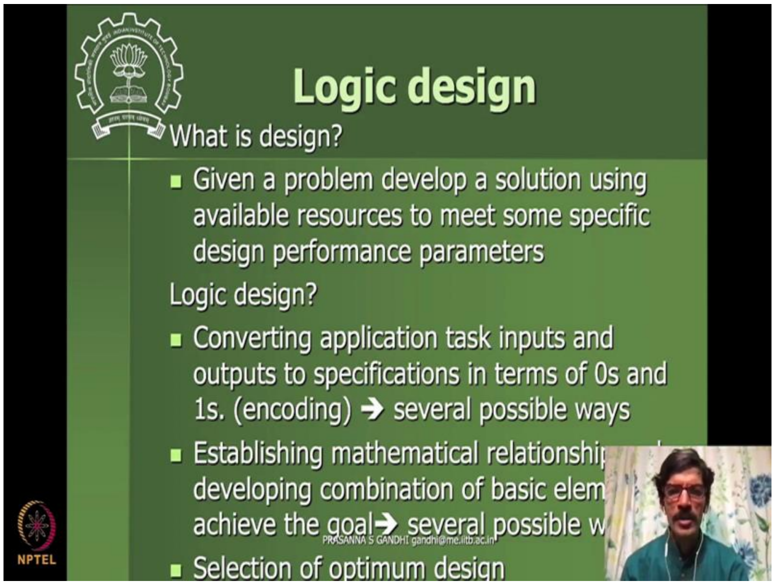
,Design of Mechatronic Systems
Professor Prasanna S. Gandhi
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Lecture 13
Microprocessor Building Blocks I-Combinational Circuits

So this lecture today's class we look at some of the fundamental building blocks of microprocessor, or microcontroller. We will start off with combinatorial circuits or combination design, logic design, and then we will move on to sequential and then memory blocks and things like that in the further classes as to go. So we will focus on combinational logic circuits in today's class.

(Refer Slide Time: 0:56)



Logic design

What is design?

- Given a problem develop a solution using available resources to meet some specific design performance parameters

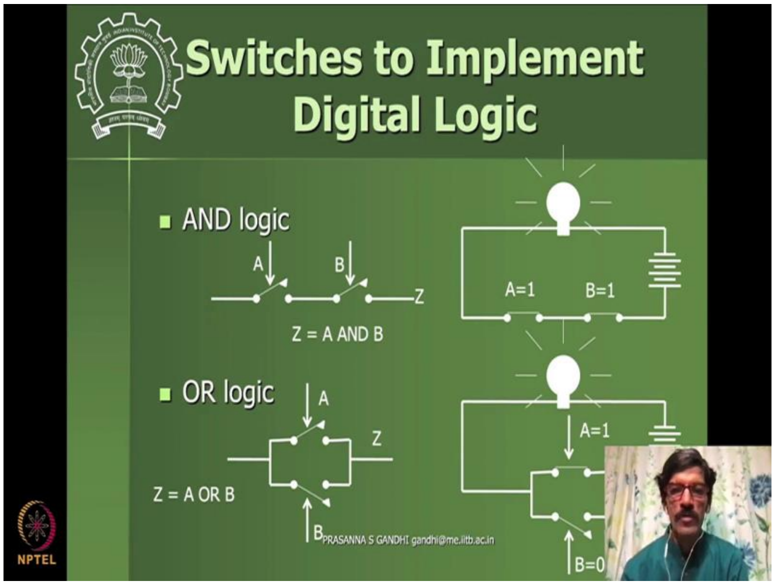
Logic design?

- Converting application task inputs and outputs to specifications in terms of 0s and 1s. (encoding) ➔ several possible ways
- Establishing mathematical relationshi developing combination of basic elem achieve the goal➔ several possible w
- Selection of optimum design

So let me get this right. So let us see if you talk of logic design, so we will start with some of the fundamental basics, I am presuming here you all are aware about basic logic gates AND, OR, NOR, NOT, that kind of gates. So, now with those gates we want to design some kind of logic. So, you want to have a solution given to a given problem basically using some of the resources that are available with us, that is typically a design process for any design.

So, in the logic design specifically what we do is given a task we convert it into some kind of mapping into zeros and ones, that is called encoding and there are several indifferent ways possible, say you say switch is, light is on is one, light is off is your zero like that you can have many different mappings possible then you establish some kind of a mathematical relationship that is required based on the laws or logic that you like to kind of incorporate into system and then select the good design or good mathematical relationship, depending upon what are the requirements, that is typically the logic design process.

(Refer Slide Time: 2:25)



Switches to Implement Digital Logic

- AND logic

Z = A AND B

- OR logic

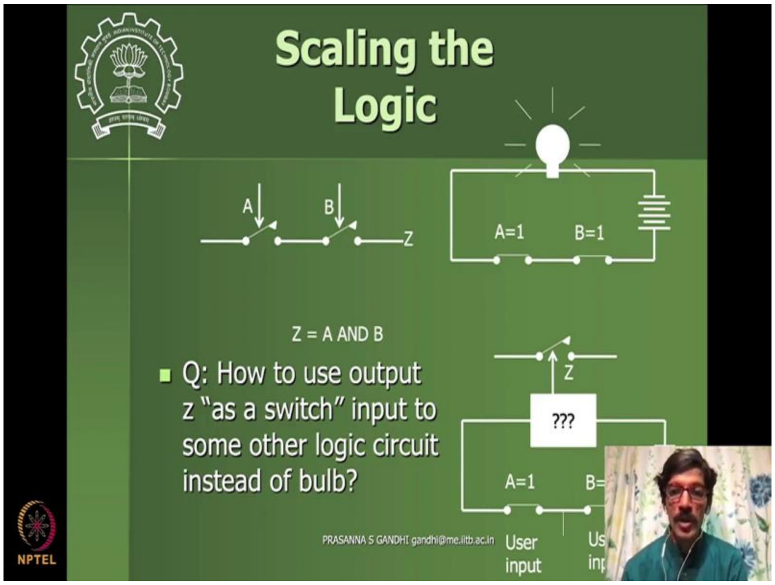
Z = A OR B

So, let us see simple like no logic that you have learned, again a little bit revision here, but it is good to do so that we can kind of get some fundamental aspects of this, logic design of the logic touched upon. So, say for example AND logic you can simply implement by these two switches that you see here.

So, A and B both switches are on then the light bulb lights and that lighting of the bulb is called as 1 then you can have OR logic where you at this circuit is put in this kind of a fashion when A is on, A is switched on and or B is switched on then also like know your light bulb close. So, this is AND and OR logic implemented.

As you can see here, A is 1, B is 1 your light is on that means that that is what I physically considered to be that physical mapping of physics into mathematics is one, is what I define as one. So switch on, I define as 1 and then like light on is defined as 1 then like I can kind of come up with this, abstract mathematical problem into a physical world problem.

(Refer Slide Time: 3:48)



Scaling the Logic

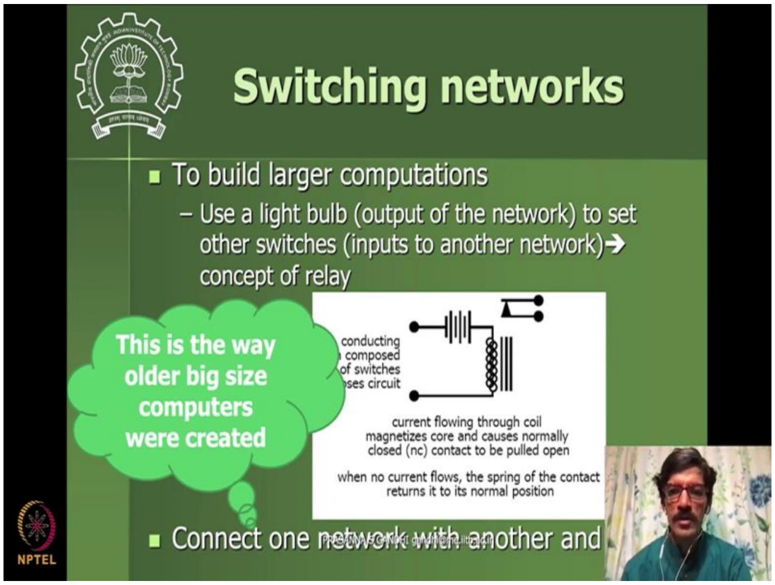
Z = A AND B

- Q: How to use output z "as a switch" input to some other logic circuit instead of bulb?

PRASANNA S GANDHI gandhi@me.iitb.ac.in

Now, the problem with this is now light is on but now I cannot use that light on as 1 as a input to any other device, so there that is where like you have a problem like you cannot scale then like this logic, so switches based kind of thing, we cannot scale and that is where like you see, what is the element that can be used here to switch the next switch on or when that is 1 like I want the switch to be closed what is this element here and that is where people started to kind of using this solenoids to do that job.

(Refer Slide Time: 4:26)



Switching networks

- To build larger computations
  - Use a light bulb (output of the network) to set other switches (inputs to another network)→ concept of relay

This is the way older big size computers were created

conducting composed of switches ses circuit

current flowing through coil magnetizes core and causes normally closed (nc) contact to be pulled open

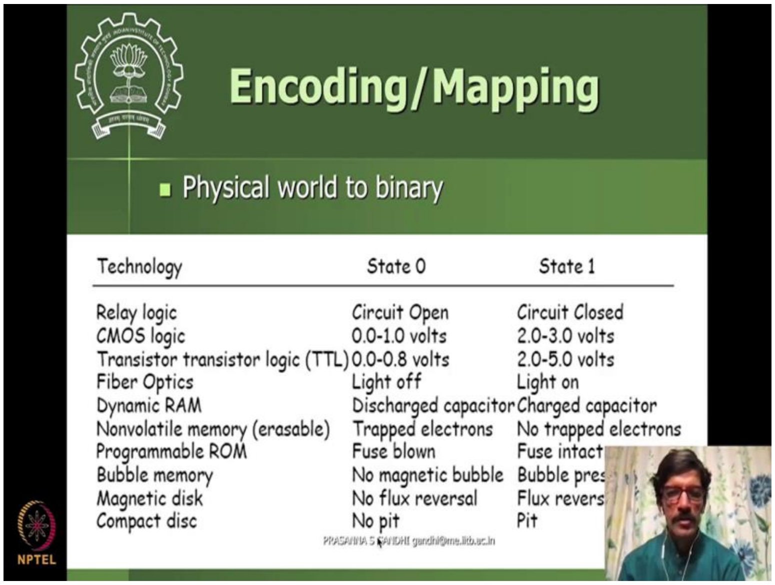
when no current flows, the spring of the contact returns it to its normal position

- Connect one network with another and

PRASANNA S GANDHI gandhi@me.iitb.ac.in

And this is a way like this whole area started developing into now you have multiple kind of solenoids switcher kind of doing switching like these and then some interesting things are happening in a circuit and then there are delays related to that, there are mechanical motion inertia, lot of lot of these problems are there. But these problems now, you know that these problems have been taken care of, by design of like this switching that happens in this transistors. So, to say like the CMOS kind of design.

(Refer Slide Time: 5:04)



Encoding/Mapping

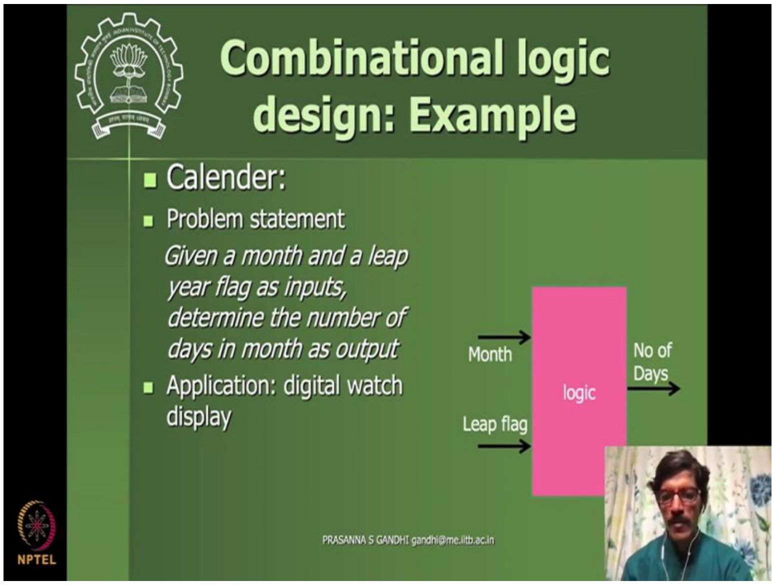
- Physical world to binary

Technology	State 0	State 1
Relay logic	Circuit Open	Circuit Closed
CMOS logic	0.0-1.0 volts	2.0-3.0 volts
Transistor transistor logic (TTL)	0.0-0.8 volts	2.0-5.0 volts
Fiber Optics	Light off	Light on
Dynamic RAM	Discharged capacitor	Charged capacitor
Nonvolatile memory (erasable)	Trapped electrons	No trapped electrons
Programmable ROM	Fuse blown	Fuse intact
Bubble memory	No magnetic bubble	Bubble pres
Magnetic disk	No flux reversal	Flux revers
Compact disc	No pit	Pit

PRASANNA S GANDHI gandhi@me.iitb.ac.in

But let us look at what is this mapping typically is, in from the physical world to the binary world you can have relay logic where your circuit is open and close you can have different states defined, CMOS logic you can have zeros and ones, 0 to 1 volt is considered as zero state and 2 to 3 volts is considered as state 1, transistor logic 0 to 0.8 volts is considered as 0 and 2 to 5 volts is considered as 1 like that you can have multiple kind of definitions here for state 0 and state 1, these are typically given definitions for different kinds of elements or technologies that you want to see.

(Refer Slide Time: 5:57)

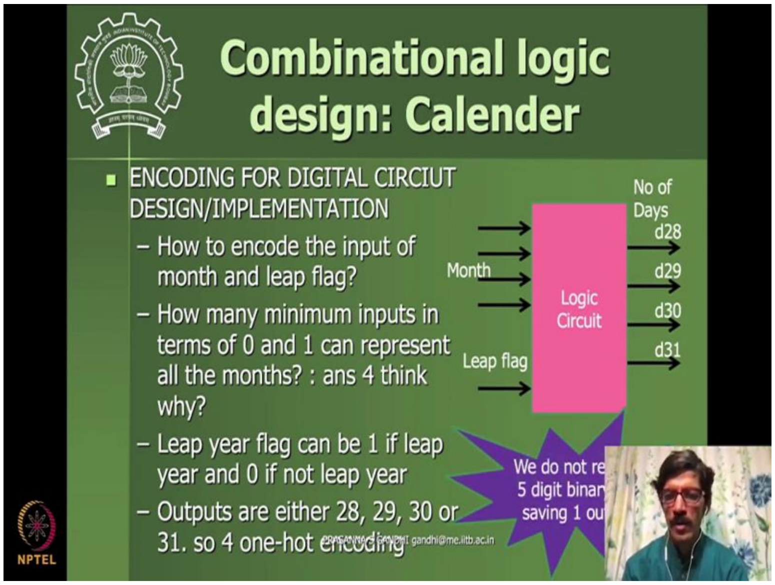


Combinational logic design: Example

- Calender:
  - Problem statement

    Given a month and a leap year flag as inputs, determine the number of days in month as output
  - Application: digital watch display

PRASANNA S GANDHI gandhi@me.iitb.ac.in



Combinational logic design: Calender

- ENCODING FOR DIGITAL CIRCIUT DESIGN/IMPLEMENTATION
  - How to encode the input of month and leap flag?
  - How many minimum inputs in terms of 0 and 1 can represent all the months? : ans 4 think why?
  - Leap year flag can be 1 if leap year and 0 if not leap year
  - Outputs are either 28, 29, 30 or 31. so 4 one-hot encoding

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, now, we want to design for example, a combinational logic for a calendar kind of a problem if this is a, this is given how do you kind of come up with a circuit which is to come consisting of these gates, typically different AND gate, OR gate or some other kind of gates and solve this problem, what is the problem statement here that given month and number of so given a month and a leap year flag has input this is a, these are the two inputs and you will need to find out number of days in a month as output.

So, number of days in the month we want them as a output here. So, this application can be different applications, but we are just looking at this as a problem statement from the design of logic perspective. So, given a month means how many months are possibility for each of the inputs. So, we will have 12 months so 12 possibilities for the input, then a leap year flag. How many inputs can be there, there can be two inputs whether it is a leap year or it is not linear like that.

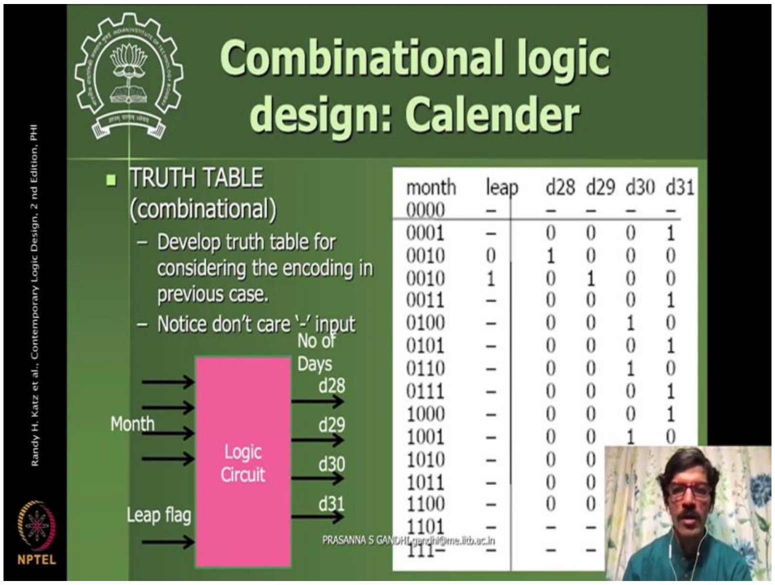
So, you need to kind of find out all the (com) like all the possible inputs that that could be there for the system and then like you map them on to some kind of outputs and then you need to now define some abstract zeros and ones kind of a combination to define like these months like say there are 12 months so you need 12 different kind of binary numbers to represent each month.

So, we can come up with many different kinds of possibilities, but typically like you say, I will have like 4 bits used as a binary input for a month thing. So that like I will start 0000 I will define as a January month like that I defined like in terms of these 4 bits, different different months and then leap year flag can be 0 or 1 depending upon if it is a leap year it is 1 and if it is not a leap year it is 0 and then number of days output I can have could be possibility 28 29 30 and 31 four possibilities only exists for a number of days.

See, look, we are not representing number here that we want 30 as a number to be represented by, no need, because we know that we are not like having any month which will have only 2 days or 5 days. So we do not need to represent 0 to 31 as entire like spectrum of numbers as a output, are you getting my point here, this is very important thing that to see, what are the distinct outputs that that we are interested in and then there are only 4 distinct outputs that are, that we are interested in.

So like that one can think about and now one can start writing a table, a table of inputs and outputs. So how do you kind of develop that table, you put all the inputs on one side and all outputs on the other side and start writing their values. So input January month, 0000, like that you can start.

(Refer Slide Time: 9:25)



Combinational logic design: Calender

- TRUTH TABLE (combinational)
  - Develop truth table for considering the encoding in previous case.
  - Notice don't care '-' input

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0		
1011	–	0	0		
1100	–	0	0		
1101	–	–	–		
111–	–	–	–		

Randy H. Katz et al., Contemporary Logic Design, 2 nd Edition, PHI

PRASANNA S GANDHI gandhi@me.iitb.ac.in

And then you see that this is typically a chart that will come. So months 0000 and then, you will have leap flag does not, you do not care because whether it is a leap year or not leap year, you always have, fixed number of days in January month. So for example, here they have used the 0001 as a January and this is getting represented here, so you can have your own definition, nobody prevents you to say that my 0000 should be a January and then you will get this output here as 31 days as 1 output here, like that you can develop this kind of a table.

And once this table is developed then you can use the combinational logic circuit laws, you know that the Karnaugh map or you have many different kinds of ways of hitting these circuits for each of the output. So, for example, for these d 28 output you can use sum of the products kind of a thing or products of the sum kind of thing, you have a lot of options possible here.

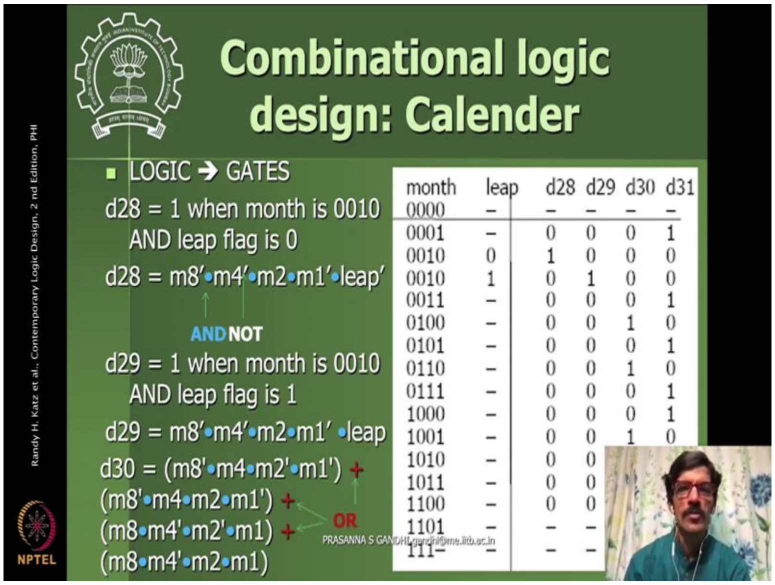
So, that is how one goes about getting like this combinational logic problem converted abstract problem given as a calendar problem, converted into some kind of a mathematical problem of a digital logic design. So, this is what is a fundamental basis of all kind of combinational logic design. So, here the important thing to note is that there is notion of memory here; given some kind of a combination of inputs will have a combination of outputs.

So, whatever conditions that we are saying are not based on like you have to remember something that previously this was something and now it is something different, no, nothing out like based on the previous thing the output is changing, no, there is no memory or notion of memory in entire of this discussion. So, given the, these kinds of fixed values here, for binary inputs, you will have fixed binary outputs here.

The combination of these like uniquely defines your combination of like outputs. So, that is a kind of way one can design. So, see now, here you can see that d 28. These are 4 kinds of outputs, I could have converted this problem into just 2 outputs as also because two binary numbers I can represent four different combinations 00 01 10 and 11. So, I can say my 00 if it is output, then it is 28 months, 28 days in the month or 01 is output 29 days in a month like that I can say.

And then this output could have been reduced and that would have saved me some circuits like that one can kind of think about so, we will not get into the details about how do we do the saving or how do we kind of like develop the circuits which are optimized, no, we are not getting into all that here. Here our basic idea is to understand these fundamentals, how these different different combination logics are developed.

(Refer Slide Time: 13:07)



Combinational logic design: Calender

- LOGIC ➔ GATES

d28 = 1 when month is 0010 AND leap flag is 0

d28 = m8′•m4′•m2•m1′•leap′

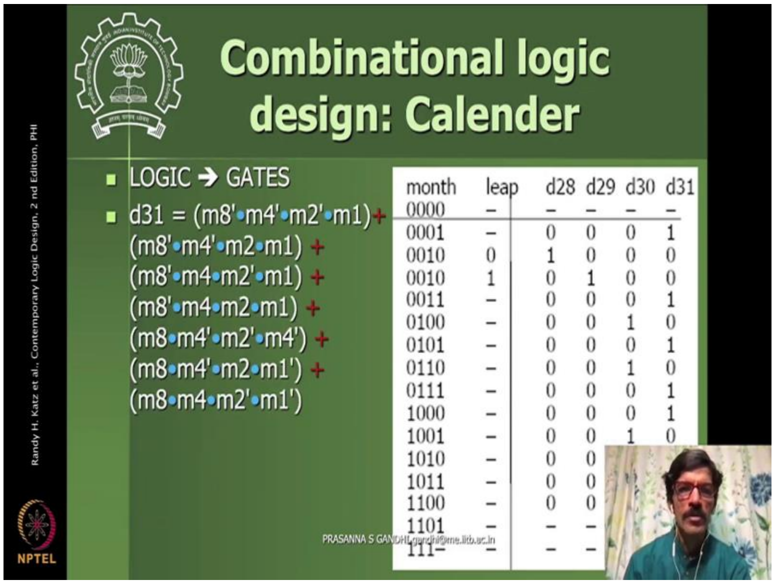
d29 = 1 when month is 0010 AND leap flag is 1

d29 = m8′•m4′•m2•m1′ •leap

d30 = (m8'•m4•m2'•m1') +
(m8'•m4•m2•m1') +
(m8•m4'•m2'•m1) +
(m8•m4'•m2•m1)

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0		
1011	–	0	0		
1100	–	0	0		
1101	–	–	–		
111–	–	–	–		

Randy H. Katz et al., Contemporary Logic Design, 2 nd Edition, PHI



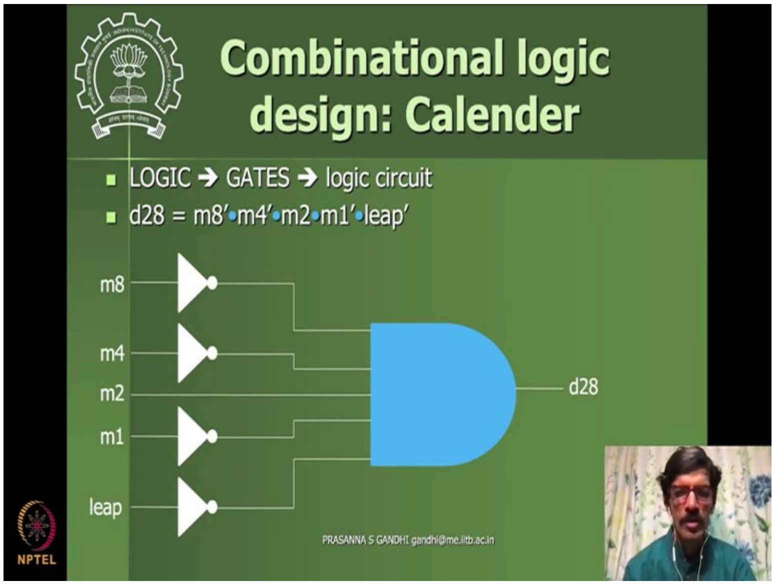
Combinational logic design: Calender

- LOGIC → GATES
- d31 = (m8'•m4'•m2'•m1)+ (m8'•m4'•m2•m1) + (m8'•m4•m2'•m1) + (m8'•m4•m2•m1) + (m8•m4'•m2'•m4') + (m8•m4'•m2•m1') + (m8•m4•m2'•m1')

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0		
1011	–	0	0		
1100	–	0	0		
1101	–	–	–		
111–	–	–	–		

Randy H. Katz et al., Contemporary Logic Design, 2 nd Edition, PHI

PRASANNA S GANDHI gandhi@me.iitb.ac.in



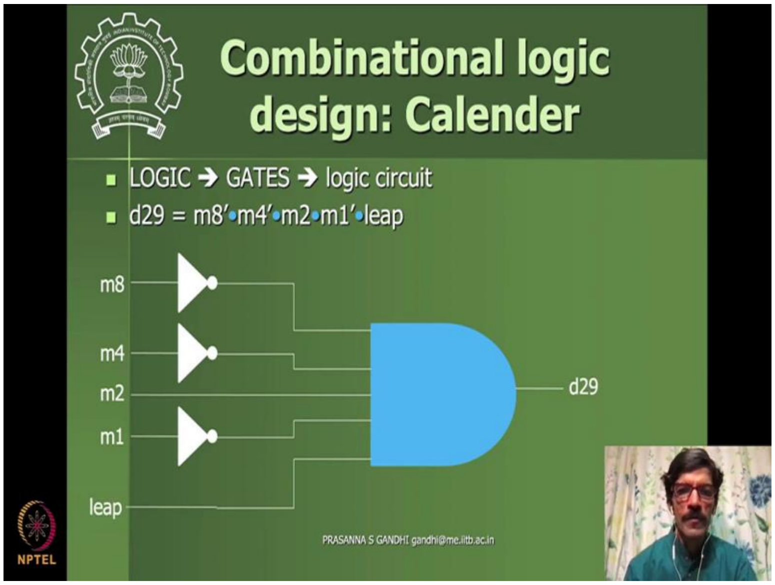
Combinational logic design: Calender

- LOGIC → GATES → logic circuit
- d28 = m8'•m4'•m2•m1'•leap'

PRASANNA S GANDHI gandhi@me.iitb.ac.in

And like that, there are, now one can use this, this is sum of products kind of a rule that I have used here to get for the output for d 28 and then it is in the combination of AND and NOT kind of gates. So, like that you get all these things developed here and you finally express everything as some kind of a logic expressions. And in this logic expressions can be converted into a circuit, you can convert them into a circuit by before simplification, after simplification is all like, you have a lot of leeway to play there, you can use Karnaugh maps or De Morgan's law to simplify things and get to some kind of a desired output in the desired form actually.

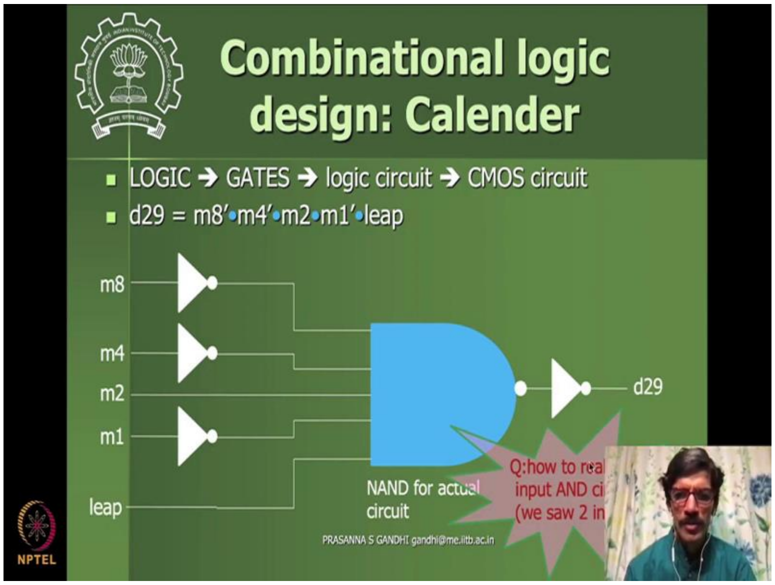
(Refer Slide Time: 14:05)



Combinational logic design: Calender

- LOGIC → GATES → logic circuit
- $d29 = m8' \cdot m4' \cdot m2 \cdot m1' \cdot leap$

PRASANNA S GANDHI gandhi@me.iitb.ac.in



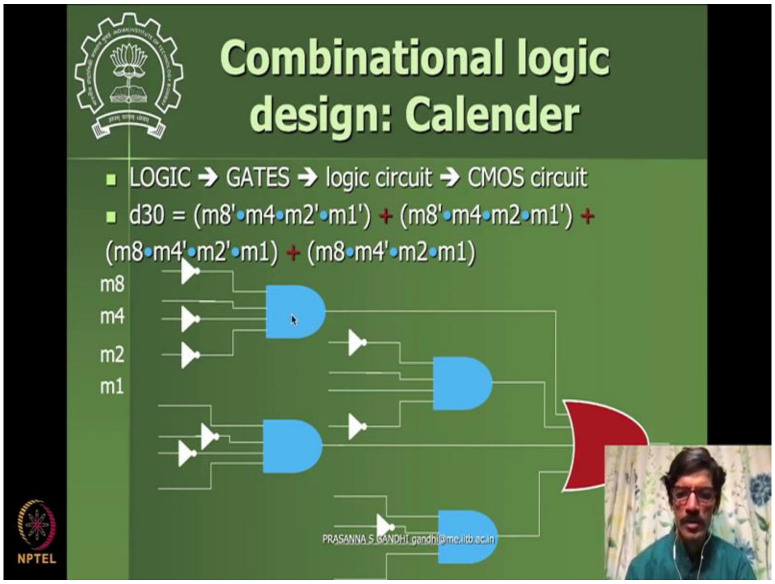
Combinational logic design: Calender

- LOGIC → GATES → logic circuit → CMOS circuit
- $d29 = m8' \cdot m4' \cdot m2 \cdot m1' \cdot leap$

NAND for actual circuit

Q:how to real
input AND ci
(we saw 2 in

PRASANNA S GANDHI gandhi@me.iitb.ac.in



Combinational logic design: Calender

- LOGIC → GATES → logic circuit → CMOS circuit
- $d30 = (m8' \cdot m4 \cdot m2' \cdot m1') + (m8' \cdot m4 \cdot m2 \cdot m1') + (m8 \cdot m4' \cdot m2' \cdot m1) + (m8 \cdot m4' \cdot m2 \cdot m1)$

So, these are like different different circuits that are for different outputs here. So, now, there are these kind of questions you can ponder over, you can see these questions and ponder over them and think about like what could be the answer for such a such a question. And then this is a say d 30 output circuit for d 30 output, and things like that.

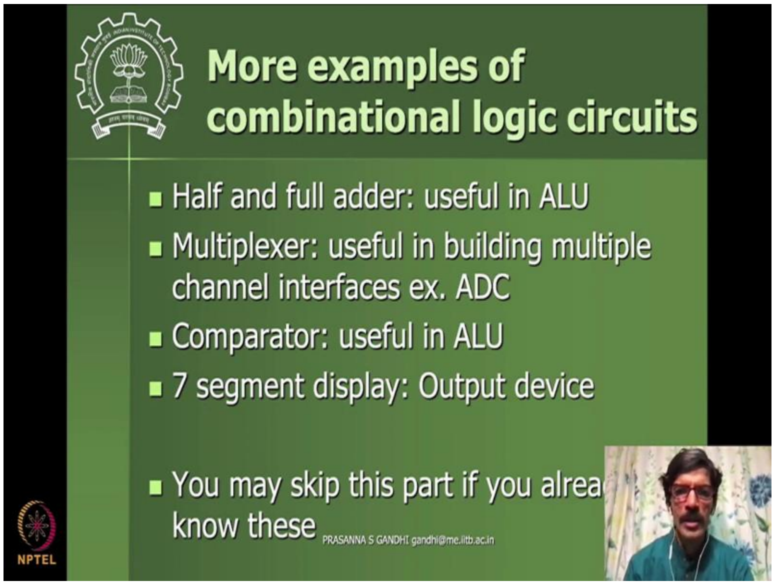
So, like that you can start developing now, these kind of logic expressions and logic circuits for many, many different kinds of combinational logic elements. So, we will see some of them but I am going to go a little faster through that, so that, the idea is to you can pause at any point of time and then like go through these examples solve them on pen and paper and then like look at the solution that will help you internalize the concepts very well.

But I am going to go a little faster to kind of because fundamentally there is nothing different there all are like in this kind of a combination of like some kind of a solutions (the) once you know this process that you express each of the outputs in the form of these inputs by using the sum of the products kind of a rule or some other kind of a rule depending upon what the problem gives a simplification to then one can kind of start developing this first expression and then use some kind of a digital circuit laws to simplify these in more details.

So that is what is, I am leaving it for you to kind of learn yourself and develop that, by the way, even if you do not know to how to simplify and how to kind of get some circuits, it is okay, there is, it is not a requirement for understanding microprocessors. See, it is like a, like when you start using computer you do not know what is inside, how it is really processing everything, but you can still use computer and like write your codes in a C language.

So C language you learn and like you do not worry about how it really processes things of each of the C language command inside, it is like that. So, even if you do not know how these circuits are doing their job, you, is enough to know that by using some kind of a rule like these or some kind of circuits like these, like your output can be produced, so that is what is most important kind of understanding here.

(Refer Slide Time: 17:05)



More examples of combinational logic circuits

- Half and full adder: useful in ALU
- Multiplexer: useful in building multiple channel interfaces ex. ADC
- Comparator: useful in ALU
- 7 segment display: Output device
- You may skip this part if you alrea know these

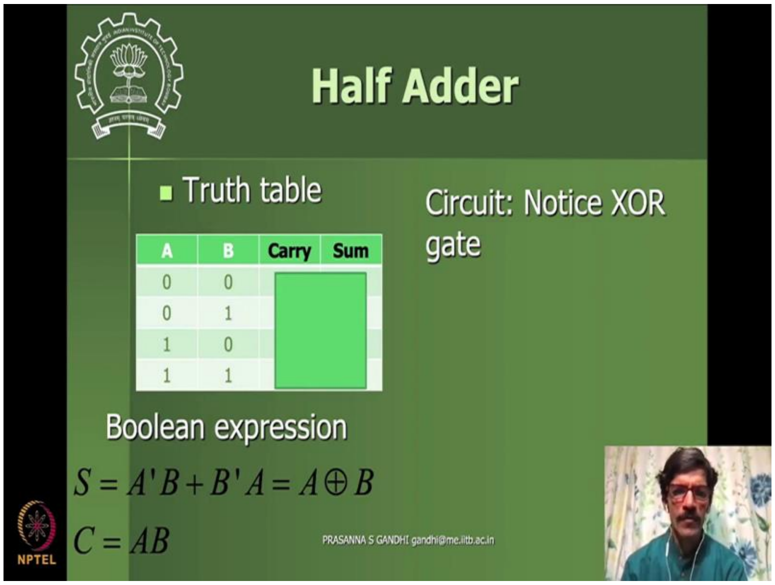
PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, the way this calendar example is done now we will see say some additional application if you want to do so half adder and full adders are which are useful in arithmetic logic unit in the microprocessor then multiplexers you can develop, they is useful in developing multiple channel interfaces for analog to digital conversion that is one of the interface for microcontroller or microprocessor, then comparators again useful in arithmetic logic units.

So, these are some few kind of examples only which we have taken, but there are many many different kinds of these circuits that can be possible and one can do a lot of operations, say logical operations and these addition operations, comparison operations like that, that those all operations can be done by using these combinational logic circuits. So, these are the at the heart of all these microprocessors particularly arithmetic logic unit in microprocessor.

And you have the seven segment display one of the examples we have which is an output device where like this seven segments will give you some kind of a representation of a number to display to the user. So, if you are aware about like know these devices already you do not need to kind of go through this part already. So, you just, because this is just same kind of application of our combination logic fundamentals.

(Refer Slide Time: 18:45)



Half Adder

- Truth table

A	B	Carry	Sum
0	0		
0	1		
1	0		
1	1		

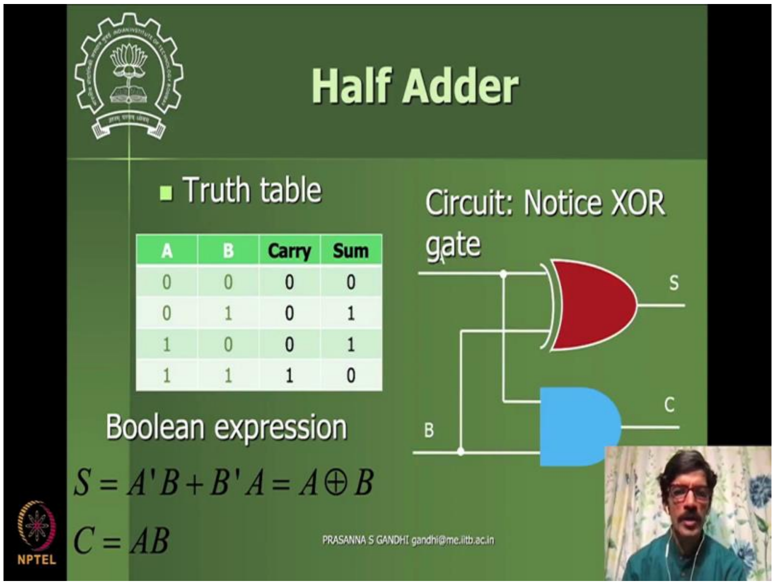
Circuit: Notice XOR gate

Boolean expression

$S = A'B + B'A = A \oplus B$

$C = AB$

PRASANNA S GANDHI gandhi@me.iitb.ac.in



Half Adder

- Truth table

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Circuit: Notice XOR gate

Boolean expression

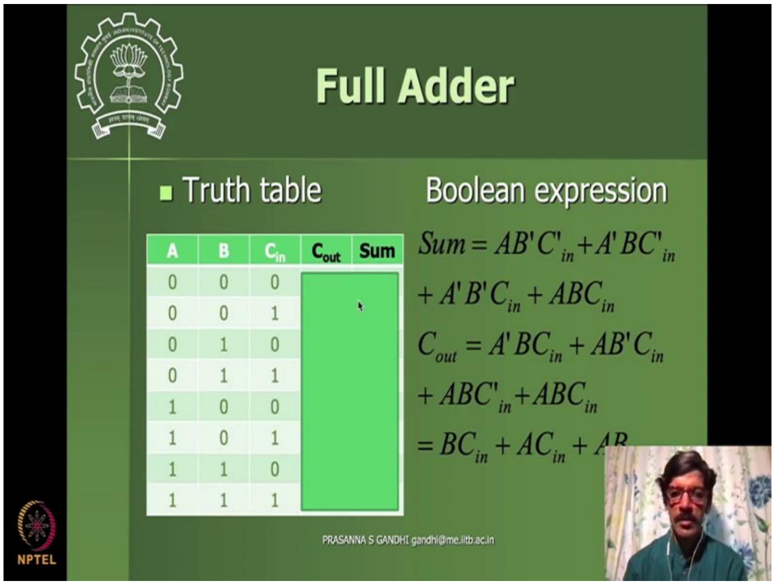
$S = A'B + B'A = A \oplus B$

$C = AB$

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, we say for example, now this half adder. So, there are these two numbers A and B binary numbers when we want to add together and then like it should have a carry. So, how, what is the expression for carry and sum. So, if A is 0, B 0 and carry is 0 and sum is 0. If A is, so like that, you fill out this table, you pause here and fit out and then again see that, it will be like this. And then for implementing this you can find out this Boolean expression and once you get this Boolean expression, you can get corresponding circuit related to that. So, there is an XOR gate coming up here.

(Refer Slide Time: 19:40)



Full Adder

- Truth table

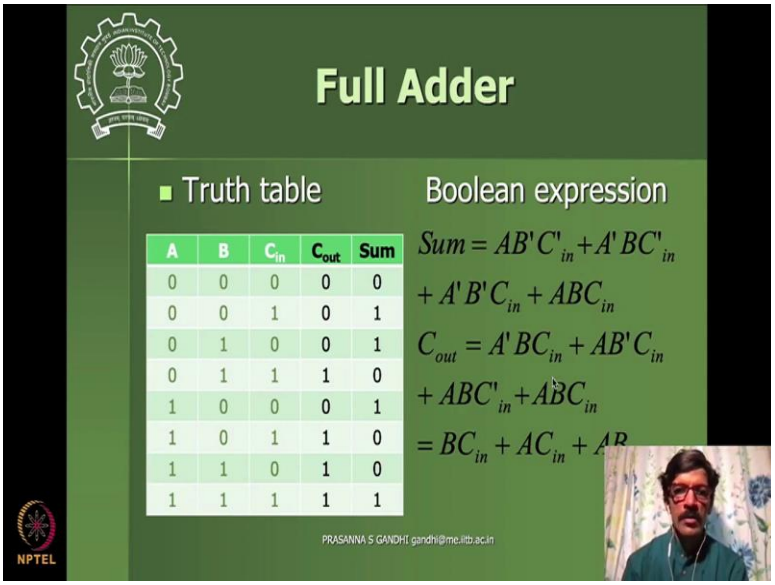
A	B	C_{in}	C_{out}	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Boolean expression

$$Sum = AB'C'_{in} + A'BC'_{in} + A'B'C_{in} + ABC_{in}$$

$$C_{out} = A'BC_{in} + AB'C_{in} + ABC'_{in} + ABC_{in} = BC_{in} + AC_{in} + AB$$

PRASANNA S GANDHI gandhi@me.iitb.ac.in



Full Adder

- Truth table

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Boolean expression

$$Sum = AB'C'_{in} + A'BC'_{in} + A'B'C_{in} + ABC_{in}$$

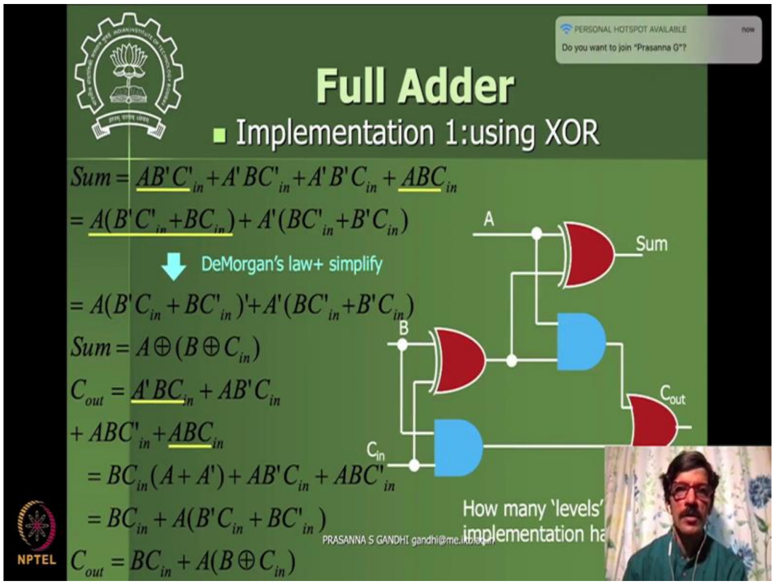
$$C_{out} = A'BC_{in} + AB'C_{in} + ABC'_{in} + ABC_{in} = BC_{in} + AC_{in} + AB$$

PRASANNA S GANDHI gandhi@me.iitb.ac.in

Then like that you can have a full adder. So, you have a carry in also now, so A B and carry in also is there so that you can now multiplex it and get more, some more the more of these (())(19:52). So then you go ahead with the same process for these and now we have three inputs, and now you see what are your outputs.

So again, pause here and then proceed, we will get these as our outputs and then you can find out corresponding expressions for that, then those expressions can be converted into circuits by using AND gates and OR gates and NOT gates, anything like that or XOR gates anything like that.

(Refer Slide Time: 20:26)



PERSONAL HOTSPOT AVAILABLE now
Do you want to join "Prasanna G"?

Full Adder

- Implementation 1:using XOR

$Sum = \underline{AB'C'_{in}} + A'BC'_{in} + A'B'C_{in} + \underline{ABC_{in}}$

$= \underline{A(B'C'_{in} + BC_{in})} + A'(BC'_{in} + B'C_{in})$

DeMorgan's law+ simplify

$= A(B'C_{in} + BC'_{in})' + A'(BC'_{in} + B'C_{in})$

$Sum = A \oplus (B \oplus C_{in})$

$C_{out} = \underline{A'BC_{in}} + AB'C_{in}$

$+ ABC'_{in} + \underline{ABC_{in}}$

$= BC_{in}(A + A') + AB'C_{in} + ABC'_{in}$

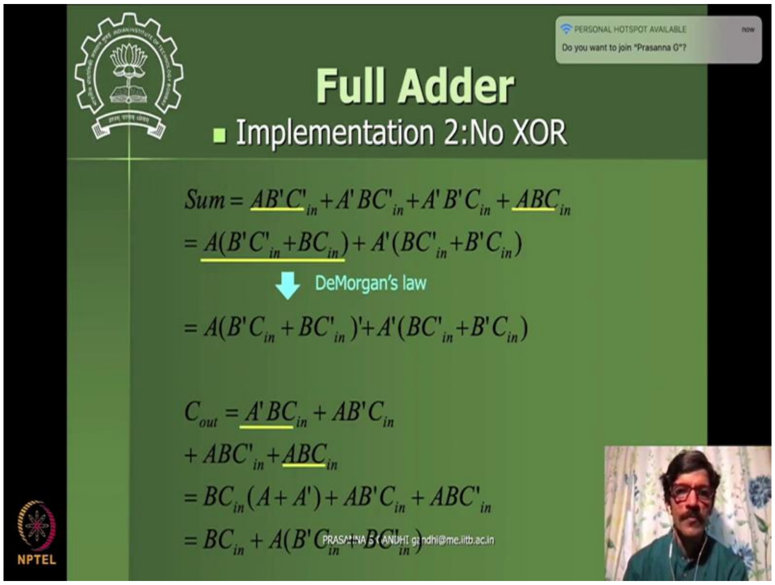
$= BC_{in} + A(B'C_{in} + BC'_{in})$

$C_{out} = BC_{in} + A(B \oplus C_{in})$

How many 'levels' implementation ha

PRASANNA S GANDHI gandhi@me.iitb.ac.in

NPTEL



PERSONAL HOTSPOT AVAILABLE now
Do you want to join "Prasanna G"?

Full Adder

- Implementation 2:No XOR

$Sum = \underline{AB'C'_{in}} + A'BC'_{in} + A'B'C_{in} + \underline{ABC_{in}}$

$= \underline{A(B'C'_{in} + BC_{in})} + A'(BC'_{in} + B'C_{in})$

DeMorgan's law

$= A(B'C_{in} + BC'_{in})' + A'(BC'_{in} + B'C_{in})$

$C_{out} = \underline{A'BC_{in}} + AB'C_{in}$

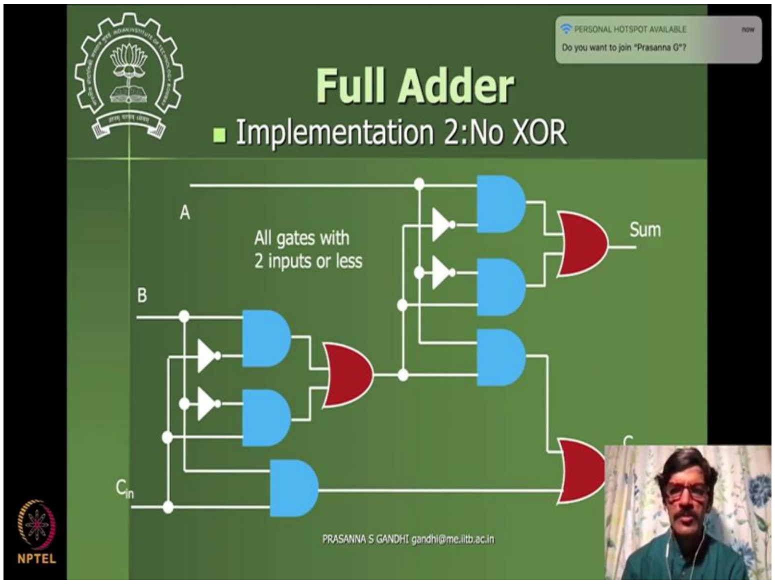
$+ ABC'_{in} + \underline{ABC_{in}}$

$= BC_{in}(A + A') + AB'C_{in} + ABC'_{in}$

$= BC_{in} + A(B'C_{in} + BC'_{in})$

PRASANNA S GANDHI gandhi@me.iitb.ac.in

NPTEL



PERSONAL HOTSPOT AVAILABLE now
Do you want to join "Prasanna G"?

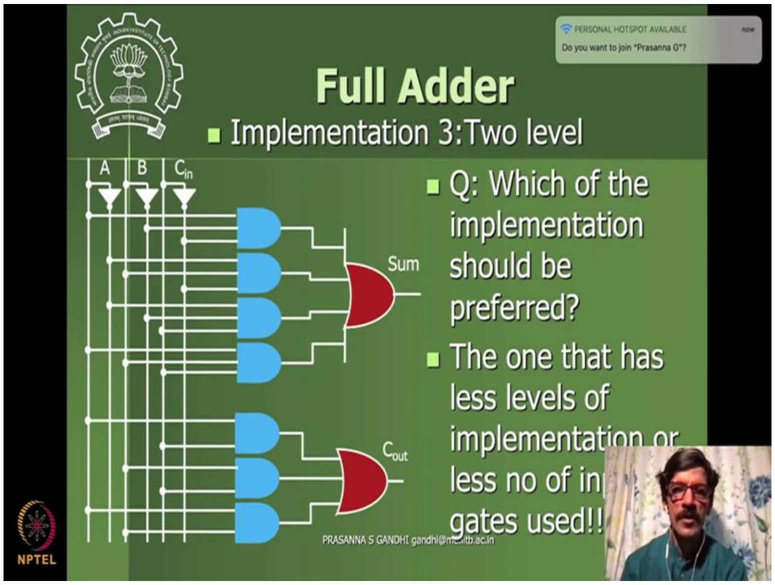
Full Adder

- Implementation 2:No XOR

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So like that this is a implementation, one of the implementations of the adder, then this is another kind of expression where you do not want to use XOR gate. So, you have to have some simplifications done to get to these points and things like that. So, you have this circuit coming up exactly the same kind of implementation carried out by this circuit also and the previous circuit also, but now, there are some kind of other simplifications in particularly here there are no XOR gates in this case. So, like that you can have some conditions put and find out that it is satisfying those conditions.

(Refer Slide Time: 21:14)



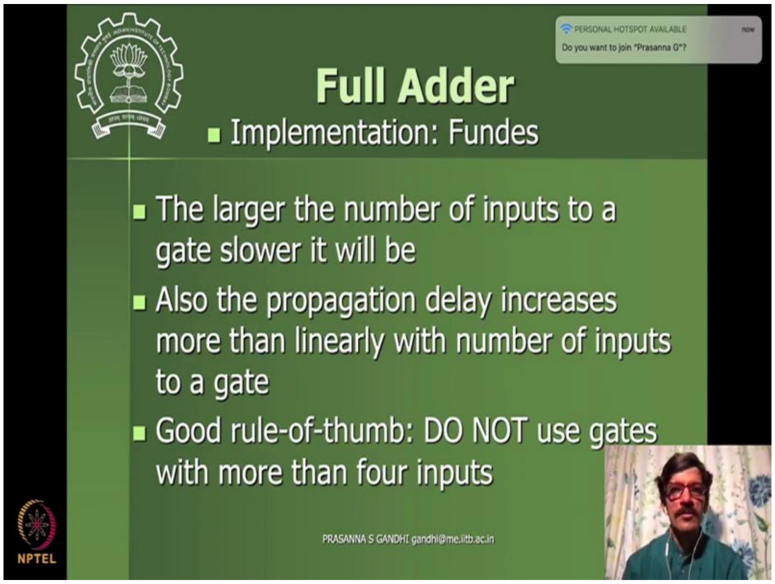
PERSONAL HOTSPOT AVAILABLE now
Do you want to join "Prasanna G"?

Full Adder

- Implementation 3:Two level

- Q: Which of the implementation should be preferred?
- The one that has less levels of implementation or less no of in gates used!!

PRASANNA S GANDHI gandhi@me.iitb.ac.in



PERSONAL HOTSPOT AVAILABLE now

Do you want to join "Prasanna G"?

Full Adder

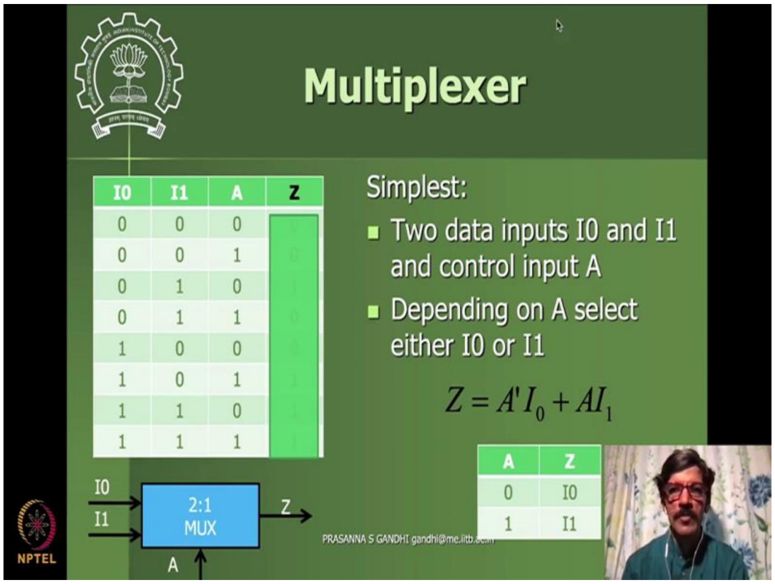
- Implementation: Fundes

- The larger the number of inputs to a gate slower it will be
- Also the propagation delay increases more than linearly with number of inputs to a gate
- Good rule-of-thumb: DO NOT use gates with more than four inputs

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, there is a levels of implementation that that will also be under kind of a constraint that can come in. So, we will not get again into details of these, but one can go into these different different kinds of aspects, so this another third implementation of the of the full adder kind of circuit. So, there are some kind of fundamental thumb rules for implementation of the circuits coming up mainly from the circuit design perspective, which we are not too much interested in getting there, we just that full adder works like this is sufficient for us to know.

(Refer Slide Time: 21:52)



Multiplexer

I0	I1	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

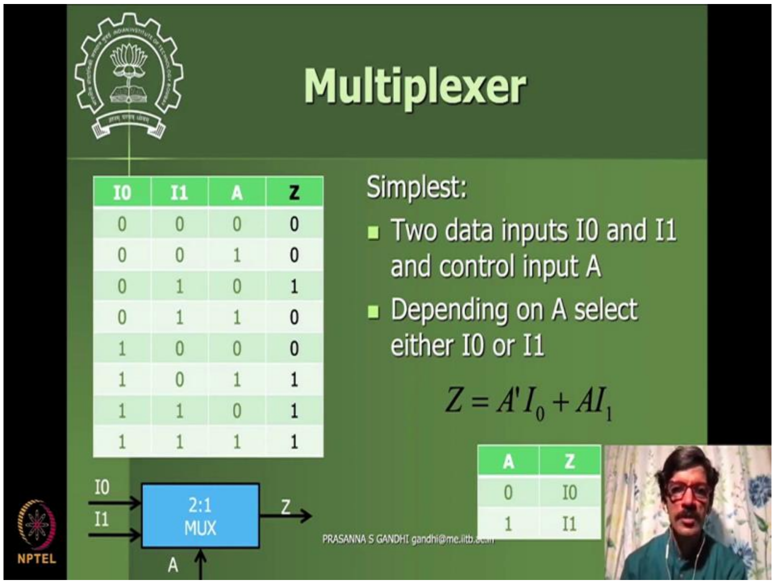
Simplest:

- Two data inputs I0 and I1 and control input A
- Depending on A select either I0 or I1

$$Z = A'I_0 + AI_1$$

A	Z
0	I0
1	I1

PRASANNA S GANDHI gandhi@me.iitb.ac.in



Multiplexer

I0	I1	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Simplest:

- Two data inputs I0 and I1 and control input A
- Depending on A select either I0 or I1

$$Z = A'I_0 + AI_1$$

A	Z
0	I0
1	I1

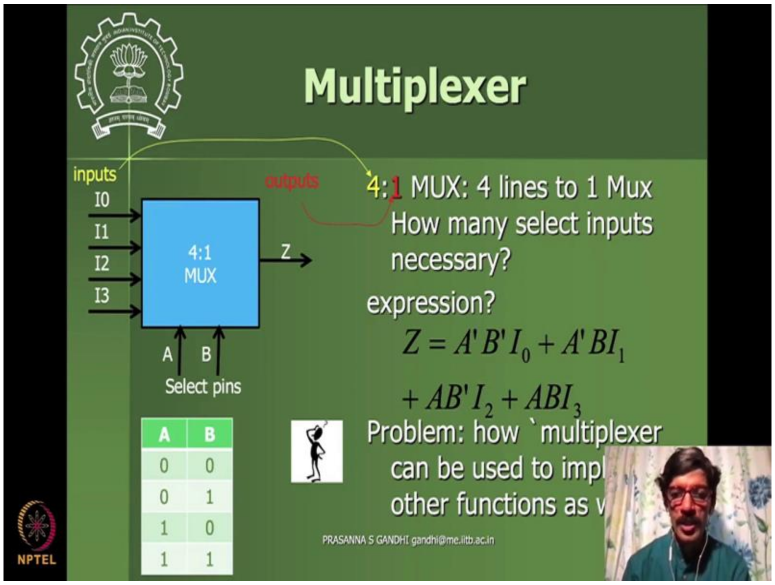
PRASANNA S GANDHI gandhi@me.iitb.ac.in

Then you have a multiplexer. So, a multiplexer has this input 0 and input 1. So, two data inputs are the I O and 1 and control input is A depending upon value of A, the one of the inputs you have selected here. So, you can see that this condition like, say for example, I O is 0 and this one is also 0 and value of A is 0 or 1 depending upon that, if A is 0, I will select I 0 as Z and if A is 1, I will select I 1 as Z, so like that that circuit will come up here. So you fill up the entire table, and then we will proceed.

So, you will see these tables something like this. So, when A is 0, it is selecting this input, A is 1 it is selecting this input. So, these inputs are like these two cases could be possible where input is same. I O and I 1 has some values which are same, but A different. So, A is 0 it will select, I think this is different in there. So, depending upon A select, so A is 0, it should select I O or I 1 depending upon like we will see what is getting selected here.

So, here A is 0 it is selecting I 1 here and A is 1 it is selecting this here, here again same thing yeah. So, A is 0 it is selecting I 1, A is 1 it is selecting I 0. So, like that. So, does not matter you can have your own way of having this also, but this is one particular kind of a definition like depending upon A you are selecting the output and then you can directly write in terms of inside this I 0 or I 1 so you can write it like this also directly.

(Refer Slide Time: 24:16)



Multiplexer

4:1 MUX: 4 lines to 1 Mux

How many select inputs necessary?

expression?

$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

A	B
0	0
0	1
1	0
1	1

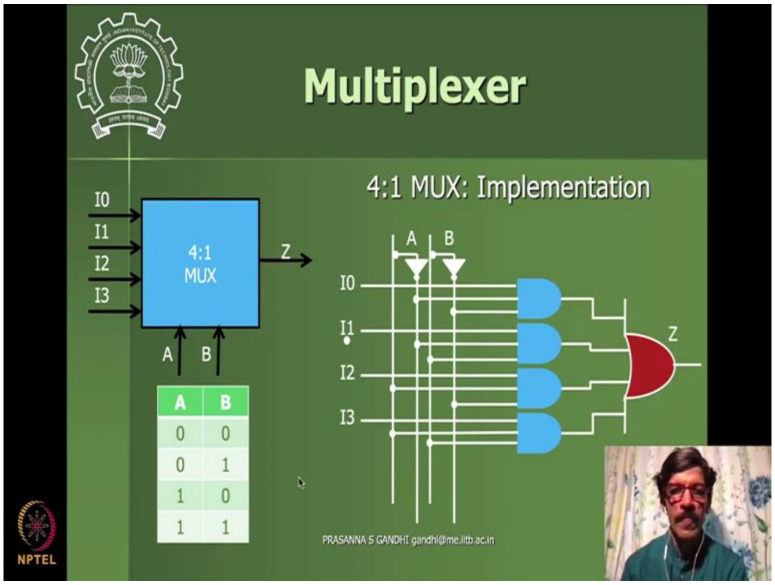
Problem: how `multiplexer can be used to impl other functions as v

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, this implementation of this is again you can convert these into a circuit. And you can have, scaling of this multiplexer as possible. So, 4 is to 1 multiplexer is here, you have 4 inputs and 1 output is there. And out of these 4, one will be selected depending upon the select pins, the select pins are inputs.

So now, to have four inputs, one of the four inputs selected, you need to have two select pins and then you will have corresponding expression for set, and so, this we will not worry about this. So, think about this other problem, this is a separate problem that you can use multiplexer instead of your AND OR gates for implementation of some of the logic circuits or logic tables.

(Refer Slide Time: 25:17)



Multiplexer

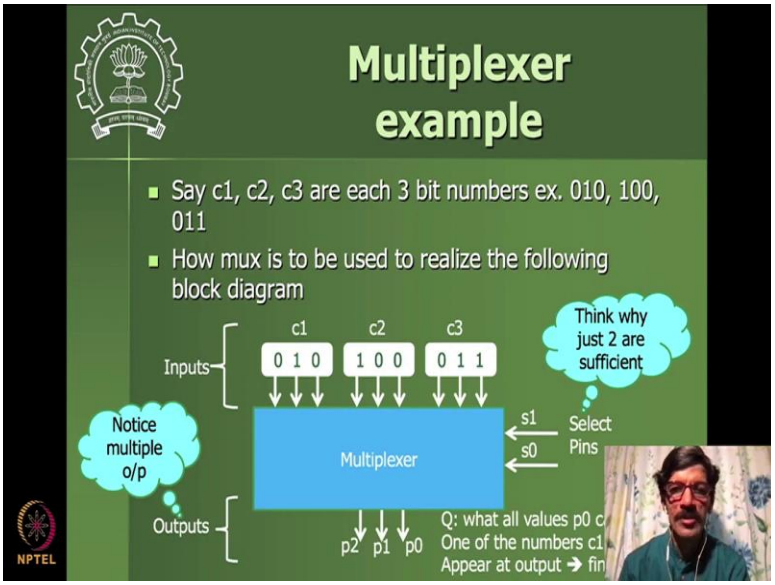
4:1 MUX: Implementation

A	B
0	0
0	1
1	0
1	1

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, this is a way the multiplexer can be implemented and you can give this I0, I1,I4, I2 these are the inputs that are coming and then these are the select pins and depending upon the select pins you can have a multiplexer implementation, the Z output is coming.

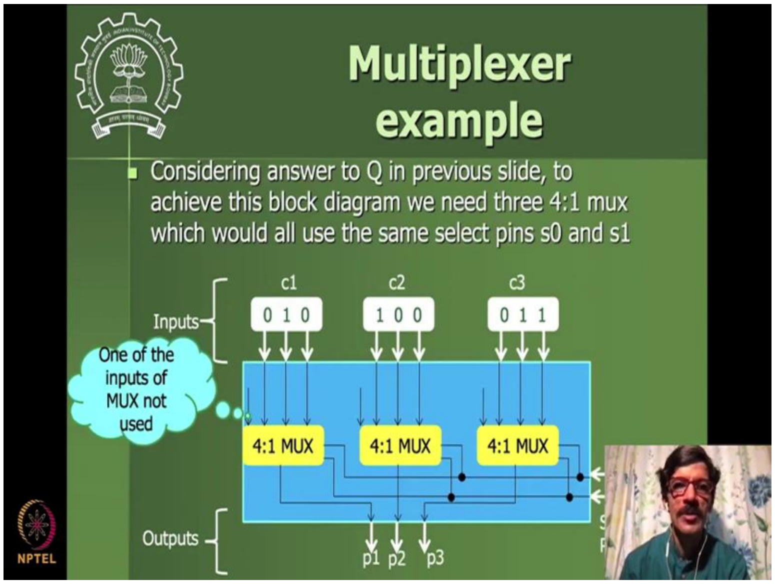
(Refer Slide Time: 25:36)



Multiplexer example

- Say c1, c2, c3 are each 3 bit numbers ex. 010, 100, 011
- How mux is to be used to realize the following block diagram

Q: what all values p0 c
One of the numbers c1
Appear at output → fin

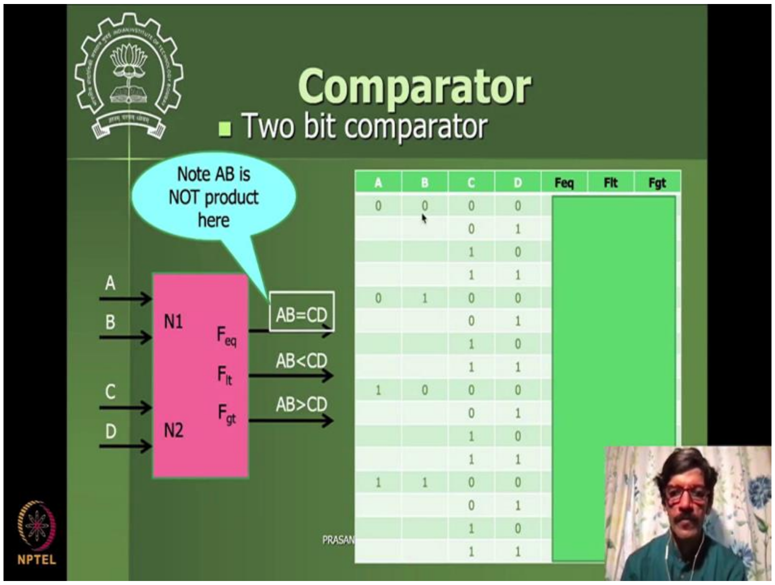


Multiplexer example

- Considering answer to Q in previous slide, to achieve this block diagram we need three 4:1 mux which would all use the same select pins s0 and s1

So, like that you can have many different other examples. So, maybe you can go through these lectures yourself, go through these slides yourself, this same kind of process we are repeating over and over again.

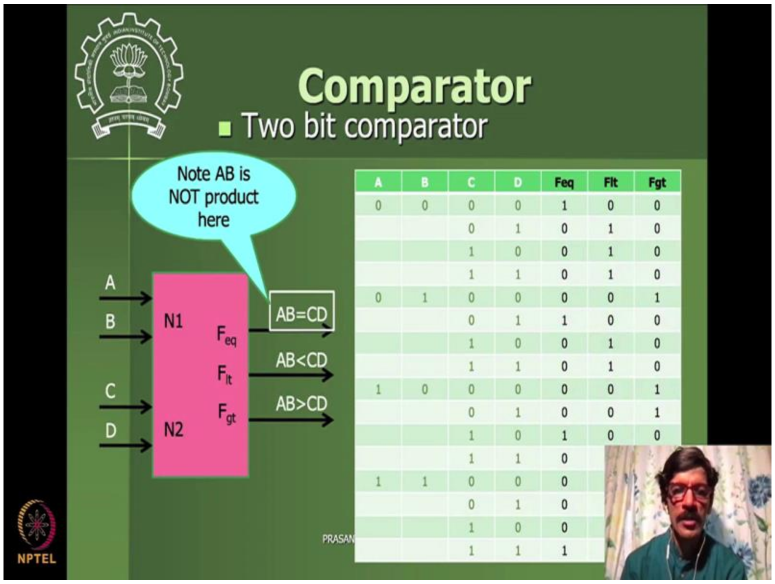
(Refer Slide Time: 25:58)



Comparator

- Two bit comparator

A	B	C	D	Feq	Flt	Fgt
0	0	0	0			
		0	1			
		1	0			
		1	1			
0	1	0	0			
		0	1			
		1	0			
		1	1			
1	0	0	0			
		0	1			
		1	0			
		1	1			
1	1	0	0			
		0	1			
		1	0			
		1	1			



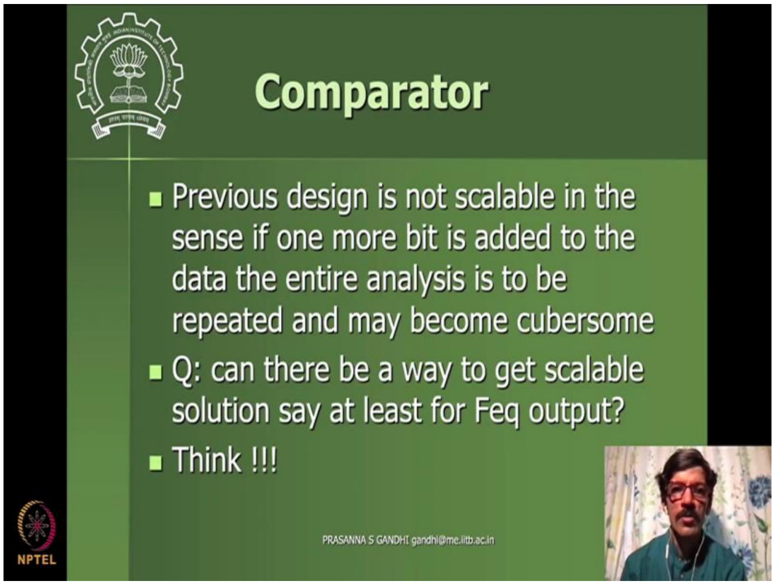
Comparator

- Two bit comparator

Note AB is NOT product here

A	B	C	D	Feq	Flt	Fgt
0	0	0	0	1	0	0
		0	1	0	1	0
		1	0	0	1	0
		1	1	0	1	0
0	1	0	0	0	0	1
		0	1	1	0	0
		1	0	0	1	0
		1	1	0	1	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	1	0	0
		1	1	0		
1	1	0	0	0		
		0	1	0		
		1	0	0		
		1	1	1		

PRASAN



Comparator

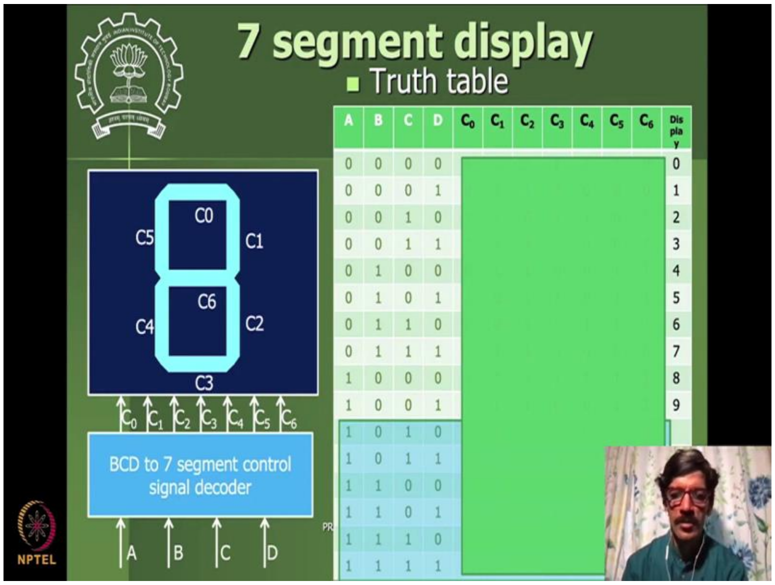
- Previous design is not scalable in the sense if one more bit is added to the data the entire analysis is to be repeated and may become cubersome
- Q: can there be a way to get scalable solution say at least for Feq output?
- Think !!!

PRASANNA S GANDHI gandhi@me.iitb.ac.in

So, in a similar way you can have definition of comparator. So, you want to compare two numbers which are represented by, one number is A B and other number is CD. So, A B are like two binary kind of bits of one number A B and bits of the other number are C and D and then you need to kind of give the output if it is less and if it is more like you can have some kind of outputs that are possible.

So, you can think of this implement, the truth table for this will come up something like that. So, A B and C D they are equal then this will be 1, if they are less like, if they are more that will be 1 like that it will have these three inputs coming up, coming out, three outputs coming out of this circuit, given based on these 2 or 4 inputs in terms of bits and then you will get this entire table represented and then you can find out further how it is a circuit could be.

(Refer Slide Time: 27:12)

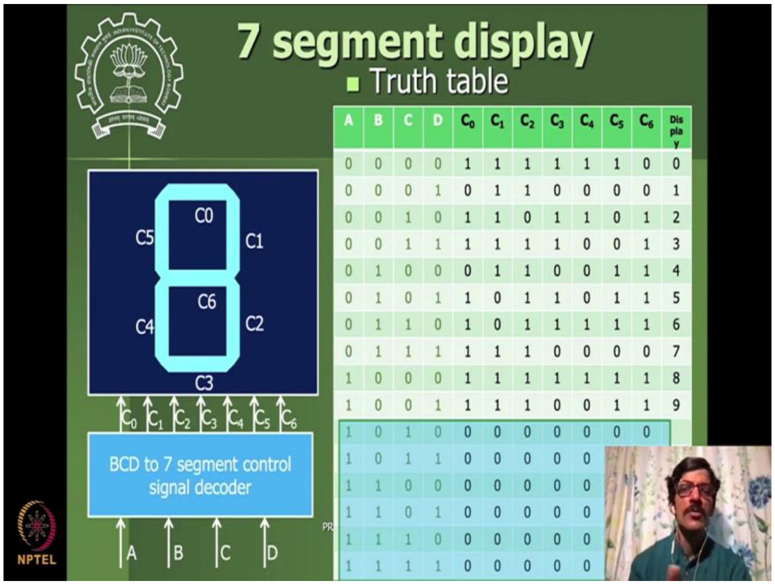


7 segment display

- Truth table

A	B	C	D	C_0	C_1	C_2	C_3	C_4	C_5	C_6	Display
0	0	0	0								0
0	0	0	1								1
0	0	1	0								2
0	0	1	1								3
0	1	0	0								4
0	1	0	1								5
0	1	1	0								6
0	1	1	1								7
1	0	0	0								8
1	0	0	1								9
1	0	1	0								
1	0	1	1								
1	1	0	0								
1	1	0	1								
1	1	1	0								
1	1	1	1								

BCD to 7 segment control signal decoder



7 segment display

- Truth table

A	B	C	D	C_0	C_1	C_2	C_3	C_4	C_5	C_6	Display
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9
1	0	1	0	0	0	0	0	0	0	0	
1	0	1	1	0	0	0	0	0			
1	1	0	0	0	0	0	0	0			
1	1	0	1	0	0	0	0	0			
1	1	1	0	0	0	0	0	0			
1	1	1	1	0	0	0	0	0			

BCD to 7 segment control signal decoder

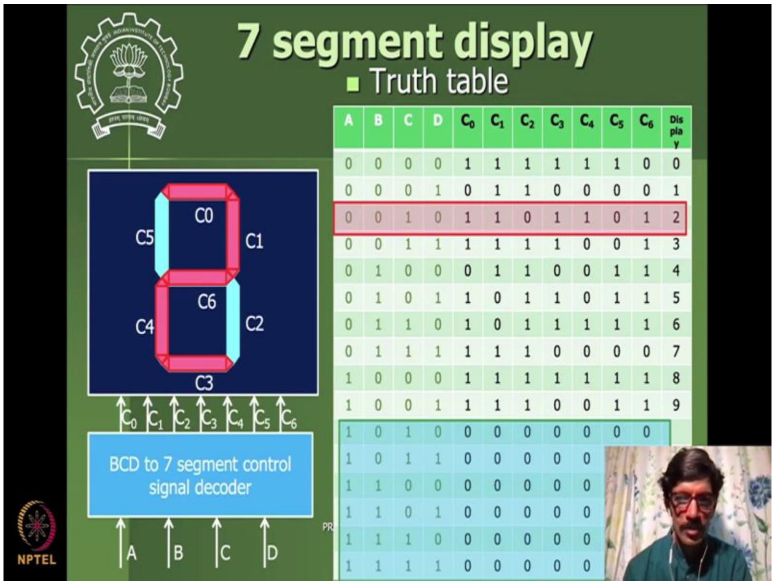
Now, there are these other kind of output elements called seven segment display this is very important for mechatronics people because a lot of these displays can be programmed with this understanding of the seven-segment display, this is nothing this just a combination kind of logic here. And you want to display say 1 here you need to see what are the, these elements of this display to be, these are a basically LEDs they should be lightened up to display 1 or 0 or whatever value you want to display here depending upon that you will have multiple outputs.

And then you can have additional thing problems seeing that, okay look my binary number needs to be getting represented here, 8 binary numbers or 10 binary numbers from 0 to 9 should get displayed here starting from 0000 to 1001, they need to get displayed here in appropriate kind of question then like how do I kind of program this circuit which can do that.

So, this is like BCD to seven segment control decoder kind of circuit that we can design by considering this logic table now, we can fill up this logic table and again the same process needs to be carried out and anyone can get like a big circuits to kind of implement this logic.

And then, so, these are different different kinds of circuits that are possible and like done in the, as a basic kind of circuits for these different different elements and then these elements now can be further put together to create your entire microprocessor.

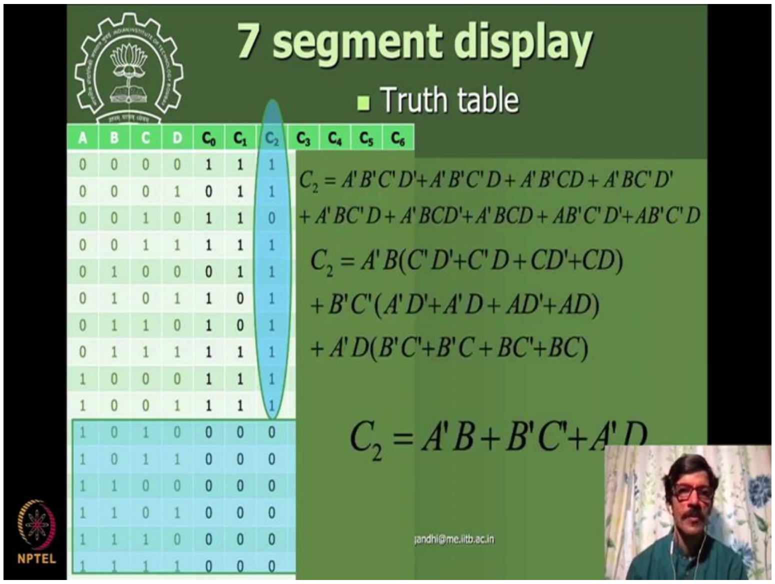
(Refer Slide Time: 29:06)



7 segment display

- Truth table

A	B	C	D	C_0	C_1	C_2	C_3	C_4	C_5	C_6	Display
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9
1	0	1	0	0	0	0	0	0	0	0	
1	0	1	1	0	0	0	0	0			
1	1	0	0	0	0	0	0	0			
1	1	0	1	0	0	0	0	0			
1	1	1	0	0	0	0	0	0			
1	1	1	1	0	0	0	0	0			



7 segment display

- Truth table

A	B	C	D	C_0	C_1	C_2	C_3	C_4	C_5	C_6
0	0	0	0	1	1	1				
0	0	0	1	0	1	1				
0	0	1	0	1	1	0				
0	0	1	1	1	1	1				
0	1	0	0	0	1	1				
0	1	0	1	1	0	1				
0	1	1	0	1	0	1				
0	1	1	1	1	1	1				
1	0	0	0	1	1	1				
1	0	0	1	1	1	1				
1	0	1	0	0	0	0				
1	0	1	1	0	0	0				
1	1	0	0	0	0	0				
1	1	0	1	0	0	0				
1	1	1	0	0	0	0				
1	1	1	1	0	0	0				

$$C_2 = A'B'C'D' + A'B'C'D + A'B'CD + A'BC'D'$$
$$+ A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'C'D$$

$$C_2 = A'B(C'D' + C'D + CD' + CD)$$
$$+ B'C'(A'D' + A'D + AD' + AD)$$
$$+ A'D(B'C' + B'C + BC' + BC)$$

$$C_2 = A'B + B'C' + A'D$$

jandhi@me.iitb.ac.in

So, this is for example, you want to display 2, you want this LED and this LED this LED this LED and this LED to glow. So, that is what is given here. So, 2 is 1 0 representation in binary ABCD four numbers are represented like that, and then your corresponding, this seven segment display LEDs are glowing they are said as 1 and then you can get this number coming up there.

So, it took 2, like that you can fill up entire table in the similar kind of fashion and once this table is filled up, then you can look at one output here and in terms of four inputs, like what combinational circuit will come up is what you can consider. So, one can do that, I mean that is very, what do you say? Simple but tedious kind of a job when we go for like this kind of a big circuits multiple such elements.

So these are the big expressions that are coming up here for example. So now like we will stop here for now and we will take, take up in the next class this sequential circuits.