

**Digital System Design**  
**Professor Neeraj Goel**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Ropar**  
**Multiplexer**

Hello everyone. Today we are going to start a new module Combinational Circuit Design. In our previous module we have seen that, given a Boolean expression how we can implement using gates? How we can optimize it? And what are the delay and area characteristics of a Boolean expression? One question which was left in our previous module was that how the Boolean expression would be initially obtained? Who will give this Boolean expression?

(Refer Slide Time: 01:09)

## Objectives



- Commonly used combinational circuits
  - Multiplexers, decoders, encoders
- Arithmetic circuits
  - Adders and multipliers
- Design and analysis of combinational circuits
- Programmable logic devices (PLDs)
  - PALs, CPLDs, FPGAs



Digital Logic Design: Combinational Circuits.

2

So we will try to answer this question in this module. And while answering this question we will try to learn about some of the commonly used combinational circuits like multiplexers, decoder, encoder. All of these combinational circuits try to implement some of the common or basically some standard Boolean expressions. Similarly we will also learn about how to implement arithmetic circuits.

In one of our previous module we have seen how numbers are represented using binary 0 and 1. Now we will see that how addition, multiplication or other arithmetic circuit can be implemented using Boolean algebra and using gates. Also while doing all this analysis, while do all the designing we will also analyze what are the area and what are the delay characteristics. So that

analysis we will do. We will also see in this module that how some of the programmable logic can also be designed as a side effect of whatever we have, we will learn.

(Refer Slide Time: 02:24)

## Boolean Expressions from plain English-1



- You should laugh at a joke if it funny, it is good in taste, and it is not offensive, or it is told in class by your professor and it is not offensive.
- Input: F (Joke is funny), T (Joke is good in taste), O (Joke is offensive), P (Told by professor)
- Output: L (laugh)
- $L = F \cdot T \cdot O' + P \cdot O'$



So the first question that, given a Boolean, so how do we derive these Boolean expressions? There are two answers. One important, one of the answer is that we try to derive this Boolean expression from our English specification. So let us say, we want to design something. So the design problem will come from the English specification or the English behavior or the behavior we specify in some of the high level language like C, C plus plus. So that is one of the input of this Boolean expression. The other input of the Boolean expression will come from the functionality. So, for example when we will study this example of addition, multiplication, so all of those Boolean expressions are coming from the functionality of the design or functionality of the specification.

So let us take one example. Let us say we want to design a Boolean circuit for this question that you should laugh at a joke if it is funny, it is good in taste and it is not offensive or it is told by your class professor and it is not offensive.

So what is the procedure? If I want to derive a Boolean expression out of this kind of a plain English language statement then I will try to decipher what is input, what is output. Here it seems

the input is, let us say F if joke is funny. And let us say T if joke is also good in taste. If it is good in taste then T would be true otherwise it would be false. Let us say O represents if joke is offensive. If offensive then O would be true otherwise false. And P means if it is told by professor. So I have identified that there are 4 inputs F, T, O and P.

What is the output? Whether I should laugh or not? Let us say L is the output. So by looking at the Boolean, by looking at this English statement I can create a Boolean expression that my output L will be one if joke is funny, F into, it should be good in taste also and it should not be offensive. Or it should be told by professor and it should not be offensive. So this could be the Boolean expression which finally we can implement using gates.

(Refer Slide Time: 5:07)

## Boolean expression from plain English-

2



- An IoT node should stop if manual stop button pressed or error occurred or no more data to process
- Input :
  - M (manual stop), E (Error), D (data)
- Output: S (Stop)
- Boolean expression  
 $S = M E D'$



Let us take one more example. So let us say the, an IoT node should stop if a manual button is pressed, or error has occurred, or no more data to process. So because there is no data it want to process, no data, it does not have any data to process it will stop; or something is manually pressed so I can identify my input. My input would be, if manual stop is there, error is there or data is there. All of these three inputs are single bit. And output is stop. So I can also find out the Boolean expression for this. S should be equal to M and E and not of D. So this way from the English statement we can find out some of the Boolean expressions.

(Refer Slide Time: 06:07)

## ADD Example



- Adding three bits A, B, C
- Sum is '1' when either only one bit is '1' or all three bits are '1'
- Carry is '1' when two or more bits are '1'
- Sum =  $A B' C' + A' B C' + A' B' C + A B C$
- Carry =  $A B C' + A B' C + A' B C + A B C$



Digital Logic Design Combinational Circuits.

5

Let us take out some more intricate example. Let us see how we add. So in add, let us say I want to add 3 bits, A, B and C. What would be the output of these three bits? Let us say if all of these three bits are 1; 1, 1, 1 the output would be two bits, 1 and 1. If these three bits are 1, 0 and 1 then the output would be 1, 0 means 2. So here carry represents two's place and sum represents one's place. So and A, B and C, all of these represent unit's place or one's place.

If we are adding these three bits then we would have this 2 bit output. One's place I am calling sum and carry is the two's place. So if I try to see then when the sum would be 1? Either when only one of the bit is 1 or all three bits are 1 then sum is going to be 1. While carry says when two or more bits are 1 then carry would be 1. So we can represent these English statements also in form of a Boolean expression. So the first statement, sum says that only one of the bit should be 1. So that means A is 1, B and C are 0. B is 1, A and C are 0. A and B are 0, C is 1. Or all three bits are 1 so that means A is 1, B is 1, C is 1. So these are the conditions when my sum has to be 1.

Similarly for carry we would like to see that when two or more bits are 1 then carry is done. So that means; let us first find out the condition for two bits to be 1. A and B is 1, C is 0. A and C is 1, B is 0. A is 0, B and C are 1. Two or more bits are 1 so that means the other condition is when all three bits are 1. A, B, C all three bits are 1. So this gives us the expression for carry.

Now after getting this expression for sum and carry, after getting this Boolean expression, now we can go for our simplification process using K-map or Quine-McCluskey method or using Boolean algebra. Or if you do not worry about course we can use them as (( ))( 8:40) also. So this is how we can create such Boolean expression.

(Refer Slide Time: 08:49)

## Output as one of the input(1)



- Two input case (A and B Input, Z output)
  - Output is A, if A is selected
  - Output is B, if B is selected
- How A or B is selected – Say one bit Control.
  - If Control is 0 => A is selected
  - If Control is 1 => B is selected
- $Z = A \cdot \text{Control}' + B \cdot \text{Control}$



Let us take one more example. So now we would like to have an output as one of the input. So let us say I have two inputs A and B. And output, that is the output. So my output Z should be equal to A if A is selected and output should be equal to B if B is selected. How these A and B would be selected?

So let us say we have another input which says selection or which says control. If that control is 0, this is one bit control, so if control is 0 then A is selected. If control is 1 then B is selected. So this is also one of this; now how do we write the Boolean expression for Z then I can write Z is equal to A if control is 0, A and control, not of control or equal to B if control is 1.

So what does it mean? It means that if control is 0 then this value would be 1 and this would be 0; because it is 0, whatever is the value of B it is going to become 0, and whatever is the value, if A is 0 Z will become 0. If A is 1, Z will also become 1. Similarly if control is 1 then it does not

depend at all on A, it depends only on the B. So if value of B is 0 then Z would be 0. If value of B is 1 it would be 1. So it this is another way of writing Boolean expression.

So this is how; this was the another example, let us say, instead of two inputs, let us consider little more input, let us say four input case.

(Refer Slide Time: 10:51)

## Output as one of the input(2)



- Four inputs (A, B, C, D) and one output Z
- Control will have two bits  $c_0$  and  $c_1$
- If  $c_1c_0 = "00"$  Z = A
- If  $c_1c_0 = "01"$  Z = B
- If  $c_1c_0 = "10"$  Z = C
- If  $c_1c_0 = "11"$  Z = D
- $Z = c_1'c_0' A + c_1'c_0 B + c_1c_0' C + c_1c_0 D$



So in four inputs my output Z should be equal to either A or B or C or D, because I was selecting, I am selecting four inputs; out of four inputs my control should have two bit at latest. Let us say, call them as  $c_0$  and  $c_1$ .

So if  $c_1 c_0$  is 0 then Z should be equal to A. If it is equal to 1, control is 1 then Z should be equal to B. If control is to 2,  $1 0$  is 2, then Z is equal to C. And if control is 3, Z should be equal to D. So if I want to write a Boolean expression for this then it will become something like this. If control is 0 0 that means  $c_1'$ ,  $c_0'$  ended with A.  $c_1'$ ,  $c_0$  ended with B.  $c_1$ ,  $c_0'$  ended with C and  $c_1$ ,  $c_0$  ended with D.

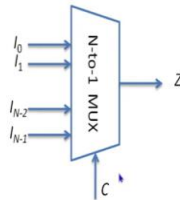
So, for example if my control is 0 0 then all these conditions would be true. So that means there would be no impact. Output will not depend on B, C and D. But because this condition is true, whatever is the value of A that would be pass to Z. So if A is 0 then Z is 0. If A is 1 then Z will become 1. Similarly, similar things would happen for all other control inputs. And so, in the end, based on these control inputs, based on what input is selected, that particular input will be copied to output or output would be equal to that particular input which is selected.

(Refer Slide Time: 12:39)

## Output as one of the input(3)



- Standard combination logic – Multiplexer



- Number of bits in C =  $\text{ceil}(\log_2 N)$
- $Z = \sum m_k I_k$



So this particular circuit if we generalize then it become a very, very powerful and one of the most commonly used circuit of our digital system. This is called multiplexer. In short we also call it mux. So this multiplexer is, generically we can say, it is N is to 1 multiplexer. There are N inputs. There is a control output, control input. And there is one output. That is why it is called N to 1. So it is selecting one of the input as the output out of N inputs.

So what is the size of C? Size of C depends on; number of bits in C depends on what is the value of N. So if I take  $\log_2$ , log to base 2 of N and take the ceiling of that; that will give me the value of C. So, for example if total number of inputs is equal to 5 then my C has to be 3 bit byte. Or if my N is 8 then C has to be again 3 bit byte. If number of inputs are 1000 then C has to be 10. So it is simple  $\log_2$  and the ceiling of that.

So, and internally we can implement this circuit as similar to whatever we have done in previous one or two slides. So we can say Z is sum of mean term k into I k. So that essentially means that, let us say mean term 0 into I 0, mean term 1 into I 1, mean term N minus 2 into I N minus 2, mean term N minus 1 into I N minus 1.

So this way, essentially if there is a N input multiplexer, N to 1 input multiplexer so there would be in each of the product term there would be, so let us say the number of bits, the number of bits in c is m then each of the product term will have m plus 1 literal. And how many such product terms are there? The number of product terms are equal to N because for each product, so for

each input there is one product term. Total number of product terms would be  $N$ . And in the end there would be one OR gate which would be taking the OR of all the  $N$  inputs, for all the  $N$  product terms.

(Refer Slide Time: 15:39)

## Cost of a multiplexer



- Cost of  $N:1$  Mux
  - $N+M$  inputs and 1 output
  - $M$  control bits
  - $N$  AND gates with  $(M+1)$  inputs ( $1^{\text{st}}$  stage)
  - One OR gate with  $N$  inputs ( $2^{\text{nd}}$  stage)



So in, so in this way let us say I can summarize the cost of this  $N$  is to 1 multiplexer would be, there are  $m$  control bits, assuming  $m$  control bits and there are  $N$  inputs so total number of input should be  $N$  plus  $m$ . And there is one output. Total number of AND gates are  $N$ . And the input of each AND gate would be  $m$  plus 1 as discussed now. And there would be one OR gate with  $N$  input. So this is the overall cost of a  $N$  is to 1 multiplexer.

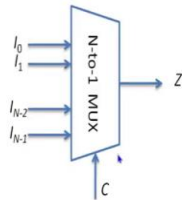


(Refer Slide Time: 16:14)

## Output as one of the input(3)



- Standard combination logic – Multiplexer



What if  $N \neq 2^k$

- Number of bits in  $C = \text{ceil}(\log_2 N)$
- $Z = \sum m_k I_k$



So what if, if my  $N$  is not exact power of 2? So let us say, my this is 5 to 1 multiplexer. So in case the multiplexer is 5 to 1 then the rest of the inputs, so it would not be there. So what it essentially means that, if the value of  $C$  is from those invalid combinations then we are not sure what would be the output. It comes into category of incompletely specified circuit. So because my circuit is not completely specified then output could be anything.

So if this multiplexer is given as a component then we will not connect those input pins and can leave them open. So that is how if the thing, the case would be  $N$  is not power of 2. If  $N$  is power of 2 then it is a kind of optimal design.

(Refer Slide Time: 17:18)

## Cost of a multiplexer



- Cost of N:1 Mux
  - N+M inputs and 1 output
  - M control bits
  - N AND gates with (M+1) inputs (1<sup>st</sup> stage)
  - One OR gate with N inputs (2<sup>nd</sup> stage)



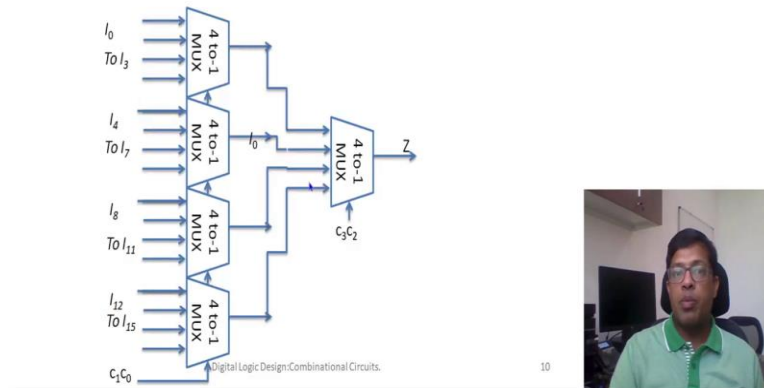
So this is how the cost of multiplexer is. Now this also gives us an idea, if the multiplexer becomes big, let us say it is a 1024 colon 1, 1024 to 1 mux. So that means I will have, in the first stage there would be 1024 AND gates, each with input of 11. Each AND gate will have 11 inputs. Similarly in the second stage there would be an OR gate with 1024, 1024 inputs. And as discussed in our one of the previous lectures, if the number of input increases our cost increases, area increases as well as delay increases. So it is not a very effective design. That is why whenever multiplexer size grows it is usually suggested that we use smaller multiplexers and we create a multi-stage design.

(Refer Slide Time: 18:27)

## Hierarchical Design of Mux



- Using 4:1 Mux to design 16:1 Mux



So let us take an example of this. This is also called hierarchical design or a multiple stage design. So let us say we want to design a 16 to 1 multiplexers using 4 is to 1 multiplexers. So in that case what we will do is we will have; how many 4 is to 1 multiplexers would required because we require 16 inputs? So we would have four 4 is to 1 multiplexer in our first stage.

So the way it would be that first four inputs would be connected to first mux, and then I4 to I7 would be connected to second mux. I8 to I11 would be connected to third mux. And I12 to I15 would be connected to fourth mux. What about the control input? So that lower two bits, lower two bits of the control, so 16 is to 1 will have 4-input control. We will divide this 4-input control as like 2 bits of LSB and 2 bits of MSB. So this 2 bits of LSB, C0 and C1 would be given as control signal to all of these multiplexers at stage 1. The output of this stage 1 multiplexers would be given as a input to another 4 is to 1 multiplexer which is controlled by MSB of my control that means C3 C2.

Now when we connect all of them, now when we connect this, when we connect the first stage multiplexer to corresponding I0 I1 I2 I3 of second stage that is how we can design a bigger multiplexer. So the similar approach can be used to design any bigger size multiplexer. We will keep on dividing the stage, stages or we will keep on using, so let us say same design of 16 cross 1 multiplexer which already have two stages can be used to design a 256 colon 1 multiplexer.

Any arbitrarily size multiplexer can be designed using this 4 is to 1 multiplexer by hierarchically creating bigger and bigger designs. So let us quickly see what would be the cost impact.

(Refer Slide Time: 21:07)

## Heirarchical Design of Mux



- 16:1 Mux cost
  - 16 AND gates of 5 inputs and one 16 input OR gate
- 4:1 Mux cost
  - 4 AND gate of 3 input and one 4 input OR gate
- Design of 16:1 Mux using 4:1 Mux
  - Four 4:1 Mux at stage 1
  - One 4:1 Mux at stage 2



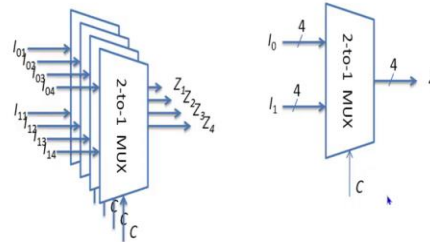
Now a 16 to 1, 16 to 1 multiplexer would have 16 AND gates with 5 inputs and one 16 input OR gate. While cost of 4 to 1 mux would be four AND gate with 3 inputs and one 4 input OR gate. So in the first stage we will have four 4 is to 1 mux and the second stage we will have one 4 is to 1 mux. So if we try to do the delay calculations, if we try to do area calculations, it has been seen that we can choose the suitable model. So we can see that the design using the hierarchical mux is many times much more efficient both in terms of area as well as in terms of delay.

(Refer Slide Time: 21:56)

## MultiBit Multiplexer



- 4 bit 2:1 Mux



Digital Logic Design: Combinational Circuits.

12



So let us see one more kind of multiplexer design which is multi-bit multiplexer. So sometime it happens that the number of the input to this mux is not single bit. So we have to multiplex, we have to multiplex multiple of these inputs which are grouped together. So let us say, this  $I_0$  was essentially 4 bits. I am calling it  $I_{01}$ ,  $I_{02}$ ,  $I_{03}$  and  $I_{04}$ . Similarly my  $I_1$  input is also of 4 bit;  $I_{11}$ ,  $I_{12}$ ,  $I_{13}$  and  $I_{14}$ . And similarly output will also be 4 bit,  $Z_1$ ,  $Z_2$ ,  $Z_3$  and  $Z_4$ .

Now all of them would, can be controlled by a single  $C$  bit. And if I see internally they are simply the copy and replication of these 2 is to 1 muxes. So the only thing is that a particular bit is connected to each of these muxes respectively. And control bit is same for all these mux. And the output is combined. So when we have such kind of a multiple bit input we call that, instead of bit we also, we called it as a bus. So the input instead of a wire would be, would be a bus, and as far as design is concerned symbolically we also represent it using this. So this is a bus.

The symbol of bus means there would be a slanted line like this and there is a number written on it. That number signifies how many bits in that bus is. So here  $I_0$  means there, in  $I_0$  are 4 bits;  $I_1$  also there is 4 bit. If there is no slant line and there is no number that means it is considered as a single bit number. In some other representation sometimes the size of the bus is not given but the difference, the diagram you will see is that the wires will be, will be of lesser width or basically not bold; while the buses will be bold. That means they are collection of wires. So they become little white.

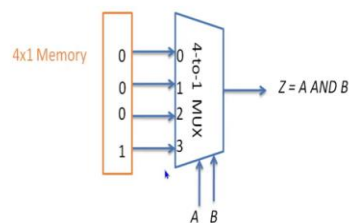
So another thing which I missed previously, so the way we will always write a, design this, like put a symbol of multiplexer, it would be a trapezoid which will have little wider side on the input side and less wider side on the output side. Control is usually shown on the side. So these are the major input, data inputs and this C determines the control input. So this is how multiple bit multiplexers, we also call it, let us say here we can call it 4 bit 2 is to 1 multiplexer. Similarly any number of bits could be there in these multiplexers. So we said that this multiplexer is one of the commonly used very powerful design. So let us try to understand what kind of applications are there for these multiplexers.

(Refer Slide Time: 25:29)

## Application of Mux (1)



- Designing any generic circuit



- $2^N$  input mux can design any N input function
- Typically used in FPGAs – Known as LUT or Look up tables

Digital System Design

13



So let us look at couple of these applications. Now one of the application which is very, very important that, that these multiplexers can be used to design any generic circuit. So let us say I want to design an AND gate from 4 is to 1 multiplexer. So what I can do? I can, at the input of, these four inputs I can write the truth table of A and B, that means at zeroth input I can tie it to 0. First input I can tie it to 0, second input also I can tie it to 0, third input I can tie to 1.

So these control inputs if they are a and b, if both input a and b are 0 so that means whatever is at zeroth input, that would be selected and given to output so that means output would be 0. If A is 0, B is 1 then whatever is there at this input will be given as output. So that means that would be 0. Similarly for A and B as 1 and 0 the output would be 0. For A and B equal to 1 1 so the output would be 1.

So in this way here we are able to design an AND gate using 4 is to 1 multiplexer. So now you can ask multiple questions here. One question that why or how much wise it is to design AND gate using 4 is to 1 multiplexer? Cost of my 4 is to 1 multiplexer is, there are four 3 input AND gates and one 4 input OR gate. It is very, very costly with respect to one AND gate.

So the reason I would like to design any generic circuit using multiplexer, though multiplexer is very much, much more costlier than my circuit which I want to derive. But it gives me genericness. So this genericness is very, very useful in some cases. The most important usefulness or use case of this generic circuit is that since the cost of fabrication is very, very high we can put, we can create all of these 4 is to 1 multiplexers fabricated on the chip. And then finally give these values 0 0 0 1 or like whatever logic we have, whatever truth table we have. That truth table we can give as an input in sort of a memory.

So we can design a let us say, 4 is to 1 memory or 4 bit memory which keep these four values and we attach those memory cells with the input of my mux and then these memory I can write at run time. But all these muxes are fabricated. So in that way I can design a programmable, I can design a programmable hardware.


So I will try to refresh things. So hardware, once it is fabricated it cannot be changed. The cost of fabrication is very, very high. So, because cost of fabrication is very, very high and we cannot re-program, we cannot redesign it. So that is why sometimes it is desirable that we design our hardware in such a way that it can be, we can design our hardware in such a way that it can be used as any circuit, whatever we want.

So the programmable part is writing whatever memory content we would like to write. So if we write the proper correct memory content then we can program these FPGAs as we want. To design 2 is to power n input, to design an n-input function, any n-input function I would require 2 is to power n input multiplexers.

So let us say I want to design any function of 4 input. Then I would require 16 is to 1 mux. So one more side point here, that designing a generic circuit is also important because, let us say, using these two inputs total number of functions which could be designed are fairly large. So from 2 inputs I can design some 15 different functions. So that is why it is required. Either number of inputs will grow. The total number of functions which could be, which are possible

using those inputs also grows exponentially. So that is why if we have a generic circuit it can implement any other functions to our like, whatever we desire.

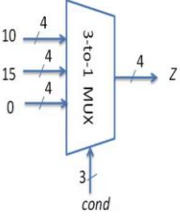
(Refer Slide Time: 31:15)




## Application of MUX(2)

- Hardware realization of if-else/switch construct
  - What if condition are not 0,1,2...

```
If(a < 0)
  z = 10
else if (a == 0)
  z = 15
else if (a > 0)
  z = 0
```



One hot encoded Mux  
Cond[0] = (a < 0)  
Cond[1] = (a == 0)  
Cond[2] = (a > 0)



Digital System Design 15

Other than this generic circuit implementation these multiplexers can also be used to realize sort of if-else construct. So let us say I have this kind of a statement. If A equal to 0 then z is equal to 10. If A is equal to 1 then Z is equal to 15. And if A is equal to 2 then Z equal to 0. So how can I design such, such a specification?

So you try to remember how we started. We are saying that in case of multiplexers, one of the input is going as the output. So that means my output is either 10 or 15 or 0 based on if that particular value is selected. So Z equal to 10, 10 would be selected as a value of Z if A is 0 and 15 would be selected if A is 1 and 0 would be selected if A is 2.

So what I can do is I can keep this A as a control and I can design a multiplexer where, 4 is to 1 I can use, 4 is to 1 multiplexer where I will tie my I0 input to 10. I will connect my I1 input to 15. I will connect it my I2 input to 0. What should I connect my I3 input to? That is the question mark. So, as we discussed before, we can leave it open. It will not harm. It will not affect our circuit anyway because we will say that I equal to, A equal to 3 is a illegal option. It is a do not care condition. It is incompletely specified machine.



So now this circuit can effectively use, so if A is 0 0, that means A is 0. 2 bits of A are 0 0. Then 10 would be passed as Z. And if 2 bits of A is 0, 1 then 15 would be passed. If 2 bits of A is 1 0 then 0 would be passed to Z. And A equal to 1 1, or A equal to 3 will never occur.

So this is certainly a very good advantage that if I want to implement such kind of a behavior or such kind of functionality this mux multiplier could be very, very efficient. Now let us consider one more example.

So before that, one more question here that cannot we simply design 3 is to 1 mux. So if we see it from the hardware perspective, if we leave the; so we remember in 4 is to 1 mux there were 4 AND gates each with three inputs. So you will leave the last AND gate with 3 input then we would effectively be able to design 3 is to 1 mux. So is there any alternative way? Or what if this, this A is not like this, A equal to zero or A equal to 1, 2. So value of A is not equal to 0, 1 or 2.

So in those cases conditions are not equal to 0, 1, 2 so, then it would be something like this. Let us say, when A is less than 0 then Z equal to 10. If A is equal equal to 0 then A equal to 15. And if A is more than 0 then Z equal to 0. So here you see the value of conditions for control is not equal to 0 1 2. So what should we do?

We would have two choices here. One choice is that, whatever number we will get, we will get some condition out of this. Either we construct 0, 1, 2; three different numbers. So we create, we call A is less than 0 should be equal to 0 and A equal, equal to 0 should be equal to 1. And A more than 0 should be equal to 2.

So other than that there is one more alternative idea. So we put these three condition bits, so this, this, these are the condition bits. All of them, we create a different condition bit. So this is, let us say, condition bit 0. This is condition bit 1. This is condition bit 2. And we combine these three bits to make it a bus. And we make sure such things will happen that such things can be helpful when we make sure that one of them is 1 at one time. If only one of them is 1 at one time, which is true to at least in this case, if A is less than 0 A cannot be equal to equal to 0. Or A cannot be more than 0.

So if all of these three conditions are disjoint so that means only one of them would be true then we create such an array. So then we can create this 3 is to 1 mux. And this kind of conditional

encoding is also called one hot encoding. So we call it this one hot encoded mux. Now this condition 0, this condition 0 can be ANDed with my 0th input  $I_0$ . And this condition 1 could be ANDed with my first input. Condition 2 could be ANDed with my second input. And all of these three could be ORed together.

So you see this one hot encoded mux is much, much cheaper in terms of hardware cost. If I consider 3 to 1 mux which is only a single bit, it is like we have a bus here but instead of that, if it is a single bit then the total number of AND gates would be, there would be three AND gates with the input 2; 2 input three AND gates and one OR gate with three inputs. The cost is also less because input of each of the AND gates has reduced if it is one not encoded.

So then you can ask another question. Then if one hot encoded is so good then why should we use that, the other standard regular mux. Question is valid, correct. But this will also increase in the number of inputs to mux and it is not always desirable. So let us say I have a very large mux like 16 to 1 mux. So there the control bits will become 16. So sometimes that kind of a flattened, this particular condition that all the condition bits has to be disjoint and only, we have to ensure only one of the bit has to be 1 at one time, that may not be true in all the cases. So that is why a regular mux has its own importance while one hot encoded mux has its own advantage.

That there would be some scenarios like these scenarios where one hot encoded mux could be useful, could be efficient. And while attempting our design problems we can see that which particular mux would be good for our purpose. Let us see. So this we have already discussed that condition 0, 0th bit of condition is A is less than 0. First bit of condition is A equal to equal to 0. Second bit is equal A more than 0.

(Refer Slide Time: 39:25)

## Application of Mux(3)



- Where selection of one input is required out several inputs
  - Arbiter
  - Device hub controller
  - Bus controllers
  - Routers
  - Network switches



Now, other than that also, we see various, numerous applications of multiplexers. So it is like, I can give at least one line description of some of them. The idea is that wherever we need to select one input out of several inputs then we have to use multiplexer. So you can think of this like, you have a Smartphone.

So in Smartphone, let us say, your television is smart. And the television can take input from your mobile phone and there are multiple mobile phones at home. And it can also take input from remote control. So finally it has to take input from only one of the guy, either one of the mobile phone or the remote control. So it is, like a, you have to have some multiplexer there.

So similarly let us say there are multiple, so you have a memory device. So now this memory device is taking input from multiple of these, multiple of the hosts, so like you can have multiple processors, you have GPU over there, you have processor, you can sometimes have accelerators or some other chips. So all of them want to access memory. Only one of them should be able to access at one time. This is called arbiter.

So such kind of different use case would be there. In our networks it is quite common that all of us would be using our network switches as well as routers. So the scenarios become very popular and very common that we have to select one of the input from the given n number of inputs. So these are the various applications of multiplexers and we will see with more examples as we go

on. So when we will do some of the design experiments we will see that how these multiplexers are used.

(Refer Slide Time: 41:46)

## Summary



- We learnt how to give generate Boolean expression from plain English
- We learnt about a useful standard combinational circuit - Multiplexer



So with this we can close this particular lecture. And in summary we can say that in this lecture we have learnt how to generate or how to create Boolean expressions through the English specification or a high level specification. So we have considered couple of these examples. And now, so other than that, we have also learnt how to write Boolean expression for multiplexers and therefore we have also learnt these multiplexers and how useful it can be as a standard combinational circuits. We have seen in various use cases, various applications of these multiplexers. Thank you very much.