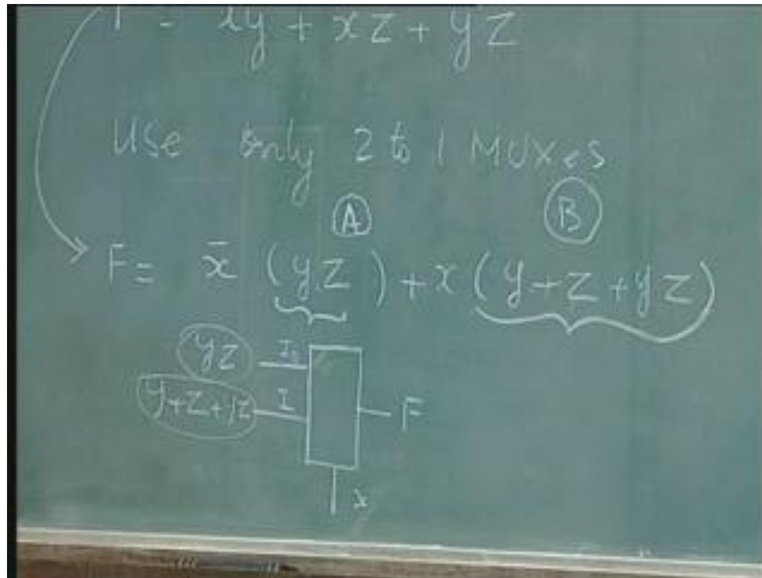
(Refer Slide Time: 42:33)



I am going to rewrite the function here and select a 2 to 1 multiplexer; we can have only 1 selector variable, whereas we have here 3 selector variables. We have to choose either x, y or z as a selector variable in each case and one of these can also be an input variable. We would use one variable in the first stage another in the second stage and the third variable would be an input variable. We shall first rewrite the function using x as the selected variable, that is, we would rewrite this as: x bar (yz) plus x (y plus z plus yz) (Refer Slide Time: 44:30). The x bar yz, combines with the x, yz to become yz, which implies that we have succeeded in rewriting this function F in this form. Therefore we can use a multiplexer with 2 inputs controlled by x, where 'x' is the selector input. So when x is 0 the input would be yz, which would be connected to the output of the multiplexer and when x is 1 the input would be y plus z plus yz, which would be connected to the output of the multiplexer. This would require gates, since yz can be realized only using gates and y or z or yz requires another 'or' gate. Even though we use only 1 multiplexer this would not be a good solution.

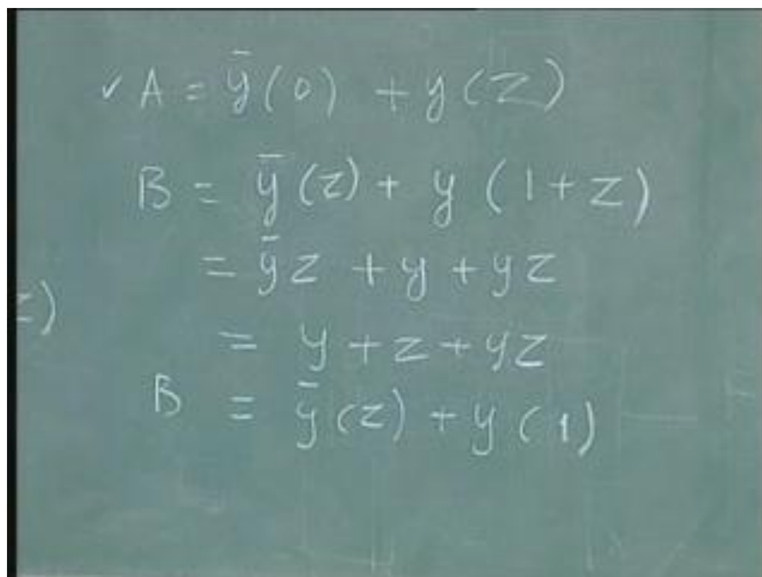
In a multiplexer with x as the selector input and I_o and I_1 as the inputs, when I_0 corresponds with x it will be yz and when I_1 corresponds with x it will be y or z or y z. The yz would require a 2 input 'AND' gate and the y or z or yz would require a 2 input 'AND' gate and a 3 input 'OR' gate. Let us try and further simplify this, by calling the function ys as A and the whole function y or z or yz as B (Refer Slide Time: 46:30).

(Refer Slide Time: 46:32)



Let us write A and B individually in terms of y. We shall write A as y bar (0) plus y (z) and B as y bar (z) ‘OR’ y (1plus z). This can be expanded to y bar z plus y plus z, since y bar z plus y is y plus z, it can be written as y plus z plus yz. B can be rewritten as y bar plus y into 1.

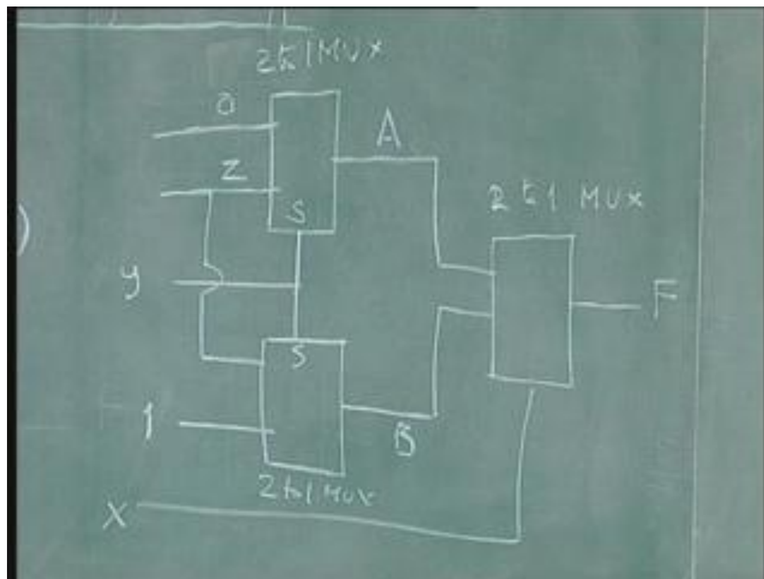
(Refer Slide Time: 47:50)



We can therefore, have A and B expressed as multiplexer expansions with y as a selector variable. This can be implemented in the first stage as; the A function, which will be

obtained by giving y as the selector input, 0 as the first input and z as the second input. The second multiplexer which will implement B will have z as the first input and 1 as the second input. Now A and B together, can implemented, using a third multiplexer with x as the selector input and F as the output. All of them use 2 to 1 multiplexers. In effect, the given function, which would normally require an 8 to 1 multiplexer, has been replaced with 3, 2 to 1 multiplexers, by rewriting these equations in such a way that we only have 2 inputs and 1 output for each, but we required 3 multiplexers.

(Refer Slide Time: 49:45)



In the FPGA there are so many multiplexer functional blocks of the same type, where the number is not restriction; we can go on adding any number of multiplexers, in order to realize the function. This is the essence of the VLSI design. That is to say, a huge function, that has several inputs and several outputs, which has several functional blocks with variable number of inputs and variable number of outputs, can be mapped onto a uniform hardware, where a uniform type of a functional block that will be available to be used for synthesis. We will stop with this and tomorrow we will see how to do using programmable logic device.

Summary of Lecture 2

Combinational Circuit Design

(Refer Slide Time: 50:58)

(Refer Slide Time: 51:05)