Switching Circuits and Logic Design

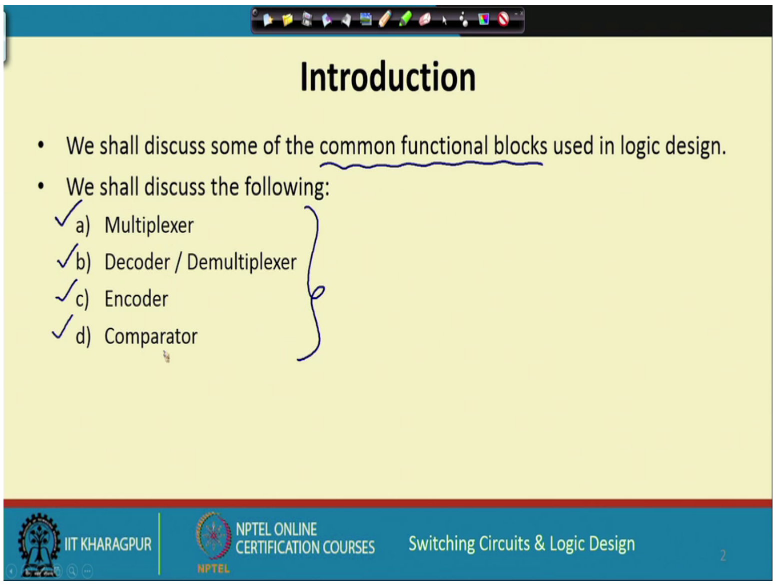
Prof. Indranil Senguta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 24
Logic Design (Part -I)

In the last few lectures if we recall, we have been talking about the design of different kinds of adders and subtractors stuff like that which are very important for the design of more complex systems. So you know addition is a very integral part of most competitions and systems that we design so in this lecture from today onwards we shall be discussing some of the other basic building blocks that are available for designing functions implementing functions and so on. So today we shall be starting our discussion on general logic design.

So, logic design is somewhat general in the sense that we are not talking about specific applications like addition or subtraction, but we shall be talking about some generalized building blocks which we can use to implement any arbitrary function let us see what we plan to cover.

(Refer Slide Time: 01:26)



Introduction

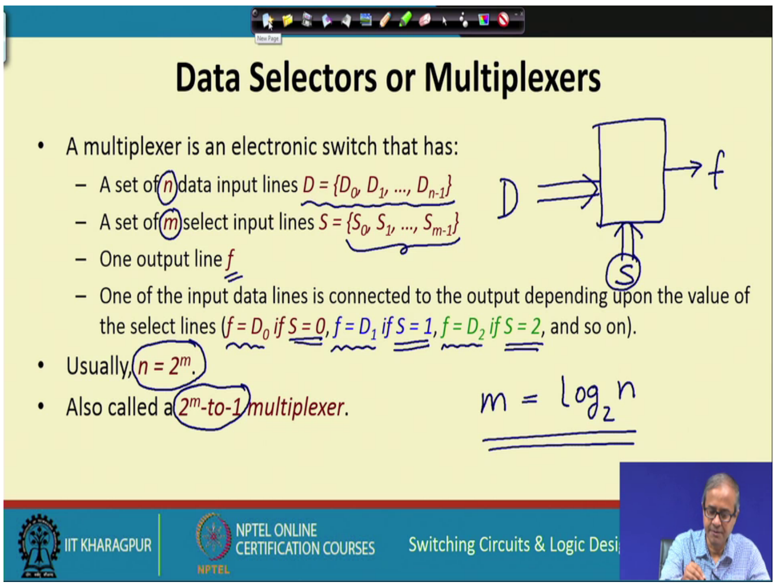
- We shall discuss some of the common functional blocks used in logic design.
- We shall discuss the following:
  - ✓ a) Multiplexer
  - ✓ b) Decoder / Demultiplexer
  - ✓ c) Encoder
  - ✓ d) Comparator

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Design

So, as I said that we shall be talking about some of the common functional blocks which we normally use in the design of digital systems. Now specifically the kind of functional blocks that we shall be considering are multiplexer which we have already seen earlier see earlier when you discussed about the CMOS switches the transmission gates we mentioned how we can very efficiently implement multiplexers using those transmission gates.

So, we shall be revisiting multiplexers and what kind of applications and designs we can do using that then we shall be talking about something called demultiplexers and decoders then encoders and comparators. Now all these blocks are quite important in the design of more complex digital systems. So for those of you will be going into the design of complex systems you will find that these modules will be used repeatedly in various scenario.

(Refer Slide Time: 02:37)



Data Selectors or Multiplexers

- A multiplexer is an electronic switch that has:
  - A set of n data input lines $D = \{D_0, D_1, \ldots, D_{n-1}\}$
  - A set of m select input lines $S = \{S_0, S_1, \ldots, S_{m-1}\}$
  - One output line f
  - One of the input data lines is connected to the output depending upon the value of the select lines ($f = D_0$ if $S = 0$, $f = D_1$ if $S = 1$, $f = D_2$ if $S = 2$, and so on).
- Usually, $n = 2^m$.
- Also called a 2^m-to-1 multiplexer.

$m = \log_2 n$

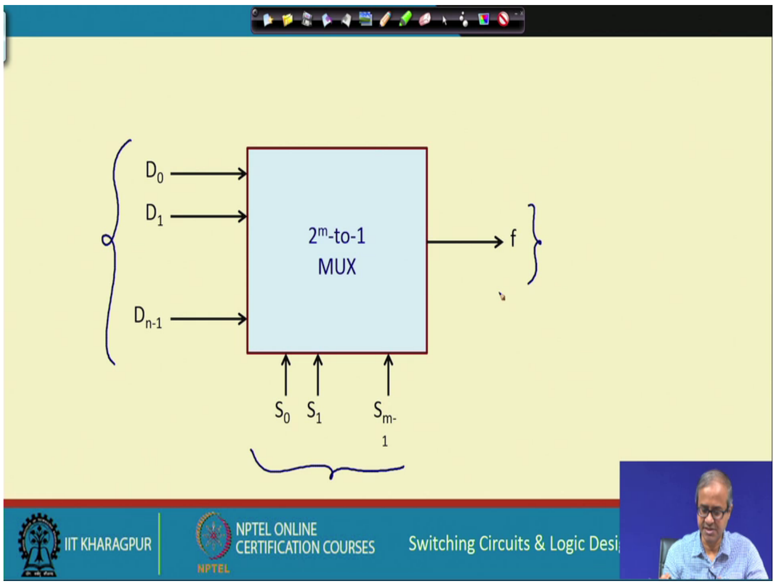
D
f
S
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desi

So, we start with our discussion on multiplexers which are also called data selectors; now why do we use the term data selector for a multiplexer, let us try to understand if you recall what is a multiplexer a multiplexer is a block if you think it as a black box which has a number of inputs and 1 output depending on some select inputs one of the input line is selected and it goes to the output.

So, in the sense we are selecting one of the several data inputs at a time that is why this is also called a data selector. So let us talk in a slightly general term so let us consider multiplexer which has n data inputs let us call them as D 0 up to D n minus 1 and let us also have m number of select lines. So we have a multiplexer as a box so in the input we have this D the n number of lines and then we have the select line S. And we have an output single output f, this is how a multiplexer is there is one output line f and as we know the way multiplexer works depending on what we apply on the select line one of the data inputs is selected and it goes to the output; so if we consider this m bit select line as a binary number.

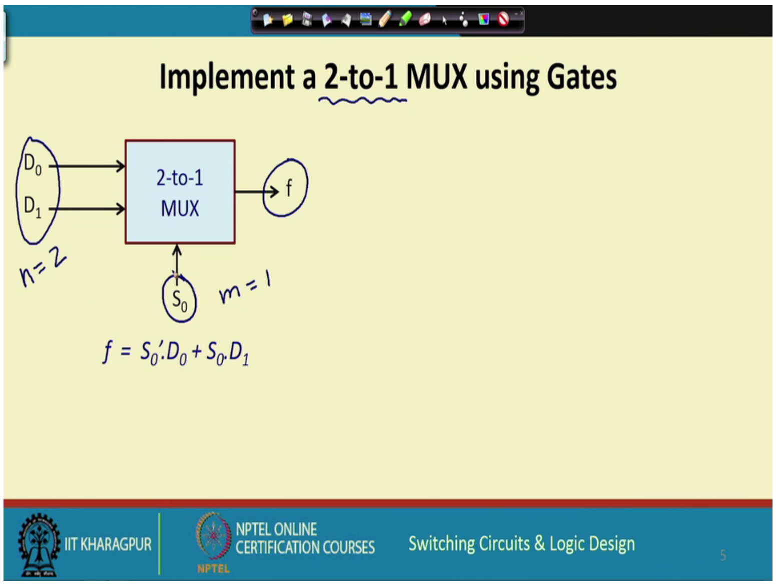
If we consider the decimal equivalent so when S is 0 the first input D 0 is selected we will have f equal to D 0, D 0 will go to f. If S is 1 then D 1 will be copied to f, f equal to D 1; if S equal to 2 similarly D 2 will be copied and so on. Now usually there is a relationship between n and m the number of select lines should be large enough so that any of the n inputs can be selected, so this condition can be expressed as n equal to 2 to the power m or alternatively we can write m equal to log to the base 2 of n these two are equivalent right. Now because the number of inputs is 2 to the power m and there is a 1 output we also referred to this kind of a multiplexer has a 2 to the power m to 1 multiplexer fine.

(Refer Slide Time: 05:41)



So, this is how this multiplexer we will look like there are n inputs, there are m number of outputs S 0 up to S n minus 1 and there is a single output as I have just said ok. This is how a multiplexer looks like.

(Refer Slide Time: 06:04)



Implement a 2-to-1 MUX using Gates

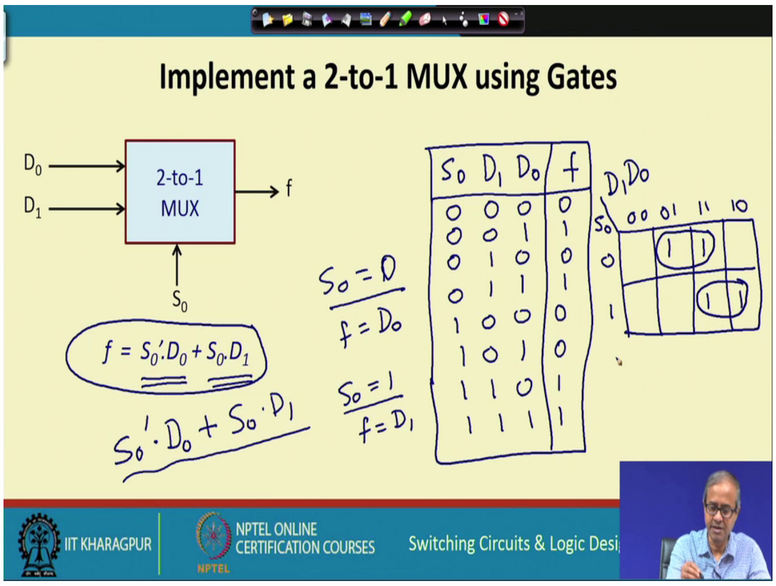
D_0
D_1
2-to-1 MUX
f
S_0
n=2
m=1
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Design

$f = S_0' . D_0 + S_0 . D_1$

Now, let us talk about specific small size multiplexer for specific values of m and n and let us see how we can design or implement such multiplexers, you recall earlier when you are discussing CMOS logic we showed how to design a multiplexer using transmission gates. But here we shall be showing the designs using basic logic gates like AND, OR, NOT, NAND, NOR etcetera ok. Let us see let us start with the simplest and smallest kind of multiplexer which is 2 to 1 multiplexer there are 2 inputs D 0 and D 1 there is 1 output, because there are only 2 inputs which is n equal to 2 we required m equal to 1, 2 to the power m equal to n ok.

So, there will be a single select input alright, here I have shown an expression before coming to this let us try to see how this multiplexer works.

(Refer Slide Time: 07:12)



Implement a 2-to-1 MUX using Gates

$$f = S_0'.D_0 + S_0.D_1$$

$$S_0' \cdot D_0 + S_0 \cdot D_1$$

$S_0 = 0$, $f = D_0$

$S_0 = 1$, $f = D_1$

S_0	D_1	D_0	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S_0 \ D_1D_0	00	01	11	10
0		1	1	
1			1	1

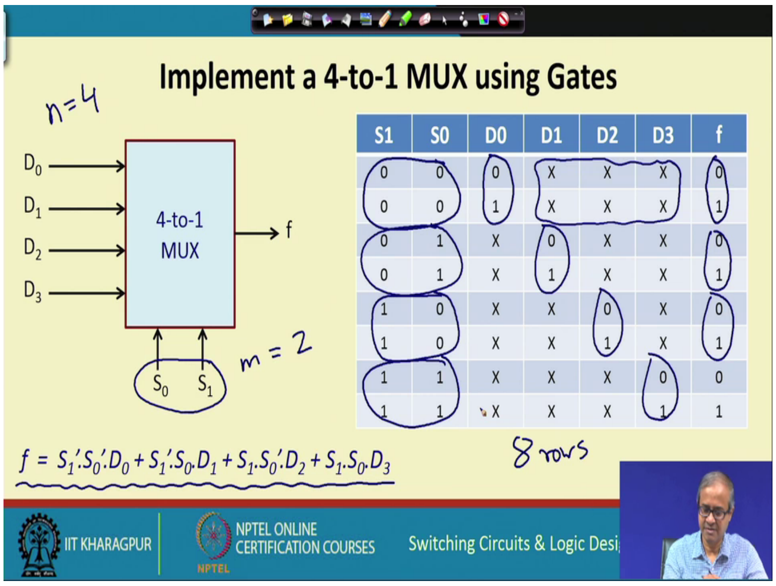
D_0
D_1
2-to-1 MUX
f
S_0
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desi

let us try to construct the truth table, so there are three select lines three inputs S 0 let us show D 1 first and D 0 then f is the output. So there are 8 possibilities 0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0 and 1 1 1. So according to the definition of multiplexer if S 0 equal to 0 if S 0 equal to 0 then we should have f equal to D 0, D 0 will be selected; so the first four rows show S 0 equal to 0, so whatever is in D 0 that will go to f 0 1 0 1. Similarly when S 0 equal to 1 then D 1 will be selected so the last four rows show this case when S 0 equal to 1 so D 1 0 0 1 1, this will be selected so this is the truth table of the multiplexer.

Now, if we construct a karnaugh map for example, and I try to minimize this let us see on this side let us show D 1 and D 0 this is 0 0, 0 1, 1 1, 1 0 and in this side we show S 0, 0 and 1 the true minterms are 0 0 1 0 0 1 here 0 1 1, 0 1 1 here then 1 1 0, 1 1 0 here and 1 1 1, 1 1 1 so you see here the minimum set of cubes will be this. And this so for that the first two indicate S 0 bar and D 0 you see in this expression S 0 bar and D 0. And the other cube is S 0 and 1 1 and 1 0 is D 1 S 0 D 1 so you see after constructing truth table and minimization we have arrived at this expression, but we would have arrived at this expression without going through all this minimization procedure just by thinking about the functional description for multiplexer. What is a multiplexer if S 0 is 0 we will get D 0 if S 0 is 1 we will get D 1.

So, how do we express it if S 0 is 0 which means S 0 bar we get D 1 which is and D 0 we get D 0 and D 0 or if S 0 is 1 which you write as S 0 we get D 1 and D 1. So we can write down this expression straight away without going through on this minimization right let us move on.

(Refer Slide Time: 10:35)



Implement a 4-to-1 MUX using Gates

S1	S0	D0	D1	D2	D3	f
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

8 rows

$$f = S_1'.S_0'.D_0 + S_1'.S_0.D_1 + S_1.S_0'.D_2 + S_1.S_0.D_3$$

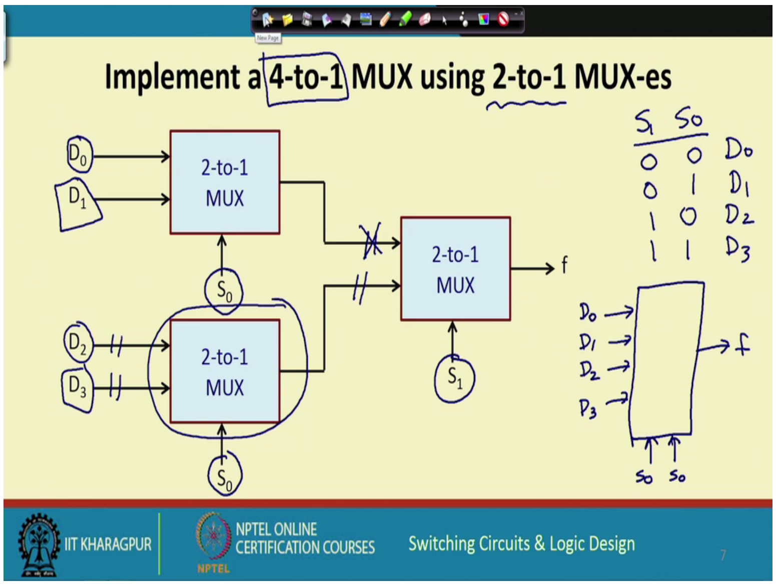
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Desi

Let us talk about a larger multiplexer or 4 to 1 multiplexer for a 4 to 1 multiplexer the number of inputs n is 4. Therefore, number of select lines m will be log to the base 2 of 4 which is 2. So S 0 and S 1 so in the same way depending on the values of S 0 S 1 1 of the inputs will be selected.

So, just following the principle I mentioned for 2 to 1 mux we can straight away write down the sum of products expression for the output function how do you write from the functional behavior if S 0 S 1 is 0 0 then we should get D 0 we write it like this S 1 bar S 0 bar indicates both are 0 0 then this will be 1 1 and D 0. So if S 0 S 1 as 0 0 and D 0 then this f will be equal to D 0 right if it is 0 1; that means, S 1 is 0 bar and S 0 then D 1 if it is 1 0. That means, S 1 is 1 S 0 is 0 then D 2 when both of them are 1 1 1 then D 3 ok. Now you see this is a 6 input function D 0 to D 3 and S 0 to S 1.

So, if I construct a truth table in the normal way there should be 2 to the power 6 or 64 rows in the table the truth table will be too big to draw, so again because of the functional characteristics of a multiplexer we can show the truth table in a much compact way using some entries as do not cares how do we do it let us see. Here we do it in this way you see first two rows of this truth table indicates the select line 0 0, so you know when this 0 0 then D 0 will be selected so we apply D 0 0, D 0 1 so the output will be f 0 f 1. Now under this condition the values of D 1, D 2, D 3 are all don't cares it is only D 0 which will be copied to f similarly when I apply 0 1 then D 1 will be copied the others will be don't cares the if I apply 1 0 D 2 will be copied. If I apply 1 1 then D 3 will be copied so you see we can very concisely express this truth table in only 8 rows this is possible because of the nice functional behavior of a multiplexer right.

(Refer Slide Time: 13:36)



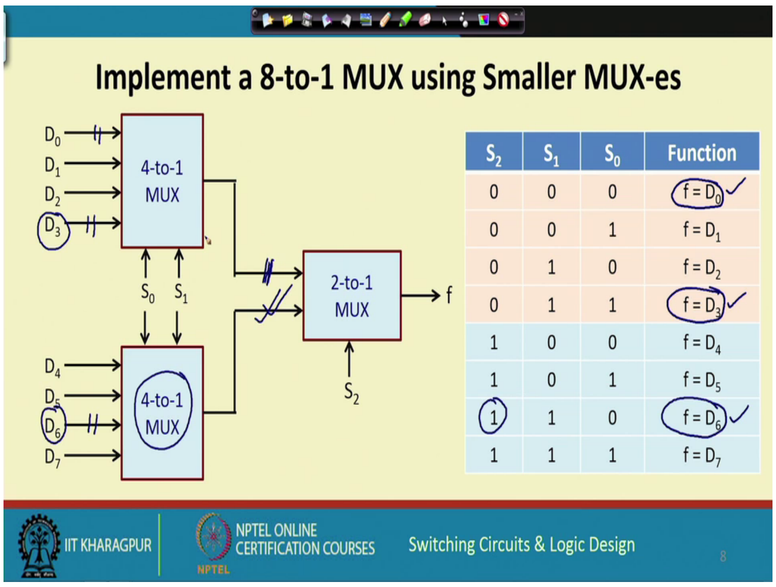
Implement a 4-to-1 MUX using 2-to-1 MUX-es

Let us now see how we can built multiplexers, let us say we have smaller multiplexers available to you and we shall be talking about how we can built larger multiplexers using this smaller multiplexers. Let us take a couple of examples the first example we take is suppose we have 2 to 1 multiplexers available to us, but you require a 4 to 1 mux; so what is the 4 to 1 mux we recall again there will be 4 inputs D 0, D 1, D 2 and D 3 the output will be f and there will be two select lines S 0 and S 1. Now you see how we have designed this multiplexer network we have used 3 multiplexers in 2 in the first level we are using 2 multiplexers and both of them we are using S 0 as the select line whereas, for the next one we are using S 1 as the select line.

Let us try to see the different scenarios, let us say S 1 is 0 suppose we have applied 0 and 0 what will happen; first is let us look at the last multiplexer first if S 1 is 0 then the first input will be selected this one. So we need not have to see what this multiplexer is doing, because it is not been selected the first multiplexer is being selected at S 0 is also 0 so S 0 0 means ultimately D 0 gets selected. So if we S 1 and S 0 I am selecting D 0, if it is 0 1 same way if it is 0 this is selected and if S S 0 is 1 then now D 1 is selected, so D 1 will go out D 1. If I apply 1 0 if S 1 is 1 then it is not this, but this line will get selected and it is the lower multiplexer which will get selected because S 0 is 0 here.

So, now this D 2 get selected and D 2 will go out D 2 and if I finally, apply 1 1 this S 0 is also 1. So now, D three will be selected. So you see that it works perfectly as a 4 to 1 multiplexer so if I have only 2 to 1 multiplexers available to us so we can very easily construct a forth one multiplexer let us take another example.

(Refer Slide Time: 16:38)


Implement a 8-to-1 MUX using Smaller MUX-es

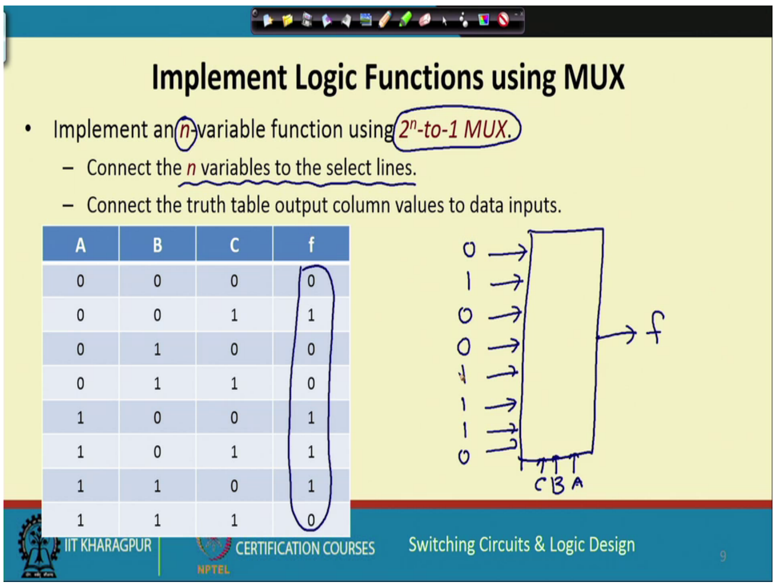
S_2	S_1	S_0	Function
0	0	0	$f = D_0$ ✓
0	0	1	$f = D_1$
0	1	0	$f = D_2$
0	1	1	$f = D_3$ ✓
1	0	0	$f = D_4$
1	0	1	$f = D_5$
1	1	0	$f = D_6$ ✓
1	1	1	$f = D_7$

Let us say we now want to have an 8 to 1 multiplexer and we assume that we have smaller multiplexers available with us smaller means we have 4 to 1 multiplexers and also 2 to 1 multiplexers, you see the network look similar we have used multiplexers in two levels which 2 for 2 1 multiplexers in the first level which are selected by S 0 and S 1 with the first 4 data inputs connected here. The last four data inputs connected here, the outputs of which are feeding to the second multiplexer which is selected by S 2.

Let us try to understand the operation with respect to the truth table, this is the functional truth table I am showing with respect to the select lines S 2, S 1, S 0 if it is 0 0 0 then D 0 is supposed to be selected 0 0 1 D 1, 0 1 0 D 2 and so on 1 1 1 D 7. Let us look at it one by one let us consider the first scenario for select lines are 0 0 0 if S 2 is 0 this will be selected, if S 1 S 0 are both 0 0 then this will be selected which is D 0 is fine; let us take another case let us say talk about this to this one.

Because S 2 is again 0 so again this is selected, but S 1 S 0 is 1 1, 1 1 means 3 this one will be selected D 3 so it is D 3 it is fine. Let us take another example let us consider this here S 2 is 1 right, S 2 is 1 so this 1 will be selected the second input. That means, this multiplexer and S 1 S 0 we have applied 1 0 or 2, 2 means 0 1 to this 1 will be selected which is D6. So you see this works exactly like a 8 to 1 multiplexer. So in this way if we have any smaller multiplexers given to you and were asked to design a larger multiplexers you can systematically construct the multiplexer network like this so I have shown 2 small examples you can think and work out larger examples similarly fine.

(Refer Slide Time: 19:20)



Implement Logic Functions using MUX

- Implement an n-variable function using 2^n-to-1 MUX.
  - Connect the n variables to the select lines.
  - Connect the truth table output column values to data inputs.

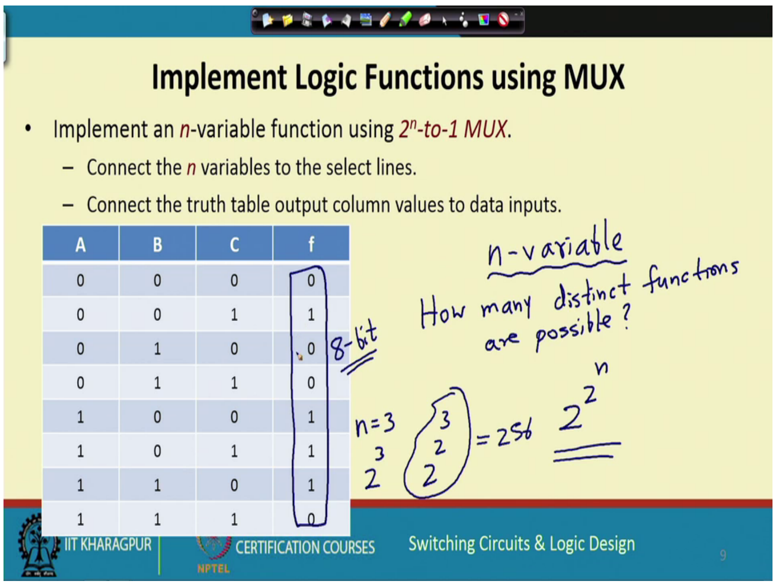
A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Now, let us talk about how we can implement logic functions using multiplexers. First let us see that if I have a 2 to power n to 1 multiplexer we want to implement and invariable function how do we do it. It is very simple here I am showing an example of a 3 variable function so I need a multiplexer with 8 inputs and 3 select lines so what we are saying is that: we shall be connecting the input variables to this select lines directly so we have A B C we connect them like this A B and C. So what will happen if A B C is 0 0 0 the first line will be selected, if it is 0 0 1 the second line will be selected, 0 1 0 the third line will be selected and so on.

So what do we do we look at the output column of the truth table and simply copy this and apply it to the inputs 0 1 0 0 1 1 1 and 0 the output is f. You see the implementation of any function using multiplexer is very trivial and the same multiplexer can be used to implement any arbitrary truth table if you just change this input 0 1 assignment.

Basically, you are applying the truth table whatever it is to the input of the multiplexer and the multiplexer implements the required function. This you can say is something like a programmable logic function generator the input you change without any change in the circuits circuit is the same still that multiplexer you only change that 0 1 pattern in the input you get another truth table and that corresponding function gets implemented right so it is very simple. Now in this context let me just ask you one thing means one thing we did not mention possibly earlier let us suppose let us do this.

(Refer Slide Time: 21:47)



Implement Logic Functions using MUX

- Implement an *n*-variable function using 2^n*-to-1 MUX*.
  - Connect the *n* variables to the select lines.
  - Connect the truth table output column values to data inputs.

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

8-bit

n-variable

How many distinct functions are possible?

$n=3$

2^3

$2^{2^3} = 256$

2^{2^n}

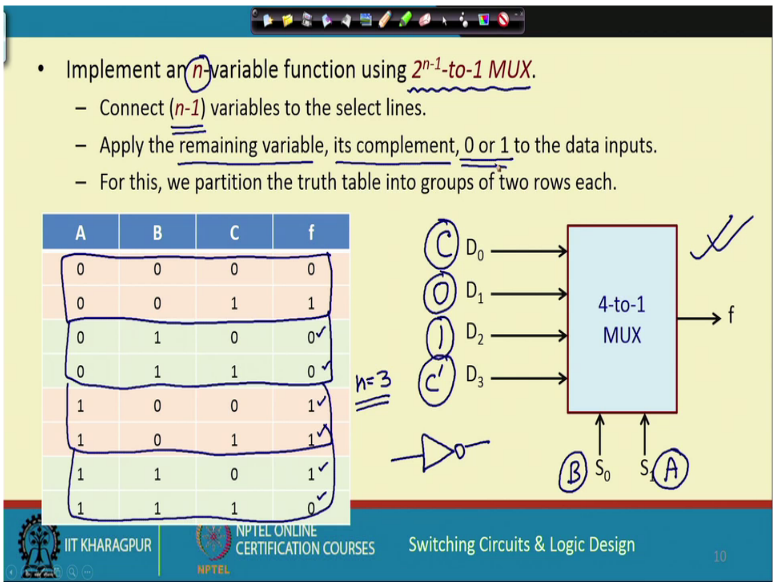
Let us suppose we have an n variable function, we have an n variable function so my question is how many distinct functions are possible, functions are possible. Well, here we are raising this question, because we are talking about programmability just by changing this input 0 1 pattern I can change the function, I can go for one function to another. So my question is in 3 variables for example, total how many functions are possible let us think in this way, think about this truth table this is a 3 variable function for n equal to 3, so I have my output specification here this output column specifies the function. So how many bits are there in the output 8, there are 8 rows how do you get 8;

2 to the power 3. Now you consider this as an 8 bit number so how many possible 8 bit numbers can be there, now you have learnt about binary numbers so in 8 bits how many possibilities can be there 2 to the power 8.

So, the total possible number of this 8 bit vectors you can have will be 2 to the power 8 means 2 to the power 2 to the power 3, in general for n variable it will be 2 to the power 2 to the power n right. This is how you get the total number of functions that are possible in an n variable function like this here for the multiplexer case also for the same three means 8 to 1 multiplexer you can implement so many functions 2 to the power 8 or 256 this is 256 functions, because there are 256 kinds of truth tables we can generate by changing this 0 1 pattern.

Now, let us try to improve upon this for an n variable function we were require in 2 to the power n to 1 multiplexer now what you are saying that we can do a little better how; we are saying for an n variable function let us use a smaller multiplexer we can do with 2 to the power n minus 1 to 1 multiplexer this is sufficient.

(Refer Slide Time: 24:17)



- Implement an *n*-variable function using *2^{n-1}-to-1 MUX*.
  - Connect (*n-1*) variables to the select lines.
  - Apply the remaining variable, its complement, 0 or 1 to the data inputs.
  - For this, we partition the truth table into groups of two rows each.

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0✓
0	1	1	0✓
1	0	0	1✓
1	0	1	1✓
1	1	0	1✓
1	1	1	0✓

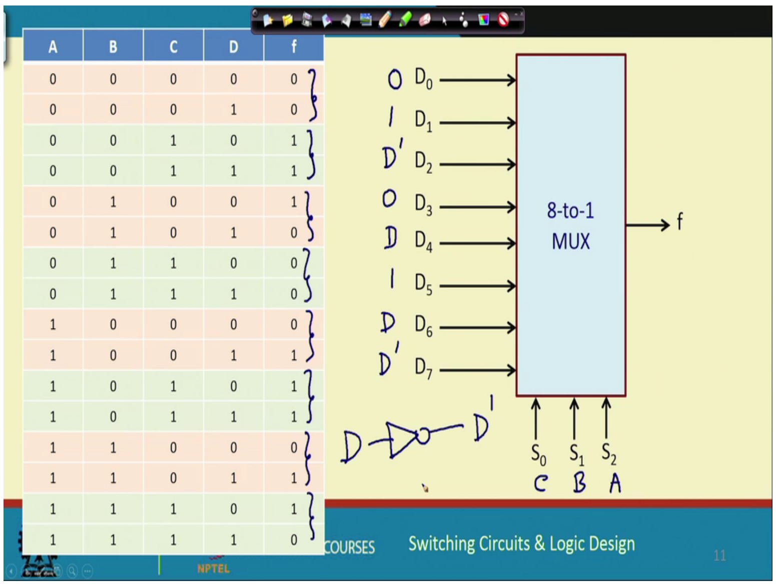
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Switching Circuits & Logic Design 10

Now here again I take that same example of a 3 variable function so n equal to 3, so how big a multiplexer I require 2 to the power 2 to 1; that means, four to 1 there are 4 inputs, 1 output and 2 select lines. Now the question is how do I apply this inputs to this multiplexer because in the earlier case there are 8 inputs straightaway we are applying the output column to there.

So, the procedure that we follow is something like, this you say it says connect n minus 1 variables to the select lines there are 3 variables A B C let us say I connect A here and B here two of the variables I connect to the select lines C is still left right and with partition the truth table into groups of 2 rows like we first consider the first two rows. Let us look at the output and the variable that we have not yet assigned we see that C is 0 f is 0 if C is 1 f is 1. That means, f and C are the same. So in the first input for 0 0 case you see A B is 0 0 we directly connect the variable C to this first data input, they directly connect C here. Let us look at the second row pair of rows now here you see that the outputs are 0 and 0 irrespective of the value of c it is always 0.

So, you apply directly a value 0 here in the second one where A B is 0 1, next two rows where A B is 1 0 you see irrespective of C the output is always 1, 1 1 so you apply a 1 here and the last pair you see for 1 1 case D 3 will be selected if C is 0, f is 1, if c is 1 f is 0; that means, the inverse of this. So I will use or not get an applies C bar to this, this will be my design. So you see if I do this then maximum I will be requiring one additional not gate to generate this C bar and of course, this constant 0 1 I may have to apply and C I may have to apply, that is why it is mentioned apply the remaining variable it is complement C bar if required and 0 or 1 at the data inputs right. Let us take another example.

(Refer Slide Time: 27:44)



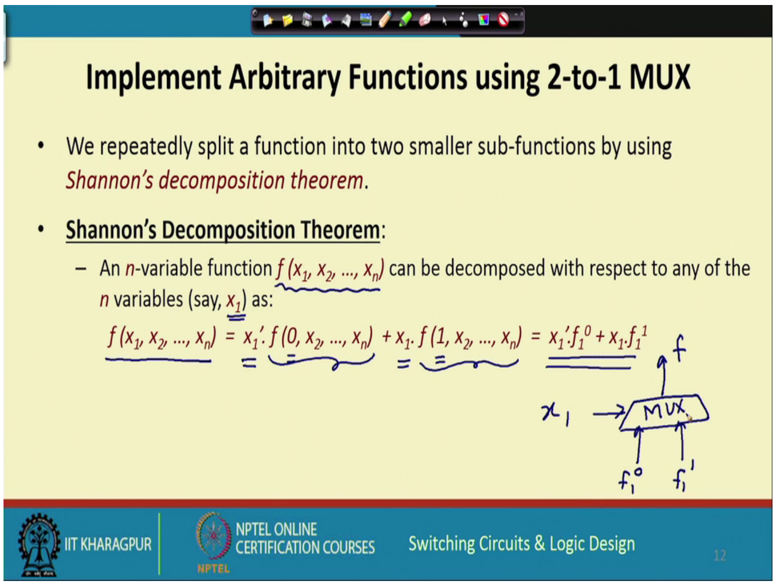
A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

This is a bigger example I consider a 4 variable function for a 4 variable function I will be requiring 2 2 2 to power 3 to 1 multiplexer 8 to 1 multiplexer.

So, this ABC this we are applying here A B and C and we are applying the same rule consider the first two rows, first two rows you see output is 0 0 irrespective of D, now D is the last variable A B C we have already applied A B C 0 0 0 D 0 is selected so we apply a constant 0 here, next two rows output is 1 and 1 always 1 you apply a constant 1 here. Next 2 rows if D 0 this is 1 D is 1 this is 0 so apply D bar D bar, next two again 0 and 0 I apply 0, next two if D is 0 f is 0 D is 1 f is 1 I apply D directly, next two it is 1 and 1 I apply 1, next two again if it is 0 is 0 1 is 1 I apply D, next two again 0 1 1 0 I apply D bar this will be my circuit.

So, so here we are inserting the truth table and we are applying either 0 1 the last variable here D or the compliment of D D bar to the input so I need a multiplexer and just to generate D bar I need a not gate that is all. So you see using a multiplexer we can so conveniently design any arbitrary logic functions. So, we sometimes called this kind of multiplexer as generalized function generator because of this property.

(Refer Slide Time: 29:50)



Implement Arbitrary Functions using 2-to-1 MUX

- We repeatedly split a function into two smaller sub-functions by using *Shannon's decomposition theorem.*
- **Shannon's Decomposition Theorem**:
  - An *n*-variable function $f(x_1, x_2, \ldots, x_n)$ can be decomposed with respect to any of the *n* variables (say, x_1) as:

$$f(x_1, x_2, \ldots, x_n) = x_1'.f(0, x_2, \ldots, x_n) + x_1.f(1, x_2, \ldots, x_n) = x_1'.f_1^0 + x_1.f_1^1$$

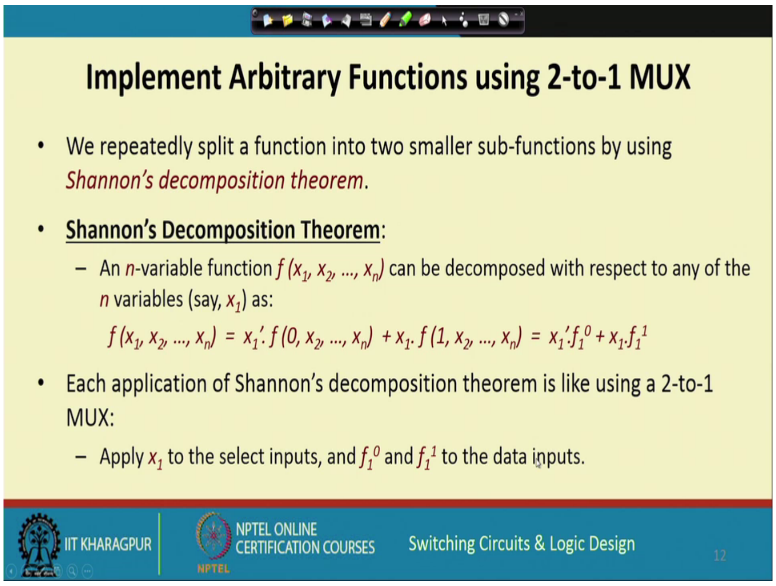
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design 12

Let us look into one more thing that we can implement arbitrary function using 2 to 1 multiplexers only there is a principle called a Shannon’s decomposition theorem. This we shall be looking again later in some later lectures so what is Shannon’s decomposition theorem say this says that if I have an invariable function.

Let us say f the input variables are x 1 to x n we can decompose this function with respect to any of these variables. Let us say we are decomposing with respect to x 1 so if I decompose with x 1 what will happen this function can be written as x 1 bar and this same function where x 1 is replaced by 0 or x 1 and the same function with this x 1 replaced by 1 this is the Shannon's decomposition theorem. This you can do with respect to any of the variable and in a compact form we write it as x 1 bar f 1 0 this is called f 1 0 which means it is with respect to x 1 we are setting x 1 to 0 and f 1 1 means this is x 1 they are setting it to 1.

Now you see if I do it in this way you think in this way suppose I have a multiplexer here this is a 2 to 1 multiplexer, I have 2 inputs, I have 1 output and I have a select line. Suppose, I am connecting x 1 to the select line the output represents my function f now if I connect this f 1 0 here and if I connect f 1 1 here so does not this multiplexer implement this if x 1 is 0 then f 1 0 if x 1 is 1 then f 1 1. So every time you do the Shannon's decomposition with respect to a particular variable we can use a multiplexer to mimic that or to realize that this is the basic idea.

(Refer Slide Time: 32:17)



Implement Arbitrary Functions using 2-to-1 MUX

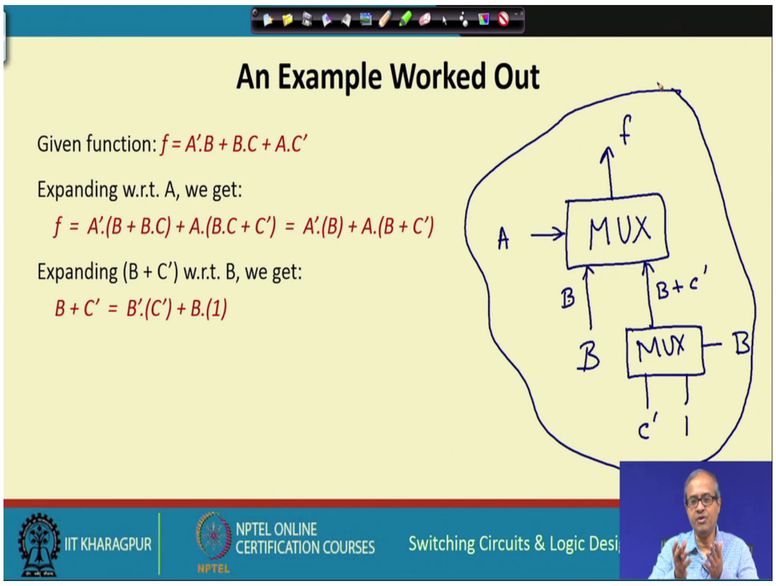
- We repeatedly split a function into two smaller sub-functions by using *Shannon's decomposition theorem*.
- **Shannon's Decomposition Theorem**:
  - An *n*-variable function $f(x_1, x_2, ..., x_n)$ can be decomposed with respect to any of the *n* variables (say, x_1) as:

    $f(x_1, x_2, ..., x_n) = x_1'.f(0, x_2, ..., x_n) + x_1.f(1, x_2, ..., x_n) = x_1'.f_1^0 + x_1.f_1^1$
- Each application of Shannon's decomposition theorem is like using a 2-to-1 MUX:
  - Apply x_1 to the select inputs, and f_1^0 and f_1^1 to the data inputs.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design 12

So, what are we saying each applications of Shannon's decomposition theorem as I said is like a 2 to 1 multiplexer, where the variable is applied to the select inputs and this f 1 0 and f 1 1 is applied to the data inputs.

(Refer Slide Time: 32:33)



An Example Worked Out

Given function: $f = A'.B + B.C + A.C'$

Expanding w.r.t. A, we get:

$f = A'.(B + B.C) + A.(B.C + C') = A'.(B) + A.(B + C')$

Expanding (B + C′) w.r.t. B, we get:

$B + C' = B'.(C') + B.(1)$

f
A
MUX
B
B+c'
B
MUX
B
c'
1
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desi

Let us take a simple example consider a function like this a 3 variable function A bar B BC plus AC bar.

Suppose we apply Shannon decomposition with respect to variable A, so what is the Shannon decomposition A bar and substitute A as 0 if A is 0 the last term will vanish this A bar is 1 only B B plus BC B plus BC plus A and substitute A by 1 the first term will vanish so it will become 0. A by 1 this will become C bar BC plus C bar and if we apply minimization B plus BC is nothing, but B BC plus C bar is nothing, but B plus C bar. So this decomposition you can express like this you can have a mux where we have applied A select input a the output implements f and you will have the 2 inputs. In the first input I am just writing the function you have this function B and the other input a of the function B plus C bar ok. Now because in the first one you have only a single variable I can directly apply B here no problem, but for B plus C bar I may have to further do a decomposition; so similarly B plus C bar.

If you let us say decomposition with respect to B we get something like this B plus C bar B bar and substitute B as 0, it is only C bar plus B substitute B as 1 so 1 plus C bar is 1. So here as if we are using another multiplexer this were decomposing with respect to B so the select line is B and the two inputs are C bar and 1, C bar and 1. So you see given any function if you systematically carry out this decomposition with respect to the variables one after the other you will be getting a network of 2 to 1 multiplexers. So this is in fact, a very general approach to implement any arbitrary switching function this we shall be discussing later. There is a data structure called binary decision diagram we shall see that this can be used in a systematic way to construct that kind of a data structure or representation.

So with this we come to the end of this lecture. Now, in this lecture we have basically considered multiplexers, its various kinds of designs how to construct larger multiplexers using smaller ones and also how we can design logic functions using multiplexers.

Thank you.