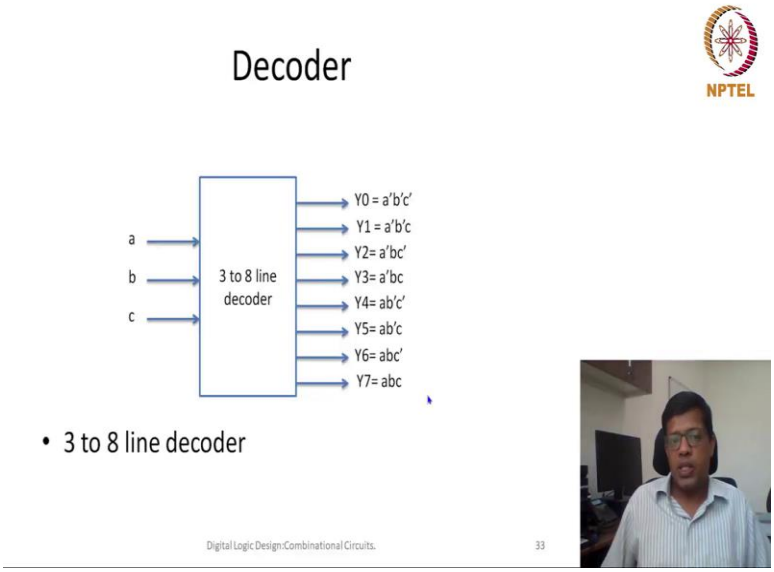
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Decoders-2

Hello everyone. One way of designing digital circuits is knowing some common building blocks. So these building blocks could be utilized in different scenarios. They could be modified and sometimes they can be used in their standard form also. So if we understand some of these blocks which are the most commonly used they would help us in designing new kind of scenarios as well as we can use them as such in some scenarios so it would save some of our design effort.

So today we are going to understand how to design very two common, two very common combination circuits. One is called decoder. Another is encoder. So we have very small glimpse of what decoder is in our previous lecture.

(Refer Slide Time: 01:20)

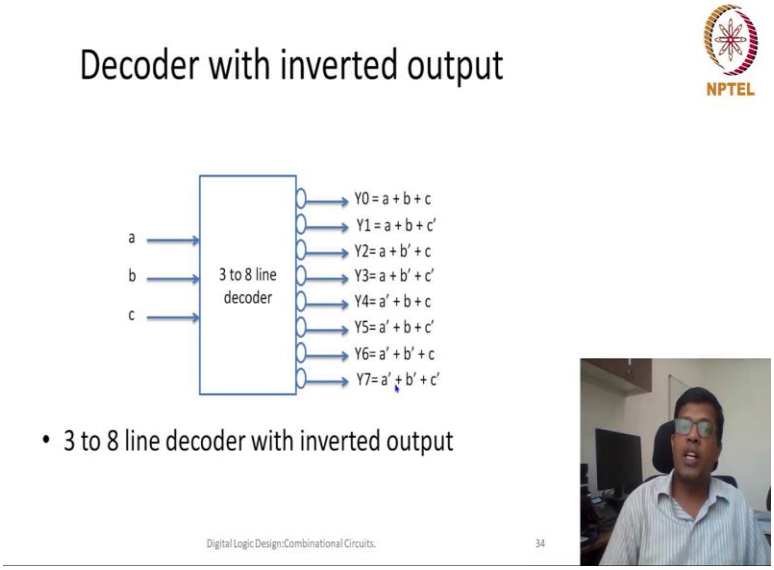


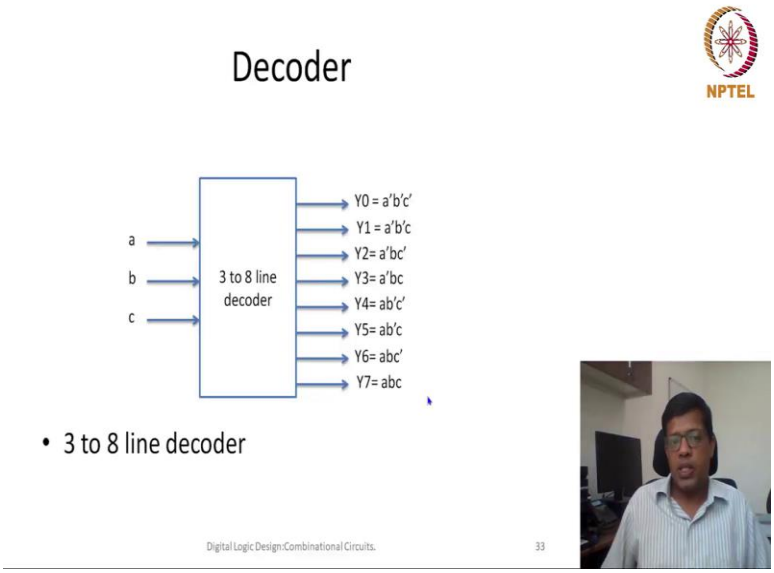
Decoder is like a demultiplexer where you are giving some n inputs. And output is 2's power n lines. And condition is that one only one of that line would be true or would be high at one of the time. All other would be low. So this because decoder is quite a common term, so to make things more narrow so we call, we are calling it n is to 2's power line decoder. That means there would

be 2's power n lines and the input is n. So there would be n signals at the input and 2's power n signals at the output, each of width 1.

So let us take an example of this 3 to 8 decoder in which there are 3 inputs and there are 8 outputs. And one of the output has to be true. So, for example, if input is 0 0 0 then Y0 would be true and all other would be false. So this we have anyway covered in our previous lecture also. So yeah, so this is a standard decoder circuit.

(Refer Slide Time: 03:42)





Now there could be some variations in this decoder circuit. One of the variation is that instead of 1 being high, 1 could be low and all other would be high. So for example if a, b, c, all three are 0

then Y0 would be 0 and all others would be 1. So it is essentially an inverted output. And we know how to, how to write function for inverted output. So one method we know that if, for example, this was my known inverted output, if I want to make it as an inverted output what should I do? I can use De Morgan's law. So in De Morgan's law what would happen is that a dash, b dash, c dash then Y0 dash will become a plus b plus c. So this is one method.

The other, we also understand that all of these represent minterms. If I want to make an invert of that, the corresponding maxterm would be, can be used as the inverted function. So either way, the inverted functions, if the output are inverted then this, the output would be represented by, Y0 would be represented by a plus b plus c. We know that a plus b plus c would be 0 if a, b, c all of them are 0 otherwise it is always going to remain 1. And this is the condition with this decoder with the inverted output that all other output would be 1 but only one would be 0 at one point of time, given like whatever is the input.

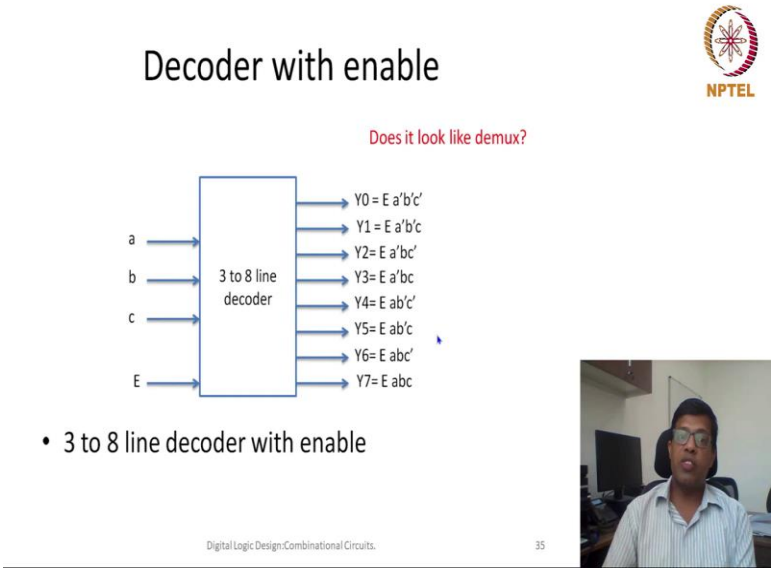
So why do people use inverted output? So mostly it is the two reasons. One is that, in some of the scenarios we understand that if the output is 1 or if the level is 1 then it will consume less power or less energy or it can also consume less delay. So here we see that in this particular decoder most of the output would remain 1 most of the time. So for example, if a, b, c is 0, 0, 0 then only this will remain 0. All others will remain 1.

So if the circuit scenario is such that if the input level or output level is 1 it is going to consume less power or delay then it is good to be, to have this inverted output. This is one. The other is, in some of the technologies like, we finally, all this gate would be implemented like a transistor. So these transistors could be CMOS. It could be BJT. It could be, and then there are logic families also, ECL, TTL etc.

So in some of the families it is preferred that if default output is 1 rather than 0, so some of the chips are fabricated with this inverted output. And this concept of inverted output or inverted input is quite generic. So if any of the block like decoder, encoder or multiplexer, demultiplexer, if there is a 0, if there is this circle before the output that means output is inverted. If the circle is before input then we can see that input is inverted.

Sometimes there are some special pins like enable, reset. If 0 is there, the circle is there that means that enable is active at 0, not at 1. So this is a very generic principle which would be applicable everywhere. Now this is about the decoder with inverted output.

(Refer Slide Time: 06:35)



Similarly there is another variation where we can have decoder with enable input. So if decoder is there with enable input the functionality would be something like this. That if enable is 1 then only decoder will perform its operation, otherwise if enable is 0 then all the output will be 0.

Now if I want to perform such an action where if enable is 0 then all the output has to be 0, and if enable is 1 then all the output has to be dependent, then the decoder operation will work, so that means I have to, whatever is this final output of my standard decoder then I can AND them with E or enable. So, or if I want to directly use my enable so I can have this 4 input AND gate where the input to these AND gates are corresponding to the minterm.

So one of the input is E or enable and other is, for example for Y0 it is a dash, b dash and c dash. For Y1 it is a dash, b dash and c, so and so forth for all others. So essentially E is being ANDed with each output or it can be given as a input of this, like we can have 4 input AND gate instead of a 3 input AND gate.

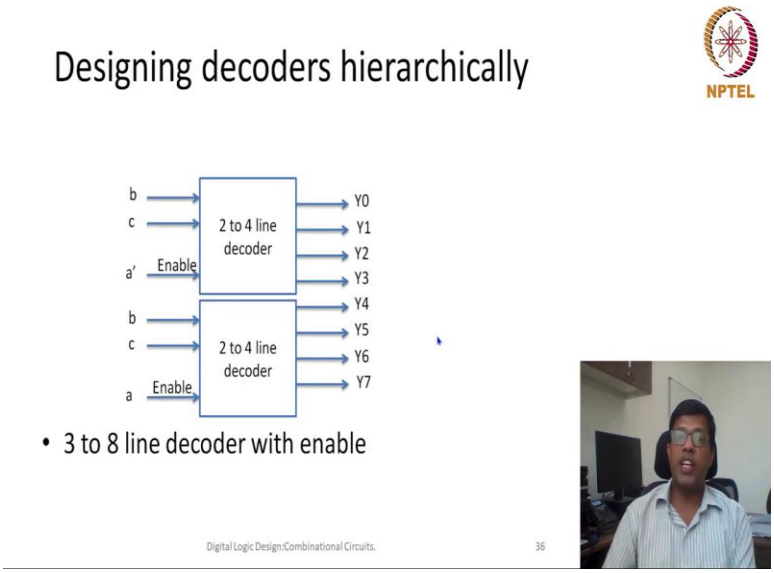
Now, so when we look at this circuit, when we look at this output functionality or these Boolean expressions we recall something. Oh yeah! This actually looks like a demultiplexer. The only

difference is, there instead of enable we were calling this as input, and this is a, b, c were called as control.

Now, in other words, if I have a decoder with the enable input I can call it a demultiplexer also. Or wherever I require to use demultiplexer I can use this decoder with enable input. So essentially this way if I make this as a standard circuit and if I keep enable all the time as 1 so then this whole circuit can work like a standard decoder which is 3 to 8 decoder without enable.

And if enable is 1, enable is a different input which is connected to enable then this would be a decoder with enable. Similarly if this enable input is connected to some other input which I want to demultiplex then this whole circuit can also work like a demultiplexer. See how different utilities can be formed with the same standard circuit or same, yeah, same standard combination.

(Refer Slide Time: 09:34)



Now there is another application of this decoder with enable. So this decoder with enable gives us a nice way to create a hierarchical decoder. So let us say I want to create decoder which is very large. Then I can use this decoder hierarchically. It is the same way we have used multiplexer in hierarchical way. And advantages are also similar. So you see, instead of having n input, so for example I want to so if I want to have large fan-in gate so I, because of this hierarchy I can break that large gate into smaller gates.

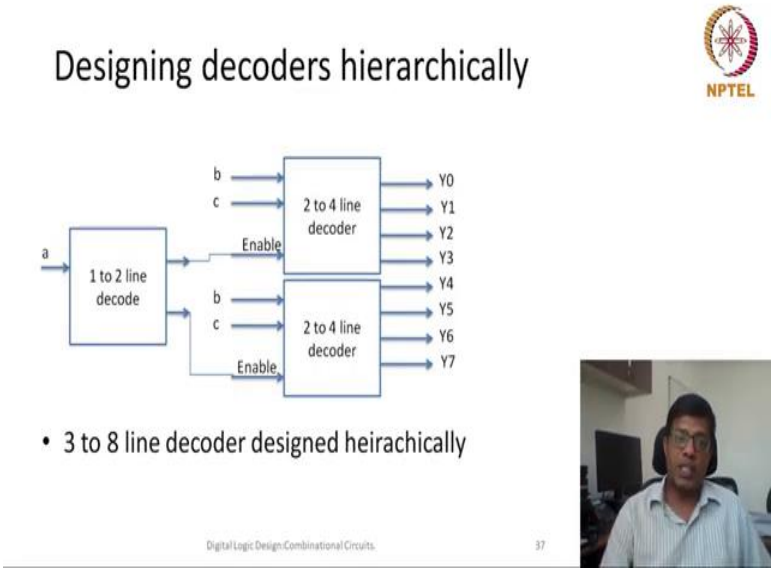
So here with an example what I am showing is that if this 8, 3 to 8 decoder I need to convert into hierarchically. So what I can do is I can create two 2 4 decoders so there would be two such decoders required. Then this enable input, for the first decoder I can give a dash as enable, and for the second one I can call this a as enable.

So first one, a dash is enable input for the second one a is enable input. So that means if a dash is 0 then this particular 1 would be active. So that means b and c would tell me about Y0, Y1, Y2, Y3 because a dash, a is 0, so this particular decoder is enable. Now if A is 1 that means this decoder will always give me 0 output and my output would be determined by the second decoder.

That means, either of Y4, Y5, Y6, Y7 would be active based on b and c values. So essentially what we are doing? We are keeping this, among this a, b, c; a was the most significant bit. So we are keeping that as enable. We are using that to generate my enable signal; more significant one I am using as a, to generate my enable signal. And the least significant one, because I am using 2 to 4 line decoder so that means I require 2 input. So the lower significant two bits I am using as the main input.

And I am keeping all of these in such a way that when, so that I can arrange my Y0 to Y7 in such a way that when a dash or basically when a is 0 then Y0, Y1, Y2, Y3; the output is Y0 Y1 Y2 Y3. And when a is 1 then the rest of the four are marked as Y4 Y5 Y6 Y7.

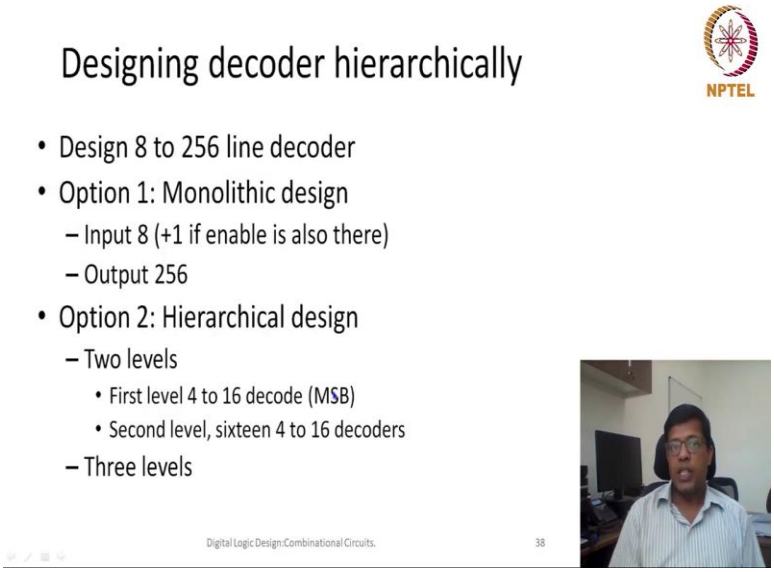
(Refer Slide Time: 12:14)

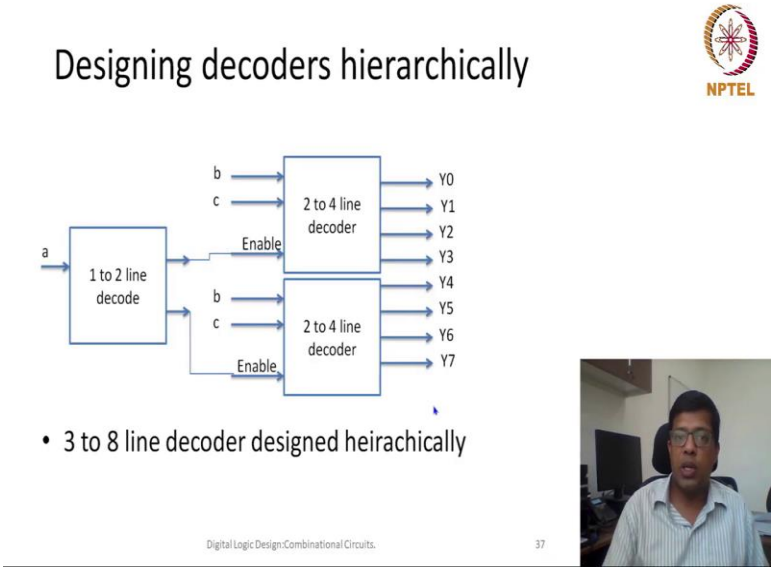


So the same thing to make it generic I can also put it like this. So this is 1 to 2 line decoder when a is the input. So that means there would be two outputs, 0 and 1. 0 means a was 0, and 1 means a was 1. And 0 again I give this to enable of first decoder, and 1 I am giving to enable of the second decoder.

So that means this a dash and a which I am giving as input was also essentially like a decoder, only 1 to 2 line decoder. Input was one bit and there were two outputs. Similarly if I want to create any kind of hierarchy I can create it like this. So these are 2 levels. This is first level. This is second level.

(Refer Slide Time: 13:06)





Now let us take a bigger example. Let us say I want to design 8 to 256 line decoder. This is more realistic scenario. We will not be using this kind of a circuit where we are trying to use only 3 to 8 decoder and we are using two levels. But it would be more concerned, more helpful when we are creating larger decoders. So we want to design 8 to 256 line decoder. Now otherwise what would happen in a monolithic design, in a single level design or in a monolithic design? There would be 256 AND gates. Each of these AND gate would have 8 inputs. If enable is there then there would be 9 inputs.

Now if I want to design a 9 input AND gate eventually that AND gate has to be again designed into multiple logic, multiple AND gates. That would increase the cost. Otherwise if I am using a monolithic design of one 9 input AND gate then the delay would be much more. So that is why it could be a better idea to go for a hierarchical design from the first place itself, could be multiple ideas here.

So, let us say I want to make a hierarchical design. I want to make it a two-level design. In two level design, one thing I can do is, at the first level there could be 4 to 16 decoder. And at the second level also there would be sixteen 4 to 16 decoders. So there are sixteen decoders, 4 to 16 at the second level. Total number of AND gates are going to be more.

So you see, in the second level itself there is going to be 16 cross 16 AND gates. But number of input to each AND gate would be 5. So because it is 4 to 16 decoders, 4 inputs plus one enable input, so there would be 5 input to all of these 256 AND gates at the second level. Similarly at the first level also, because it is 4 to 16 decoder, now in the 4 to 16 decoder there would be again 16 AND gates with 5 inputs each.

So because 5 input again will be large so instead of two levels we can again go for 3 levels. So whenever we are breaking things at multiple level what would happen that your MSB will be go in the first level, then your next level of bits would go into second level and the next level of bits will go in third level. So these bits will be divided like this. You can always ask a question that why not to divide bits in the other way? So why cannot we have, let us say, bits in a reverse order. We keep our LSB.

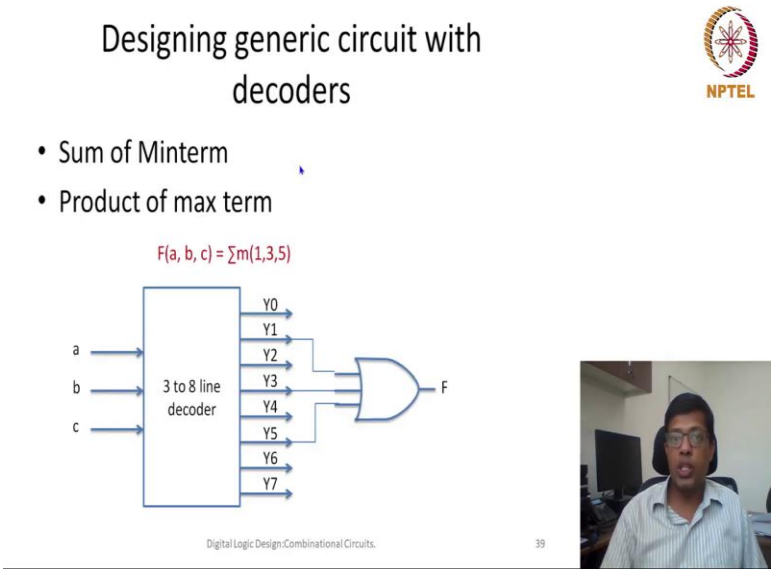
So this can be understood from this example. So let us say, instead of A if this has been C, least significant one. And A and B we are keeping it here. A and B we are keeping it here. What does it mean is then this would have been because A is the least significant one. So A means 0. So whenever A is 0 then these would be up.

So that means this output would have been Y0 Y2 Y4 Y6.And when A is 1, then this particular decoder would be active. So that means these output would be Y1 Y3 Y5 Y7. So although other combinations are possible, there is no harm in that, but the issue would be that you have to jumbled up these lines. So this is the simplest example.

The example is like this where you are creating 8 to 256 line decoder then this jumbling would be so much that we will be crazy in solving them back or in basically strengthening, straightening them back. So that is why it is a good idea that we keep our MSB at the first level and then going for the lower significant bits to the next levels. And then we can arrange them in such a way that we can simply call that as a decimal number 0, 1, 2, 3, 4, 5, 6 up to 256, 255 sorry. So this way these decoders can be designed hierarchically so that they are efficient and there is another advantage of designing things hierarchically.

So some of these decoders like 3 to 8 decoders, so or 4 to 16 decoders, they are also available as package ICs or package chips. So because they are available as packaged chips means pre-fabricated chips so we can use them in bread board. Because they are pre-fabricated they are available as IC so we can use them in bread board and we can use them directly. So if we are able to use them directly then we can use multiple of them to create even larger pieces. So that is why it is good to have generic component at lower level so that we can create a higher component or a better component, bigger component using them.

(Refer Slide Time: 18:44)



So there is also now, I will see one more advantage, one more application where we can use these decoders. So these decoders can essentially be used, like multiplexers these decoders can also be used to design any generic circuit.

So in previous module we have seen any generic circuit means either it would be sum of product or product of sum. And if it is sum of product then I can also further generically tell it as sum of minterms, classical sum of product or canonical sum of product form would be sum of minterms. Or similarly canonical product terms could be canonical product of maxterms. So if those sum of minterms and maxterms are there then also we can create it.

So it is a pretty straight forward to understand. You see that a, b, c was the input. And all of these output were representing only one minterm because they were only representing, each output was representing only one minterm so if, whatever is the function, let us say function was this. F a b c we want to have a sum of minterm 1, 3 and 5. Then I can have an OR gate which is doing an OR with line number 1, line number 3 and line number 5. So I can have this function.

Similarly product of maxterm can also be there. So there the decoder would be with inverted outputs. And if it is with inverted output then each of this line would be representing one maxterm. And then in the end we can have AND gate. This AND gate can represent that function which is sum of product. So if it is in the form of inverted output, my decoder is in the form of inverted output then I can design it using product of maxterms. Or if it is as a standard form then I can use it as a sum of minterms. So this way decoder can be used, yeah to design or to design any generic circuit. I require additionally one OR gate or one AND gate. So this is about decoders.