

Digital Circuits



Prof. Santanu Chattopadhyay
IIT KHARAGPUR

INDEX

<u>S.No.</u>	<u>Topic</u>	<u>Page No.</u>
<i>WEEK- 1</i>		
1	Introduction	01
2	Introduction (Contd.)	19
3	Number System	38
4	Number System (Contd.)	62
5	Number System (Contd.)	86
<i>WEEK- 2</i>		
6	Number System (Contd.)	104
7	Number System (Contd.)	127
8	Boolean Algebra	145
9	Boolean Algebra (Contd.)	167
10	Boolean Algebra (Contd.)	174
<i>WEEK- 3</i>		
11	Boolean Algebra (Contd.)	190
12	Boolean Algebra (Contd.)	205
13	Boolean Algebra (Contd.)	222
14	Logic Gates	242
15	Logic Gates (Contd.)	263
<i>WEEK- 4</i>		
16	Logic Gates (Contd.)	279
17	Logic Gates (Contd.)	295
18	Logic Gates (Contd.)	308
19	Logic Gates (Contd.)	328
20	Arithmetic Circuits	347
<i>WEEK- 5</i>		
21	Arithmetic Circuits (Contd.)	370
22	Arithmetic Circuits (Contd.)	402
23	Decoders, Multiplexers, PLA	413

24	Decoders, Multiplexers, PLA (Contd.)	431
25		449

WEEK- 6

26	Decoders, Multiplexers, PLA (Contd.)	462
27	Decoders, Multiplexers, PLA (Contd.)	476
28	Sequential Circuits	493
29	Sequential Circuits (Contd.)	513
30	Sequential Circuits (Contd.)	528

WEEK- 7

31	Sequential Circuits (Contd.)	543
32	Sequential Circuits (Contd.)	558
33	Sequential Circuits (Contd.)	574
34	Sequential Circuits (Contd.)	591
35	Finite State Machine	618

WEEK- 8

36	Finite State Machine (Contd.)	638
37	Data Converters	658
38	Data Converters (Contd.)	676
39	Data Converters (Contd.)	697
40	Data Converters (Contd.)	713

WEEK- 9

41	Memory	728
42	Memory (Contd.)	746
43	Memory (Contd.)	764
44	FPGA	783
45	FPGA (Contd.)	799

WEEK- 10

46	VHDL	814
47	VHDL	833

	(Contd.)	
48	8085 Microprocessor	849
49	8085 Microprocessor (Contd.)	866
50	8085 Microprocessor (Contd.)	879

WEEK- 11

51	8085 Microprocessor (Contd.)	895
52	8085 Microprocessor (Contd.)	908
53	8085 Microprocessor (Contd.)	925
54	8085 Microprocessor (Contd.)	942
55	8085 Microprocessor (Contd.)	963

WEEK- 12

56	8085 Microprocessor (Contd.)	980
57	8085 Microprocessor (Contd.)	997
58	8085 Microprocessor (Contd.)	1014
59	8085 Microprocessor (Contd.)	1031
60	8085 Microprocessor (Contd.)	1047
61	8085 Microprocessor (Contd.)	1063
62	8085 Microprocessor (Contd.)	1085
63		1098

8086 Microprocessor

64	8086 Microprocessor (Contd.)	1118
65	8086 Microprocessor (Contd.)	1145

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 01
Introduction

Welcome to this course on Digital Circuits. So, if you look into this VLSI design process or VLSI industrial development where these integrated circuit chips are being manufactured. So, you will see that most of the designs that we have so, they are digital in nature.

So, the basis of all these developments are on these type of circuits where we consider the digital circuits as the building block.

(Refer Slide Time: 00:45)

The slide has a yellow header and a blue footer. The title 'The Start of the Modern Electronics Era' is at the top. Below it are two black and white photographs: one of three men (Bardeen, Shockley, and Brattain) in a lab, and another of a small electronic device. The text below the left photo reads: 'Bardeen, Shockley, and Brattain at Bell Labs - Brattain and Bardeen invented the bipolar transistor in 1947.' The text below the right photo reads: 'The first germanium bipolar transistor. Roughly 50 years later, electronics account for 10% (4 trillion dollars) of the world GDP.' The footer contains the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

So, the start of this modern electronics era if you look back in to the history so, we will see that it started around the revolution started in 1947, where a group of scientists at Bell Labs they invented the bipolar transistors in 1947. So, 1947 is a very important year for India as well we got freedom, and similarly the electronics industry. So, this is another landmark that we had achieved in 1947.

So, from that point onwards the electronics design it has advanced a lot. And today we are at a stage where the complexity of systems and their performance have gone up by

leaps and bounds. And now it is almost whatever functionality we think about so, we will look for an integrated circuit chip which can do that. And majority of them are digital in nature.

So, we will why this has happened. So, if you look into the, these slides on the next in the right side, it shows the first germanium bipolar transistor. So, this left side. So, this was silicon, based on silicon. So, this is the bipolar transistor based on germanium. So, 50 years later so, this electronics account for about 10 percent of the world's total GDP.

(Refer Slide Time: 02:14)

Electronics Milestones

1874	Braun invents the solid-state rectifier.	1958	Integrated circuit developed by Kilby and Noyce
1906	DeForest invents triode vacuum tube.	1961	First commercial IC from Fairchild Semiconductor
1907-1927	First radio circuits developed from diodes and triodes.	1963	IEEE formed from merger of IRE and AIEE
1925	Lilienfeld field-effect device patent filed.	1968	First commercial IC opamp
1947	Bardeen and Brattain at Bell Laboratories invent bipolar transistors.	1970	One transistor DRAM cell invented by Dennard at IBM.
1952	Commercial bipolar transistor production at Texas Instruments.	1971	4004 Intel microprocessor introduced.
1956	Bardeen, Brattain, and Shockley receive Nobel prize.	1978	First commercial 1-kilobit memory.
		1974	8080 microprocessor introduced.
		1984	Megabit memory chip introduced.
		2000	Alferov, Kilby, and Kromer share Nobel prize

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this is the milestones that we have in the electronics industry. In 1874, Braun invented the solid state rectifier. Then 1906 DeForest, DeForest invented triode vacuum tube. So, that way this the whole development has taken place. So, it started with solid state rectifier then the vacuum tube diode the triodes, then the radio circuits, then field effect devices the patent was filled in 1925.

1947 as we already said so, there is a bipolar transistor was invented. Then 1952, Texas instruments they started making these bipolar transistors commercially. 1956 that there was a Nobel Prize given to Bardeen, Brattain and Shockley. So, they receive the Nobel Prize for this transistor.

Then IC was developed in 1958 by Kilby and Noyce, and then the first commercial IC came from the Fairchild semiconductor in 1961. So, as this industry was advancing so, so

the scientists also started doing lots of research works, and there was an institution of engineers ie which was formed - IEEE so, electrical and electronics engineers.

So, that was found by merger of IRE and AIEE. So, IEEE is one of the very popular societies now that we have and they apart from just bringing peers close to each other. They make a lot of standards for the new devices and systems that are coming up, and that way they are helping in the electronics industry.

So, whenever we are looking for some development of some device or system, we normally look for some standards and that standardization is generally done by IEEE whatever we have today. The operational amplifier; so, this is the analog part. So, though we will not discuss much about the analog revolution, but this analog part also has started being developed from 1968.

So, first operational amplifier IC came up in 1968. The other important part that we have in it is system, digital system; so, apart from during computation. So, you need to store lot of information and in that direction the memory chips. So, they are the main part that we have. So, they so, transistors are again used for making the memory chips. So, this dynamic RAM that we see in almost all the computer systems today. So, they are they were invented in 1970 at IBM by Dennard.

So, in the involution process as this computations and computations and memory storage. So, they went on improving over the generations. So, this computational module. So, they were clubbed together into a structure called microprocessor, where it can do some set of operations in taking help of some registers are may be taking executing some stored program, which it can access through some address and data lines. So, that microprocessor came in 1971. So, it was Intel 4004 microprocessor.

So, and now you see that after that it has gone a long way, and today we have got this Pentium processors and other RISC machines and all so, they are all very advanced versions of this microprocessors. So, this is another digital component that has involved a lot. Then this commercially memory chips they started coming in 1978. So, first one kilo bit memory chip was available in 1978. 1974 another very popular microprocessor 8080 came up.

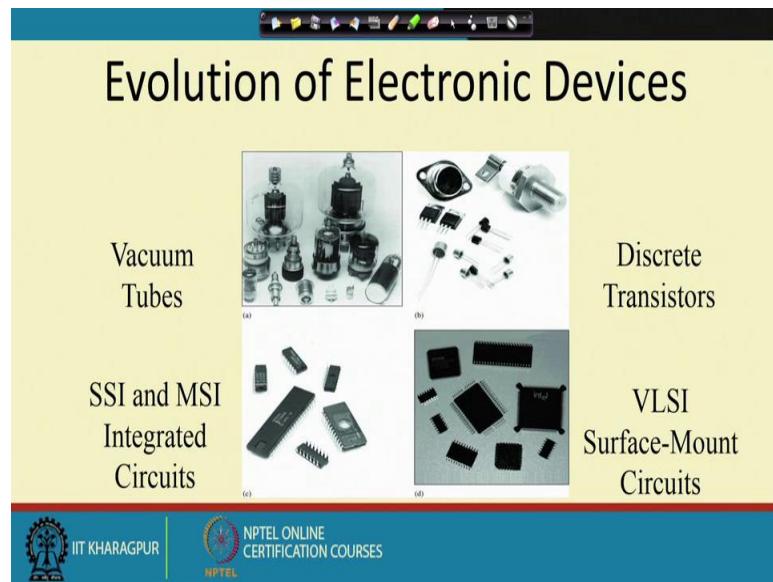
So, this is one of the, I should say with the one of the first well known microprocessor 8080 of the, again from Intel. And that so, based on this microprocessor there has been they lot of developments and all this later microprocessors that are available; so, they are actually developed from this microprocessor architecture.

We will see one such microprocessor in this particular course which is 8085. So, it has it is bit improved version over 8080. In 1984, we have got mega bit memory chip that was introduced so, memory capacity went up. So, this was a because of because of this we could we can now stored lots of memory and today storage is not a problem. Because this memory chips they have advanced so much like even this tiniest devices that we have in our day to day life, electronic devices. So, they have got large amount of memory that we can have in the system. So, they can store lots of information.

So, that is the, that is the point so, we this memory chip development that is also digital in nature. So, this digital circuits so, they helped us in developing this memory chips to get the functionality implemented by the systems. Then the, this another Nobel prize came up in 2000 to Alferov, Kilby and Kromer. So, they have got another Nobel Prize for this memory and all.

So, this way this electronic industry came. So, after 2000 so, we do not have the information here, but as you are all familiar. So, lots of development are taking place every day where this electronics industry is advancing towards better and better processors with higher and higher capabilities, and better and higher memory storage capacities.

(Refer Slide Time: 08:22)



So, if pictorially so, this shows that. So, this is the first generation where we have got this is the vacuum tube diode triodes and all. Then they came that discrete transistors. So, you see the volumes or size of the devices. So, that has decreased price so much you see. So, this is a power, power transistor so, this is another transistor. So, these are some diode some transistors and all so, they are all discrete transistors.

Then after that came this one. So, where we have got this small scale integrated circuits and medium scale integrated circuits, MSI and SSI. So, they are actually this IC chips started coming. So, they do not have large number of transistors in them, SSI chips they will have about 10 transistors, MSI will have about thousand, 100 transistors. LSI will have thousand transistors, and then VLSI is going beyond that.

So, that way we have got large number of transistors. So, today this we have got even further terms like ULSI and all so, that is actually having larger and larger number of transistors in them. And now this is not a limitation; like, if we want to pack large number of transistors on a single chip that is possible because of the technology advancement. So, we can have transistor size is very small so, we can pack large number of transistors on a silicon floor.

So, that way size is not a problem, but the difficulty comes in terms of power consumption in terms of heat generation and all. So, those are of course, not part of this course discussion. So, they will go to some course on VLSI design.

(Refer Slide Time: 10:09)

The slide has a yellow header and a red footer. The title 'Microelectronics Proliferation' is in bold black font. The list below consists of six bullet points. A note at the bottom cites 'Gordon Moore's Plenary address at the 2003 International Solid State Circuits Conference'. The footer features the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' next to the NPTEL logo.

- The integrated circuit was invented in 1958.
- World transistor production has more than doubled every year for the past twenty years.
- Every year, more transistors are produced than in all previous years combined.
- Approximately 10^9 transistors were produced in a recent year.
- Roughly 50 transistors for every ant in the world .

*Source: Gordon Moore's Plenary address at the 2003 International Solid State Circuits Conference.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, integrated circuit it was invented in 1958, then this world transistor production has more than doubled every year for the past 20 years. So, every year so, total transistor productions so, it has doubled. So, you can imagine the amount of functionality that is becoming digital every year, digital or electronic going to electronic components every year.

So, more transistors are produced every year more transistors are produced which is more than all the previous year previous years combined. So, if you sum up are all the transistors that are produced till last year. So, this year it is going to be double of that sum. So, that is an exponential growth in terms of transistor production you can say.

So, approximately 10^9 transistors were produced can be, in some recent year. So, though it is not mentioned which year. So, it is around that. So, we can and roughly 50 transistors for every ant in the world. So, this is another statistical information. So, if you think about all the ants that are available in the world. So, each of them that multiplied by 50 so, that give the total number of transistors in the world, this is just a speculation of course.

So, this is obtained from 2003 source so, so, you all this figures. So, they have gone up significantly by the time today.

(Refer Slide Time: 11:40)

5 Commandments

- Moore's Law : The number of transistors on a chip doubles annually
- Rock's Law : The cost of semiconductor tools doubles every four years
- Machrone's Law: The PC you want to buy will always be \$5000
- Metcalfe's Law : A network's value grows proportionately to the number of its users squared

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, there are a few laws which actually predicted the growth of this electronic industry. So, first one is the Moore's law, it says that the number of transistors on a chip doubles annually. So, this is a prediction and more or less it has been followed, then there is a Rock's law it says that the cost of semiconductor tools doubles every 4 years.

So, this is another problem because as the complexity of the system is going up so, you cannot think about designing the system by hand. So, you cannot just take draw say 10 12 transistors, connect them in some fashion and get your circuit design. So, that is not possible so, we have to take help of these semiconductor tools; that may be based on, that maybe for this design specification, that maybe for simulation, that maybe for synthesis, that may be for testing.

But the costs are going up for every 4 years. Because the complexity of this systems are going up, complexity of this chips are going up. So, testing them the corresponding testing and implementing them the corresponding tools so, they are also becoming very complex, and the cost is going up significantly.

Then there is another law due to Machrone, which says that the pc you want to buy will always be dollar 5000. So, what does it mean it means two things, first of all it says that the cost will not go up significantly, because of the advancement in the semiconductor industry. So, the cost of production will go down so, that way it is going like this. So, you

can say the cost are producing a transistor is coming down. So, that way it is it will in some sense say.

So, it will balance the cost increase in the in the tools that we have, in the cost of the semiconductor tools that we have so, that that will get balanced. So, if we say on the other side that your cost of this manufacturing has gone down, but what has gone up is the cost of the software. So, software's are becoming more and more complex, as a result this the PC will always be a more or less the same price so, if you look at any point of time.

Then there is a Metcalfe's law which says that the network's value grows proportionately to the number of users of it is users squared so, that you know very well. Today in the in the era of this mobile networks so, you know so, if they are as the number of subscribers to the network increase we value that network further because that gives us connectivity both in our day today life and social life.

So, that way this is another law which says that if you if you are making the network complex, and you can accommodate more number of users. So, you are going to be paid back at a square rate. So, that is also very important thing.

(Refer Slide Time: 14:43)

5 Commandments(cont.)

- Wirth's Law : Software is slowing faster than hardware is accelerating
- Further Reading: "5 Commandments", IEEE Spectrum December 2003, pp. 31-35.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a Wirth's law it is say that software is slowing faster than hardware is accelerating. So, this is another important issue which says that the software is slowing. So, why because the computational power is increasing. So, as a software designer people

think that we can pushing lot of functionalities into it, and when we do that, the overall software speed that is coming down. So, so, essentially again there are research which will try to make that software faster.

So, hardware engineer they come up with some assisted tools by which you can make this hardware faster to run those operating system and software modules, but it is actually the software which is pooling back the developments in hardware, the speed advancement in hardware to a limit so that you can always you will always have the more or less the similar type of speed if you are if you are using if you are not using very advanced software.

So, this is a reference that talks about these commandments further.

(Refer Slide Time: 15:46)

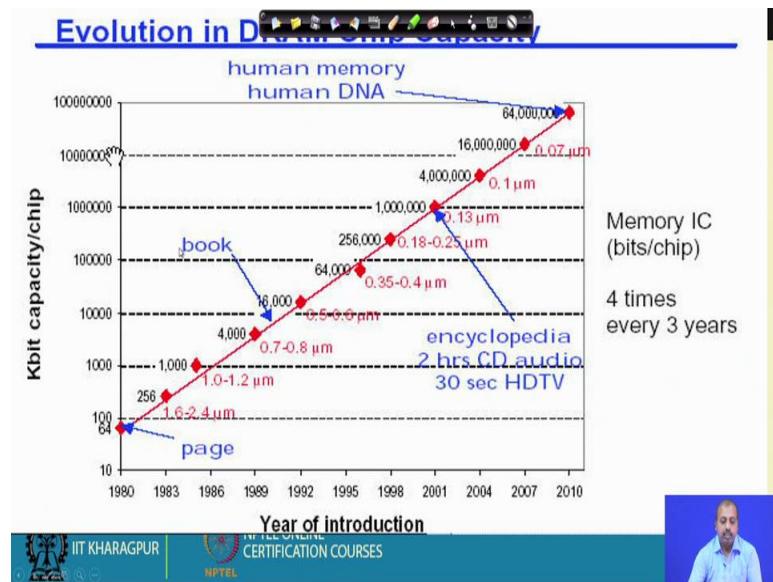
The slide has a yellow background and a blue header bar. The title 'Moore's law' is centered in a large, bold, black font. Below the title is a bulleted list of six items. At the bottom of the slide, there is a blue footer bar containing the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Moore predicted that the number of transistors that can be integrated on a die would grow exponentially with time.
- Amazingly visionary – million transistor/chip barrier was crossed in the 1980's.
- 16 M transistors (Ultra Sparc III)
- 140 M transistor (HP PA-8500)
- 1.7B transistor (Intel Montecito)

So, if you look into the Moore's law; so, Moore predicted that the number of transistors that can be integrated on a silicon, on a on a die so that would grow exponentially with time. So, it is says that it will double every year, every 2 years or so.

So, what was happened is a it is amazingly visionary law that Moore has predicted. So, you see that million transistor per chip barrier was crossed in the 1980s. So, and then I mean Ultra Sparc III, had 16 million transistors, then HP's PA 8500 140 million transistors. So, that way the number of transistors that we have so that is more or less following the Moore's law.

(Refer Slide Time: 16:29)

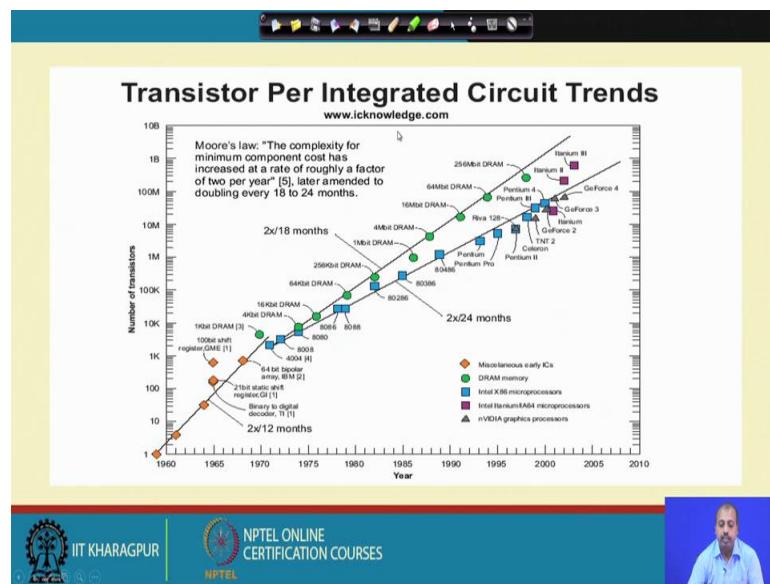


So, this is the evolution if you see that this is a. So, from 1980 to above 2010. So, we if we just look into this DRAM capacity. So, dram how this DRAM capacity is growing so, you see that this is it was about say 64 kilo bit per chip in the around 19 in the year around 1980 so, which was sufficient for holding say one page of information.

Then around 1989 1990 around that time we have got the capacity which us sufficient for a book. Then in the year 2001, it reached to support suppose as a it can hold the entire encyclopedia. So, 2 hours of CD audio or 30 second of HDTV, high definition television high definition television signals.

Then this then in 2010. So, the capacity has increased so much that even the total human memory or human DNA, we can store in that chip. So, memory IC is it is becoming 4 times every 3 years. The capacity of the memory chips are it is becoming 4 times every 3 years roughly. So, that way it is memory capacity is going to increase.

(Refer Slide Time: 17:51)

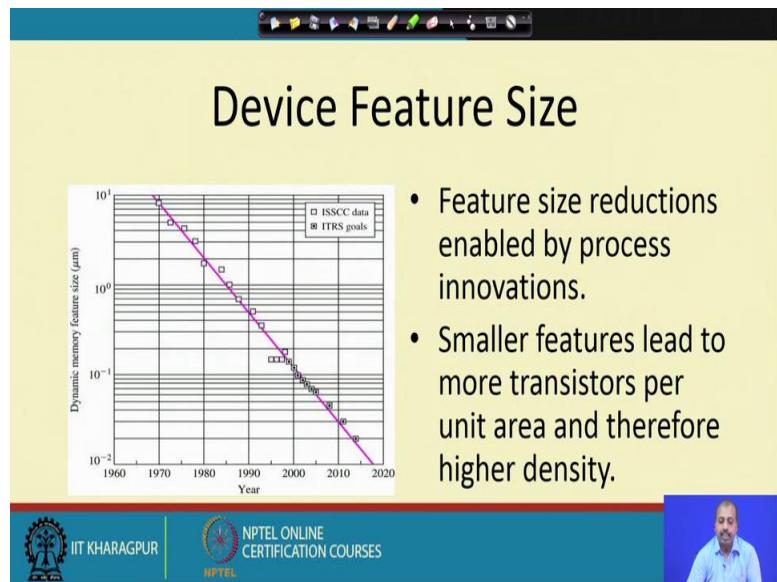


If you plot the number of transistors per integrated circuit so the trend is like this. So, it is started in 1960 so, if you say that it just started then it has gone up.

So, this Moore's law, which says that the complexity for minimum component cost has increased at a rate of roughly a factor of 2 per year. So, it was later amended to 18 to 24 months ok. So, this is a 2 year so, this was shortened further. So, if this was brought down to 18 months also.

So, this way where is the transistor per integrated circuit so, number transistor so, it has increased following Moore's law and that is going on, and as the technology is advancing further and further. So, whenever it comes to a saturation point you see that the technology advances it to the next state, next stage and this Moore's law still continue to hold in the next generation.

(Refer Slide Time: 18:54)

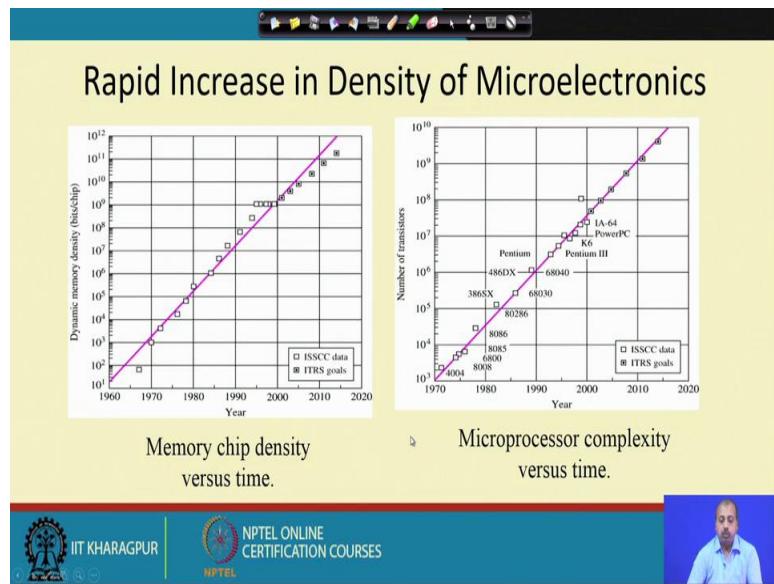


Feature size or device size so, feature size feature, when you say feature so, it is a size of a transistor you can say. So, that is the feature size in the VLSI terminology so, this feature size is reducing that will enable by process innovations. So, we in the VLSI manufacturing process so, it is advancing so, we can produce a smaller and smaller sized devices, transistors.

So, you can have say some something like say 16 nano meter designs now. So, that way it is a big advantage that we have so, the feature size is reducing so, overall size of the system is coming down so on the same silicon floor so, you can put in lot of transistors.

So, they put more transistors per unit area and therefore, we can have higher density. So, this is a plot till 2000 predicted up to 2020. So, after 2005 so, these are the goals that is said up to this much this was fine and then it is so, it is dynamic memory inter feature size in micron so that has come down significantly.

(Refer Slide Time: 20:06)



So, if you look in to this memory chip density versus time so, that is also density is also increasing significantly, as you say see over the year. So, that is following a more or less an exponential development and this microprocessor complexity. So, over this if you measure in terms of number of transistor so, that is also going up exponentially.

So, you see that from 1970's where we have we if you just start around 1975 or so, this 4004, Intel 4004 was introduced, then 1980 this 8008, was introduced so, like that it went. So, there are lots of developments that are taking place and they more or less following this Moore's law in the rate of increase of this device density and this computational power.

(Refer Slide Time: 21:01)

Analog versus Digital Electronics

- Most observables are analog
- But the most convenient way to represent and transmit information electronically is digital
- Analog/digital and digital/analog conversion is essential

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you look into these developments in the electronics industry so, you find development both in the analog domain and in digital domain. So, to on the analog domain it is mainly the operational amplifier, introduction of operational amplifier and circuit development around that, so, which is which has got the momentum on the digital domain of course, we have got different type of logic families that are being developed and we can realize some switching circuitry based on those logic on those logic elements, and accordingly we can have some digital design implemented in the IC.

So, if you just try compare between analog and digital electronics so, most observables are analog in nature. So, why do I say so, because nature is analog so, there is nothing digital in nature. So, if you look into the temperature. So, it is not the temperature is the digital quantity, if you look into the amount of light so; it is never a digital quantity. So, that way it is a most of the observables that we have. So, they are analog in nature so, weight of some person so, that is also an. So, that is also analog in the sense that. So, if you measure the weights of different people. So, they will be you cannot always say that after say 75 kg. So, next weight will definitely be 76 kg or 78 kg. So, in between you can get number of other values.

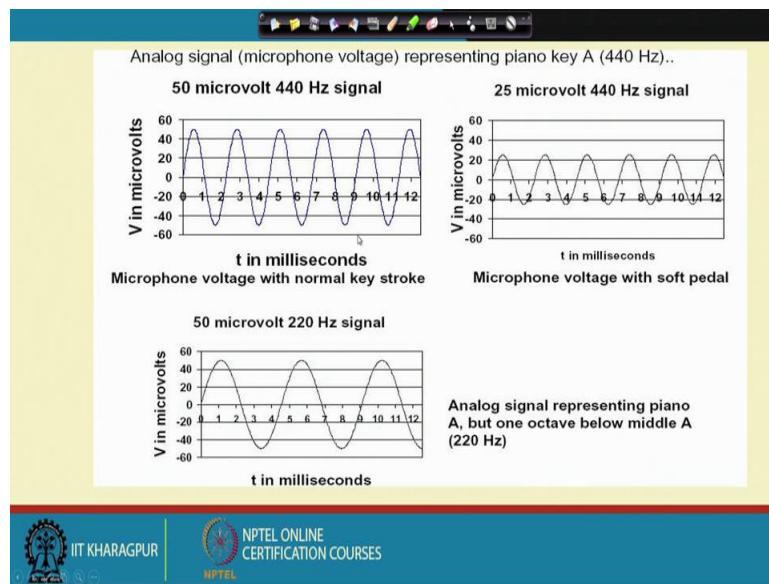
So, they are observe there will be a values are analog in nature, most of the; but most convenient way to represent and transmit information electronically is digital. So, we will see why so, once we have got it in digital format. So, we know that with the transmission

becomes easy from our say if this our devices the electronic devices that we are handling. So, we are nor- normally going for digital design so, why this digital? So, that will try to explain in the part of this course.

So, all the processing job that we do so, they are digital in nature whereas, the nature the environment is inherently analog. So, what is required is that if we are trying to process something in the analog in the something related to the environment. So, we should have this interface, which will convert the analog input signal into digital and again that after processing the actuation should be done in terms of digital values, which should be converted to analog value, for controlling the environment.

So, this way we have got analog to digital conversion on the input side of this digital processors and on the output side will we may have a digital to analog conversion. So, which will convert these digital values to analog actuation signals.

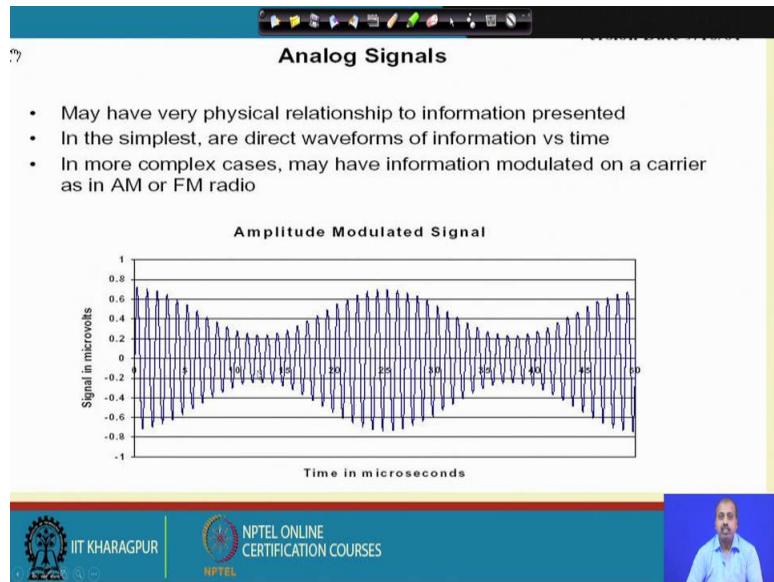
(Refer Slide Time: 24:00)



So, just to have a understanding so, this analog signal; so, suppose we are using this piano and thus this key A is place so, that is the 440 hertz. So, if you so, you can represent it in terms of a sinusoidal signals sinusoidal voltage signal which has got a maximum volt of 50 maximum peak of around 50 microvolt, so, this 440 hertz 50 microvolt sinusoidal signal may be like this. So, in this side if you plot the time in milliseconds and this side if you plot the volt voltage in microvolts, so the signal generated from the piano key A so, it may be like this.

So, on the same time, if you say that if you are using microphone voltage with soft pedal, then this is 20, it is a 25 microvolt signal, frequency may remain same. So, it is 25 microvolt peak to peak, and 25 microvolt peak and we have got 440 hertz signal. Or you can have this piano A, but one octave below the middle A so, that is about 220 hertz so, frequency is less. So, this is the peak value is 50 microvolt and this frequency is going to be 220 hertz. So, this way we can have the analog signals which are coming from the instrument.

(Refer Slide Time: 25:31)



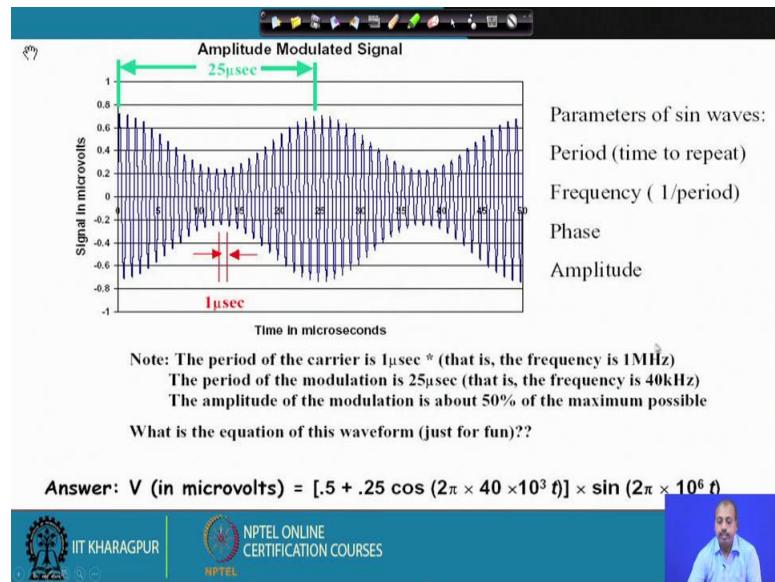
Now, this analog signals that we are talking about. So, they may have a very physical relationship to information presented. So, naturally this the direct signal that we have so they are information versus time. So, the so, this is basically the wave form the, that is generated is in terms of that is giving us some information in terms of the different time instance. So, in, but it is a normally what is done is that we never have this analog signals transmitted or analog signals processed directly. So, we convert it into some other form, which is known as the process of modulation.

So, we with the information that we want to send. So, that or we that we want to transmit. So, it is modulated on a carrier, such as in amplitude modulation or this frequency modulation the AM radio or FM radio that we have.

So, what is happening? So, this is an example of this amplitude modulated signal. So, here these high frequency signal that we have here so, that is basically the carrier signal. So, we

have got this high frequency, this blue colored one so, this is the high frequency carrier signal. And so, this is used for transmitting a relatively low frequency sinusoidal signal. So, if you look into this signal so, this high frequency signal so, this it frequency is much higher. Whereas, this low frequency signals so, it is time period is pretty high.

(Refer Slide Time: 27:20)



So, accordingly so, this if this high frequency carrier signal is multiplied by this low frequency sinusoidal signal. So, we will get some wave form like this. So, this is the amplitude modulated signal.

So, this the diagram it shows the parameters like see this is the; so, this parameters of the sin waves like period time to repeat then frequency. So, these are the parameters for any sin wave so, we can have so, in this case. So, this is the 25 micro second. So, this is the period of the modulating signal that we have. So, thus the information that we want to transmit so, that is the, that is that period is 25 microsecond whereas, the frequency sin wave. So, it is time period is one microsecond so, this is high frequency.

So, the period of the carrier is one microsecond that is the frequency is one megahertz, and the period of the modulation is 25 microseconds. So, the sinusoidal signal that we want to transmits so, that periodicity is 25 microsecond. So, it is frequency is 40 kilohertz, much lesser than this one megahertz signal.

Then so, this amplitude so what so, amplitude of this carrier signal is getting modulated by this. So, this amplitude of the modulation is about 50 percent of the maximum possible value. So, you can just do some mathematics and try to figure out, say like what is the corresponding equation for this ultimate output signal. So, this is going to be an expression like this ok.

So, this way we can have this amplitude modulated signal which will be which will be used for this analog communication purpose.

(Refer Slide Time: 28:55)

Digital signal representation

- By using binary numbers we can represent any quantity.
- For example a binary two (10) could represent a 2 volt signal.
- We generally have to agree on some sort of "code" and the **dynamic range** of the signal in order to know the form and the minimum number of bits.
- Possible digital representation for a pure sine wave of known frequency.
 - We must choose **maximum value** and "**resolution**" or "**error**," then we can encode the numbers.
 - Suppose we want 1V accuracy of amplitude with maximum amplitude of 50V, we could use a simple pure binary code with 6 bits of information.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

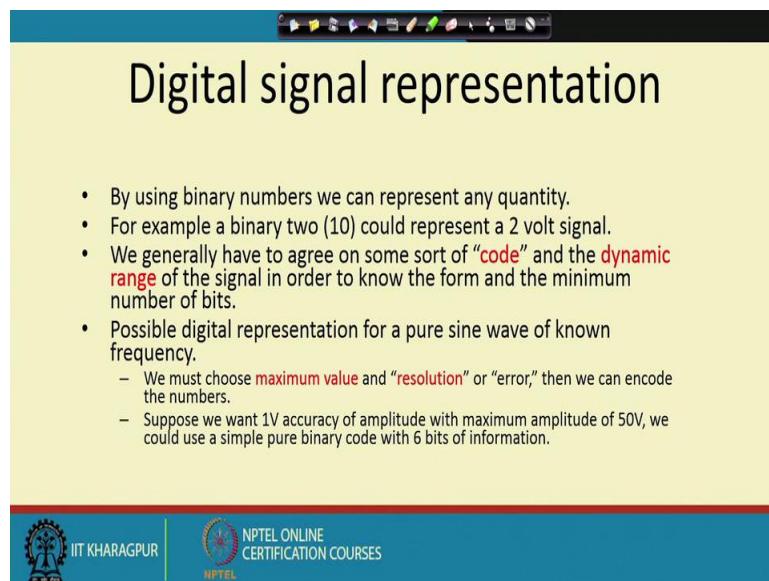
On the other hand, when we go for digital signal processing so, we normally use this binary representation. So, this will help us in making the system much more robust. So, we will see it in the next class.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 02
Introduction (Contd.)

So, this is a digital signal representation so, this gives us the flexibility, by which we can think about digital processing of this signals from the environment. So, this analog signals that we are getting so, if we want to process them digitally. So, we have to think about their representation, because anything that you want to do first of all what is required is to store the information that we have got from the outside world and for storing the information. So, you have to go for this digital form only.

(Refer Slide Time: 00:50)



The slide has a blue header bar with various icons. The main title 'Digital signal representation' is centered in a large, bold, black font. Below the title is a list of bullet points:

- By using binary numbers we can represent any quantity.
- For example a binary two (10) could represent a 2 volt signal.
- We generally have to agree on some sort of "code" and the **dynamic range** of the signal in order to know the form and the minimum number of bits.
- Possible digital representation for a pure sine wave of known frequency.
 - We must choose **maximum value** and "**resolution**" or "error," then we can encode the numbers.
 - Suppose we want 1V accuracy of amplitude with maximum amplitude of 50V, we could use a simple pure binary code with 6 bits of information.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' next to the NPTEL logo.

So, by using this binary numbers we can represent any quantity. So, we will see in our course that this by the different number systems are possible, and in a computer system, the number system that is used is the binary number system so, where this individual digits are only ones and 0s.

So, what we can do this we can represent 2 different state of some signal by means of these 2 different notations 0 and 1. So, may be if it is some voltage value we can say that this 0 is 0 volt, and 1 is 1 volt like that. Or if it is a say 0 is 0 volt, 1 is 5 volt, or it may be ,+ 12

volt and 1 is -12 volt. So, like that we can have different type of conventions that we follow.

But ultimately the information that we store so, that is in some digital format. So, for example, you can have this binary 2. So, 2 is represented by 2 bits 1 0 that could represent a 2 volt signal. So, if we say that one, binary one corresponds to a one volt signal. So, you can say that the binary value 2, it corresponds to a 2 volt signal.

So, we generally have to agree on some sort of code. So, what that is what I was talking about some time back. So, you can say that say -12 volt is 0 and +12 volt is 1, or somebody may say that the +12 volt is 0, and -12 volt is 1. So, of them are possible so, if you look into different systems, we will see that the all these types or different types of representations are possible.

So, we must agree on some sort of coding that we have and the dynamic range of the signal that we want to represent. So, see whenever we are talking about some representation of a number. So, we are dedicating certain number of bits in it, certain number of digits for that purpose so, using that many digits. So, you cannot represent any arbitrarily high number. So, if I say that I have a number of system, I have a space to write only say 3 digits. So, I cannot use it for representing the number 2556 in that system; because that requires 4 digits 4 digits space, but we have only a single digits 3 digits space so, I cannot have 4 digits there.

So, this way we can think about this range. So, this range dynamically what the, what the values that signal can pick up so, we must know that and accordingly we have to decide the minimum number of bits that will be using for storing that information. Possible digital representation of a pure sine wave of known frequency, for that purpose we must choose the maximum value. So, what is the maximum peak value that we have, and resolution or error that is at what level at what difference of values you are going to take, ok.

So, we will take suppose you want to one volt accuracy ok. So, suppose that is the, and the maximum amplitude is -50 volt. Maximum amplitude is 50 volt that is the signal goes from say -25 volt to +25 volt in the 2 ranges.

So, if we say that the -25 volt, I will represent as 0, then -24 volt, I will represent it as 1, minus 23 volt, I will represent it as 2. So, like that if we think about this type of levels,

then ultimately for this -25 volt to +25 volt, this 50 volt range I will have 50 different levels of the signal, that we can have so, if we think about one volt accuracy.

So, as a result so, this 50 different values have to be represented, and in a binary number system will see that it will require 6 bits to store that information. So, if you are given less number of bits. So, you will not be able to represent this 50 different levels. So, or so, you have to suffer on the accuracy side ok. So, suppose instead of say 50 different levels so, I want to store I want to have I am given only say 5 bits. So, with 5 bits so, I can have 32 different levels possible only. So, accordingly my accuracy will be less for the signal.

So, this way this digital signal representation itself it introduces some amount of error. So, if we are going, if we are looking for these exact values if we are looking for exact value when we have to go for the analog signals definitely, but as soon as we go for this digitization process some amount of error is introduced into the system so, that is there.

But we will see that there are different types of errors that can crop up into this analog systems, and that makes the digital systems much better from that point, but as far as the initial quantization is concerned. So, digital signals they have got some amount of error associated with it.

(Refer Slide Time: 06:15)

Digital representations of logical functions

- Digital signals also offer an effective way to execute logic. The formalism for performing logic with binary variables is called switching algebra or Boolean algebra.
- Digital electronics combines two important properties:
 - The ability to represent real functions by coding the information in digital form.
 - The ability to control a system by a process of manipulation and evaluation of digital variables using switching algebra.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, how can we represent digital logical functions in digital representation? So, digital signals they offer an effective way to execute logic. The formalism of performing logic with binary variable is called switching algebra or Boolean algebra.

So, when we talk about algebra so, the term algebra means that there will be some values that you can take, that individual variables can pick up so, that is there. And I can do some operations on that, on those values. So, there are certain axioms or fixed accepted norms for that particular system, and then on that we can build up the total set of computations.

So, in case of digital signals, so, we will see that, this Boolean algebra or switching algebra. So, that makes the basic, basic foundation stone for developing the system. And you may be knowing that this Boolean algebra. So, this was discovered long before these digital circuits came into existence, and that time it was pure mathematical interest, but once this digital systems got introduced, then this again this Boolean algebra got its importance back.

So, this digital electronics when we are talking about it combines 2 important properties. One is the ability to represent functions by coding the information in digital form. So, that is one, one property. And the other property is that it can control a system by a process of manipulation and evaluation of digital variables is in switching algebra. So, you can do some computation you can find out, you can have a set of conditions for which certain signals are to be turned on.

For example, if you are in a plant control so, there are many different conveyor belts moving. Then items are being put on to conveyor belt, then they apart from the there may be some fire safety system so, if you have smoke detector is sending signals. So, like that there may be a large number of signals the inputs that you can get from the plant, and accordingly you may have to do a set of other operations.

So, in this process they can be represented by means of some logical equation or logical functions, and those logical functions or those controls functions they can be implemented by means of this, they can be represented by means also switching algebra. And once you have represented them using switching algebra so, you can implement them using digital circuits.

(Refer Slide Time: 09:02)

Digital Representations of logic functions (cont.)

- Digital signals can be transmitted, received, amplified, and retransmitted with **no degradation**.
- Binary numbers are a natural method of expressing logic variables.
- Complex logic functions are easily expressed as binary function.
- With digital representation, we can achieve arbitrary levels of "dynamic range," that is, the ratio of the largest possible signal to the smallest than can be distinguished above the background noise.
- Digital information is easily and inexpensively stored

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, digital signals so, this is the next we going to why digital signals are preferable. So, digital signals can be transmitted, received, amplified and retransmitted with no degradation. So, why do I say so, because if you are transmitting some digital value. So, you know that the ultimately, what is transmitted is a bit 1 or 0. So, when you are transmitting a bit as 1 or 0 at the receiving station.

So, if I say that 1 in my system, 1 is represented by +12 volt and 0 is represented by -12 volt. So, at the receiving station, if you get some intermediary value; say, -5 volt. So, you can say that the -5 volt is close to -12. So, that is that can be taken as 0. So, that way we can say that even if I have got some values degraded, but digitally the value is the not degraded digitally the, we can still take it back to 0.

So, this binary numbers so, there they are natural method for expressing this logic variable. And complex logic functions can be easily expressed as binary functions. So, we will as I was telling that, in a plant there may be different inputs coming from different regions. And then we can have the overall plant function specified in terms of this control operations that it is doing, and that may be a set of logic functions, and they are represented as binary function and from there we can get the function implemented in digital circuits. So, we can achieve arbitrary levels or dynamic range that is the ratio of the largest possible signal to the smallest one can be distinct that can be distinguished above the background noise.

So, what happens is that so, whenever you are transmitting a signal there is always some amount of noise that we will get added into the systems. So, if you are transmitting some analog signal so, you can say that the analog signal say suppose I am transmitting a value of say 12 volt. And due to the this noise so, this value becomes degraded to say 7 volts; say, there is a noise introduced of -5 volt and at a some instant, and it becomes 7 volt.

So, at the receiving end I do not know whether value was a value is really 7 volt or it is 6 volt or it is 12 volt or something intermediary so, that way we cannot find out the thing. But if we say that in case of in case of digital one so, if I know that plus they one will be transmitted as 12 volt, and 0 will be transmitted at -12 volt. So, we can safely say anything above 0 is possibly 1 and anything below 0, 0 volt so, it is possibly a bit 0. So, this is the amount of that the noise margin the noise immunity that we have in the digital system. So, it is going to be much more compared to the analog system.

So, digital and the last point is the digital information it can be easily and inexpensively stored. So, we can store this values in the; so, with the advancement of this DRAM technique **tech-** technology and all, SRAM technology and all. So, we store large amount of information digitally in the system, and the quality is also much better than the analog storage.

(Refer Slide Time: 12:26)

Signal Types

- Analog signals take on continuous values - typically current or voltage.
- Digital signals appear at discrete levels. Usually we use binary signals which utilize only two levels.
- One level is referred to as logical 1 and logical 0 is assigned to the other level.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

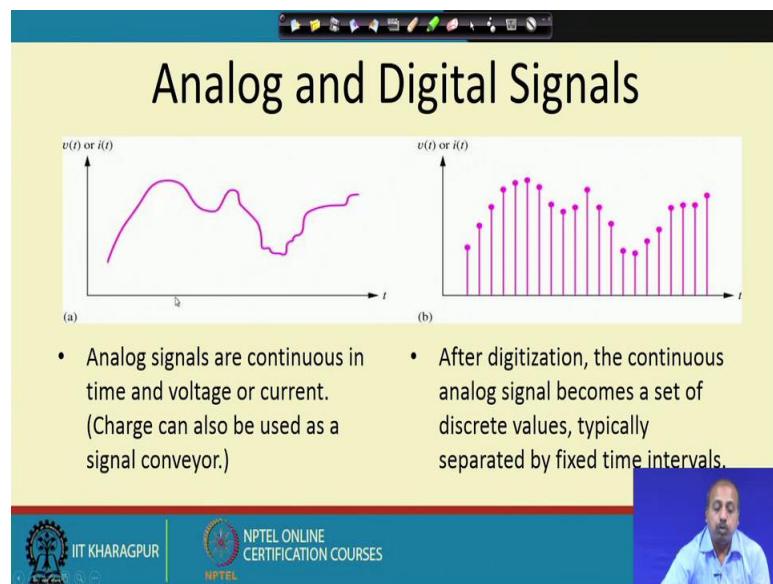
So, so, what you mean by this analog signal or digital signal? So, the first diagram that you see here so, this is an analog signal. So, for the time so, this it may be a voltage signal

or a current signal. So, it is varying like this so, this is the analog signal. So, analog signals take on continuous value so, there is you cannot say that at difference there is nothing like at different. So, this is normally specified into terms of some function over time or maybe a say differential equation and all, but ultimately this is a continuous function of time. So, typically a current or voltage may be representing an analog signal.

A digital signal so, they appear at some discrete levels. So, usually we use the binary signals that it only 2 levels like here you see that we can have a low level and a high level. So, this may be for this for this much of time the signal was low. Then for the this range of time the value is 1, then again for this range the value is 0 again it is 1. So, one level is refer to as logic 1, and the other level is refer to as logic 0.

So, of course, it is not mandatory that this high level should be one and low level should be 0. Somebody may say that my high level is 0 and the low level is 1 that is very much possible. But what is what is required is that there are 2 distinct levels. So, one of them is called logic 0, another one is called logic 1. So, we have got different this is a digital signals. So, they are going to be discrete in terms of level. So, it is not a continuous function like that.

(Refer Slide Time: 14:04)



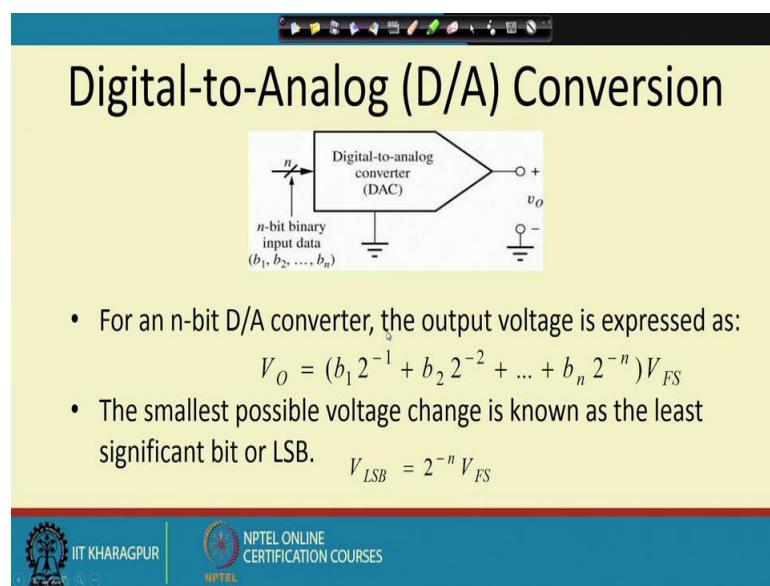
So, with that so, if we have got an analog signal. So now, I cannot have information's told for every time instance in my digital system. Like so, analog signals are continuous in time

and voltage or current. So, like say so, we can so, or charge so, that way we can have this analog signals represented over time, but when you do a digitization.

So, at every point we basically do a sampling. So, at some regular intervals of time so, we sample these digital signal, and then what we get is over the time intervals after separated by time intervals we get some signal samples. So, the after digitization this continuous analog signal it becomes a set of discrete values only. So, they are typically separated by this fixed time interval. So, which is known as the, which is this is a typically this is known as the sampling interval after this sampling intervals, the value is a sampled again and this values are stored.

So, ultimately for digital signal so, what we will need to do is that we need to store the values of the signals at this discrete points only. So, if you are willing to do a good approximation of the analog signal the, we have to do a sampling at a much higher rate, and if you are not so, then you can do where sampling at a lower rate also. And from the communication theory you can find out like what is the minimum rate at which we have to do the sampling and all for periodic signals etcetera for reconstruction purpose.

(Refer Slide Time: 15:41)



The important modules that we have in this process, one is the digital to analog converter so, or DAC. So, digital to analog converter will come when this after doing this digital processing. So, we are trying to give some signal to the environment. So, the the environment is analog so, we have to give some analog signal.

So, these n bit digital value, that is coming as input to this digital to analog converter module so, it produces a voltage level. So, you can say that if this in general. So, it is ah, if V_{FS} is the voltage the full scale voltage we say $V_{FS} 0$ to V_{FS} are the values. Then based on these values that we have got so, if all these b_1, b_2, b_n . So, if all these bits are 0s, then what you get is 0 at the output. So, if you are giving all 0 here you are getting a 0 output here.

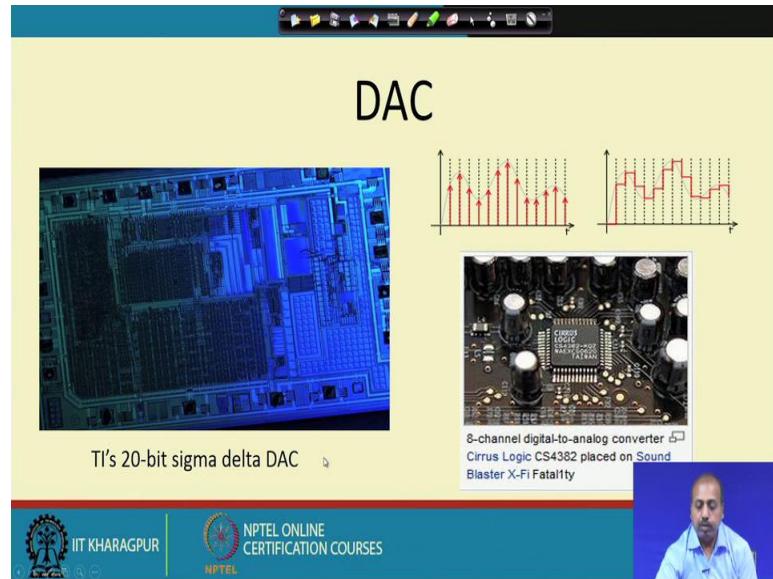
On the other hand if you are giving all these bits as 1, so, you will getting a very close, you will be getting the expression V_o will be very close to V_{FS} , ok. It is be just slightly less than this V_{FS} so, which is basically because of so, that that way you will get a value which is very close to V_{FS} so, you can say that. This is the voltage level that you are getting so, that you are getting the full scale range.

So, that way this digital to analog converters; so, they are useful for converting the digital value that we have got, after processing into some analog signal for, to be sent to the environment. So, what is the smallest possible voltage change? So, change the value can you can vary this bits, and since this quantity $b_n 2^{-n}$.

So, these has got the minimum contribution so, if you are going to change from the smallest possible change that you can do is by changing this flipping this particular bit b_n . And that way this minimum change that is possible is the $2^{-n}V_{FS}$. So, this we called the least significant bit or LSB.

So, that is the 2 this nth bit is the significant bit or the LSB. So, by changing this LSB, we can get this much of change so, this is the quantum this is the quantization level. So, you cannot represent voltage change which is finer than these $2^{-n}V_{FS}$. So, you cannot get better than that. So, so, that way this will have the error part.

(Refer Slide Time: 18:25)



Similarly we have so, this is another, this is the commercial some commercial DAC. So, this is TI's 20 bit sigma delta DAC, then this is an 8 channel DAC from Cirrus. So, like that so, essentially what is happening is; so, this light colored signal. So, this is the analog signal that we have so, and these are the samples.

So now, if you are if you are doing a digitization so, this is basically getting approximated like this. And then so, so, this is digital values that we have. So, that is converted into levels like this. So, this signal that we are getting is a close approximation of what we actually want. So, that way we say that these digital to analog converters are doing the conversion.

(Refer Slide Time: 19:11)

Analog-to-Digital (A/D) Conversion

- Analog input voltage v_x is converted to the nearest n-bit number.
- For a four bit converter, $0 \rightarrow v_x$ input yields a $0000 \rightarrow 1111$ digital output.
- Output is approximation of input due to the limited resolution of the n-bit output. Error is expressed as:

$$V_{\epsilon} = \left| v_x - (b_1 2^{-1} + b_2 2^{-2} + \dots + b_n 2^{-n}) V_{FS} \right|$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The other component the opposite if this is the analog to digital conversion. So, where on this analog side some value is coming, and then we want to convert it into some digital value. So, like a say one conveyor belt is moving, and you are just trying to see what is the speed at which it is moving.

So, the speed is coming as an analog quantity a some analog or some transducer will be there, and it will be coming as a analog signal. And that analog signal we want to convert digital values so, that we can do some processing we can either increase or decrease it like that. So, that way we can have the analog to digital converter on that side. So, this analog input voltage v_x so, it will be converted to nearest n bit number.

So, again the same thing that for a 4 bit converter so, 0 represent the v_x so, the so, if you 0 to v_x . So, that will input yield 0 0 0 0 to 1 1 1 1 digital output. So, when you give a 0 as input so, you will get 0 0 0 0. When you give the maximum value v_x as input so, you get 1 1 1 1 as the digital output. So, that is the design principle of this analog to digital converter.

So, we will see later in our course how to design this ADC's and DAC's in different ways. But however, ultimately this is the thing that we want so, if we give a 0 voltage. So, it should 0 0 0 0 if we give v_x the maximum possible voltage, then it should give all one.

So, output is again approximated, approximation of the input data, to the due to the limited resolution of this n bit output. So, error that we have so, this is the actually the v_x that we want to represent, but what we actually get is this one. So, these bits that are coming. So, if all these bits are one then you will be getting a value which is very close to V_{FS} , but this value is V_{FS} .

So, this V_{FS} minus that approximation that gives the amount of error that you have in the process. So, this way we can have the analog to digital conversion so, this DAC and ADC they are two very important modules that we have in a digital system when it is interfaced with the analog domain. So, we will see that due course of time.

(Refer Slide Time: 21:38)

Analog and Digital Signals

We seem to live in an analogue world – things can be louder or quieter, hotter or colder, longer or shorter, on a “sliding scale”.

If we record sound on a tape recorder, we’re putting an analogue signal onto the tape.

Digital signals aren’t on a sliding scale – they’re either ON or OFF. (We call these “1” and “0.”) There’s no “in between”.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you look into this analog and digital signal so, it seems that we are, we live in analog worlds. So, things like we make the statements like louder, quieter, hotter, colder longer shorter. So, this is the sliding scale so, may be if I say that make it louder. So, you just increase you will not turn the knob a bit, and may be the may be unsatisfied that is ok or, I say the no, no reduce it a bit. So, make it quieter so, like that so, that is an analog signal.

When you are recording some sound on a tape recorder. So, we are putting the analog signal on to the tape, and digital signals are not on a on a sliding scales so, they are either on or off like switches. So, the either the switch is on or the switch is off there is nothing like a light is turned half on. Of course, if there may be intensity control of the light so, that maybe if they are may be some on some analog some sliding scale may be there.

(Refer Slide Time: 22:39)

The slide has a yellow background. At the top center, it says "Analog and Digital Signals". Below that, a question is asked: "Are these analogue or digital?". To the left of the question, there is a list of five items: "Volume control on a radio", "Motor bike throttle", "Dimmer switch", "Water tap", and "Music on a tape". To the right of the question, there is a list of four items: "Traffic lights", "Light switch", "Music on a CD", and a small video thumbnail of a man speaking.

Analogue or digital?

- Volume control on a radio
- Motor bike throttle
- Dimmer switch
- Water tap
- Music on a tape

- Traffic lights
- Light switch
- Music on a CD

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if we just look into some of the signals and try to see whether it is analog digital. So, this volume control on a radios so, this is an analog signal traffic lights. So, they are either on or off so, they are digital. So, this green red and yellow, either they are on or they are off. Motor bike throttle, the amount of fuel injected into the system. So, that is going to be an analog signal, because with the place of the as we turn this accelerator. So, it puts in more and more fuel.

The dimmer switch; so, if we you can control the amount of illumination. So, that is also an analog signal whereas, a light switch with just turn on or off the light. So, that is going to be a digital signal. Water tap so, this is again an analog signal ok, we can turn the tap as much as you want and accordingly the amount of water passing will vary. Music on a CD is a digital signal. So, CD is the compact disc, they store information on a digital format. And music on a tape so, that is going to be an analog signal. Because they are using analog principle to store the information.

So, this way different, different signals so, you can think about them in digital or a analog, but most of the signals or the most common signals that we have so, they are all analog in nature. So, we have to pass them to an analog to digital convertor to get the corresponding digital versions. And if we are trying to produce an analog signal through some digital processing, we have to produce a, we have to have digital to analog convertor for converting the digital value to analog.

(Refer Slide Time: 24:24)

Analog and Digital Signals

A security floodlight switches on when you approach.



It has an **analogue input** (how much infra red it sees from you), and produces a **digital output** (the floodlight is either on or off). We could call it an “**analog to digital converter**”.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this is another typical example of analog and digital signals so, a security floodlight. So, it switches on when you approach so, it may be it has got an analog input. So, it senses the infrared the amount of infrared that it sees from you. And produces a digital output so; floodlight is either turned on or off. So, when it is doing that sensing of this infrared component.

So, that is an analog signal, it is an analog input, but when it is producing this floodlight, once it is turned on it is either turned on or turned off so, that is going to be a digital signal. So, we can call it an analog to digital converter in that sense, ok. So, so that is one way of the ADC that you can see.

(Refer Slide Time: 25:06)

Analog and Digital Signals

- The problem with analog signals is **noise** – hiss on the sound and speckly dots on the picture.
- When we send a signal over a long distance, the signal gets weaker, so we need to boost (amplify) it.
- The problem is that we end up boosting the noise as well.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

What is the problem with analog signal is the noise. So, the lot of noise gets introduced; like, if you are the for the sound, they are may be a that get introduced. Or if it is an image or picture these speckly dots may get introduced into the picture due to these errors, the, or the, that got introduced from the noise.

So, when we are sending a signal over a long distance the signal will get weaker because of losses, and we need to boost the signal; otherwise, you have transmitted as I was telling. So, you are transmitted 12 volt and over the length. So, it got reduced maybe so much that you get a very low voltage there. So, that way you need to amplify in between, ok.

So, normally what we do is that we put some periodic amplifiers on the path, and then the amplifier amplifies the signals, it boosts up the signal so that at receiving end you get reasonably correct signal.

So, the problem is that we end up boosting the noise as well the process. So, if we have got a boosting station so, a previous to that in the previous segment some noise got introduced. So, that noise will also get boost up in the process. So, this noise also gets amplified or noise also get boosted. So, that is the problem with the analog signal.

(Refer Slide Time: 26:29)

Analog and Digital Signals

If we convert the signal into digital form, then send it, it still gets weaker and noise still creeps in.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if we on the other hand, the digital signals if we convert the signal to the digital form and then send it, still gets weaker, and noise will still, noise still creeps in. Because that is the ultimately what we transmit is some analog voltage values. So, that will be, that will make the signal weaker and the noise will come. So, there is no doubt about it, but it is like this.

(Refer Slide Time: 26:55)

Analog and Digital Signals

Example: if you have a bad photocopy of a piece of text, and you photocopy that, you'll get a worse photocopy.

But if you **read the text yourself**, the “software” in your brain can “reconstruct” the text, because **you know what the letter shapes are supposed to be** even though they’re blurred.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

If you read the text yourself, the software in your brain can reconstruct the text, because you know that, because the letter shapes are supposed to be even though they are blurred.

So, like if you do a photocopy of a page with the page which is not very clear ok. So, if you do a photocopy so, after photocopy we will get an even worse page.

But if you do it like this if you read the text from the page. So, possibly you will be able to figure out those blurring of the different letters that we have on the page, and we can just correct it. So, this is just what this digital thing does. So, maybe the voltage value has reduced a bit, but it is not that much degraded and still possibly we can figure out that this was a 1 and this was a 0. We can reconstruct the signal and then again transmit. So, that away the amount of noise that got introduced so, that may get reduced further.

(Refer Slide Time: 27:56)

The slide has a blue header bar with standard presentation icons. The main title is "Analog and Digital Signals". Below it is a section titled "Summary". The text in the summary section is divided into two parts: one for analog signals and one for digital signals. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the NPTEL logo.

Analog signals suffer from noise, but don't need such complex equipment.

Digital signals need fast, clever electronics, but we can get rid of any noise.

IIT KHARAGPUR | **NPTEL** | **NPTEL ONLINE CERTIFICATION COURSES**

So, analog signals suffer from noise, but do not need such complex equipment. On the other hand, the digital signals need fast, clever electronics, but we can get rid of any such noise. So, this is the so, that tells the, summarizes is the analog and digital signals.

So, one of the very basic components when we are doing this the digital switching and digital circuits so, is, is a transistor and the most recent transistor that is being used in the digital industry today.

(Refer Slide Time: 28:22)

2.3

The CMOS Transistor

- CMOS transistor
 - Basic switch in modern ICs

The diagram illustrates the internal structure of a CMOS transistor. It shows a cross-section of a silicon substrate with a gate oxide layer on top. Below the gate oxide are the source and drain regions. A positive voltage is applied to the gate, which attracts electrons from the substrate into the channel between the source and drain. This creates a conductive path for current flow. The diagram also shows the IC package and the IC itself.

A positive voltage here...
...attracts electrons here, turning the channel between source and drain into a conductor.
nMOS gate
(a)
Silicon – not quite a conductor or insulator: Semiconductor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, they are CMOS transistors so, in CMOS transistor so, we will discuss in detail later just to tell you how it works.

This is a semiconductor block, so, so that so, normally. So, this is on that semiconductor block we have got 2 dedicated region one is called a source another is called drain. So, they have got some type of diffusion, they are some amount of some type of diffusion basically. So, if this semiconductor block that we have. So, if this is of say p type semiconductor then on this 2 regions so, we put some n type diffusions. So, one region is called source, other region is called drain.

So, the region in between so, it is covered by some silicon dioxide layer, and on top of that we put another layer of polysilicon gate. Now normally what happens is the if we apply a voltage on to this gate, positive voltage on to this gate so, it will it will attract the electrons from this from the substrate, we call it a substrate. So, the, it will call, it will attract this electrons into this into this channel between the source and drain, and now if there is a potential difference between source and drain.

So, if we apply a battery between the source and drain, then these electrons will move through this channel they will go from source to drain or drain to source depending on the polarity of the signals.

So, you get a current flow through the, through the device if we apply a positive voltage here. So, it will attract electrons turning the channel between source and drain into a conductor. So, this region will behave as a conductor. So, that is why it is called a semiconductor so, will be. So, will be, for turning off conduction so, you just make the gate 0. And so, this is track this channel will go. So, there is no conduction, and if you want this channel to be on so, you just apply a voltage here. So, this channel will become on.

So, pictorially it is represented like this. So, we have got source and drain at the 2 ends, and there is a gate control. So, if you apply logic high here, high voltage here, then the channel will come into existence as a result you can see that the as if this transistor is now open. So, transistor is now on so, you can get a conduction from this side of the transistor to this side of the transistor through the channel.

The other hand if you put a logic 0 here a low voltage here or you withdraw the voltage that you are apply. So, this channel will low will not exist as a result, this transistor will behave as if the transistor is open. So, this is, this is, this will behave like a open switch.

So, in one case it is working as a closed switch, when this, when this gate is high so, this will act as a closed switch. So, you will get connection from here to here, and another case you will be getting an getting an open situation where there is no connection from drain to source. So, this way this transistor will behave in the form of a switch.

So, we will come back to these transistors again later, but what I essentially means is that these are the, these small transistor. So, they are the tool by which we represent this switches in the digital circuits. So, they will be used again and again for when in the circuit design.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 03
Number System

So, we will be looking into the number system. So, Number System that tells us how can we store or represent numbers in a digital system. So, this is a very important part because this will tell us how can we do processing over those numbers and all, and what are the limitations and what are the advantages that we can get of different number systems, that we can think about. Or what are, what are in general the number systems that are available in the computer, or say for the sake of simple mathematical interest also, you can look into this topic which will be telling about different ways in which you can represent numbers.

(Refer Slide Time: 01:01)

System	Base	Symbols	Used by humans?	Used in computers?
Decimal	10	0, 1, ... 9	Yes	No
Binary	2	0, 1	No	Yes
Octal	8	0, 1, ... 7	No	No
Hexa-decimal	16	0, 1, ... 9 A, B, C, ... F	No	No

The slide includes a logo for IIT Kharagpur and NPTEL Online Certification Courses. A video player window in the bottom right corner shows a man speaking.

So, any number system that we look into so, it will have 2 important components in it. One is called the base of the number system, one is called the symbols of the number system so, where we talk about some quantity that we represent like how many digits, can be the how many symbols can be there different types of symbols and the other part is the symbols.

So, any number system that we talk about. So, these are the 2 things that we have to tell. So, to tell me a base so, as soon as you tell me a base we can assume that there will be so

many different types of symbols that you can have. Now, a number system maybe a consisting of different number of digits. Like I can have say a decimal number system where base is 10. So, as soon as we know that the base is 10, we know that there are 10 symbols, and the, those symbols are conveniently represented as the say 0, 1, 2 up to 9 ok.

So, if you are talking about a binary number system, the base is 2, and in that case the symbols only 2 symbols are needed so, we represent them as 0 and 1. So, so, we that there is of course, some value associated with this. So, the convention that we follow is that in a decimal number system. So, 0 we represent the integer value 0, 1 represents the integer value 1 and similarly the 9 represent the integer value 9. In a binary number system the base is 2, and the symbols that we have are 0 and 1. So, here also we have got the assumption that 0 represents the integer 0 integer value 0, and 1 represents the integer value 1.

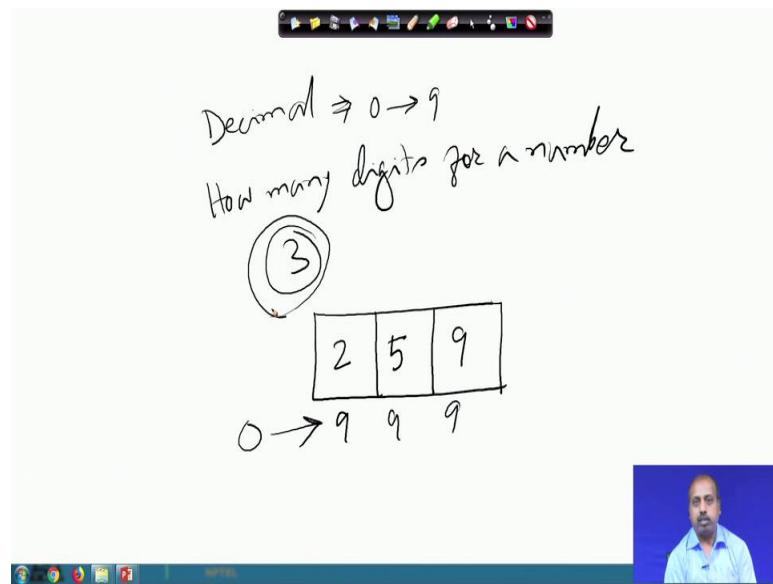
In an octal number system base is 8, and naturally there will be 8 symbols and we have got the symbols 0 to 7. Hexadecimal number system the base is 16, now here is the problem. Because so far whatever we had so, we have sufficient number of symbols to represent individual values of the symbols, now we do not have that. So now, you see that up to 0 to 9 we take. So, these are these are the 10 numbers that we have after that so, we do not know any symbols. So, what is done is that this ABCDEF so, they have been taken as symbols.

And here actually comes the thing that I was talking about the value associated with the symbols. So, this A it is associated with the value 10, the integer value 10. Similarly this B so, this is associated with the integer value 11. So, C associated with integer value 12, that way F associated with integer value 15. So, you can you can say that instead of this ABCD, I will be using some other notation $\alpha, \beta, \gamma, \delta$ like that.

So, there is nothing wrong in using some different notations for these symbols, but what is required is that you have to tell what is the corresponding integer value, for the symbols otherwise we will not be able to represent the some quantity in that number system ok.

The next thing that we need to know or need to understand is like, whenever you are representing some numbers so, you have to tell how many digits are you allowing for the numbers. Like if I say that I have a, if I say that I have got decimal number system ok.

(Refer Slide Time: 04:29)



So, as soon as I say decimal. So, I know that the digits that I can have are 0 to 9, the next question is how many, how many digits I can use, how many digits for a number. So, if you are trying to say that so, if I say that I will be using 3 digits. So, you can say as if I have box like this and in this box so, there are 3 places ok. So, in the first place you can put a digit say 2.

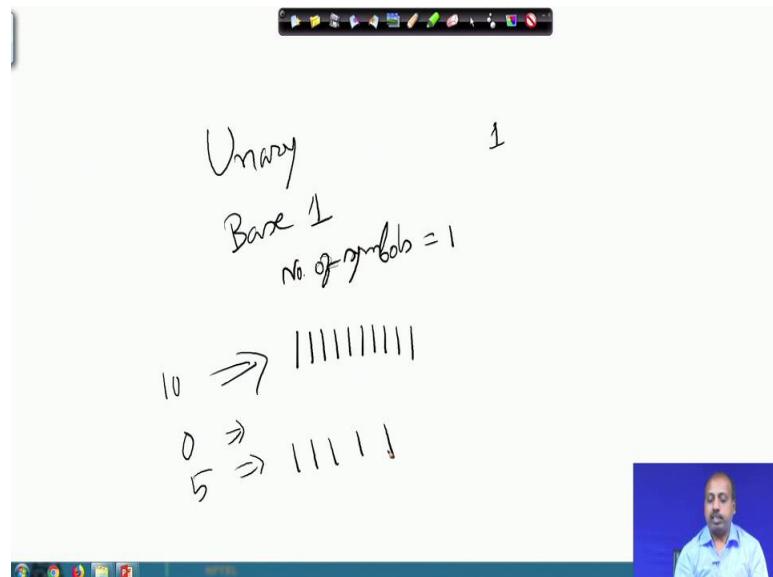
So, here you can you can put a symbol 2, here you can put a symbol 5, here you can put a symbol 9. So, as soon as I limit this value to 3 so, you can understand the maximum value that I can have in this system is 999, not more than that. So, I can this system can represent numbers from 0 to 999. So, this number of um digits that we are using for representation that is vital. So, whenever we are talking about. So, these computers or the processors we see that it is a 16 bit processor or 32 bit processor. So, essentially what it means is the information that it can handle the size of the data that it can handle as the basic processing limit. So, that is, that is going to limit the number of digits here.

So, going back to the point that we are discussing, we have got this number systems the, their base values and their symbols are like this. Then we can say answer like whether they are used by human or not. So, of course, this is a bit tricky like somebody may be very comfortable with hexadecimal number system, and may be always writing in terms, in terms in terms of the hexadecimal digits. So, that is very much possible, but normally it is not so. So, from our school days we are familiar with decimal number systems so, we will

be we are we say that it is used by humans whereas, the others binary, octal and hexadecimal are not, ok.

Then we convey so, so they are used in computers. So, decimal number system is not used in computers, and then we can have this binary number systems used, octal and hexadecimal number system. Now you see that while talking about this one thing I have I have purposefully avoided; where can we have a number system with base 1, so, that we have not talked about. So, answer is definitely yes we can have a base one number system as well.

(Refer Slide Time: 07:14)



So, so, here if I say it is a base one number system so, that is then that is a unary number system. So, since base is 1, base is 1. So, I can have only a single symbol so, number of symbols, number of symbols that you can have is equal to 1. So, if I decide that, but the symbol that we will be using is also 1. Then if I ask you to represent 10 the quantity 10 in this number system, then what we will do? You will write 10 such ones, fine? Because I do not have any other symbols. So, as many value I want so, I needed a quantity 10 to be represented. So, I put 10 such 1's, I need to, if I need to represent the quantity 0. So, I do not put any of these things so, none of the ones are there.

So, if I want to represent 5, then 5 such ones are there. So, this is unary number system so, it can be used, but this is not used, because of it does not, does not make it amenable to the processing that we have possible with the other number system the other bases basically.

So, that is the reason why this is not done, but it is very much possible the unary number system ok. So, in our discussion we will be concentrating mostly on these number systems and so, this decimal number system.

(Refer Slide Time: 08:44)

The slide has a yellow header bar with the title 'Quantities/Counting (1 of 3)' and a black navigation bar at the top. The main content is a table with four columns: Decimal, Binary, Octal, and Hexadecimal. The table is as follows:

Decimal	Binary	Octal	Hexa-decimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

At the bottom left, there are logos for IIT Kharagpur and NPTEL. The IIT Kharagpur logo features a temple gopuram. The NPTEL logo features a circular emblem with text. On the right side of the slide, there is a small video window showing a man speaking.

So, if we say the quantity 0, 1, 2, 3, 7. So, the 0 in binary will be represented as 0 in octal also 0 hexadecimal also 0. 1 will also be represented same, but 2 onwards there will be problem, because 2 cannot be represented by the symbols of binary number system. So, I need to use 2 symbols, 1 0 to represent 2, similarly once 2 is done. So, 2 3, they can be represented by binary number systems. And then octal will be there octal and hexadecimal they do not find problem till 7, so, we have got enough symbols to represent the numbers.

(Refer Slide Time: 09:25)

Decimal	Binary	Octal	Hexa-decimal
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

But after that of course, problems starts coming like octal faces problem for 8 onwards. So, 8 has to be represented by 2 symbols 2 digit like 1 0, 9 by 1, 1 like that. So, that way till 15 hexadeciml does not face any problem.

(Refer Slide Time: 09:47)

Decimal	Binary	Octal	Hexa-decimal
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

But if you are going to 16 then of course, hexadeciml will have to have 2 digits now ok. So, this will be 16 so, that way it will grow. So, this way this numbers are or the quantities they will be represented in different number systems.

(Refer Slide Time: 09:58)

Conversion Among Bases

- The possibilities:

```
graph TD; Decimal((Decimal)) <-->|bidirectional| Binary((Binary)); Decimal((Decimal)) -->|bidirectional| Octal((Octal)); Binary((Binary)) -->|bidirectional| Hexadecimal((Hexadecimal)); Octal((Octal)) -.->|crossed out| Decimal((Decimal))
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this talks about the conversion between the bases ok. So, these are the possible conversion like all these number systems we can convert from one form to another form. So, decimal to binary, binary to decimal, decimal to octal, octal to decimal. So, like that we can convert between this number systems very easily, and the so, ultimately in a computer system so, it understands the binary number systems.

So, they are of course, it does not have any bearing other number system it does not have any bearing so, truly speaking, if we are learning about this digital circuit class. So, digital circuit course we do not need to learn about the other number systems ok. So, that is just for our understanding of the binary number systems.

So, truly speaking so, if we know the binary number system well, we are done we do not need to know others, but because we are familiar with the other number systems. So, we will be looking into the conversion to make us feel comfortable with the number system.

(Refer Slide Time: 10:59)

Quick Example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

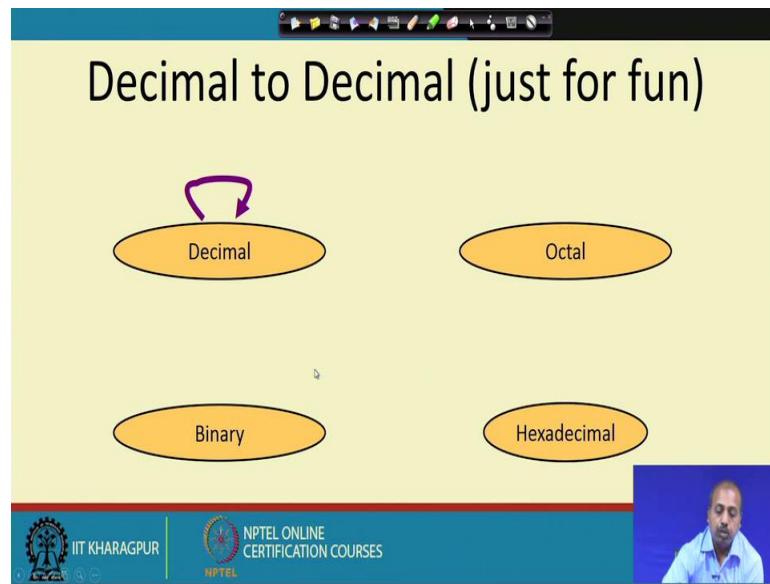
Base

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, 25, so 25, so, normally while writing the numbers so, we do not write this base part, when the because in our day to day life or from our school days with the numbers that we know they are all base 10 number system.

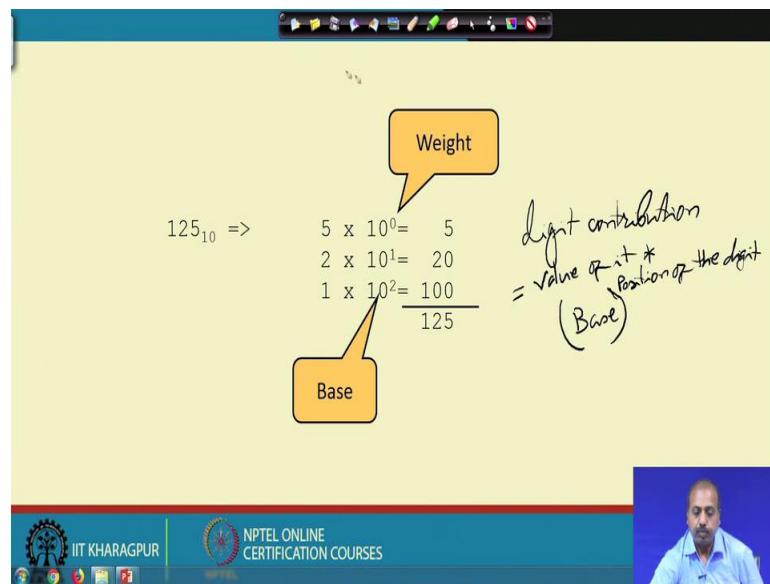
So, we will be, but if you if you want to make it explicit, then we do it like this after the number at the suffix we write the base. So, this $(25)_{10}$ is equivalent to $(11001)_2$. So, in a binary number system so, it will be 1 1 0 0 1 it is equivalent to $(31)_8$ octal number system and it is again equivalent to $(19)_{16}$ that is hexadecimal 19 so, all these numbers are equivalent.

(Refer Slide Time: 11:46)



So, we will be looking into some conversions so, between these number systems. So, just to understand this conversion process so, we will be looking into the conversion of decimal to decimal, the decimal numbers how is it converted to decimal.

(Refer Slide Time: 12:03)



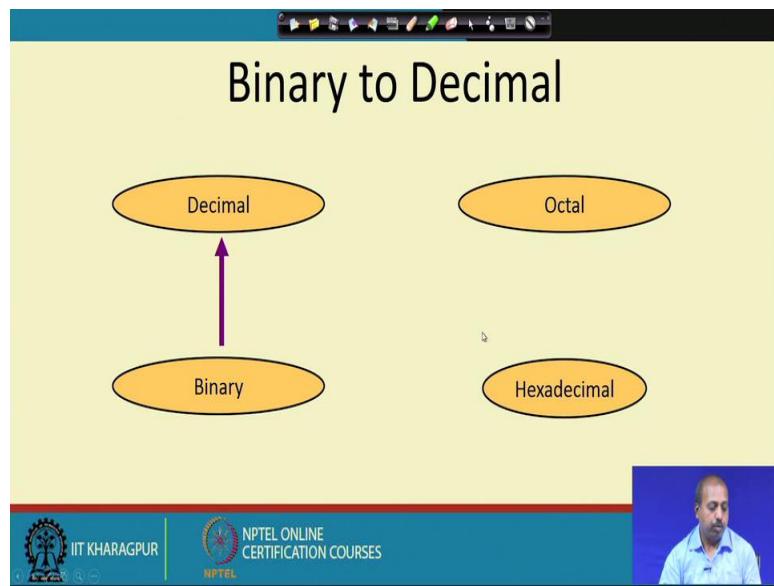
So, this will give us the understanding so, you see that we are not going to so, decimal to decimal conversion as such is meaningless. But this will this is just give us a recapitulation of our school day's knowledge, about how these numbers are actually representing some quantity.

So, in this in a number like 125, this, that there are 3 digits 1 2 and 5. But all of them are not representing values of equal strength ok. So, the number 5 so, when you are looking into the number 5, say actually it is representing the number 5×10^0 , that is equal to 5. So, this 2 here so, this is actually representing 2×10^1 , that is 20, and this 1 here is representing 1×10^2 , that is 100.

So, the symbol that we have so, the value of it multiplied by the some weighted value of the base. So, that is going to be the contribution of a particular digit. So, a digit is; so, whenever we are talking about a digit so, digit contribution, digit contribution is equal to the value of the digit, value of it multiplied by base taken to the power of the position of the digit. So, this is the overall formula that we are getting.

So, this is digit contribution is value of it multiplied by base to the power of position of the digit. So, this position is 0 so, it is contribution $base^0$, this 2 is a position 1. So, this is 10^1 , and this one is a position 2. So, it is 10^2 . So, this is a weight varies with the position, ok. So, so, next we will look into so, so, this will tell us like how do how do we do the conversion.

(Refer Slide Time: 14:25)



Now, when we are converting say binary to decimal.

(Refer Slide Time: 14:33)

Binary to Decimal

- Technique
 - Multiply each bit by 2^n , where n is the “weight” of the bit
 - The weight is the position of the bit, starting from 0 on the right
 - Add the results

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, the technique that we have to follow is multiply each bit by 2^n , where n is the weight of the bit. As I said that any number, when we are trying to get the value of a contribution of a bit or a digit it is the value of the digit multiplied by weight of the digit. So, the weight of the digit is coming from the $base^n$; where n is the position of the digit

Now, in case of binary number system so, we have got only 2 digits possible 2 symbols possible 0 and 1, and anything multiplied by 0 gives 0 only. So, we can say that why it is a so, we can just write it multiply each bit by 2^n . So, these weights is the position of the bit starting from 0 on the right and then add the results.

(Refer Slide Time: 15:21)

Bit "0"

101011₂ =>

1 x 2 ⁰ = 1	
1 x 2 ¹ = 2	
0 x 2 ² = 0	
1 x 2 ³ = 8	
0 x 2 ⁴ = 0	
1 x 2 ⁵ = 32	

$\underline{ }$

43_{10}

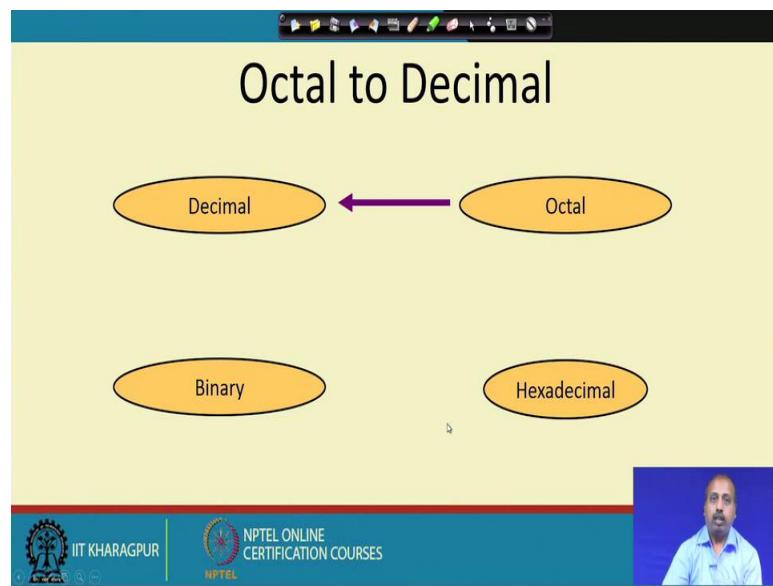
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, here is an example so, this is the, this is your binary number, say a 6 bit binary number so, this is the bit 0, ok.

So, this 1×2^0 , so that the position 0, then 1×2^1 then 0×2^2 . So, that way it is growing so, if you just, I will get the corresponding contribution. So, this is one this is 2 this is multiplied by 0 so, the contribution of the bit 2 is 0. Then contribution of bit 4 is also 0, others are contributed to some value. So, that way it gives 43 in the decimal number system, because this values that we are taking so, they are the decimal values ok.

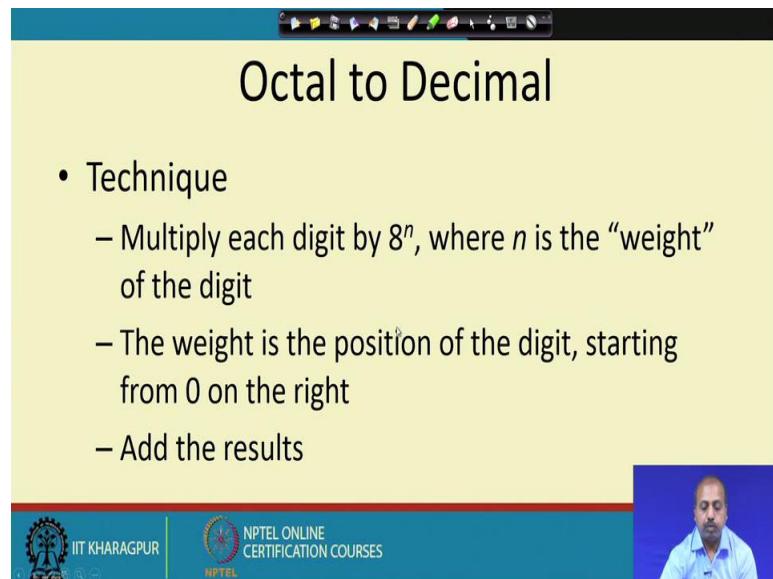
So, since they are the decimal values so, that is ultimately giving me a decimal number 43.

(Refer Slide Time: 16:08)



Now, in a similar line so, we can convert octal number to decimal number. So, multiply each digit by 8 to the power n. So, in case of octal number system so, base is 8.

(Refer Slide Time: 16:14)



So, we multiply each digit by 8 to the power n; where n is the weight of the digit, and the again the same thing that weight is the position of the digit starting from 0 on the right, and then we add all the results.

(Refer Slide Time: 16:33)

The slide has a yellow background with a black header bar containing standard window controls. The title "Example" is centered in a large, bold, black font. Below the title, the octal number 724_8 is shown followed by \Rightarrow . To its right is a vertical column of calculations:

$$\begin{array}{rcl} 4 \times 8^0 & = & 4 \\ 2 \times 8^1 & = & 16 \\ 7 \times 8^2 & = & 448 \\ \hline & & 468_{10} \end{array}$$

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video camera icon indicating a live video feed.

So, this is $(724)_8$ so, 4×8^0 , that is a contribution is 4, 2×8^1 contribution is 16, 7×8^2 the contribution is 448. So, you sum them up so, it becomes 468.

(Refer Slide Time: 16:58)

The slide has a yellow background with a black header bar containing standard window controls. The title "Hexadecimal to Decimal" is centered in a large, bold, black font. Below the title, four orange ovals represent different number systems: "Decimal", "Octal", "Binary", and "Hexadecimal". A purple arrow points from the "Octal" oval down towards the "Binary" oval, indicating a conversion path. At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video camera icon indicating a live video feed.

So, that way we can convert this um, we can convert an octal number into a decimal number. Hexadecimal to decimal again the similar way we can do multiply each digit by 16^n . So, base is 16 so, we take it to the power n; where n is the weight of the digit, and then we the weight. So, that the then we just add the results.

(Refer Slide Time: 17:17)

The slide has a yellow background with a black header bar containing standard window controls. The title 'Example' is centered in a large, bold, black font. Below the title, there is a mathematical calculation:

$$\begin{array}{l} \text{ABC}_{16} \Rightarrow \\ \text{C} \times 16^0 = 12 \times 1 = 12 \\ \text{B} \times 16^1 = 11 \times 16 = 176 \\ \text{A} \times 16^2 = 10 \times 256 = 2560 \\ \hline 2748_{10} \end{array}$$

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video camera icon.

So, in a similar way say ABC to the base 16 so, $C \times 16^0$, and the value of a C is 12 ok. So, the as I said that each symbol it has got some corresponding integer value, in the number system; so, and for hexadecimal number system so, A is 10, B is the integer value is 11 and C is integer value is 12. So, that way so, this values are multiplied by the powers are 16 so, that way we get the value 2748. So, this is the corresponding decimal value for this ABC to the base 16 in hexadecimal system, ok.

(Refer Slide Time: 17:55)

The slide has a yellow background with a black header bar containing standard window controls. The title 'Decimal to Binary' is centered in a large, bold, black font. Below the title, there is a flowchart diagram:

```
graph TD; A([Decimal]) --> D([Binary]); B([Octal]); C([Hexadecimal]);
```

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video camera icon.

So, that way if you want to go to the decimal so, you can do it in this fashion. So, you can we can multiply by the, we can multiply by the weight of the base, and then we can go to the decimal system.

(Refer Slide Time: 18:18)

Decimal to Binary

- Technique
 - Divide by two, keep track of the remainder
 - First remainder is bit 0 (LSB, least-significant bit)
 - Second remainder is bit 1
 - Etc.

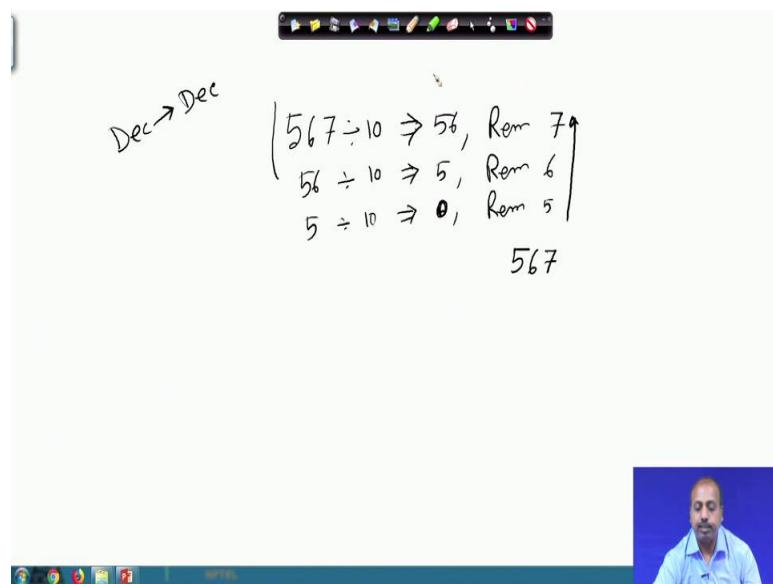
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player window shows a man speaking, likely the instructor.

But the other side the decimal to binary, octal and hexadecimal conversion so, that for that we need to divide. For decimal to binary, if you want to convert so, you can you can convert into we can divide by 2, and we can keep track of the remainder. So, the first remainder is bit 0 second remainder is bit one etcetera. So, for the sake of understanding, if we think about a decimal to decimal conversion, how do we do this thing? Ok so, we have seen by multiplication technique, but you can, we can if you thing about this decimal to decimal conversion, say the number say 567, ok.

(Refer Slide Time: 18:57)



So, 567 so, I am looking for decimal to decimal conversion. So, what it says is that you divide by the base of the number system. So, if you divide this $567/10$ so, what you will get the value 56, as the quotient and the remainder is 7. Then you divide again $56/10$ so, what you get is 5 with remainder as 6, and what is $5/10$? So, you get you get this 0, and the remainder is 5.

Now, what you do for getting the converted form. So, you just write down this remainders from this side first remainder of the latest remainder is 5, previous to that 6 and previous to that 7. So, this is the way we are converting the numbers, so, same thing applies to any other number system. So, this is with respect to decimal so, you can do it with respect to any other number system. So, we will see that while we are converting from decimal to binary. So, in case in the since in binary system, the base is 2 so, we will go on dividing by 2, and we will track the reminder value. Say, what is 125 which decimal to binary, ok.

(Refer Slide Time: 20:28)

125₁₀ = ?₂

2 | 125
2 | 62 1
2 | 31 0
2 | 15 1
2 | 7 1
2 | 3 1
2 | 1 1
0 1

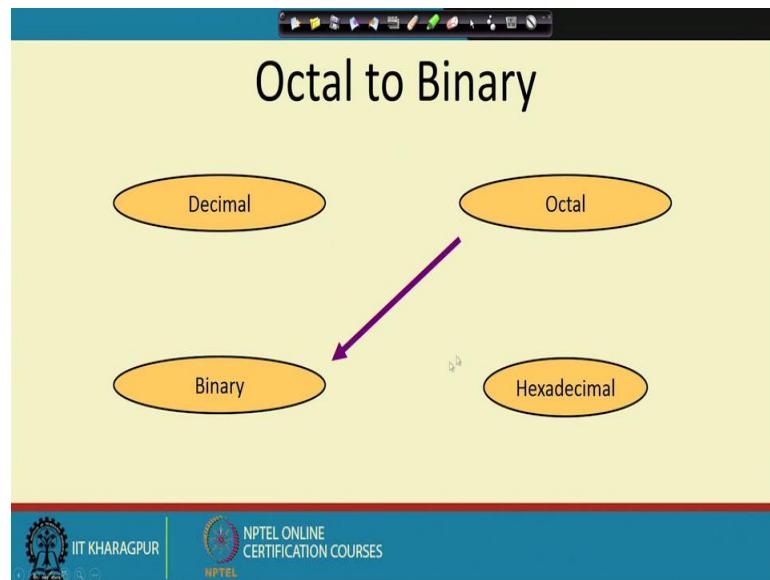
125₁₀ = 1111101

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you divide 125/2 so, 62 is the quotient and 1 is the remainder. So, we divide again 62/2 so, you get 31 as the quotient and 0 as remainder, divide 31 by 2 you so, you get 15 as quotient and 1 as remainder. Divide 15 /2 you get 7 as quotient 1 as remainder divided by 2, 3 as quotient, 1 as remainder, then again divide by 2, you get one as quotient and one as remainder. Again divided by 2 the quotient becomes 0 and remainder is 1.

So, at this point we stop, the similarly what we did in the in the decimal number system. So, we do the same thing here and so, we get this so now, we start writing from this side. So, the 1 1 1 1 1 0 1 so, this is the thing so, this is the conversion. So, this way we can convert a decimal number into binary number.

(Refer Slide Time: 21:35)



So, if we are looking for other way the octal to binary number system so, conversion so, one thing that you can do is octal. So, you can you can come this way so, you can convert this octal to decimal, and then from decimal we can come to binary.

(Refer Slide Time: 21:39)

- Technique
 - Convert each octal digit to a 3-bit equivalent binary representation

So, this is one avenue other straightforward avenue for octal to binary conversion is you can do it some sort of grouping of bits, because in octal system the base is 8, and in binary system the base is 2. So, you can group 3 bits of binary numbers, sorry, you can convert

each octal digit by a 3 bit binary number. And because of this the divisibility of this 2, base is of this base 2 of binary and base 8 of octal.

So, this octal digit boundaries are similar to this binary number boundary. So, that that is why so, this each octal digit will form a 3 bit boundary for the binary number system so, you can convert octal to binary. So, you can directly convert individual digits of this octal number system into binary number and get the result.

(Refer Slide Time: 22:57)

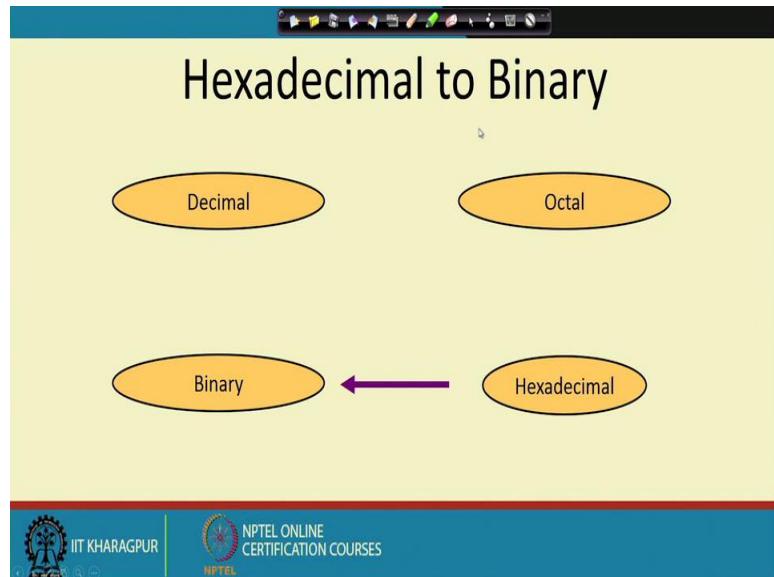
So, what is $(705)_2$ so, 7 is 1 1 1, 0 is 0 0 0 and 5 is 0 0 1 so, you can get it like this. So, or other way you can say that ok, I can convert it in the so, 7 0 5. So, if you convert this 7 0 5 to decimal so, you will get $7 \times 8^2 + 0 \times 8^1 + 5$, it is 64×7 , ok.

So, so, it is 64×7 yeah, ok. So, this is 448, plus we have got this plus 5. So, that is 453. Now this 453 so, so this 453, as I was telling. So, you can convert it into binary. So, this is 22, 13 that is 6. So, 1 then 2, 1 1 3 0 then 2 so, this is 56, 1, then 2, 28, 0 then 2 ,14 0, then 2 divided by 2, 7, 0, remainder is 0. So, this is 3 1, this is 2, 1, 1, this is 2, ok.

So, then we have to write in this order for the conversion into a binary so, this is a 1 1 1 0 0 0 1 0 1. So, either you can do it like this so, we can take this individual digits, and then convert them directly into 3 bit binaries. So, we can get it directly from octal to binary, or you can just you can convert to decimal number system, and then from the decimal number

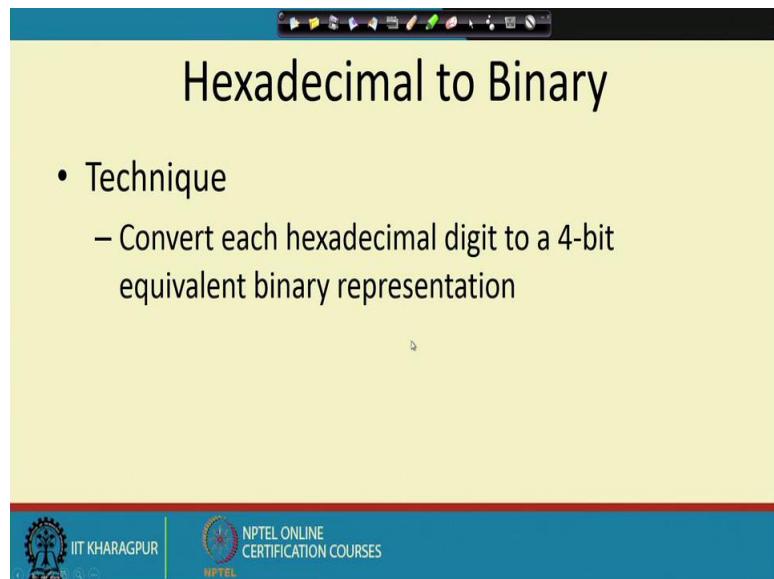
system again divide by 2. So, it is a slightly longer way of doing it, but that we are we are doing it for the sake of our confidence that this is the method is correct, ok.

(Refer Slide Time: 25:43)



So, next we will look into this hexadecimal to binary conversion. So, here just like the octal systems so, we took 3 bits each octal digit was taken separately and they converted into 3 bit pattern.

(Refer Slide Time: 25:46)



So, here also we will be taking the individual hexadecimal digits and then convert it into 4 bit equivalent because that hexadecimal 16. So, that can we can have it can represent a

values from 0 to 15, and we will need 4 bit for representing this 0 to 15 in the binary number system.

(Refer Slide Time: 26:15)

10AF₁₆ = ?₂

1 0 A F

0001 0000 1010 1111

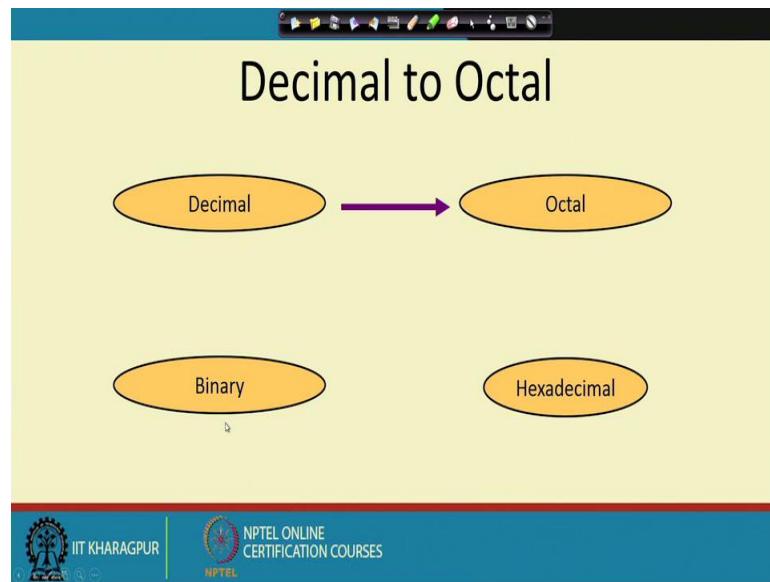
10AF₁₆ = 0001000010101111₂

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the same thing that is suppose (10AF)₁₆. So, what is the value in the binary number system? So, you can just take the 4 digits and then convert them into binary numbers 4 bit binary numbers and that is the result. So, you can just verify this by doing the similar exercise that is converting this into decimal and from decimal converting it into binary, but that is a lengthy process so, we are not doing it. Of course, we can say that this 0 0 0. So, this 3 zeroes are not necessary, because they are the leading zeroes and they do not have any contribution to the overall value of the numbers so, we can ignore them.

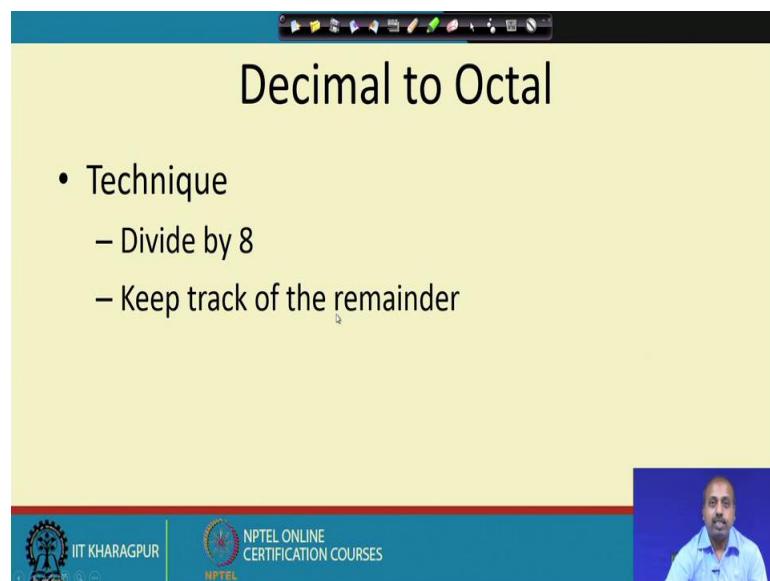
So, but this is just for the sake of completeness so, it has been done like this.

(Refer Slide Time: 27:02)



Now, how to convert decimal to octal?

(Refer Slide Time: 27:07)



Ok, so, just like decimal to binary we divided by 2 and kept track for the remainder. So, here also we do the same thing we divide by 8 and keep a track for the remainder.

(Refer Slide Time: 27:21)

Example

$$1234_{10} = ?_8$$
$$\begin{array}{r} 8 \overline{)1234} \\ 8 \overline{)154} \quad 2 \\ 8 \overline{)19} \quad 2 \\ 8 \overline{)2} \quad 3 \\ 0 \quad 2 \end{array}$$

$1234_{10} = 2322_8$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is a 1 2 3 4 to the base 10 converting it to hexadecimal sorry, octal number system divided by 8. So, remainder is 2, proceed another step, remainder is 2, then another step, remainder is 3 and then another step, so, remainder is 2.

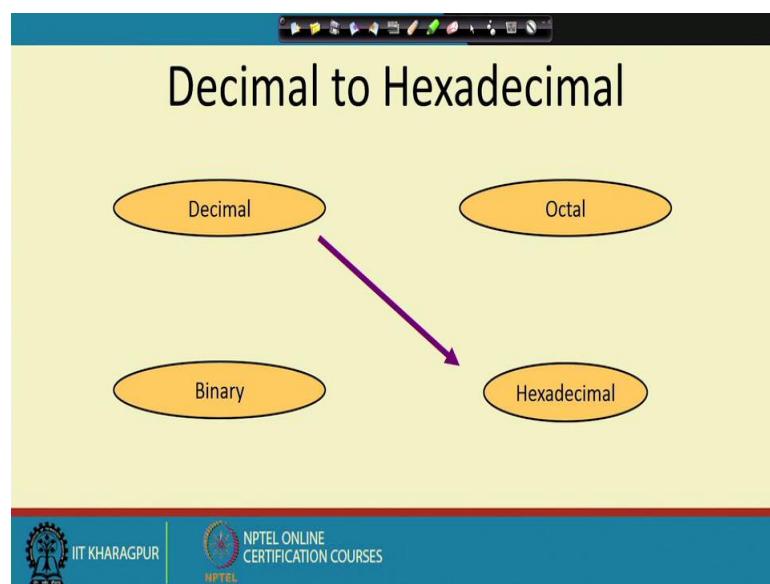
So, the number that we get is $(1234)_{10}$, the decimal number is basically $(2322)_8$ in the hexadecimal number system in the octal number system. So, in this way we can convert between the number systems, and by dividing by the base of the number system so, and it keeping track of the remainders.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 04
Number System (Contd.)

Decimal to hexadecimal conversion so, this is also similar in nature.

(Refer Slide Time: 00:20)



So, here we have to divide by 16 as you can understand by this time.

(Refer Slide Time: 00:29)

The slide has a yellow background. At the top, the title 'Decimal to Hexadecimal' is centered. Below the title, there is a bulleted list under the heading 'Technique':

- Divide by 16
- Keep track of the remainder

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, there is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a blue shirt.

So, this base conversion so, will be dividing it by 16. So, divide by 16, and keep track of the remainder.

(Refer Slide Time: 00:33)

The slide has a yellow background. At the top, the title 'Example' is centered. Below the title, the conversion equation is shown: $1234_{10} = ?_{16}$. To the right of the equation, there is a division diagram:

$$\begin{array}{r} 16 \overline{)1234} \\ 16 \overline{)77} \quad 2 \\ 16 \overline{)4} \quad 13 = D \\ 0 \quad 4 \end{array}$$

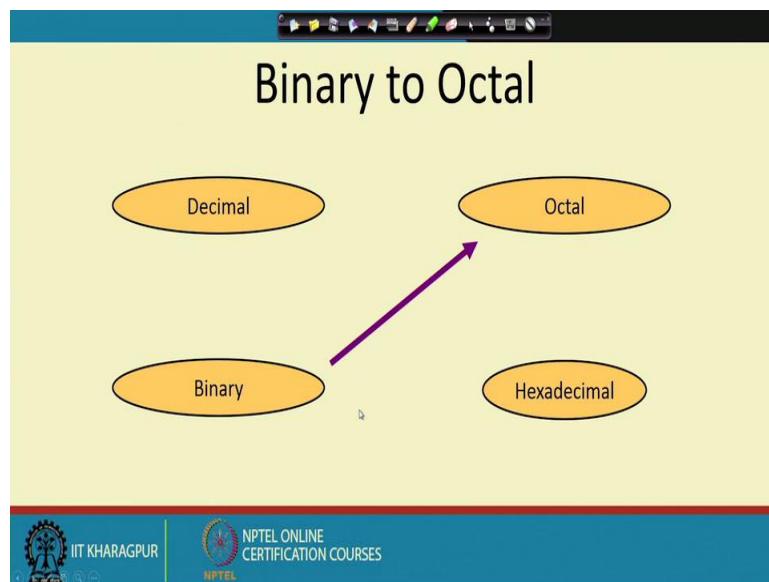
A red arrow points from the remainder 'D' down to the result '4D2₁₆' at the bottom right. At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, there is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a blue shirt.

So, $(1234)_{10}$ converting to hexadecimal. So, only thing is that division becomes a bit complex. So, divide by 16 so, remainder is 2, then divide by 16, remainder is 13. And 13 the corresponding hexadecimal symbol is D, so, 13 is basically D. So, this is the integer value, and this is the hexadecimal symbol corresponding to that. And then is 4/16,

remainder is 4 so the number that you get is 4D2. So, this 4D2 so, this is the number corresponding to this.

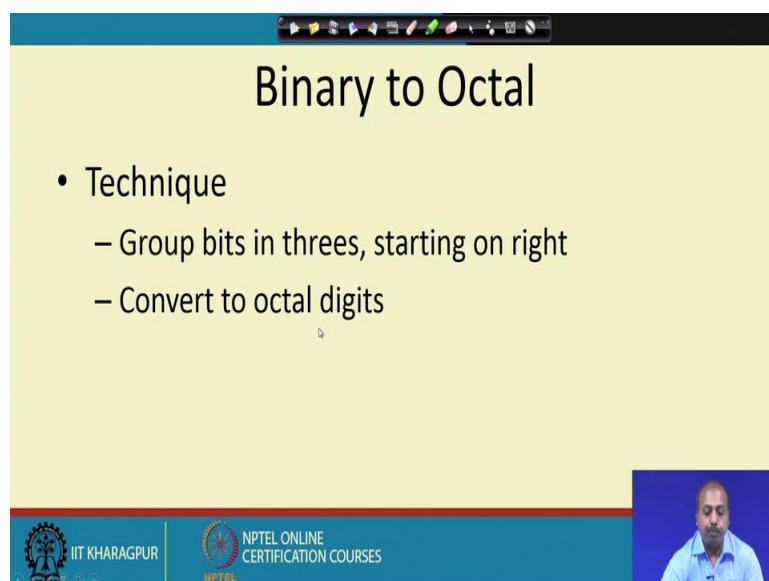
So, this way we can convert by a decimal number to different hexadecimal numbers.

(Refer Slide Time: 01:13)



Now, the other way binary to octal so, how can you do this thing? As I already said the one possibility is that is binary number, you convert to decimal and from this decimal you go to octal. So, that is one avenue, but we can say that that is two conversions will be necessary in the process. But that is not required so, you can do it directly.

(Refer Slide Time: 01:35)



So, you can group bits in 3s starting on the right and convert to octal digits so, will explain it with an example.

(Refer Slide Time: 01:44)

001011010111₂ = ?₈

001 011 010 111
↓ ↓ ↓ ↓
1 3 2 7

1011010111₂ = 1327₈

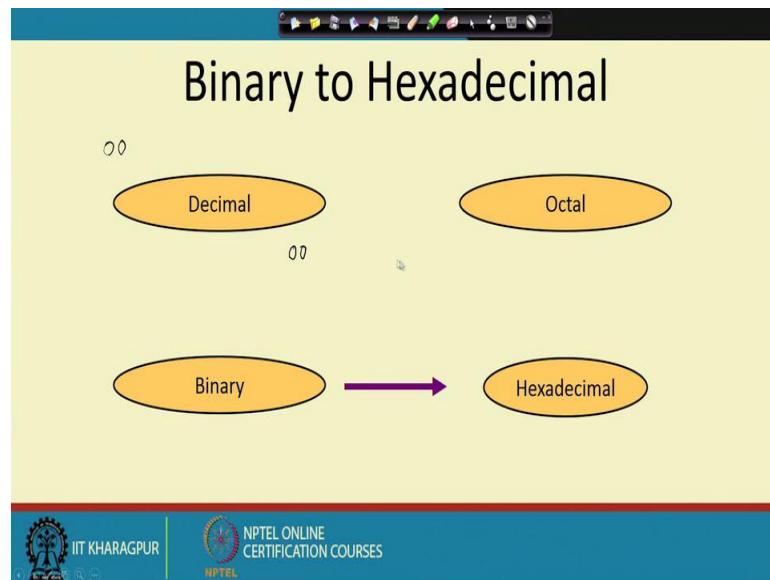
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Suppose this is the binary number that you have, I want to get the corresponding octal number. So, what is done? So, I make groups of 3 starting from the right side. So, you must keep in mind so, should not start from the left side so, you should start from right side.

Then this first 3 bits 1 1 1, they form the first group then this 0 1 0, from the next group. Then this 1 1 0 from the third group, and here I do not have 3 bits. So, you can safely assume that we have got this bits has 0 0's. Two 0's are there so, as a, because I can always assume that there are two 0's at this point. And because they will not contribute anything to the number. So, I can do that so, that way I can have this.

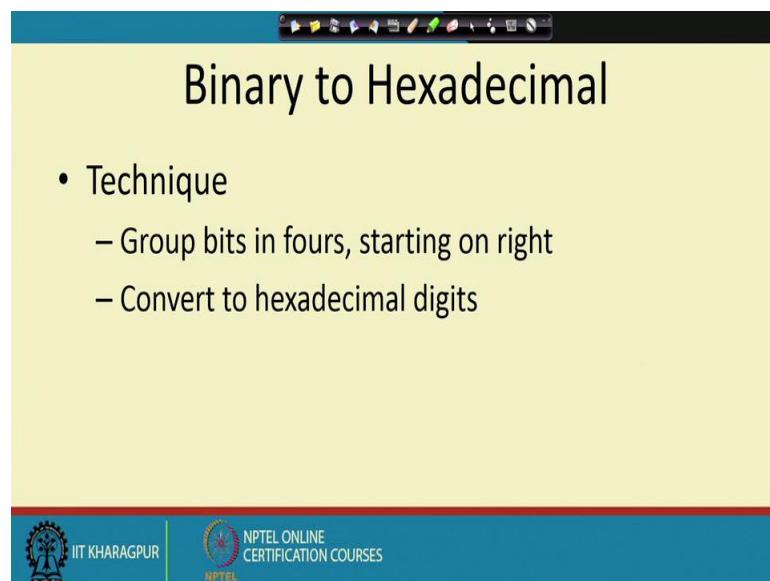
So now you convert the bits the 3 bits pattern into corresponding octal numbers. So, this 111 is 7, 010 is 2, 011 is 3 and 001 is 1. And the corresponding number becomes, this 1327.

(Refer Slide Time: 02:53)



So, this is the corresponding number so, if you are going for this binary to hexadecimal conversion. So, here also the same thing, so, but now the grouping that will do will be in terms of 4 bits.

(Refer Slide Time: 03:08)



So, we will group in terms of 4, 4 bits starting from the right side, and then that the groups the bit pattern group that we have so, they will be converted to hexadecimal digits.

(Refer Slide Time: 03:21)

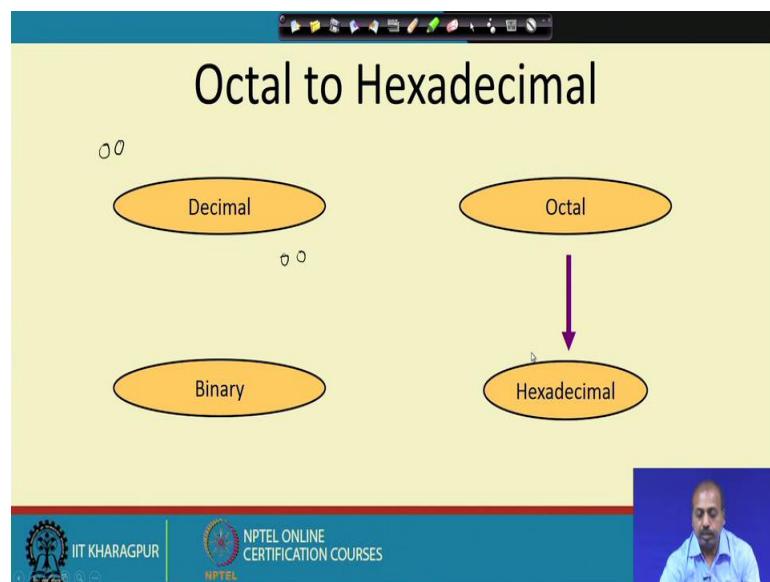
001010111011₂ = ?₁₆

0010 1011 1011
↓ ↓ ↓
2 B B

1010111011₂ = 0BB

So, here you are grouping in terms of 4 bits from the right. So, this 110 1 they from the first group then this 1 1 0 1 from the next group and then I have got only 2 bits left 0 and 1. So, 0 1 and as I did in the previous case so, you can safely assume that we have got 0's here, and those 0's are brought here ok. So, you can have this 0010. So, ultimately the pattern becomes 2BB, and this 2BB is the hexadecimal number corresponding to the binary number that we have.

(Refer Slide Time: 03:57)



So, then how to convert octal number to hexadecimal numbers.

(Refer Slide Time: 04:08)

Octal to Hexadecimal

- Technique
 - Use binary as an intermediary

IIT Kharagpur NPTEL ONLINE CERTIFICATION COURSES

So, it says that you can use a binary number as the intermediary. So, we take what we do is we convert the octal number to binary number first and from the binary number you convert to hexadecimal number.

(Refer Slide Time: 04:25)

Example

$$1076_8 = ?_{16}$$

1 0 7 6

↓ ↓ ↓ ↓

001 000 111 110

2 3 E

1076₈ = 23E₁₆

IIT Kharagpur NPTEL ONLINE CERTIFICATION COURSES

Like say this is an octal number $(1076)_8$, what is the value in the hexadecimal number.

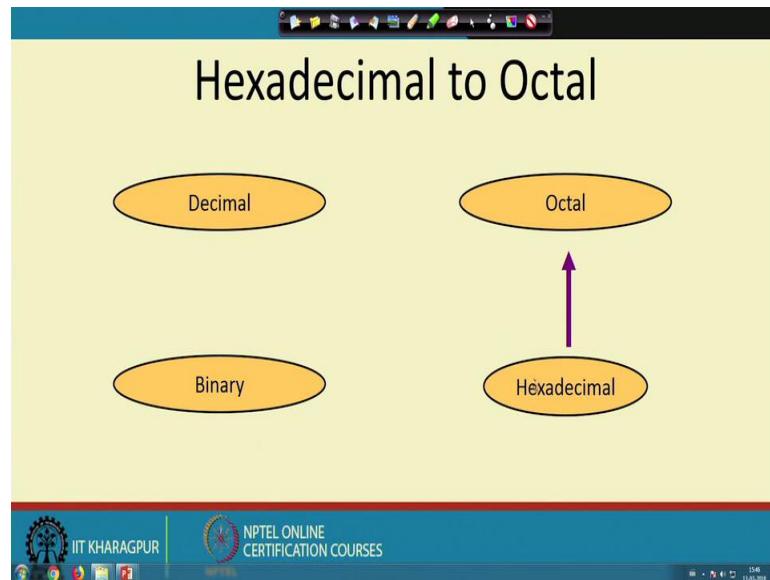
So, first this 1076 so, since this is an octal number so, individual digits I can convert it into 3 bit patterns and getting, the corresponding binary bits 001 000 111 110. And now I will be grouping them in terms of 4 bits from the right side. So, this so you see that this part so,

this portion up to this much. So, it is can be it is taken as one group, then this part is taken as another group, this part is taken as another group, and then this part is taken as another group.

And again for making it 4 so, the 2 extras 0's have been added, at the beginning to make it a group of 4. And then you convert it into the corresponding hexadecimal numbers. So, this 1 10 so, this is 14 so, 14 in hexadecimal number system is E. So, that is E, similarly this is 0011 that is 3. So, that is in hexadecimal also this is 3, and 0010 that is that is 2 in hexadecimal system it is true 2.

So, you can do this conversion, and accordingly you get this 1076 in hexadecimal in octal number system is equivalent to 23E in the hexadecimal number system.

(Refer Slide Time: 06:02)



Now, hexadecimal to octal so, this also you can guess what are we trying to do what will be doing. So, will be converting hexadecimal to binary, and from binary you will be converting into octal. So, will be following this avenue, ok, so, let us see how it is done using binary as intermediary.

(Refer Slide Time: 06:25)

1F0C₁₆ = ?₈

1 F 0 C

↓ ↓ ↓ ↓

0001 | 1111 | 0000 | 1100

 | 7 | 4 | 4

1F0C₁₆ = 17414₈

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, suppose we have got this hexadecimal number (1F0C)₁₆, and we want convert into corresponding octal number.

So, this so, first of all this 1F0C so, that we right down the corresponding the binary representation. And we know that straight way we can convert these individual digits to binary 4 bit binary patterns, and then that gives us the binary representation of this number.

Now, after that since I am trying to go to octal so, I have to make the groups of 3 so, I make groups of 3 from the right. So, the 001, that is there, next group is 100, next group is again 001, next group is 111 and the last group is 100. So, 3 extra 0 have been added so, that we can make it we can get 4 bit pattern here.

So, that way the number that we get in the octal number system is 17414 in the octal number system. So, this way we can convert very easily between this octal and hexadecimal number systems using this binary number system as the intermediary.

So, otherwise we have to convert to decimal, and then by from multiplying by powers of 8, and or dividing by power after the dividing by powers of 16. If you are trying for octal to hexadecimal conversion or hexadecimal to octal conversion you have to convert to decimal by multiplying by powers of 16, and then from that decimal number to octal by dividing by powers of 8.

(Refer Slide Time: 08:05)

Decimal	Binary	Octal	Hexa-decimal
33			
	1110101		
		703	
			1AF

And do a very simple exercise like the, we have got assume numbers, and how to what the values will be in different number system so, this is the answer.

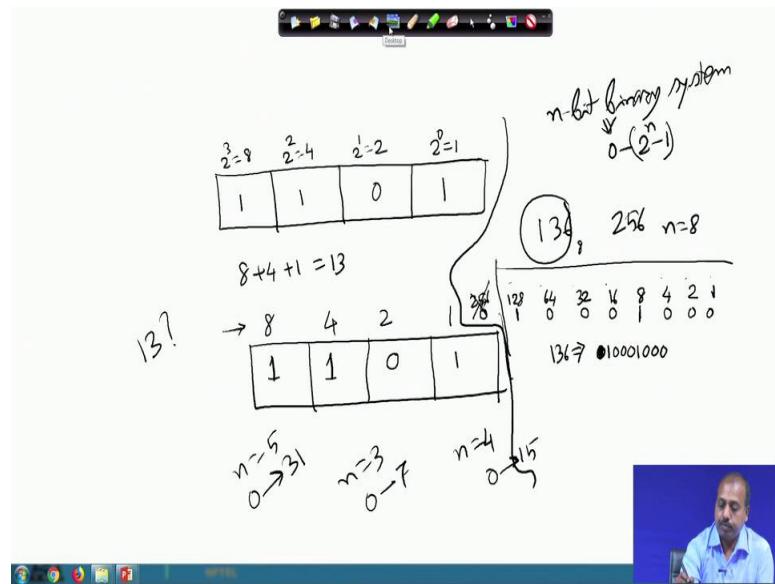
(Refer Slide Time: 08:15)

Decimal	Binary	Octal	Hexa-decimal
33	100001	41	21
117	1110101	165	75
451	111000011	703	1C3
431	11010111	657	1AF

So, 33 in binary number system so, this will be 1 0 0 0. Actually whenever you are doing these operations this binary number system so, that can be and easier way of doing this conversion so, you did not always go on dividing by 8 ok. So, will looking to a technique, and that will make it simple ok.

So, let us say that if we are looking at the individual digits of a binary number system. So, the first so, if this is, if suppose this is the, this is a 4 bit number; for example 4 bit binary number.

(Refer Slide Time: 08:53)



So, what is the contribution of first number? So, that is $2^0 = 1$. Contribution of the next digit is $2^1 = 2$. This is $2^2 = 4$, and this is $2^3 = 8$.

So, any number that you are representing so, they will be summing of the values of this 8, 4, 2 and 1 so, if I take the number say 1101 so, the number is $8 + 4 + 1$ so, that is 13.

So now if I ask you the question in the other direction, I tell you what is the representation of 13 in the binary number system. And you have in your mind this particular scale, ok, that on this scale I have got these values. So, this is 8, this is 4, this is 2, this is 1, these are the weights. So, what you do? From 13, so, since the most significant value, the weight is 8 so, this has to be given to reach 13 so, from 13 I have given 8. So, I am left with 5, sorry I am left with 5 next value is 4 so, I have to take it because I need to reach 5. So, I have to take it so, this 1 also I take.

So, what is now I am left with 1, and but the next weight is 2. So, I cannot take this so I put a 0 here, and the next one I am still left with 1 so, I take a one here. So, you see that if I just remember this particular scale then I can just put them 1's and 0's to convert

it into the binary number system. So, any arbitrary number let us take a say 136. First of all we have to see how many bit representation will be required for 136.

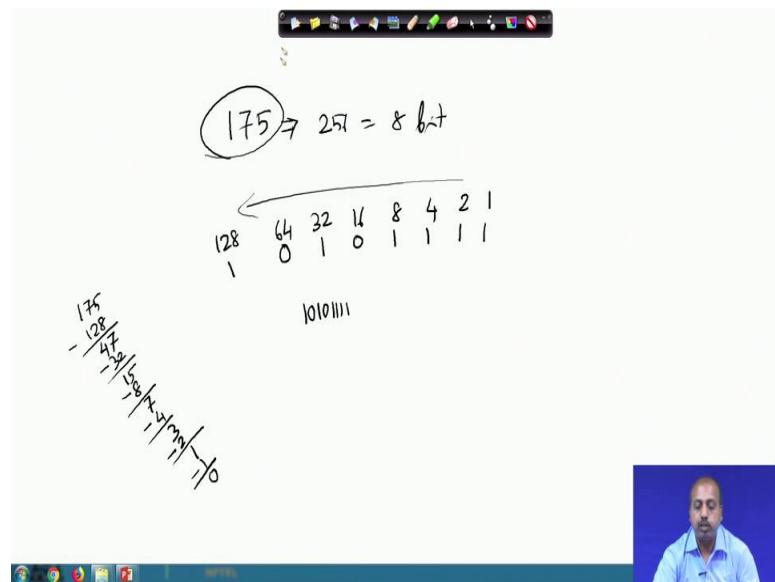
See if I have got n bit so, if I have got n bit binary number system, n bit binary number system, then it can represent numbers in the range of 0 to $2^{(n-1)}$. So, n bit binary numbers for example, if you take n equal to 4. So, you can represent numbers from 0 to 15. If you take n equal to 3, then you can represent the number 0 to 7 if you take n equal to 5 you can represent numbers from 0 to 31.

So, for 136 what will be the value of n? So, value of n will be the next power of 2 after 136 ok. So, the next power of 2, 2 is 256 that is n equal to 8. So, I will need an 8 bit number system to represent 136. So, in an 8 bit number system what happens to the values like I have got the first value is if I write from the LSB side, first is one, next is 2. So, I am just writing this position that we have written here say 1 2 4 like that. So, if you are writing like this 1, 2, 4, then 8, then 16, 32, 64, 128, 256. This is that 8 bit range that we have.

Now, for the number 136, we do not need this 256 bits. So, this is 0, fine? Now 128 I need because I want represent 136, 128 I need. So, after I have taken 128, I am left with only 8 ok. So, 64 I do not take I do not take 32, I do not take 16, I have to take 8 after that value has become 0 so, none of these bits are necessary.

So, 136 is actually represented as the bit pattern 010001000. So, this. So, 8 bit so, I am sorry this 120 after, this 256 will not come here, because I am going up to n equal to 8. So, this is 1 2 3 4 5 6 7 8 so, after this much will be necessary. So, this will be up to 128 will be necessary. And in that 128 so, for representing 136 128 will be necessary so, this 0 is also not there. This 136 will be necessary, after that I will be taking this only 8 will be left so, this 64, 32, 16. So, they will be not necessary only, this 8 will be necessary again, ok.

(Refer Slide Time: 14:21)



So, let us take another example say 175. So, for 175 again the same thing that the next power is 256 so; that means, I need an 8 bit representation. So, 8 bit if I take so, 1, 2, 4, 8, 16, 32, 64, 128. So, these are the 8 bit weights.

So, for 175 I will need this 128, after I have taken 128. So, $175 - 128$ so, that gives me 7, 47 so, more 47 more has to be represented. So, I cannot take this 64 so, that is 0. I have to take this 32 so, if you have subtract 32. So, you are left with 15 so, I do not take this 16, I take this 8 so, minus 8. So, left with 7 so, I have to take this 4, minus 4 and left with 3. So, I take this 2, minus 2 and left with one, I take this one, minus 1 left with 0.

So, the value 175 in the binary number system will be given by 10101111. So, you can check the corresponding decimal value to be equal to 175. So, this way we do not need to always do that division by 2 and all. So, if we just remember these powers of 2, and we can just go and assign we can take go on taking those positions by the turning on the corresponding bits to 1, ok.

So, here also is 33, say 33 means, I will require 6 bit representation. And in that 6 bit representation so, this is 1. So, this is 2, this is 4, this is 8, this is 16, this is 32 so, $32 + 1 = 33$. 117 - so, I will need 7 bit representation, because $2^7 = 128$. And then this is again the thing 1, 2, 4, 8, 16, 32, 64. So, say $64 + 32 + 16 + 4 + 1 = 117$ so, that gives us 117.

So, this way I can do the conversion, and once you have convert it into binary, then octal and hexadecimal conversion conversions are very simple, because for octal conversion what I need to do is I have to make groups of 3. So, I make a groups of 3 here I make another group of 3 here. So, the first group is one and the second group is 4. For hexadecimal I make a group of 4, and then I make a group of 4; like this so, that way it is 21, ok.

Similarly, here I make groups of 3 so, this is one group, this is another group and this is another group. So, I get 165 in octal and if you are taking hexadecimal, then this is one group and this is another group. So, the first group is so, sorry 0111 so, that is 7, and the next one is 0101 that is 5. So, this way you can you can very easily to a conversion from decimal to binary by taking help of that number scale that powers of powers of 2 scale. And then from there you can go to octal and hexadecimal numbers very easily. Do not need to do those multiplication divisions and all.

(Refer Slide Time: 18:25)

Common Powers (1 of 2)

- Base 10

Power	Preface	Symbol
10^{-12}	pico	p
10^{-9}	nano	n
10^{-6}	micro	μ
10^{-3}	milli	m
10^3	kilo	k
10^6	mega	M
10^9	giga	G
10^{12}	tera	T

IIT KHARAGPUR

NPTEL
NPTEL ONLINE
CERTIFICATION COURSES

So, next we common powers so, this is these are the different names that are given base 10. So, 10^{-12} is called pico 10^{-9} power minus 9 is called nano, 10^{-6} is micro. So, these are the standard thing up to 10^{12} is tera. So, in representation we represent 10 by p, n, I have then micro m, k for kilo then M for mega, G for giga, T for tera. So, this is so, this is common powers of base 10.

(Refer Slide Time: 18:59)

Common Powers (1 of 2)

- Base 10

Power	Preface	Symbol	Value
10^{-12}	pico	p	.00000000001
10^{-9}	nano	n	.000000001
10^{-6}	micro	μ	.000001
10^{-3}	milli	m	.001
10^3	kilo	k	1000
10^6	mega	M	1000000
10^9	giga	G	1000000000
10^{12}	tera	T	1000000000000

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And the corresponding values are like this ok, so, 10^{-12} will be this on so, this is known to us.

(Refer Slide Time: 19:06)

Common Powers (2 of 2)

- Base 2

Power	Preface	Symbol	Value
2^{10}	kilo	k	1024
2^{20}	mega	M	1048576
2^{30}	Giga	G	1073741824

- What is the value of "k", "M", and "G"?
- In computing, particularly w.r.t. memory, the base-2 interpretation generally applies

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if a base 2 so, we have got 2^{10} , 2^{20} , 2^{30} , they are called kilo mega and giga. So, those are the corresponding symbol and the values are 1024, then this one and this one 2^{20} , 2^{30} like that very big numbers. What is the value of kilo, mega and giga? So, these are the values actually.

In computing so, particularly with respect to memory so, will be using base 2 interpretation. So, in incase of, by incase of decimal number system so, kilo means thousand, but in case of binary number system. So, the kilo is 1024, and in computer systems or in digital processors. So, whenever we are talking with respect to memory so, will be talking about kilo as 1024, and then mega is 104857 that 2^{20} , 2^{30} ; like that, they are not the powers of 10, ok.

(Refer Slide Time: 20:05)

Review – multiplying powers

- For common bases, add powers

$$a^b \times a^c = a^{b+c}$$
$$2^6 \times 2^{10} = 2^{16} = 65,536$$

or...

$$2^6 \times 2^{10} = 64 \times 2^{10} = 64k$$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, some common rules, like say for common basis we can add the powers, like if you are multiplying 2 numbers $a^b \times a^c$, then we can add the power so, we can say a^{b+c} so, $2^6 \times 2^{10} = 2^{16}$. So, 65536 or say $2^6 \times 2^{10} = 64 \times 2^{10}$. So, I can, I the convert it in this and then you can say it is 64 k. So, either we can visualize it like this 65536 or we take it to the power of 2, ok. So, it say that it is 64 k, where the actual value is 65536.

(Refer Slide Time: 20:50)

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	10

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, when you are adding binary numbers to add 2 one bit values so, if this is the rule. So, we both the bits are $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 2$. So, which will be requiring 2 bits for representing it so, that is 1 0 so, that is 2.

(Refer Slide Time: 21:12)

- Two n -bit values
 - Add individual bits
 - Propagate carries

E.g.,

$$\begin{array}{r} 10101 \\ + 11001 \\ \hline 101110 \end{array}$$
$$\begin{array}{r} 21 \\ + 25 \\ \hline 46 \end{array}$$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if you are adding 2 n bit values so, you have to add individual bits and propagate the carry's just like in decimal addition so, we are propagating the carries, here also the same thing, like in decimal system; so, if you are adding say 45 and 96, what are doing? So, $6+5=11$ so, we write 1 here, and this 1 carry is taken to the next position. And then $9+4+1$

so, that is 14 so, 4 is written here, and one is taken has a carry and then that carry appears at this point.

So, that way we can have the same thing in case of binary number system as well. So, we can say that we can add individual bits and propagate the carry's. So, here $1 + 1$ is 0 and carry is propagated then $1 + 0 + 0$ is 1. So, there is no carry, then $1 + 0$ is 1, $0 + 1$ is 1, $1 + 1$ is 0 and this one is propagated so, this is one. So, this gives us the number 46 so, that is you can check it that this is really 46.

(Refer Slide Time: 22:21)

Multiplication (1 of 3)

- Decimal (just for fun)

$$\begin{array}{r} 35 \\ \times 105 \\ \hline 175 \\ 000 \\ 35 \\ \hline 3675 \end{array}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Multiplication so, you can multiply decimal number, like this so, this is the way you do multiply. So, first we multiply by the first digit, then do a shifting, multiply by the second digit, then do another shifting, multiply by the first digit, and then we just sum all this partial result, to get the overall, overall multiplication value.

(Refer Slide Time: 22:45)

The slide title is "Multiplication (2 of 3)". Below it is a bulleted list: "• Binary, two 1-bit values". A 4x3 grid table follows:

A	B	$A \times B$
0	0	0
0	1	0
1	0	0
1	1	1

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

So, here also in case of binary number system. So, the rule is like this 0×0 is 0 0×1 is 0 1×0 is 0 and 1×1 is 1.

(Refer Slide Time: 22:55)

The slide title is "Multiplication (3 of 3)". Below it is a bulleted list:

- Binary, two n -bit values
 - As with decimal values
 - E.g.,
$$\begin{array}{r} 1110 \\ \times 1011 \\ \hline 1110 \\ 1110 \\ 0000 \\ \hline 10011010 \end{array}$$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

And then if you are doing some multi bit multiplication, then we have to do the similar rules that we do for the decimal 1 so, first this is multiplied by once. So, if it is a one basically you get back the other numbers so, this 1110. So, then after that there is a shift, for the second bit multiplication so, you get again get 1110. Then this is another shift, but

this is all 0 this multiplying by 0. So, all are 0's, and then another shift and multiplying by 1 so, you are getting 1110. So, ultimately you are getting this one as the result.

So, this way can the rules of multiplication addition they remain same.

(Refer Slide Time: 23:31)

Fractions

- Decimal to decimal (just for fun)

$$\begin{array}{rcl} 3.14 & \Rightarrow & 4 \times 10^{-2} = 0.04 \\ & & 1 \times 10^{-1} = 0.1 \\ & & 3 \times 10^0 = \underline{\underline{3}} \\ & & 3.14 \end{array}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, for the fractions ok. So, how are the fractions represented? So, just for fun so, you can look into the decimal fraction, like how they are actually what is the interpretation of this fractional numbers in decimal numbers system. Like, we represent something like 3.14, and then what we do is that we say if we are trying to assign weight so, you start assigning weights from this less significant position. So, so, this is this position is 0, this position weight is 0. So, after that so, after the decimal point, the first one this weight is -1, the second one weight is -2, so, it goes like this.

So, in case of for the sake of conversion so, you can say it is $4 \times 10^{-2} + 1 \times 10^{-1} + 3 \times 10^0$. So, ultimately you get 3 point so, this is the decimal interpretation of this fractional numbers. So, whenever we are converting binary to decimal so, the same thing has to be done.

(Refer Slide Time: 24:38)

• Binary to decimal

$$\begin{array}{r} 10.1011 \Rightarrow \\ \downarrow \downarrow \downarrow \downarrow \downarrow \\ 10 \cdot 1 \cdot 0 \cdot 1 \cdot 1 \\ 1 \times 2^{-4} = 0.0625 \\ 1 \times 2^{-3} = 0.125 \\ 0 \times 2^{-2} = 0.0 \\ 1 \times 2^{-1} = 0.5 \\ 0 \times 2^0 = 0.0 \\ 1 \times 2^1 = 2.0 \\ \hline 2.6875 \end{array}$$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Because for so the, you can say that for this 1011 part. So, this one so, what is the weight? So, if, if you just right down so, this is the weights are; so, this is this is this weight is 0 so, this is -1, -2, -3 and -4, so, this weight is 1.

So now the rule is straight forward so, $1 \times 2^{-4} + 1 \times 2^{-3} + 0 \times 2^{-2} + 1 \times 2^{-1} + 0 \times 2^0 + 1 \times 2^1$. So, from right to left so, you just sum them up. So, it then after getting the values so, you just take the sum so, the value is 2.6875. So, this way you can convert the binary numbers so, which is a fractional number into the corresponding decimal values.

(Refer Slide Time: 25:47)

• Decimal to binary

$$\begin{array}{r} .14579 \\ \times 2 \\ \hline 0.29158 \\ \times 2 \\ \hline 0.58316 \\ \times 2 \\ \hline 1.16632 \\ \times 2 \\ \hline 0.33264 \\ \times 2 \\ \hline 0.66528 \\ \times 2 \\ \hline 1.33056 \\ \vdots \end{array}$$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

For decimal to binary conversion the rule is just the reverse. So, you have to take the 2 parts separately so, what we do is for this decimal part, for the integer part so, we take it we convert it separately, and for the fractional part we do it separately.

So, for this 3, if we if we are doing it if we are doing for 3. So, 3 is represented as 11, 3 is represented as 11. And for this 0.14579 so, we go on multiplying it by 2. So, 0.14579 multiplied by 2 it gives 0.29158, the portion that comes before this decimal point is 0. And then this 0.29158 is again multiplied by 2. So, you get point 0.58316, again this value that that is getting is 0. And again you are multiplying it by 2 so, this is getting 1, 1.16 something.

So, this is so, this value is 1. So, the portion that is coming before the fractional point after doing the multiplication is taken into consideration. And this value we go on multiplying so, this is multiplied by 2 again. So, you get a 0 here after that 0.33264 multiplied by 2 so, you get 0 point something. Again multiplied by 2 so, you get one point something so, it is not ending.

So, after this so, you can continue further, you can continue further multiplying again by 2 so, you will get is this before decimal you will get a 0. So, this way it can continue so, it is not ending here. But, but so, you so, the so, if it ns at some point of time everything become 0. So, you can stop at that point, otherwise it can go till infinity, but definitely for computer system. So, there is a finite storage. So, you cannot representing infinitely all the bits after the decimal point. So, we have to stop at some point of time.

So, if you say that will stop after so many bits. So, this is so, this is 0.001001. So, for the portion before decimal point so, you divide by 2 for the portion after decimal point, you multiplied by 2 . So, this 0 that you have got, it is coming into this 0 is coming into say this 0, then this 0 is coming into the next 0. This one is coming as the next one so, it comes like this successive bits they will be coming to the binary system like that.

So, in this way you can represent fractions in the binary, in the binary number system.

(Refer Slide Time: 28:43)

Decimal	Binary	Octal	Hexa-decimal
29.8	11101.11001100	35.63...	1D.CC...
5.8125	101.1101	5.64	5.D
3.109375	11.000111	3.07	3.1C
12.5078125	1100.10000010	14.404	C.82

So, this is an exercise so, 29.8 so, if you convert 29 into binary. So, you this is the number, and 0.8 if you go on multiplying by 2 so, 0.8×2 is 1.6. So, you get the one here now 0.6 that remains. So, multiplied by 2 so, you will get 1.2. So again get you will get a 1 here. Now 0.2 is remaining, 0.2 multiplied by 2 is 0.4. So, you get a 0 here. That 0.4 multiplied by 2 you will get again 0.8. So, that 0 comes here, the 0.8 multiplied by 2 you will get again point 1.6 the one comes here so, that way it goes on.

So, this way we can convert this decimal numbers into binary numbers. Now once the decimal number has been converted into binary so, you can do the conversion to other number systems; like, here so, to convert into octal number system so, what we are doing? So, we are making groups of 3. So, you make a group of this 3 bits, and then we make a group of this 3 bits for the integer part. So, this is 35, for the fractional part also we do grouping, but here grouping starts from this side.

So, grouping should start from left side. So, this is one group, this is the next group. So, first group is giving the number digit 6 next digit may be giving as 3. For hexadecimal so, you take groups of 4. So, that is the first one so, for the integer part, you start from the less significant bit position. And here you do you take 2 more you take 2 more 0's. So, that way you get this part is one and this part is $8 + 4 + 1$ that is 13; which is D, and for this fractional part you start grouping from this side. So, this part we group this 4 bits then you take two 0's and you group this part.

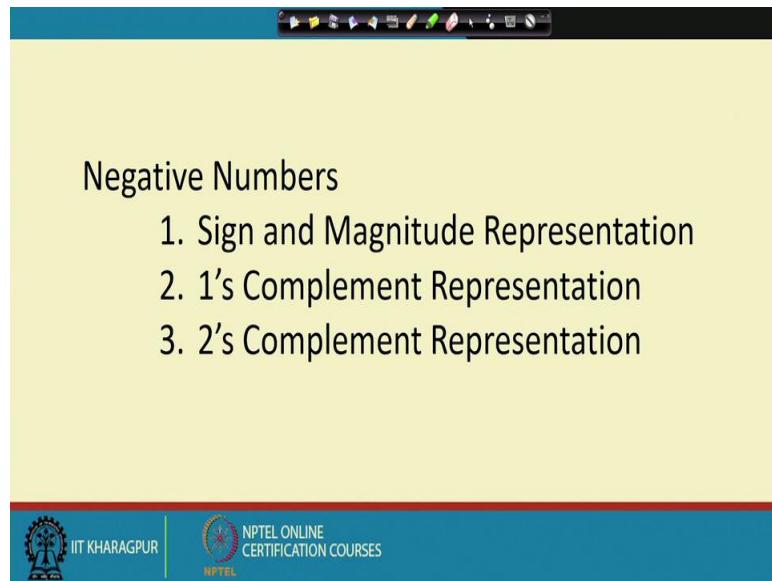
So, this is $8 + 4 = 12$, that is C so, you get 1DCC. So, this way you can do this conversions, so, for you can convert the decimal number to binary number first, and then from there using the conversion rules. So, you can go to octal number or hexadecimal numbers.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 05
Number System (Contd.)

So far we were talking about positive numbers, fine? So, but the numbers may be negative also.

(Refer Slide Time: 00:22)



Negative Numbers

- 1. Sign and Magnitude Representation
- 2. 1's Complement Representation
- 3. 2's Complement Representation

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will be looking into how to represent negative numbers. So, there are 3 distinct ways in which we represent negative numbers in computer systems. One is called sign magnitude representation, one is called 1's complement representation, the other one is 2's complement representation.

So, we will be looking into these 3 types of representation in our next few classes.

(Refer Slide Time: 00:44)

Goal of negative number systems

- Signed system: Simple. Just flip the sign bit
 - 0 = positive
 - 1 = negative
- One's complement: Replace subtraction with addition
 - Easy to derive (Just flip every bit)
- Two's complement: Replace subtraction with addition
 - Addition of one's complement and one produces the two's complement.

So, why do we need this? What is the goal of this negative number system? So, the signed system so, this is simple so, you just flip the sign bit so, 0 is positive and 1 is negative. So, if we have to if we. So, you can say that in your system. So, if I have got say 4 bits so, if I have got say 4 bits for representing numbers. So, in 4 bits I can represent the numbers 0 to 15, fine?

Now so, in signed system what will happen is that; so, we will have 4 + 1 5 bits. So, so these 4 bits are there, + there will be extra bit which is for the sign part, ok. Now this sign parts so, if it is 0; that means, I am representing positive numbers. So, naturally I can represent the numbers + 0 to + 15. On the other hand if this bit is one, then the number is a negative number. So, you can say as if it is - 0 to - 15, because this value can go up to 15.

So, of course, this is a bit confusing so, I have got a + 0 and a - 0, but that is there, but apart from that. So, this a representation is very simple. So, if we have to convert a number from positive to negative or negative to positive you just flip the sign bit of the number. So, that way the sign magnitude represent the sign system. So, this is very simple and it can be used for representing the negative numbers.

The 1's complement representation so, it is says that replace subtraction with, the goal is to replace subtraction with addition. So, the, this add this subtraction process so, we can do using addition. And here the idea is that we just flip every bit. So, if the number was

originally say if I, if I representing the number say + 5 in a 4 bit representation if the number if the number is 0101 for representing - 5.

(Refer Slide Time: 02:55)

Goal of negative number systems

- Signed system: Simple. Just flip the sign bit
 - 0 = positive
 - 1 = negative
- One's complement: Replace subtraction with addition
 - Easy to derive (Just flip every bit)
- Two's complement: Replace subtraction with addition
 - Addition of one's complement and one produces the two's complement.

We just flip all the bits so, it is 1010, ok. So, that is the, it is the 1's complement number system.

In 2's complement number system so; again we will be replacing subtraction with addition. So, addition of 1's complement and one produces the 2's complement. So, this is the 1's complement representation of - 5, we are looking for 2's complement representation, then with that you have to add a 1. So, if you were adding a 1 so, it is becoming 1011. So, that is the 2's complement representation of - 5.

So, this way these number systems they are the negative numbers can be represented, and we can use them for computer representation.

(Refer Slide Time: 03:42)

Given a positive integer x, we represent -x

- 1's complement:
 - Formula: $2^n - 1 - x$
 - i.e. n=4, $2^4 - 1 - x = 15 - x$
 - In binary: $(1111) - (b_3 b_2 b_1 b_0)$
 - Just flip all the bits.
- 2's complement:
 - Formula: $2^n - x$
 - i.e. n=4, $2^4 - x = 16 - x$
 - In binary: $(10000) - (0 b_3 b_2 b_1 b_0)$
 - Just flip all the bits and add 1.

$2^n - x + x = 2^n - x$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if we are, for the 1's complement, the overall formula is like this, if the original number is x, and you want to represent - x now, then for 1's complement the formula is $2^n - 1 - x$. So, for example, if we have a got n = 4 then for the formula is $2^4 - 1 - x = 15 - x$.

So, in binary so, what is happening is this 15 is 1 1 1 1, -x suppose the x is b₃ b₂ b₁ b₀. So, essentially what will happen is that all bits will get flipped. So, you were subtracting so, this will be it is subtracting from one. So, the bits will be getting flipped ok.

(Refer Slide Time: 04:32)

Definitions: 4-Bit Example

id	b ₃	b ₂	b ₁	b ₀	Signed	One's	Two's
0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1
2	0	0	1	0	2	2	2
3	0	0	1	1	3	3	3
4	0	1	0	0	4	4	4
5	0	1	0	1	5	5	5
6	0	1	1	0	6	6	6
7	0	1	1	1	7	7	7
8	1	0	0	0	8	7	6
9	1	0	0	1	-1	-6	-7
10	1	0	1	0	-2	-5	-6
11	1	0	1	1	-3	-4	-5
12	1	1	0	0	-4	-3	-4
13	1	1	0	1	-5	-2	-3
14	1	1	1	0	-6	-1	-2
15	1	1	1	1	-7	0	-1

Handwritten notes on the left side of the table explain the conversion of signed binary numbers to their one's and two's complements. It shows the process of flipping bits and adding 1 for two's complement.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, anything like say if I have got say, 5 sorry if I have got say 5, then in binary number system 4 bit binary it is 0 1 0 1 now say ah. So, I have got the numbers so I want to represent - 5. So, as per this formula it says 1 1 1 1, from there have to subtract 0101. So, 1 - 1 is 0, then this 1 - 0 is 1 1 - 1 is 0 1 - 0 is 1. So, what you get is 1010 which is the 1's complements.

So, if you take 1's complement of this. So, all the bits are flipped. So, you get 1010. So, you see that these two are same these two are same. So, this is same as flipping all the bit. So, the this is the mathematics so, we have to take $2^n - 1 - x$, but ultimately, it boils down to a very simple state you just flip all the bits. Then in the 2's complement the formula is $2^n - x$. So, in 1's complement we have already done this 1, $2^n - 1 - x$. So, that is why it says that if we with 1's complement if we add this 1 so, this is $2^n - 1 - x$. So, we if we that if we add one then these ones will cancel out. So, you will get $2^n - x$, fine?

So, we have got this 2's complement representation. So, so $n = 4$, we have got $2^4 - x$ so, that is 16 - x. So, so, was a 16 is 10000 - this one. So, in this if we take the previous example that for the, that - 5 so, it is 10000 - 00101 ok. So, so 0 - 1 is 1 with 1 carry. So, so basically so, 0 - 1 is 1, now 0 - 0 is 0 0 - 1. So, there will be one carry so, this 1 - 0 will be this will be 1. And then this, then 0 - 1 is one with a one carry then 1 borrow rather. So, 1 - 0 is sorry this is not anyway so, this is I will I will come back to this later this is not very simple here anyway.

So, basically what we have to do is, we have to do a this is done by means of some addition it is not, it is not doing the subtraction in 2's complement form that is why it created some problem so, we will come back to this later. But this is basically flipping all the bits and taking and adding one to that. So, so in say the, if these are number so, 0 so, in normal or representation is 000, signed representation.

So, this is 0, 1's complement representation is also 0 2's complement is also 0. So, for up to this much there is no change. So, up to 7 since it is a 4 bit representation up to 7 all are same. But when you go to 8, then you have got this 1000. So, since they have signed representation so, this will be using a sign bit. So, most significant bit is the sign bit so, this part is 0 so, it represents - 0.

Similarly, 9 in the sign representation will be - 1, because these 3 bits represent 1. So, this sign bit is one so, this is - 1, similarly this one is - 2. So, this is - 3 like that in 1's complement form so, this 1 0 0 0. So, that is your 15 - x. So, 15 - so, that is basically that 15 - 8 that is 7 and that is a negative number so, it is - 7. So, it is 15 - 9 is 6 that is - 6. So, like that in 2's complement number system it is 16 - x. So, it is 16 - 8 is 8.

So, this 1000 is - 8, 1001 is - 7 like that. So, this way from the formula will directly get all these values. But the point that I would like to mention is the range of values that you can represent in the number system so, in case of signed representation. So, you can represent the numbers - 7 to + 7 in a 4 bit if I give you only 4 bit, then you can represent the numbers - 7 to + 7. In 1's complement representation, your negative side you can go up to - 7, and the positive side we can go up to + 7; so, here also it is - 7 to + 7.

In 2's complement representation you can go from - 8 to + 7. So, this is something extra that you have got. And what was so, in so, I can say in general. So, if I have got say n bit. So, n bit representation, then in this signed and this signed and this 1's complement. So, they are giving you, you can represent numbers from $-(2^{(n-1)} - 1)$, that is for example, here it is $2^{(n-1)} - 1$. So, $-(2^{(n-1)} - 1)$ to $2^{(n-1)} - 1$. So, if we put n = 4 so, this is basically your - 7 to + 7.

In 2's complement notation, in 2's complement, so, you have got the range, which is $-2^{(n-1)}$ to $2^{(n-1)} - 1$, ok. So, for n equal to, say n = 4 so, you have got here - 8 to + 7. So, as you see here so, you can go up to - 8 on the negative side and + 7. On the positive side on 1's complement and all so, I can go to - 7 on the negative side at most. And + 7 on the positive side at most similarly here also I can go up to - 7 on the negative side and + 7 on the positive side.

So, why this thing has happened you see that so, in 2's complement notation. So, I can represent one number extra. So, this $-2^{(n-1)}$ so, this was not representable in 1's complement and signed representation, but it is representable in 2's complement form. This thing has happened because in case of this signed representation and 1's complement representation, so we are representing this + 0 as well as - 0. Similarly, in 1's complement also we are representing + 0 as well as - 0.

So, the value 0 it has got 2 different representations in signed representation and 1's complement representation. So, that is actually I should say utilizing one extra code ok; of the values so, but in case of 2's complement you have got only one 0. So, there is nothing like - 0 in 2's complement notation. So, we have got one extra pattern so, which can which is being utilized to represent say one extra number.

So, this way we can have this 2's complement number system to be doing better than this 1's complement, and signed magnitude there are of course, other advantage that we will see later, let us see.

(Refer Slide Time: 13:09)

Given n-bits, what is the range of my numbers in each system?

- 3 bits:
 - Signed: -3, 3
 - 1's: -3, 3
 - 2's: -4, 3
- 5 bits:
 - Signed: -15, 15
 - 1's: -15, 15
 - 2's: -16, 15
- 6 bits
 - Signed: -31, 31
 - 1's: -31, 31
 - 2's: -32, 31

Formula for calculating the range →

Signed & 1's: $-(2^{n-1} - 1), (2^{n-1} - 1)$

2's: $-2^{n-1}, (2^{n-1} - 1)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the thing that is given n bits what is the range of my numbers in each system, the 3 bits signed - 3 to + 3 1's complement - 3 to + 3 2's complement - 4 to + 3. So, if you have given 8 bits then signed is - 127 to + 127 1's complement - 127 to + 127. 2's complement is - 128 to + 127, and this is the rule that I was talking about formula for calculating the range signed and 1's complement is $-(2^{(n-1)} - 1)$ to $2^{(n-1)} - 1$. And for 2's complement it is $-2^{(n-1)}$ to $2^{(n-1)} - 1$ so, this way we can have ranges, ok.

(Refer Slide Time: 14:00)

Arithmetic Operations:
Derivation of 1's Complement

Theorem 1: For 1's complement, given a positive number $(x_{n-1}, x_{n-2}, \dots, x_0)$, the negative number is $(\bar{x}_{n-1}, \bar{x}_{n-2}, \dots, \bar{x}_0)$ where $\bar{x} = 1 - x$

Proof:

- (i). $2^n - 1$ in binary is an n bit vector $(1, 1, \dots, 1)$
- (ii). $2^n - 1 - x$ in binary is $(1, 1, \dots, 1) - (x_{n-1}, x_{n-2}, \dots, x_0)$.

The result is $(\bar{x}_{n-1}, \bar{x}_{n-2}, \dots, \bar{x}_0)$

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

So, how do I derive 1's complement? For 1's complement given a positive number x_1 ,

x_{n-1}, x_{n-2} these are the individual bits of the number, x_{n-1}, x_{n-2} and to x_0 . The negative number is given by this $\bar{x}_{n-1}, \bar{x}_{n-2}$ up to \bar{x}_0 where $\bar{x} = 1 - x$. So, so, $\bar{x}_{n-1} = 1 - x_{n-1}$ like that. So, it is because of the fact that from the definition of 1's complement, it says that it is from the definition of 1's complement it says $2^n - 1 - x$.

So, this $2^n - 1$ is all 1 and so, $2^n - 1 - x$ is 1, 1 all 1 - this so, naturally all the bits are getting complemented. So, for 1's complement so, we can get the number the 1's complement representation will be negative number in this fashion.

(Refer Slide Time: 15:09)

Arithmetic Operations: Derivation of 2's Complement

Theorem 2: For 2's complement, given a positive integer x , the negative number is the sum of its 1's complement and 1.

Proof: $2^n - x = 2^n - 1 - x + 1$. Thus, the 2's complement is $(\bar{x}_{n-1}, \bar{x}_{n-2}, \dots, \bar{x}_0) + (0, 0, \dots, 1)$

<u>Ex:</u> $x = 9$ (01001) 1's -9 (10110) 31 - 9 = 22 2's -9 (10011) 32 - 9 = 23	<u>Ex:</u> $x = 13$ (01101) 1's -13 (10010) 31 - 13 = 18 2's -13 (10011) 32 - 13 = 19
--	---

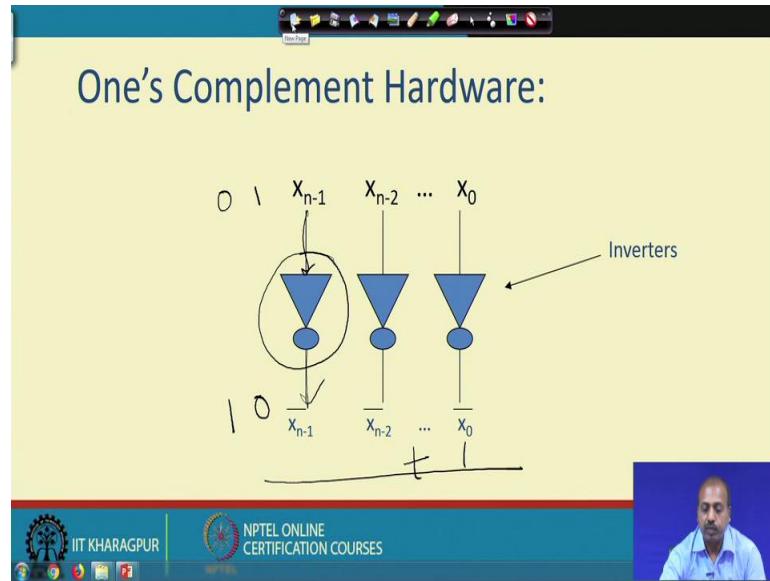
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For 2's complement the operation is like this that you have to the negative number is the sum of it is 1's complement and 1. So, we can prove it like this that for 2's complement this is the formula $2^n - x$. So, it can be written $2^n - 1 - x + 1$. So, now this part, $2^n - 1 - x$ so, this is the 1's complement ok, with that we are adding this one. So, 2's complement is basically the 1's complement + 1.

So, these are some examples like x equal to 9. So, this is the, or number 9 positive number 9. So, if we take 1's complement so, I will be getting this one. So, $31 - 9$ is 22 so, you can check that this number is really 22. So, it is 1 2 4 8 16 so, $16 + 4 + 2 = 22$. So, that is $31 - 9$ 2's complement of -9 is $22 + 1$. So, that is 23 so, that is $32 - 9$ that is 23.

Similarly, here for the 13 also we can get it is 1's complement as 10010, 2's complement as 10011. So, this way you can do this you can prove this 1's complement and 2's complement rules for doing the purpose.

(Refer Slide Time: 16:34)



So, 1's complement implementation is very simple. So, we will see we are not yet familiar with these symbols, but we will see after a few classes, that they are actually the inverters. So, inverter means if you whatever give value you give if just complements that bit and produces the output. So, if you give a one here, you will get a 0 here, we will put a 0 here, you will get a 1 here.

So, that is what you need for 1's complement. So, for 1's complementing the hardware is very simple, you just do this thing. But for 2's complement it is not that simple because for 2's complement, after doing this you have to have some adder which will be adding a one with this. So, that way the 2's complement hardware is going to be a bit complex compared to 1's complement. But anyway 2's complement is fine, 1's complement is fine.

(Refer Slide Time: 17:27)

Arithmetic Operations: 2's Complement

Input: two positive integers x & y ,

1. We represent the operands in two's complement.
2. We sum up the two operands and ignore bit n .
3. The result is the solution in two's complement.

Arithmetic	2's complement
$x + y$	$x + y$
$(x - y)$	$x + (2^n - y) = 2^n + (x - y)$
$-x + y$	$(2^n - x) + y = 2^n + (-x + y)$
$-x - y$	$(2^n - x) + (2^n - y) = 2^n + 2^n - x - y$

Handwritten notes on the right side of the slide:

$\begin{array}{c} \cancel{x} \\ \downarrow \\ 2 - \cancel{x} \end{array}$

NPTEL ONLINE CERTIFICATION COURSES

IIT KHARAGPUR

Now, whenever we are doing some arithmetic operation in 2's complement; suppose, we have 2 positive integers x and y has input so, we want to do some operations on x and y . So, for doing the operations first we represent the operands in the 2's complement form, and then we sum up the 2 operands and 2 operands and ignore the bit n . So, we it, we do not need to consider whether it is they are addition operation or subtraction operation, like say $x + y$. So, there is nothing to do so that is simply if the operation is $x + y$. But this $x - y$ what we do is that we do not use a subtractor rather this $-y$ is represented in 2's complement form.

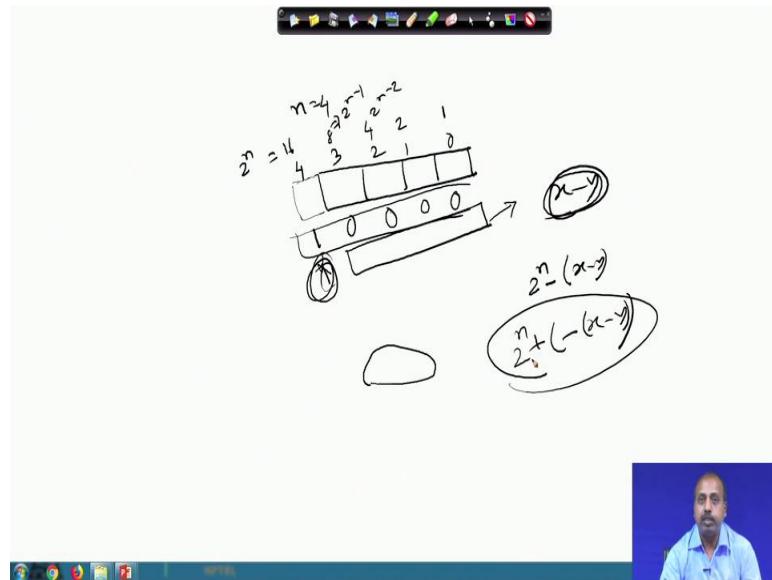
So, we have got the number y with us, the positive y with us. So, we take the 2's complement of this y so, 2's complement of y is $2^n - y$. Now when is so, if you just simplify it further. So, it is $2^n + x - y$ so, if this number turns out to be negative, then it will be subtracted from 2^n directly.

So, you will directly get the 2^n representation. So, basically, so, you remember that if a number is negative, if a number is negative, then suppose I have a number a , which is a negative number. So, it is represented as $2^n - a$ in the 2's complement notation.

Now, you see that I am doing this $x - y$. So, what I am doing? I am taking 2's complement of this y . So, accordingly I am getting this value $2^n - y$. Now after doing this operation if the number is positive then there is no problem, but if the number is negative ok. So, if the

number is positive then this 2^n , so, this 2^n is basically so, if you look into this individual bits ok.

(Refer Slide Time: 19:46)



Let me take it fresh page and so, if I have got say n equal to 4 fine. So, if n equal to 4 then I have got the bits 0123. And this bit, this bit is the 4th bit. So, that is basically for, if you look into the corresponding weights. So, this is 1 2 4 8 16. So, this is basically this is equal to 2^n , this is equal to 2^{n-1} , this is 2^{n-2} like that.

Now, the operation that I was doing there, the operation that I was doing there is this one. So, $2^n + x - y$. So, this 2^n o, where is it going to affect? So, it is 2^n is represented by this one, fine? $+ x - y$ so, if $x - y$ is a positive quantity if $x - y$ is a positive quantity. So, it will be restricted to this 4 bits only so, it is not going to affect this bit. So, whatever the bit is coming, if I simply ignore these bit, then I will get the sum I will get the result $x - y$.

If this result is negative if $x - y$ is negative then of course, that is no problem, because then these whole number is going to be 2 power the this the as per the representation. So, this is $2^n - (x - y)$ so, I am getting the result directly. So, if the number $x - y$ is negative, then I am. So, by doing this addition I am what I am essentially getting is $2^n + (x - y)$ so, I am getting the same result.

So, by doing this subtraction, instead of doing this subtraction so, if I do addition then also I am getting this subtraction done ok. So, then this $- x + y$ if I have to do it, I do not need

to do subtraction, I just take 2's complement of this x which is $2^n - x$, and then with that I add y .

So, as a result I will get $2^n + (-x + y)$; the same logic if this $-x + y$ is a positive quantity then this n th bit can simply be neglected. And if this result is negative then this will be this is anyway the 2's complement representation of that negative number. And $-x - y$ so, this is I am taking the 2's complement of them, and then doing the addition. So, again I am getting the same thing. So, this $2^n - x - y$ so, that is the 2's complement representation.

So, whatever we are doing whenever we are doing this subtraction operation. So, I can just take the 2's complement representation and do the addition operation.

(Refer Slide Time: 22:36)

Arithmetic Operations: Example: $4 - 3 = 1$

$$4_{10} = 0100_2$$
$$3_{10} = 0011_2 \quad -3_{10} \rightarrow 1101_2$$
$$\begin{array}{r} 0100 \\ + 1101 \\ \hline 10001 \end{array}$$

$10001 \rightarrow 1$ (after discarding extra bit)

We discard the extra 1 at the left which is 2^n from 2's complement of -3. Note that bit b_{n-1} is 0. Thus, the result is positive.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for example, this one is a $4 - 3$ is equal to 1 are we getting it by the formula that we are doing. So, 4 in 2's complement is it is a positive 4. So, this is 0100, 3 is 0011, now I need -3 . So, 2's complement representation of that is 1101.

Now, if I am adding now I am adding this, ok. So, this is the addition and this is the result ok. So, after doing the addition, so, if we discard this since the result is. So, we discard this extra one at the left. So, this extra one at the left is discarded. So, which is 2^n from 2's complement of -3 . So, and this b_{n-1} bit is 0 so, the result is positive. So, I will get a positive number 0001 so, this bit can always be neglected.

(Refer Slide Time: 23:34)

So, we will see with another example like say $-4 + 3$ that is equal to 1. So, how is it happening? So, 4 is 0100 so, -4 in 2's complement representation is; so, this is 1's complement + 1 so, 1100, 3 is 0011. So, $1100 + 0011$ so, this produces this, ok.

Now, there is no carry produced here to be neglected so, that carry is 0. Now you see that since this bit is 1 so, this means that this is a negative number and if we want to get the original the decimal number, so, what you do is you take the 2's complement of this. So, this is $0000 + 1$ that is 1 so, our answer is - 1. So, if the left most bit is 1, so that means, it is a negative number, and if you convert it if you convert this into it is 2's complement. So, you will get the corresponding positive number. So, here I am converting this 1111 into 2's complement. So, I am getting one so, which is the corresponding positive number.

So, in this way we can convert so, I never, I have never done any subtraction operation; though, it was necessary that I will be doing a subtraction operation of $3 - 4$, but that was not necessary.

(Refer Slide Time: 24:54)

Arithmetic Operations: 1's Complement

Input: two positive integers x & y,

1. We represent the operands in one's complement.
2. We sum up the two operands.
3. We subtract $2^n - 1$, if there is carry out at left.
4. The result is the solution in one's complement.

Arithmetic	1's complement
$x + y$	$x + y$
$x - y$	$x + (2^n - 1 - y) = 2^n - 1 + (x - y)$
$-x + y$	$(2^n - 1 - x) + y = 2^n - 1 + (-x + y)$
$-x - y$	$(2^n - 1 - x) + (2^n - 1 - y) = 2^n - 1 + (2^n - 1 - x - y)$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

What about 1's complement operation? So, 1's complement operation is a bit complex. So, here we have got if we have got 2 input integers x, positive integers, x and y. So, first of all we represent them in 1's complement, we sum the 2 operand, and we subtract $2^n - 1$ if there is a carry out left. So, in case of 2's complement we did not have to do anything. So, if we represent a 2's complement, then sum up. So, it was over, but in 1's complement it is not over at that point.

So, if there is a carry, then you have to subtract $2^n - 1$ and the result will be in 1's complement formula. So, this $x + y$ is $x + y$, $x - y$ is $x + 2^n - 1 - y$. So, this is 1's complements, and if then you are getting this thing and after that if there is a carry, then you have to subtract $2^n - 1$.

(Refer Slide Time: 25:50)

Arithmetic Operations: Example: $4 - 3 = 1$

$4_{10} = 0100_2$

$3_{10} = 0011_2 \quad -3_{10} \rightarrow 1100_2$ in one's complement

$$\begin{array}{r} 0100 \text{ (4 in decimal)} \\ + 1100 \text{ (12 in decimal or } 15-3\text{)} \\ \hline 1,0000 \text{ (16 in decimal or } 15+1\text{)} \\ 0001 \text{ (after subtracting } 2^n-1\text{)} \end{array}$$

We discard the extra 1 at the left which is 2^n and add one at the first bit.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us take an example suppose $4 - 3$ ok. So, 4 is 0100, 3 is 0011. So, -3 in 1's complement is 1100. Now we are adding this so, after doing this addition you see there is a carry. So, carry one is generated so, it says that to get the result you have to subtract $2^n - 1$. So, $2^n - 1$ is. So, you have to subtract 15 from here so, after subtracting 15 you so, this value is 16. So, 2 to the power $n - 1$ is 15 so, from 16 I have to subtract 15 I will get a one.

So, we discard the extra one at the left which is 2^n and add one at the first step. So, this is the process of subtraction. So, you discard this so, that way you are losing this 16. So, you are subtracting 16 and then you are adding one. So, essentially you are subtracting 15 so, essentially we are subtracting $2^n - 1$ that is 15 in this case.

(Refer Slide Time: 26:54)

So, another example so, $-4 + 3$ is -1 how is it coming. So, 4 is 0100 so, -4 1's complement is 1011 , 3 is 0011 . So, you are doing this addition so, after doing this addition it comes 1110 . Now so, there is no carry generated. So, I do not have to do anything so, know the $2^n - 1$ subtraction that is not necessary. So, it is the so, these itself is the number that I have got the, but it is in the 1's complement form. So, to get the actual number, what you have to do is that you have to convert it to 1's complement that is 0001 . So, that is one, but since this bit is one so, it is a negative number so; the value that you get is -1 .

So, if the left most bit is one so, you were getting a -1 . So, the rule is so, the rule is this one that after doing the summation if there is a carry, then you subtract $2^n - 1$. Then either case the result will be available in 1's complement notation. So, this way in the, in the, in this example there was a carry generated. So, we had to subtract 15, and for subtracting 15 the clever way that we did is we ignore this ones. So, as a result I have subtracted 16, and then I have added I have turned on this I have added one at the least significant position.

So, that way I have made + 1's so, $- 16 + 1$ so, actually 15 is subtracted. So, in this case since this carry was generated so, I had to subtract 15, but in this case we have got there is no carry generated. So, no subtraction is necessary so, this result is fine, but only thing is that to get the corresponding value what it is, to understand what is the corresponding value. So, we have to convert it take it as 1's complement you take the 1's complement of

this so that is one and since this bit is one. So, this is a negative number so, it is - 1 so, - 4 + 3 is - 1.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 06
Number System (Contd.)

So, in our last class we were looking into negative numbers and we have seen that they can be represented either in sign magnitude format or 1's complement format or 2's complement format. Now, how to get back the number like once it is a negative number and represented in 1's complement, how do we know what is the number.

(Refer Slide Time: 00:38)

Recovery of the Numbers

1's Complement Let $f(x) = 2^n - 1 - x$ Theorem: $f(f(x)) = x$ Proof: $f(f(x))$ $= f(2^n - 1 - x)$ $= 2^n - 1 - (2^n - 1 - x)$ $= x$	2's Complement Let $g(x) = 2^n - x$ Theorem: $g(g(x)) = x$ Proof: $g(g(x))$ $= g(2^n - x)$ $= 2^n - (2^n - x)$ $= x$
---	---

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, it is like this for 1's complement system so, you can say it is so, we define the function $f(x) = 2^n - 1 - x$ then $f(f(x)) = x$. So, that is if we apply this particular function on itself ok so, you will get back the x . So, why because if we are trying to find this $f(f(x))$ then $f(x) = 2^n - 1 - x$. So, we put it here.

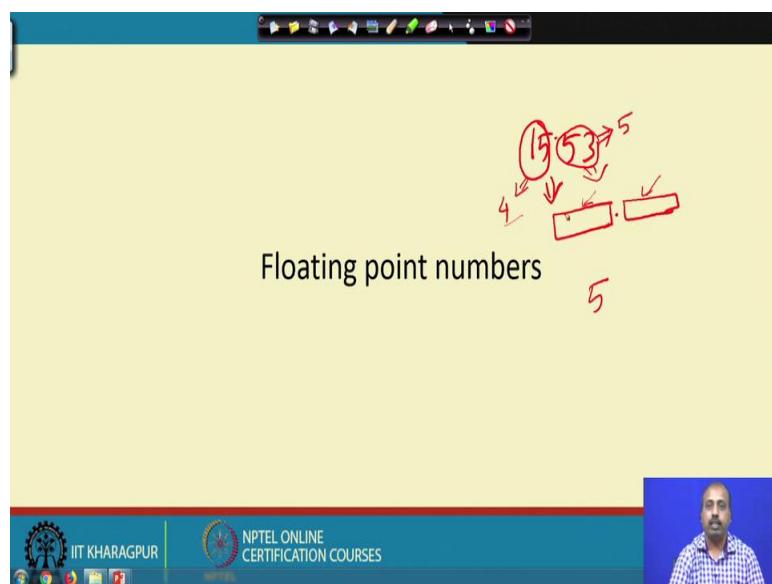
Now, $f(2^n - 1 - x) = 2^n - 1 - (2^n - 1 - x) = x$ So, this 2^n and 1, they will cancel out what remains is the x . So, if you so, this is the 1's complement representation of the number $-x$. So, from there if you want to get back what is the original x so, you can apply the function on it and get back x . So, if you apply on this f , if you apply on apply the function f on this input, so, you will get back the original number ok.

So, whose negative is actually represented by this representation. Similarly in the 2's complement case also suppose so, this is the formula for getting the 2's complement of a number. So, $g(x) = 2^n - x$. So, we have we have a similar theorem, which says the $g(g(x)) = x$, that is if we apply the function g on the result $g(x)$. So, you will get back x .

So, as a proof we can see that $g(g(x)) = x; g(x) = 2^n - x$. So, you put it here. So, $g(2^n - x) = 2^n - (2^n - x) = x$ So, this you this way we get this one so, you get back the x .

That means both in 1's complement and 2's complement notation, if you want to get back the original number whose negative was represented as $2^n - x$ or $2^n - 1 - x$ in 1's complement or 2's complement, to get back x . So, you take the 1's complement or 2's complement of the number. So, you will get back the original number x . So, it is helpful for our understanding ok. So, if you are getting a negative number. So, what was the original positive number. So, we can get back by applying the function again.

(Refer Slide Time: 03:17)



So, next we will see the representation for floating point numbers. So, far we have considered only integers positive and negative, and we have also considered fractions like say we have seen that this numbers where the number of digits that we are using for representing it is fixed. So, like we can say we can we have representation like say 15.53.

So, we have seen that this 15 part, we are converting it into some integers some integer part then this point is there and then this 53 also we are convert into binary to represent it.

So, that way this representation of a problem comes when this number of bits that are allocated for this part and number of bits that are allocated for this part they are same, they are fixed ok. So, you cannot change them. So, in so, like if I say the so, for example, for representing 15, you will need at least 4 bits and for a representing 53 properly, so, you will be again requiring say suppose I give you say 5 bits. So, 5 plus 4 total 9 bits are available; but if 9 bits are needed, but in a reality so, if we give you say only say 5 bits.

Then for the sake of accuracy how do you choose like how many bits we can give for this integer part and how many bits we can give for the fractional part. So, if you are using this real number representation in the form of fixed point notation, so, there are problems in terms of accuracy. So, we will see that we can do better by realizing them in the floating point form.

So, we will be looking into this floating point representation in the, in the next few slides.

(Refer Slide Time: 05:17)

Real Numbers

- Two's complement representation deal with signed integer values only.
- Without modification, these formats are not useful in scientific or business applications that deal with real number values.
- Floating-point representation solves this problem.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this real numbers so, two's complement numbers that we have discussed so, far they this they deal only with signed integer values and without modification these formats are not suitable in scientific or business applications that deal with a real number values because of the accuracy and all.

So, floating point representation, they will solve this problem. So, we can see that this floating point representation so, they can help us in having a better accuracy and all.

(Refer Slide Time: 05:43)

Floating-Point Representation

- If we are clever programmers, we can perform floating-point calculations using any integer format.
- This is called *floating-point emulation*, because floating point values aren't stored as such; we just create programs that make it seem as if floating-point values are being used.
- Most of today's computers are equipped with specialized hardware that performs floating-point arithmetic with no special programming required.
 - Not embedded processors!

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, if in fact, before going into this discussion on this floating point representation one thing we should keep in mind that ultimately this floating point is a representation is nothing, but collection of a, some integers. It is a, every floating point number is represented by means of a collection of integers.

So, if a, if in the programming we can use some tricks by which we can do this floating point operations ourselves. So, if underlying machine or say operating system, it is not supporting that floating point representation also. So, as a programmer so, you can always do that and you can use the integer formats for representing different components of the floating point number and represent it that way.

So, this is known as the floating point emulation because the so, because the floating point number is not stored as such. So, we just create programs that so, that it seems like we as if we are handling floating point numbers. But most of the computers that we have today or the processors the advanced processors, they are equipped with specialized hardware that can do this floating point arithmetic without any special programming requirement.

So, this is very important, because otherwise whenever you are you whenever you represent this floating point operation by means of software, the overall operation will

become will become slower, because software is always much slower compared to the hardware.

So, if the hardware directly supports the floating point format. So, it is better. So, most of the processors that we have now the advanced processors so, they support these floating point formats in the hardware itself. However, these are not embedded processors because these are not dedicated for the floating point operation only. So, they do many other things, but they also support the floating point operation.

(Refer Slide Time: 07:37)

Floating-Point Representation

- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
 - For example: $0.5 \times 0.25 = 0.125$
- They are often expressed in scientific notation.
 - For example:
 $0.125 = 1.25 \times 10^{-1}$
 $5,000,000 = 5.0 \times 10^6$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, how do we represent a floating point? So, floating point numbers they allow an arbitrary number of decimal places to the right of the decimal points.

For example so, floating point means so, we can have any number of digits after this decimal point. So, that is the floating point; obviously, for actual representation so, it cannot be arbitrary. So, there has to be some higher limit on the number of digits that you have after the decimal point, because a computer has got any processor or computer it has got a finite word size. So, we cannot go beyond that anyway. So, if you are multiply 0.5 by 0.25 you get 0.125.

So, you see the problem the point that is the this point the that this bullet tells is that these numbers, 0.5 and 0.25, they have 1 digit after decimal here and 2 digits after decimal here and in the result I have got 3 digits after the decimal. So, after decimal how many digits

will come. So, that is not fixed. So, that is why it is a floating point. So, in a scientific notation so, we can represent it like this, say this 0.125 that we have got so, it can be represented as 1.25 into 10 power minus 1, say this 5 they are 6 zeroes after 5 is a very large number.

So, you can represent it as 5×10^6 . So, you see that if we are representing it in this format, then the numbers they have got two components. The first component is a number and the second part is a power of 10 ok. So, that way we can have different levels of we can the range of the numbers that we are going to store. So, it becomes larger because we are we are converting it into this format.

(Refer Slide Time: 09:26)

Floating-Point Representation

- Computers use a form of scientific notation for floating-point representation
- Numbers written in scientific notation have three components:

Sign Mantissa Exponent

+ 1.25 × 10⁻¹

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, computers they use this, but this type of scientific notation for floating point representation. So, numbers written in scientific notation they have got three components; one is the sign another is called mantissa, another is called exponent. So, the sign is whether the number is positive or negative that is all right, then the mantissa part.

So, mantissa we will have a particular format like in this case so, we are assuming that we have got 1 digit before the decimal point so, only 1 digit before the decimal point. So, this is a 10^{-1} so, that is the exponent part. So, I need to remember this sign bit this 1.25 and this -1 to know what is the number that I am going to store. So, they are called sign mantissa exponent representation.

(Refer Slide Time: 10:17)

Floating-Point Representation

- Computer representation of a floating-point number consists of three fixed-size fields:

Diagram illustrating the fields of a floating-point number:

- Sign (represented by a blue oval)
- Exponent (represented by a green rectangle)
- Significand (represented by a pink rectangle)

• This is the standard arrangement of these fields.

Note: Although "significand" and "mantissa" do not technically mean the same thing, many people use these terms interchangeably. We use the term "significand" to refer to the fractional part of a floating point number.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Computer representation of floating point numbers so, they consists three fixed size field. The sign field is definitely there so, that is the, that is a sign then the exponent part that is the power of 10 that we are talking about so, the exponent part and the significand part.

So, significand part is the in the previous example where you when you are talking about mantissa. So, that is actually told as significand here. So, exponent is same. So, this is a very standard arrangement of this field that is the after sign the exponent will come and then the significand will come. So, depending upon the number of bits that you allocate to different portions, the sign will definitely be 1 bit, but this exponent and significand. So, you can assign different number of bits to tell like how many up to what power we can represent or up to how many digits after decimal point so, we can represent in the significand form.

So, although this significand and mantissa they do not technically mean the same thing. So, they can be used interchangeably ok. So, we will see why they are not same.

(Refer Slide Time: 11:25)

The diagram illustrates the structure of floating-point representation. At the top, the title "Floating-Point Representation" is displayed. Below it, a light blue box contains three fields: "Sign" (a single bit), "Exponent" (a green box), and "Significand" (a pink box). An arrow points from the "Sign" field down to the "Exponent" field. A list of bullet points explains the significance of each field:

- The one-bit sign field is the sign of the stored value.
- The size of the exponent field determines the range of values that can be represented.
- The size of the significand determines the precision of the representation.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side, there is a small video window showing a person speaking.

So, the one-bit field is the sign value that is stored here, the size of the exponent field it determines the range of values that can be represented, as I have already said. The size of the significand it determines the precision of the representation.

So, if I say that this significand is 32 bit, then after decimal point we are going to represent 32, up to 32 bit we are taking the precision. So, if more number of bits are available for the significand part. So, I will have better precision and less number of bits means, I will have less precision. So, that way this exponent mantissa and significand parts so they are.

(Refer Slide Time: 12:08)

The diagram illustrates the structure of floating-point representation. At the top, the title "Floating-Point Representation" is displayed. Below it, a light blue box contains three fields: "Sign" (a single bit), "Exponent" (a green box), and "Significand" (a pink box). An arrow points from the "Sign" field down to the "Exponent" field. A list of bullet points explains the significance of each field:

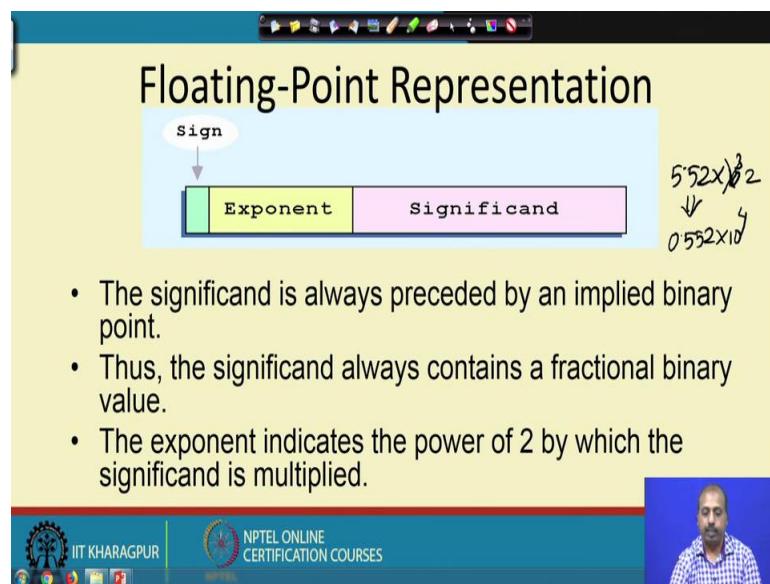
- We introduce a hypothetical "Simple Model" to explain the concepts
- In this model:
 - A floating-point number is 14 bits in length
 - The exponent field is 5 bits
 - The significand field is 8 bits

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side, there is a small video window showing a person speaking.

So, we will take a very simplistic example ok. So, for our discussion so, a hypothetical simple model to explain the concepts. So, we will think that the floating point number altogether I am giving it 14 bits ok. So, this entire from this entire structure that I have so, that is given 14 bits. So, out of that sign will take 1 bit.

So, we are left with 13 bit. Out of the 13, so, exponent part we are giving 5 bits and the significand part we are giving 8 bits. So, this is the, this is our simplistic model a 14 bit representation of floating point numbers.

(Refer Slide Time: 12:49)



So, what can we do with this? The significand is always preceded by an implied binary point. So, it is suppose I am going to represent a number say a 5.52×10^3 .

So, what we will do. So, we will not represent it like this. So, we will take it as 0.552 into 10 to the power of 4 ok. That that the, that is why the significand is always preceded by the implied binary. So, binary point is always at this at this point. So, at the very first one is the binary point ok. So, we do not need to store it explicitly, meaning that it starts with the binary point. So, significand always contains a fractional binary value like here it is 0.552 as I was talking about.

The exponent part so, it is a power of 2. So, instead of since I am representing in a computer. So, instead of 10 power. So, I will be calling it 2 power ok. So, it is a power of 2. So, it is. So, it is 5.52×2^x . So, 10^3 .means 1000. So, it is may be 5.52×2^{10} is 1024.

So, it will be close to that ok. So, that way I can so, I that the exponent part is power of 2 instead of power of 10 that we are that we are looking into.

(Refer Slide Time: 14:18)

The slide has a yellow background with a black header bar. The title 'Floating-Point Representation' is centered at the top. Below the title is a bulleted list:

- Example:
 - Express 32_{10} in the simplified 14-bit floating-point model.
 - We know that 32 is 2^5 . So in (binary) scientific notation $32 = 1.0 \times 2^5$.
 - In a moment, we'll explain why we prefer the second notation versus the first.
 - Using this information, we put 110 ($= 6_{10}$) in the exponent field and 1 in the significand as shown.

Below the list is a diagram showing a 14-bit binary number in three fields: a sign bit (0), an 8-bit exponent field (00110), and a 6-bit significand field (100000). To the right of the diagram is a small video window showing a man speaking.

The footer of the slide includes the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

So, so, suppose I am trying to represent the number 32 in decimal system into this 14 bit floating point notation that we have introduced. So, 32 is 2^5 so, in binary notation by scientific notations. So, 32 will be 1.0×2^5 . So, scientific notation means, before decimal point we have got 1 digit and in case of this floating point representation that is followed in the computer system. So, we have got a decimal point it starts with the decimal point.

So, this is 32 in scientific notation is 1.0×2^5 , in our notation it is 0.1×2^6 ok. So, why do we prefer the second one? So, we will see slightly later. So, this 6 ok so, 6 is 110. So, in the exponent part we put 110 and total 5 bits are allocated to the exponent part out also in this in this 5 bit location, we stored the value 6, 110.

And then on the significand part, so, we have to keep only this 1 that we have so, because this point is implied here. So, we have to I have to keep this 1 and 1 is represented like this. So, this is so, 10000 like that. So, it is 10000. So, it will be 0.1×2^6 . So, that represent again 32. So, so this is. So, we put 110 in the exponent part and 1 in the significand part as we have shown here.

(Refer Slide Time: 16:03)

Floating-Point Representation

- The illustrations shown at the right are *all* equivalent representations for 32 using our simplified model.
- Not only do these synonymous representations waste space, but they can also cause confusion.

0	0 0 1 1 0	1 0 0 0 0 0 0 0
0	0 0 1 1 1	0 1 0 0 0 0 0 0
0	0 1 0 0 0	0 0 1 0 0 0 0 0
0	0 1 0 0 1	0 0 0 1 0 0 0 0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, these are all equivalent like say the. So, this is the representation they are all representing 32. It is 32 into 2 to the power it is it is 6. So, it is 0.1×2^6 . So, this is point 0 1. So, this is 0.01×2^7 then it is 0.001×2^8 . So, that way these multiplying by 2 will be shifting it by 1 position in the binary representation.

So, that way all these representations, they have the same value which is 32. So, they are they are synonymous ok. So, they, but they can also cause some confusion. So, we need to have some standardization, because otherwise what will happen is that different computers may be storing information in different format. So, as a result understanding may become confusing ok. So, we have to use some fixed notation.

(Refer Slide Time: 17:06)

The diagram illustrates the floating-point representation of a number. At the top, the title "Floating-Point Representation" is displayed. Below it, a light blue box contains three fields: "Sign" (represented by a small circle), "Exponent" (represented by a green rectangle), and "Significand" (represented by a pink rectangle). An arrow points from the "Sign" field to the "Exponent" field. A red callout box contains the text: "All of these problems can be fixed with no changes to our basic model." At the bottom of the slide, there are two logos: IIT Kharagpur and NPTEL.

- Another problem with our system is that we have made no allowances for negative exponents. We have no way to express $0.5 (=2^{-1})$! (Notice that there is no sign in the exponent field.)

All of these problems can be fixed with no changes to our basic model.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, another problem with the system is that we have no allowance we have for the negative exponent. So, we have got this sign bit, but that is for the entire number, but suppose I have to represent say 2^{-3} . So, I do not have any of the any technique by which I can represent -3.

So, we have no way to express this 0.5. So, that is 2^{-1} why? Because this I cannot have I cannot tell that my exponent part is -1. So, I can tell that exponent part is 1, but I cannot tell it to be -1, there is no sign bit. So, these problems can be fixed with no changes to our basic model. So, we can keep this exponent both positive and negative.

(Refer Slide Time: 17:53)

Floating-Point Representation

- To resolve the problem of synonymous forms, we establish a rule that the first digit of the significand must be 1, with no ones to the left of the radix point.
- This process, called *normalization*, results in a unique pattern for each floating-point number.
 - In our simple model, all significands must have the form 0.1xxxxxxxxx
 - For example, $4.5 = 100.1 \times 2^0 = 1.001 \times 2^2 = 0.1001 \times 2^3$. The last expression is correctly normalized.

In our simple instructional model, we use no implied bits.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, how do we do this? We will see slowly. So, to resolve this problem of synonymous forms that is for the same number we have got multiple representation so, we have to have a rule ok.

So, some rule has to be followed and the rule is that, the first digit of the significand must be 1 with no ones to the left of the radix point because it is a binary representation. So, it is zeroes and ones. So, I will say that my significand can never start with a 0. So, if you just look into this slide. So, you see that this representation we are telling ok, but these three representations we are telling that they are not correct because my as per my convention. So, this significand part must start with a 1. So, it cannot start with a 0 ok.

So, that way we put we make a rule, that the first digit of the significand must be 1 and to the left of it to the and there will be no 1 to the left of the radix point. So, before radix point there is no digit or no bit, but after radix point we have always it always starts with a 1. So, this is called normalization ok. So, this process is called normalization like say 4.5.

So, 4.5 when you are representing so, this is 100.1×2^0 . So, we can tell it like this. So, it can also be like 1.001×2^2 it can also be 0.1101×2^3 . So, we will say that the, this representation is correct because it does not have any one to the left of the radix point, to the left of the radix point there is only 0.

Similarly, here to the left of the radix point we have 1. So, this representation is not correct, but this is correct. And in the right side, so, it, it starts with a 1. So, the in the here if you do. So, it is right side this starts with 0. So, it is not correct. So, here it is starting with a 1. Here also its starts the right side starts with a 1, but on the left side we have got some digits ok. So, that is not correct.

So, there should not be any digit to the left of the radix point. So, the only this representation is correct in as per our convention. So, all significands must have the form 0.1xxxx like that. So, its starts it must start with it the bit before decimal point should be 0, and after decimal point the first bit should be 1. Accordingly we have to adjust the exponent part ok. So, we can adjust the exponent part and that way we can do it.

So, in this model so, we have we use no implied bits ok. So, but it is it is 0 directly, but the implied bits we will see later. So, this is known as the process of normalization.

(Refer Slide Time: 20:46)

- To provide for negative exponents, we will use a *biased exponent*. \downarrow 16
- A bias is a number that is approximately midway in the range of values expressible by the exponent. \downarrow $+16$
We subtract the bias from the value in the exponent to determine its true value. \downarrow 16
- In our case, we have a 5-bit exponent. We will use 16 for our bias. This is called *excess-16 representation*.
- In our model, exponent values less than 16 are negative, representing fractional numbers. \downarrow 1111

Now, how do we go for these negative exponents ok? So, negative for the for the purpose of negative exponent. So, the technique that is used is known as biased exponent. So, a bias is a number that is approximately midway in the range of values expressible by the exponent, and we subtract the bias from the value in the exponent to determine its true value.

So, we have got in our example we have got 5 bit exponent. So, 5 bit exponent means. So, it can go from. So, if I am using negative numbers also. So, it can go from a -16 to +16. So, 5 bit is all the bits may be 1 ok. So, it may be 1. So, by so, I may be so, it is -16 to +16. So, this out of this total the in 5 bits. So, I can have 32 different patterns 32 patterns. So, 32 different exponents can be stored.

So, I can make it I can divide this range on the negative side minus 16 and the positive side say plus 16 and then what we do? We with the value with we stores add another 16 ok. So, we add another 16 what whatever be the result that is actually stored. So, this is called excess 16 representation. So, we will take an example that will make it clear.

(Refer Slide Time: 22:29)

See this one says we are again coming back to example of 32. So, 32 representation we have seen that we are following this notation 0.1×2^6 . So, in our previous case the, a significand part was storing this 1 and the exponent part was storing this 6, but we do not store 6 there. So, what we do with this 6 we add 16. So, that way it becomes 22. So, we will be storing 22 in the exponent.

So, here if you look into the all possible bit pattern so, you can if you look into sorry if you look into all possible bit patterns here, then it is all 0 to all 1 so; that means, I can go from 0 to 31. So, in that range what I do. So, I add 16. So, if I add 16, then this will become this 6 will become $16+6 = 22$. So, I stored the pattern of 22 here knowing that when we

are calculating the value. So, this is in this is. So, 16 extra has been added. So, I subtract 16 from here and I will get back 6 ok.

So, if I have a negative number if I have a negative number. So, the negative can go till say -16. So, that way that even 16 will be added to that. So, that number will become 0 the exponent will become 0. So, this way we do not have to store the negative exponents separately. So, in the following this excess notation so, this mid value will be added to the exponent and that will be stored. So, the negative numbers will also come as positive numbers only.

So, in the storage so, that is known as the biased exponent concept; so, exponent has been biased by the middle of the range. So, that everything becomes positive.

(Refer Slide Time: 24:35)

The screenshot shows a presentation slide titled "Example 2". The slide content includes a bulleted list of steps to express 0.0625₁₀ in a 14-bit floating-point model. The list points out that 0.0625 is 2⁻⁴, converts it to scientific notation as 1.0 × 2⁻⁴ = 0.1 × 2⁻³, and notes that using an excess 16 biased exponent requires adding 16 to -3, resulting in 13₁₀ (=01101₂). Below the list is a diagram of a 14-bit floating-point number: the sign bit is 0, the exponent field is 01101, and the fraction (mantissa) field is 10000000. At the bottom of the slide are logos for IIT Kharagpur and NPTEL, along with a small video thumbnail of a speaker.

So, next we have another example suppose it is 0.0625, in the in the decimal representation. So, this 0.0625 is 2⁻⁴ because, so, in so, this is 2⁻⁴. So, in the scientific notation that we are having so, this will be represented as 1 × 2⁻⁴ or in our notation. So, it is as per our rule. So, it is there should be only 1 digit after there should be no digit before decimal. So, this is 0.1 × 2⁻³.

So, this is the value that we are trying to store finally, in the computer for example, now what we do we take an excess 16 notation. So, with this -3, 16 is added. So, the value becomes 13. So, ultimately in the exponent part we store 13 we do not store -3, but we

store 13 knowing fully well that when we are looking back for the value. So, from this whatever is stored there. So, I have to subtract 16. So, it was excess 16. So, 16 was added while storing it. So, while getting it back. So, I have to subtract 16 and then only I can get it back ok.

So, that way I can do we I can go back to 2^{-3} and this part this significand part remains unaltered. So, this is known as the biased exponent concept.

(Refer Slide Time: 25:58)

The slide is titled "Example 3". It contains a bulleted list of steps to express -26.625_{10} in a 14-bit floating-point model. The list includes:

- Example:
 - Express -26.625_{10} in the revised 14-bit floating-point model.
- We find $26.625_{10} = 11010.101_2$. Normalizing, we have:
 $26.625_{10} = 0.11010101 \times 2^5$.
- To use our excess 16 biased exponent, we add 16 to 5, giving 21_{10} ($=10101_2$). We also need a 1 in the sign bit.

Below the list is a diagram showing the 14-bit binary representation. It consists of three colored boxes: a green box containing the sign bit '1', a yellow box containing the exponent '10101', and a pink box containing the significand '11010101'. A small cursor arrow is positioned below the boxes.

The footer of the slide includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video thumbnail of a person speaking.

Another example so, suppose and to we represent the number minus 26.625 in this 14 bit representation. So, since this is minus. So, this sign bit should be 1 now this 26.625. So, if you represent it in binary notation. So, it is like this 11010.101. So, we do normalization. So, after doing normalization so, it is like this. So, this is the format we want to store in the computer, but. So, this significand part is storing it 1 1 0 1 1 0 1, but for the exponent part for the exponent part. So, we take again excess 16.

So, with this 5 16 will be added. So, this becomes 21. So, this 21 is stored in the exponent part. So, this way we can store the numbers in the system.

(Refer Slide Time: 26:51)

Floating-Point Standards

- The IEEE has established a standard for floating-point numbers
- The IEEE-754 *single precision* floating point standard uses an 8-bit exponent (with a bias of 127) and a 23-bit significand.
- The IEEE-754 *double precision* standard uses an 11-bit exponent (with a bias of 1023) and a 52-bit significand.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now so, this is the standard that we were, we have developed the model that we have the 14 bit representation of model that we were using. So, IEEE has got some standards for this floating point numbers. So, there it is IEEE 754 standard and they have got one standard for single precision number and another standard for double precision number.

So, for 60 for a single precision number so, it uses 8 bit exponent with a bias value of 127 and a 23 bit significand; On the other hand this a double precision number, it has got 11 bit exponent with a bias of 1023 and 52 bit significand. So, that is a very, very huge number. So, that way we can represent very large numbers in double precision format with higher precision values.

(Refer Slide Time: 27:42)

Floating-Point Representation

- In both the IEEE single-precision and double-precision floating-point standard, the significant has an implied 1 to the LEFT of the radix point.
 - The format for a significand using the IEEE format is:
1.xxx...
 - For example, $4.5 = .1001 \times 2^3$ in IEEE format is $4.5 = 1.001 \times 2^2$. The 1 is implied, which means it does not need to be listed in the significand (the significand would include only 001).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, both single precision and double precision use the significand has an implied 1 to the left of the radix point. So, in our model to the left of the radix point we have only 0, but in case of this float this IEEE standard so, of to the left we have got a 1. For example, this 4.5 is actually this one, in our notation it will be like this, but in IEEE format so, it will be 1.001×10^2 because that will be so, the this 1 is implied. I do not need to store this 1 explicitly I in the in the storage I will only have this 001 in the in the significand part, but this 1 is implied.

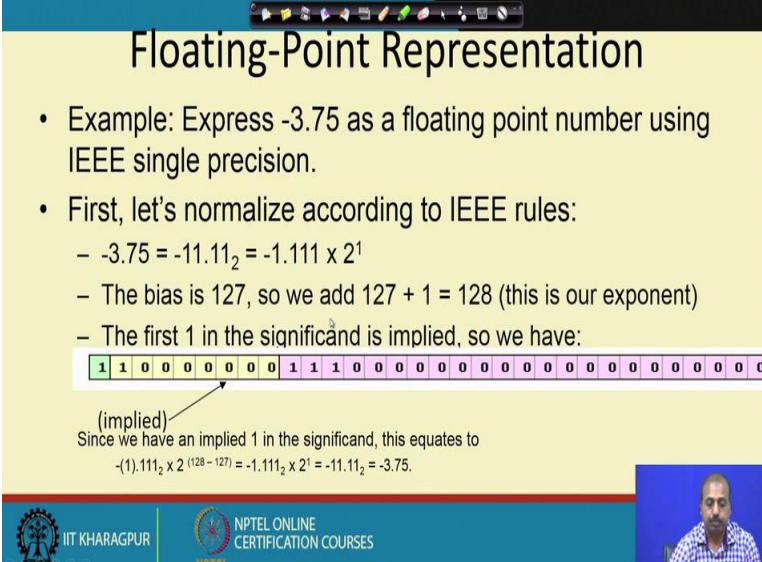
So that means, so, in our case what was happening is this 1 we were storing separately and, but that way we were actually wasting a bit ok. So, we were wasting 1 bit of storage, but using this 1.001×2^2 . So, we can we so, that we can say 1 bit ok. So, that way we can have a better representation.

(Refer Slide Time: 28:53)

Floating-Point Representation

- Example: Express -3.75 as a floating point number using IEEE single precision.
- First, let's normalize according to IEEE rules:
 - $-3.75 = -11.11_2 = -1.111 \times 2^1$
 - The bias is 127, so we add $127 + 1 = 128$ (this is our exponent)
 - The first 1 in the significand is implied, so we have:

Since we have an implied 1 in the significand, this equates to
 $-(1).111_2 \times 2^{(128-127)} = -(1).111_2 \times 2^1 = -11.11_2 = -3.75$.



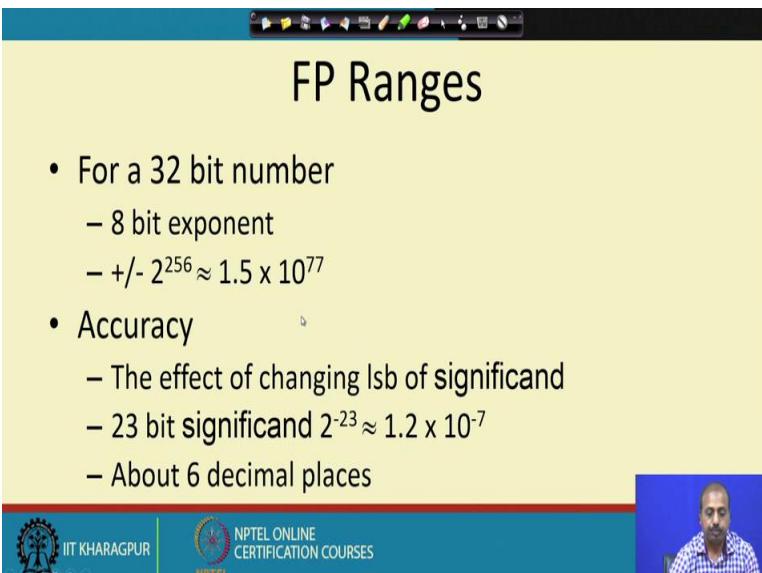
So, this for example, this -3.75 so, this floating point representation so, it will be like this. So, minus 3.75 is -11.11. So, -1.111×2^1 .

Now, this bias is 127. So, 127 is added to 1 so, it becomes 128. Now the sign is negative. So, the so, this is the sign now, this part is so, before decimal. So, this is this is the exponent. So, 127 and then we have got this implied 1 and after that we have got this 111000. So, this 1 is already implied. So, this 1 this 1 need not be stored separately.

(Refer Slide Time: 29:35)

FP Ranges

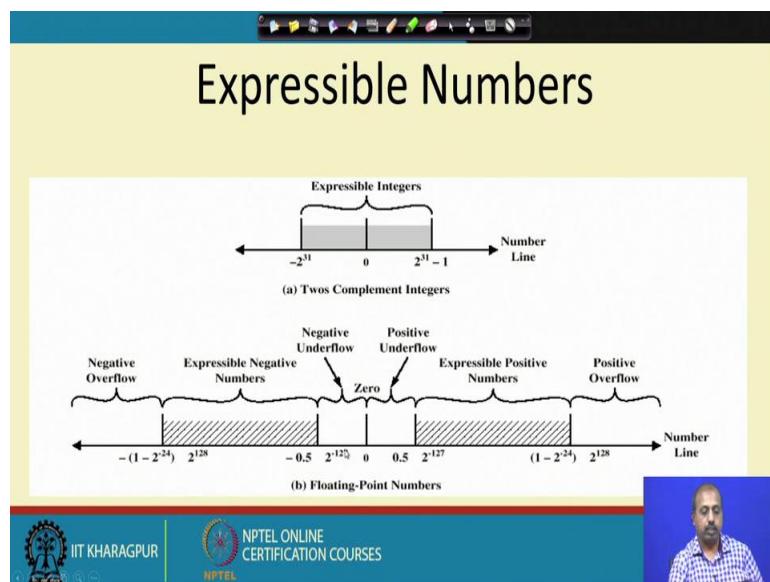
- For a 32 bit number
 - 8 bit exponent
 - $\pm 2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
 - The effect of changing lsb of significand
 - 23 bit significand $2^{-23} \approx 1.2 \times 10^{-7}$
 - About 6 decimal places



Floating point number range for 32 bit number with 8 bit exponent and $\pm 2^{256} = 1.5 \times 10^{77}$ a huge number.

Now, accuracy so, this is the change that will come if you change the LSBs of the significand part. So, if I have got 23 bit significand then the accuracy is 2^{-23} . So, that is 1.2×10^{-7} . So, that is up. So, if this turns out to be 6 places after decimal. So, that is a huge number accuracy is good.

(Refer Slide Time: 30:08)



The numbers that you can express so, that way you can see that in 2's complement range. So, it is -2^{31} to $+2^{31} - 1$, but in floating point numbers. So, you have got a huge range ok. So, this is $-(1 - 2^{-24})$ to $(1 - 2^{-24})$. So, that way so, they that is a huge number of numbers that you can store here ok.

(Refer Slide Time: 30:35)

The slide has a yellow header and a blue footer. The title 'Floating-Point Representation' is at the top. The main content is a bulleted list about IEEE-754 floating-point standards:

- Using the IEEE-754 single precision floating point standard:
 - An exponent of 255 indicates a special value.
 - If the significand is zero, the value is \pm infinity.
 - If the significand is nonzero, the value is NaN, "not a number," often used to flag an error condition.
- Using the double precision standard:
 - The "special" exponent value for a double precision number is 2047, instead of the 255 used by the single precision standard.

The footer contains the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. There is also a small video window showing a person's face.

So, similarly we can have this floating point representation for this single precision floating point. So, these are this representation can be there and this flow 0, 0 over this 14 bit model that we have used.

(Refer Slide Time: 30:38)

The slide has a yellow header and a blue footer. The title 'Floating-Point Representation' is at the top. The main content is a bulleted list about IEEE-754 floating-point standards:

- Both the 14-bit model that we have presented and the IEEE-754 floating point standard allow two representations for zero.
 - Zero is indicated by all zeros in the exponent and the significand, but the sign bit can be either 0 or 1.
- This is why programmers should avoid testing a floating-point value for equality to zero.
 - Negative zero does not equal positive zero.

The footer contains the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. There is also a small video window showing a person's face.

So, that is that will also have similar such thing, but this 0 with this IEEE 754 so, this allows two representation of zeroes.

So, we have got minus zero indicated by all zeroes in the exponent part and the significand part, but the sign bit can be either 0 or 1. So, as a result there is a +0 and a -0 in the in our

representation as well as IEEE 754 presentation. So, this suggests that we should not have this floating point we should not check whether the value is 0 or not, because of the sign bit mismatch. So, it may not be giving the correct result in a program. So, you should not try to check whether x equal to 0, where x is the floating point number and this negative 0 does not equal to positive 0.

(Refer Slide Time: 31:41)

The slide has a yellow background and a blue header bar. The title 'Floating-Point Representation' is centered in the header. Below the title is a bulleted list of four points:

- Floating-point addition and subtraction are done using methods analogous to how we perform calculations using pencil and paper.
- The first thing that we do is express both operands in the same exponential power, then add the numbers, preserving the exponent in the sum.
- If the exponent requires adjustment, we do so at the end of the calculation.

At the bottom of the slide, there is a blue footer bar with two logos: IIT Kharagpur on the left and NPTEL on the right. The NPTEL logo includes the text 'NPTEL ONLINE CERTIFICATION COURSES'.

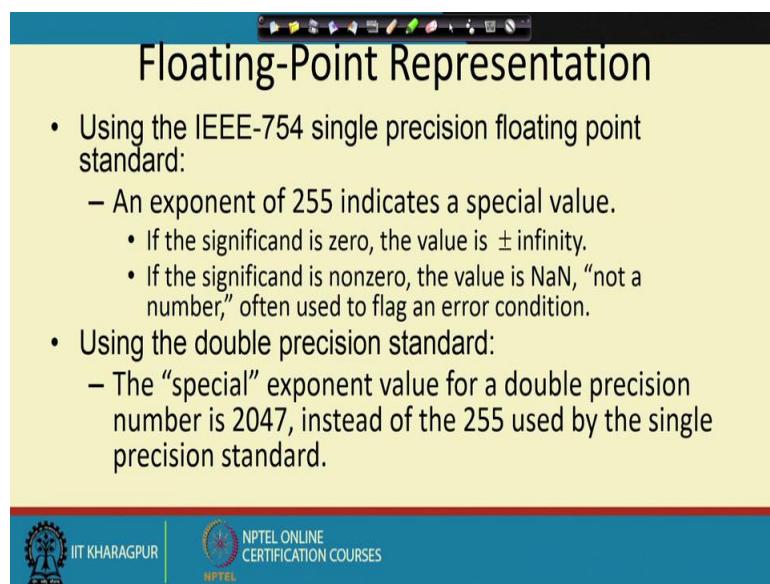
So, this floating point representation. So, it can we can do addition subtraction etcetera. So, that will be doing that we will do using a similar to integer arithmetic. So, this we will see in the next class.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 07
Number System (Contd.)

So, in the floating point representation there is one more point regarding these, these numbers that we are going to store.

(Refer Slide Time: 00:20)



The slide has a title 'Floating-Point Representation' and two main bullet points:

- Using the IEEE-754 single precision floating point standard:
 - An exponent of 255 indicates a special value.
 - If the significand is zero, the value is $\pm\infty$.
 - If the significand is nonzero, the value is NaN, "not a number," often used to flag an error condition.
- Using the double precision standard:
 - The "special" exponent value for a double precision number is 2047, instead of the 255 used by the single precision standard.

At the bottom, there are logos for IIT Kharagpur and NPTEL.

Like in IEEE 754 single precision floating point standard. So, an exponent 255 it indicates a special value. So, if the significand part is 0. So, this represent a $+\infty$ or $-\infty$ depending upon the sign bit ok. So, this has got a special representation of infinity.

On the other hand if the significand part is nonzero, then it means it is a not a number. So, many a time you might have seen while writing programs using floating point numbers. So, it prints the message that not a number NaN like that. So, that actually comes from here. So, if the number that you are getting is of this format that this the exponent part is 255, and the significand part is nonzero then it will be telling that it is a it is not a number. And if it is zero, if the significand part is zero then it is $+\infty$ or $-\infty$ depending upon the sign.

In the double precision format also the special exponent value for double precision number is 2047. So, instead of 255, it is 2047 otherwise it is same. So, if you are getting exponent 2047 then this has got a special meaning. So, this maybe if the significand is 0 then it is $\pm\infty$ depending upon the sign bit and if it is if the significand is nonzero, then it is not a number. So, this way this IEEE the standard so, it has kept special provision for representing infinity and this not a number type of flags.

(Refer Slide Time: 02:00)

Floating-Point Representation

- Both the 14-bit model that we have presented and the IEEE-754 floating point standard allow two representations for zero.
 - Zero is indicated by all zeros in the exponent and the significand, but the sign bit can be either 0 or 1.
- This is why programmers should avoid testing a floating-point value for equality to zero.
 - Negative zero does not equal positive zero.

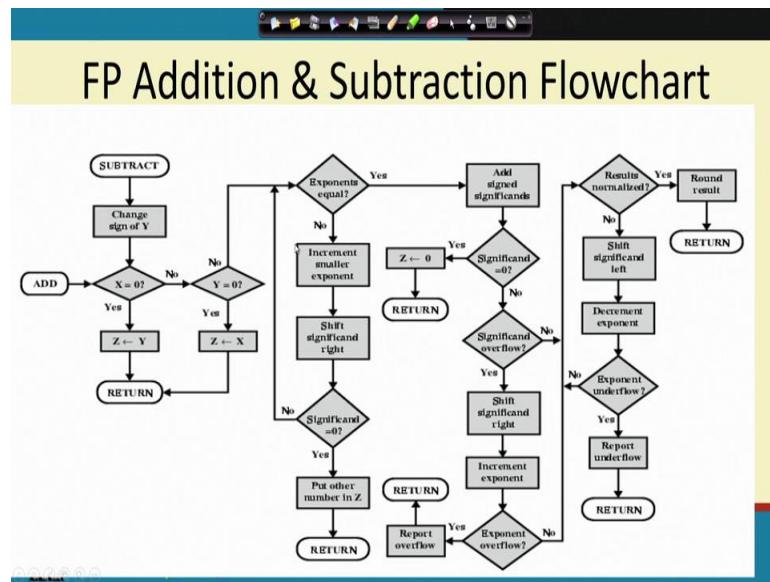
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, another caution that we have discussed in the last class that, both in our model as well as IEEE. So, there is +0 and -0 that representation is possible. So, we should be careful while writing programs and trying to check between whether a variable **b** has become 0 or not because this plus 0 and minus 0 they are not same.

So, you can get some surprising results because of that either the check may be successful or the check may be fail, maybe a failure though in both the cases the value is 0 only ok. But the x equal to 0 in some cases that check may be very check may come out as pass some cases, it may come out as a fail. So, we have to be careful.

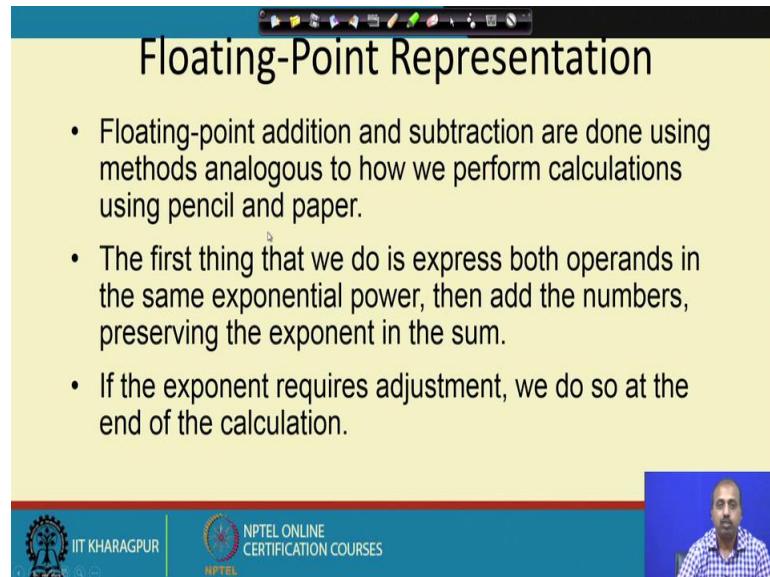
So, maybe we will do some other check ok. So, it will normally what we do is it is less than some very, very small quantity. So, maybe I can I will write like $x < 0.000000000001$. So, that way it is a very small number. So, it is it may be considered to be equal to 0; the absolute value of x of course.

(Refer Slide Time: 03:14)



Next we look into how do we do this addition and subtraction in the floating point format.
So, floating point addition and subtraction

(Refer Slide Time: 03:19)



So, we use methods which are analogous to our paper pencil method. So, first of all the operands must have same exponent otherwise you cannot do the addition subtraction. So, the exponent has to be preserved and the after doing the addition and subtraction. So, we may we may need to normalize and that way the exponent may be changing after the

addition or subtraction. So, if the exponent requires adjustment so, we do so, at the end of the calculation. So, this is the process.

Suppose I am doing trying to do an addition of $X + Y$. So, if X equal to 0 then the answer is Y directly so, we go back. If the $X \neq 0$ then we check whether Y equal to 0 or not. If $Y \neq 0$ then we, if Y is equal to 0 in that case the answer is Z . So, it simply goes out otherwise X and Y both are nonzero. So, it comes to this point and it checks whether the exponents are equal or not.

So, if the exponents are equal, I don't need to do any exponent adjustments. So, I just simply add the add sign significands and if the significand becomes 0, then the answer is 0. If the significand is not is nonzero then there may be a possibility that significand there is an overflow or there is no overflow in the significand with contain containing the number of bits.

If there is no overflow then we have to check whether the result has been normalized or not that is after decimal point, the first digit should be 1 and all before decimal point there should not be any digit. So, all those rules so, it is checking those rules and accordingly if the results are not normalized, then we have to shift the significand towards left, decrement the exponent part and that way this if there is it resulting exponent underflow. So, if it is not so, we and it is result is not yet normalized. So, we may just shift by a number of bits the significant part till the exponent and go on decrementing the exponent.

So, till the exponent becomes this significand is correct as per our rule. So, ultimately when this everything is correct, then there we have far we have got the result then we go then we round up round our do a rounding operation of the result and then it returns. And otherwise if there is an if there is an overflow in the significant part. So, you have to shift the significant right increment the exponent part and if there is no exponent overflow, then we will go for result normalization and go back.

And if this exponent part also overflows; that means, after doing this addition both significant and exponent they have over-flown. So, we cannot represent this addition the added addition result into the given format. So, it will report overflow and it will return. So, there is an overflow in the addition process.

Similarly, for the subtraction part so, we just change the sign of Y, and then rest of the thing is same as we are doing for the addition part. So, this is simply pen and paper type of addition subtraction. So, that is followed here also.

(Refer Slide Time: 06:39)

Floating-Point addition example

- Example:
 - Find the sum of 12_{10} and 1.25_{10} using the 14-bit "simple" floating-point model.
- We find $12_{10} = 0.1100 \times 2^4$. And $1.25_{10} = 0.101 \times 2^1 = 0.000101 \times 2^4$.
- Thus, our sum is 0.110101×2^4 .

+	0	1	0	0	1	0	0					
	0	1	0	0	1	0	0					
	0	1	0	0	1	0	0					

So, suppose we are trying to do this so, $(12)_{10}$ and $(1.25)_{10}$. So, this we are adding using 14 bit simple floating point model that we have. So, 12 is represented as in our normalized model say it is 0.1100×2^4 and $1.25 = 0.101 \times 2^1$.

So, I have to make both of their exponents same so, we take both of them 2 exponent 4 larger one definitely. So, after that so, when this is converted into 2^4 . So, this becomes the representation knows now we add this significant with this significant. So, getting the significant part as this so, our overall sum becomes 0.110101×2^4 . So, here no adjustment is necessary because before decimal we have 0 and after decimal we have one. So, no adjustment is necessary.

(Refer Slide Time: 07:35)

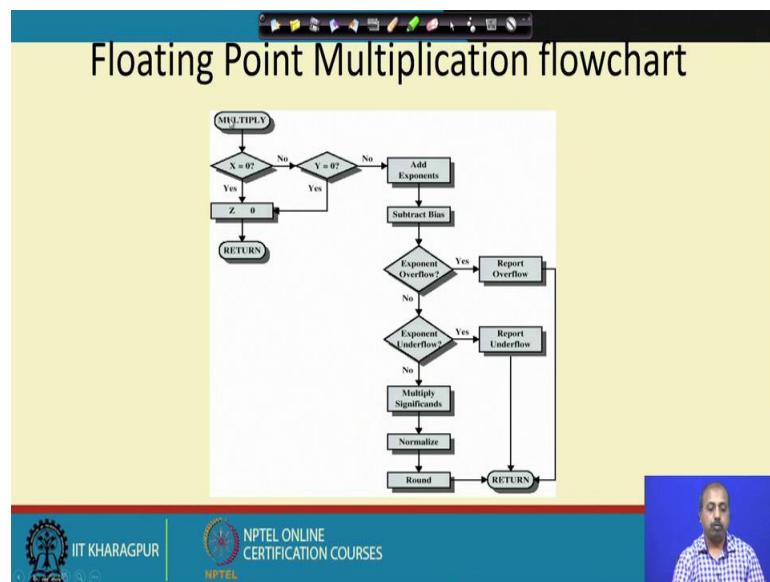
Floating-Point Multiplication

- Floating-point multiplication is also carried out in a manner akin to how we perform multiplication using pencil and paper.
- We multiply the two operands and add their exponents.
- If the exponent requires adjustment, we do so at the end of the calculation.

IIT Kharagpur | NPTEL Online Certification Courses

So, in case of multiplication so, we do similar thing so, by this is also like say paper and pen a paper and pencil type of multiplication that we do. So, we multiply the operands and add the exponents. So, significant parts are multiplied and the exponent parts are added. So, if after that we you or you need to do some adjustment with the exponent. So, we do that adjustment after the calculation.

(Refer Slide Time: 08:00)



So, this is the multiplication routine. So, we multiply so, first check whether X is equal to 0 or not, if X equal to 0 then we say that we check whether we say that the result Z is also

equal to 0 or if Y equal to 0 then also we say that the result Z is equal to 0 and it returns from there.

And if both are nonzero then we add the exponents because while doing the multiplication the exponents will get added. So, we add the exponent and we subtract the bias and if there is an exponent overflow so, we report overflow. If there is no exponent overflow then we check whether exponent under flows or not, in that case we will report an underflow. And that is not the case then we multiply the significands, they do some normalization and then do rounding and ultimately it return. So, that way this floating point multiplication can take place.

(Refer Slide Time: 08:58)

Rounding and Errors

- No matter how many bits we use in a floating-point representation, our model must be finite.
- The real number system is, of course, infinite, so our models can give nothing more than an approximation of a real value.
- At some point, every model breaks down, introducing errors into our calculations.
- By using a greater number of bits in our model, we can reduce these errors, but we can never totally eliminate them.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you can take a paper pencil type of example and see how to do this thing, and I hope you are already familiar with these multiplications from the school days; So, instead of taking the base 10 so, it is taking base 2, that is the only difference otherwise the rules are same.

So, another very important issue with this floating point representation is the rounding and errors. So, no matter how many bits we use in a floating point representation. So, our model is finite. So, as I was telling that there is this floating point numbers they can go up to infinity. So, this after the decimal point how many digits you are keeping. So, that way it can go up to infinity. So, that way there will be restriction. So, we will have to be careful and then when there is a restriction. So, we will introduce some amount of error.

So, real number system is infinite. So, our models can give a nothing more than an approximation of the real value. So, at some point every model breaks down. So, whatever be the size the that we allocate for the exponent part and the significand part. So, there is there will be a limit. So, it will introduce some error in the calculation. So, if you use more number of bits, then we will can reduce error, but we can never eliminate it totally. So, these errors will remain it will may not go.

(Refer Slide Time: 10:22)

Rounding and Errors

- Our job becomes one of reducing error, or at least being aware of the possible magnitude of error in our calculations.
- We must also be aware that errors can compound through repetitive arithmetic operations.
- For example, our 14-bit model cannot exactly represent the decimal value 128.5. In binary, it is 9 bits wide:
 $10000000.1_2 = 128.5_{10}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES
NPTEL



So, how to reduce the error? So, we will try to reduce the error or we should be while do while doing arithmetic with this floating point. So, we should be and we should be having an idea like how much error may get introduced into my calculation and another thing is that, as we are doing some operation repetitively.

So, this may add this may go increase the error part ok. For example, if we are adding all numbers all floating point numbers in an array then as we are adding successive numbers or successive entries of the array. So, this the error also grows.

So, in our 14 bit model we cannot exactly represent the value 128.5. So, in binary it is 9 bit wide so, it is like this. So, in case of floating point representation what will happen is, I have to represent it as though this 0.1 something and then 2 to the power some value, but that it will not be possible to represent in 14 bits.

(Refer Slide Time: 11:35)

Rounding and Errors

- When we try to express 128.5_{10} in our 14-bit model, we lose the low-order bit, giving a relative error of:

$$\frac{128.5 - 128}{128.5} \approx 0.39\%$$

- If we had a procedure that repetitively added 0.5 to 128.5, we would have an error of nearly 2% after only four iterations.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, what is what will happen is that so, if you represent we try to represent 128.5 in our 14 bit model we will lose the lower order bit. So, it will become 128 only. So, this 0.5 will not be coming, because the last bit will be ignored it will not have so much of space. So, the error that is introduced is about 0.39 percent. So, if we if we. So, so just for representing the number, we are losing 0.39 percent.

So, if we if the go on repeating the process by adding 0.5 to 128.5. So, just we go on adding 0.5 to this again and again, then within 4 iterations. So, it will introduce about 2 percent error in the result. So, that will come because of every time we are doing it so, it is not representable in this system. So, as a result it will introduce some error into the system. So, that is that is how so, these errors can creep in floating point computations.

(Refer Slide Time: 12:35)

Rounding and Errors

- Floating-point errors can be reduced when we use operands that are similar in magnitude.
- If we were repetitively adding 0.5 to 128.5, it would have been better to iteratively add 0.5 to itself and then add 128.5 to this sum.
- In this example, the error was caused by loss of the low-order bit.
- Loss of the high-order bit is more problematic.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, floating point errors can be reduced when we use the operands that are similar in magnitude; so, that is one possibility and if we are for example, if we are repeatedly adding 0.5 to 128.5. So, it would be better to iteratively add 0.5 to itself and then add 128.5 to the sum. So, what it says is that since every time we were adding 0.5.

So, you add this 0.5 separately how many whatever be the number of 0.5 is you need to add with 128.5. So, many 0.5's are added together and then the resulting sum is added to 128.5. So, in this way while you are doing this 0.5 additions so, error will not be introduced. So, it is possible that while adding with 128.5 the resulting number. So, the some error will be introduced, but that will be introduced only once. So, that way the so, if the error caused so, the error possibility will reduce.

So, in this particular example so, this error when we are adding 1.5 to 128.5 we lost the lower order bit, but if the values are higher, then we may start losing the higher order bits as well and as a result we can a the situation may become more problematic. So, it may become the number that we are representing maybe far away from the actual number.

(Refer Slide Time: 14:00)

Rounding and Errors

- Floating-point overflow and underflow can cause programs to crash.
- Overflow occurs when there is no room to store the high-order bits resulting from a calculation.
- Underflow occurs when a value is too small to store, possibly resulting in division by zero.

Experienced programmers know that it's better for a program to crash than to have it produce incorrect, but plausible, results.

EVEN BETTER, CAUGHT THE ERROR and DEAL WITH IT!

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, floating point overflow and underflow can cause programs to crash ok so, this is there. So, though it is a bit of programming side, but it is important because if the if there is an underflow the number becomes 0 and then when you are doing some division for example, by that number. So, it is a divided by 0 error. So, as a result the program may crash.

Similarly over flow also it may go out of range as a result it may generate some wrong values and then the program may crash. So, this overflow and underflow so, these are the two cases and if we are whenever we are using this floating point representation so, we should be careful. Like if you are writing a program, then it should be careful that this type of cases are caught, and then we avoid the operations using those values, and come up with say some sort of error message ok. So, that may the program does not crash or the system does not crash.

(Refer Slide Time: 15:05)

The screenshot shows a presentation slide titled "Rounding and Errors". The slide content is as follows:

- When discussing floating-point numbers, it is important to understand the terms *range*, *precision*, and *accuracy*.
- The range of a numeric integer format is the difference between the largest and smallest values that can be expressed.
- Accuracy refers to how closely a numeric representation approximates a true value.
- The precision of a number indicates how much information we have about a value

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". To the right of the footer, there is a small video window showing a man speaking.

So, these floating point numbers. So, we have to there are three important terms with respect to floating point numbers the range, that is the range of values it can represent the precision, with what precision it can do it and the accuracy level. So, range is simple.

So, range of any integer format is the difference between the largest and smallest values that can be expressed. In case of accuracy it refers how closely a numeric representation approximates a true value and this precision means, how many the number how much of information we have about a value. So, apparently it seems that if the precision is higher than the accuracy will also be high, but that is not always the case.

(Refer Slide Time: 15:50)

The screenshot shows a presentation slide titled "Rounding and Errors". The slide content includes a bulleted list of points about floating-point precision and operations:

- Most of the time, greater precision leads to better accuracy, but this is not always true.
 - For example, 3.1333 is a value of pi that is accurate to two digits, but has 5 digits of precision.
- There are other problems with floating point numbers.
- Because of truncated bits, you cannot always assume that a particular floating point operation is commutative or distributive.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video thumbnail of a person speaking.

For example suppose we are representing the number pi and we have stored the number as 3.1333. Now, you see that value of pi that we have here is accurate only up to two places after decimal ok, but while storing it I am storing value which has got 5 digits of precision. So, 3 and this 1333 so, total 5 digits are stored. So, that way the amount of accuracy and the amount of precision, they are not necessarily same ok. So, they may be different.

So, there are other problems in floating point numbers also because when you truncate the bits, you cannot always assume that a particular floating point operation is commutative or distributive; like what do you mean by if I have an operation say $x + y$ ok.

So, this addition operation so, in general $x + y = y + x$. So, this is a commutative operation or $x(y+z) = xy+xz$. So, the first one is the commutativity property, second one is the distributive property. So, these operations addition and multiplication say they follow this type of properties.

So, for integers it is true, but for floating point numbers. So, it may or may not be true. So, in reality it may not happen because of this accuracy and all ok. So, some of the bits may be truncated. So, you so, we cannot always assume that $x + y$ will always be equal to $y + x$ or in particular $x(y+z)=xy+xz$. So, that is another very problematic situation. So, while handling this floating point number. So, you have to be careful with them.

(Refer Slide Time: 17:45)

The screenshot shows a presentation slide titled "Rounding and Errors". The slide content includes:

- This means that we cannot assume:
$$(a + b) + c = a + (b + c) \text{ or}$$
$$a * (b + c) = ab + ac$$
- Moreover, to test a floating point value for equality to some other number, it is best to declare a "nearness to x" epsilon value. For example, instead of checking to see if floating point x is equal to 2 as follows:
if $x == 2$ then ...
it is better to use:
if $(\text{abs}(x - 2) < \text{epsilon})$ then ...
(assuming we have epsilon defined correctly!)

The footer of the slide contains logos for IIT Kharagpur and NPTEL, along with a small video thumbnail of a person speaking.

So, for example, so, this $(a+b)+c=a+(b+c)$. So, if you are doing this addition first and then adding c with it, in another case you are doing $b + c$ first and then adding a with that then or. So, this is the distributive property or say $a(b+c)=ab+ac$ as I was telling so, here this for floating point number. So, we cannot assume this type of equality is to hold always; like if you are do a plus b first.

So, the result may have may have some got some truncations with that c is added. So, on the other hand when I am being b plus c first so, this gives another this may result in some other type of truncation, with that when a is added the final results may not match of that may be between the two cases ok.

So, this another important thing is that when you are checking the value of a floating point number. So, we have to say like say we want to check say x whether x is equal to 2 or not where x is a floating point number. So, this check may not be correct because this the 2 the number 2 may not be represented with same level of accuracy as x ok.

So, as a result what will happen is that, even if the mathematically this x and 2 should be same, but in the represent from the representation point of view the x and 2 their values may be different.

So, while if you put it in a program. So, you may find that it is creating some problem. So, it is better that you write it like this the $|x - 2| < \epsilon$ so, where ϵ is a very, very small

positive quantity. So, we find out the difference between x and 2 take the absolute of that. So, it becomes positive and then we check, whether it is less than epsilon or not.

So, this way so, normally in floating point problems instead of writing it like this. So, we have to write it in this format to avoid this type of problems ok.

(Refer Slide Time: 19:50)

The slide has a yellow background and a blue header bar with various icons. The title 'Some Exercises' is centered at the top. Below the title is a bulleted list of four questions:

- If $(75)_x = (45)_y$, then possible values of x and y are
- Possible values of x and y satisfying $(2x)_5 = (3y)_6$ are
- If $x_6 = (32)_9 * (24)_5$ then x is equal to
- The number 267_8 in base-11 number system is

At the bottom, there is a red bar containing the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window showing a person's face.

So, next that finishes our discussion on this floating point this number system. So, next we will be doing a few exercises which are very interesting. So, the first one is says that I have got 2 numbers 75 and 45. Out of this 75 is a base x number and 45 is a base y number. So, what are the possible values of x and y . So, if we want to do this.

(Refer Slide Time: 20:19)

$(75)_x = (45)_y$ ✓

$\Rightarrow 7x + 5 = 4y + 5$

$\Rightarrow 7x = 4y$

$x=4 \quad y=7$ X

$x=8 \quad y=14$ ✓

So, let us take this one so, $(75)_x = (45)_y$ where this 7 5 so, they are the digits of that number system and they are taken they are numeric values are taken to be equal to decimal 7 and decimal 5 ok. So, if I just convert it so, it becomes $7x + 5 = 4y + 5$. So, that gives me the condition that $7x = 4y$. fine.

So, how can I satisfy? So, any value of x and y that satisfies this equality so, that is good enough for the mathematical point of view. So, we can say that I will choose x equal to 4 and y equal to 7, x equal to 4 and y equal to 7 that satisfies this condition. But this is not a correct result because if you look into the number here 75 x so; that means, this x must be more than 7, otherwise the digit 7 is not defined for x. Because if I have got a base a d number system, then the digits are 0, 1, 2 up to $d - 1$. So, if 7 is the base then I will have the digits 0, 1, up to 6.

Similarly, for this if so, this x equal to 4, I cannot get the digit 7 there; similarly if y equal to 7, y equal to 7 is of course, all right from the point of view. So, this is fine. So, this does not satisfy the relation. So, what else can be satisfied? So, we can have this the next one so, I can take x equal to 8 and y equal to 14. So, if I take x equal to 8 and y equal to 14, then it is fine ok. So, then this relation is correct. So, we have to so, this is the correct value for this particular problem.

(Refer Slide Time: 22:25)

Some Exercises

- If $(75)_x = (45)_y$, then possible values of x and y are
- Possible values of x and y satisfying $(2x)_5 = (3y)_6$ are $\cancel{(2x)_5 = (3y)_6}$
- If $x_6 = (32)_9 * (24)_5$ then x is equal to $(24)_5 = (34)_6$
- The number 267_8 in base-11 number system is

$$\begin{array}{rcl} 10+x=18+y & & \\ \cancel{4} \quad \Rightarrow \cancel{x-y=8} & x=9 & = 10+4 \\ & \cancel{4-y=8} & y=1 \\ & & \end{array}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we will take another example say this one so, $(2x)_5 = (3y)_6$. So, by a similar logic so, it is 2 into the number is $10 + x = 18 + y$. So, you get a condition that x minus y should be equal to 8. So, if I put any value of x and y such that $x - y = 8$ then we are done. So, can I take say x equal to 9 and y equal to 1.

So, that satisfies this relationship, but the problem is this one. So, this y should be at least y , y should be less than 6 and this x , x cannot be 9. So, x cannot be 9 so, x has to be less than 5. So, x has to be less than 5.

So, naturally I cannot have any relation that satisfies this particular property. So, I cannot choose any value of so, if I choose y equal to 1 then I will require x equal to 9. So, which is not set possible so, I can choose x at most equal to 4 fine. So, x at most since, this is 5 so, I can have x to be at most equal to 4.

So, if I put x equal to 4 ok so, $4 - y = 8 \Rightarrow y = 4$. So, that is a possibility. So, I take x equal to 4 and y equal to 4. So, $(24)_5 = (34)_6$ is it correct. So, 10 + 4 sorry then of course, so, this $x - y$ no sorry this is not correct because this is this 4 - y. So, y is becoming - 4.

So, the so, I cannot choose any value of x and y which satisfies this relationship. So, this I cannot get any satisfying value for this problem.

Then this one the third problem is simple. So, here we have got $(32)_9 = (24)_5$. So, you multiply them and then so, for multiplication purpose. So, we have to we can first convert

both of them to their decimal values do the multiplication, and then convert it into a base 6 number system. So, you have to you have to go on dividing the number by 6 and see what are the remainders and then the remainders you can represent it like the.

(Refer Slide Time: 25:15)

Some Exercises

- If $(75)x = (45)y$, then possible values of x and y are
- Possible values of x and y satisfying $(2x)_5 = (3y)_6$ are
- If $x_6 = (32)_9 * (24)_5$ then x is equal to
- The number $\underset{8}{\overset{2}{\underset{\sim}{\overset{6}{\sim}}}} \underset{8}{\overset{7}{\sim}}$ in base-11 number system is

$$\begin{array}{r} 183 \\ \times 11 \\ \hline 183 \\ -183 \\ \hline 0 \end{array}$$

$$2 \times 64 + 6 \times 8 + 7$$

$$= 128 + 48 + 7$$

$$= 183$$

$$\begin{array}{r} 183 \\ \div 11 \\ \hline 16 & 7 \\ -11 \\ \hline 5 \\ -5 \\ \hline 0 \end{array}$$

$$(157)_{11}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then the fourth one this $(267)_8$ in the base 11 number system so, here first of all these $(267)_8$. So, this has to be converted into decimal number system. So, this is $2 \times 64 + 6 \times 8 + 7 = 128 + 48 + 7 = 183$

Now, this 183 if you divide by 11 so, it is 1 then 7 so, 6 so, 66 then this is 73. So, this is your 7 then again by 11. So, you get 1 and 5 and then 11, 0 and 1. So, $(157)_{11}$ number system. So, this way you can convert numbers from one number system to another using this division, okay.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 08
Boolean Algebra

So, we will now start with another algebra method which is known as Boolean algebra. So, this is the algebraic systems that are used in computer systems and particularly in the digital circuits. So, which is this Boolean algebra; so this was this was invented longer, before these digital computers came into existence, but these are. So, previously it was only for say mathematical interest, but after these digital circuits came so this Boolean algebra got a new life. So, that way it is being used very much in this digital systems computer and other digital accessories.

(Refer Slide Time: 01:00)

The slide has a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the word "Introduction" is written in a large, bold, dark red font. A bulleted list follows:

- Developed by English Mathematician *George Boole* in between 1815 - 1864.
- It is described as an *algebra of logic* or an *algebra of two values* i.e *True* or *False*.
- The term *logic* means a statement having binary decisions i.e *True/Yes* or *False/No*

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is the IIT Kharagpur logo and the text "IIT KHARAGPUR". Next to it is the NPTEL logo with the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer, there is a small video player window showing a man speaking.

So, to introduce this was developed by English mathematician George Boole, in between 1815 and 1864. So, you can see that it was long before these digital circuits came into existence. So, it is described at the algebra of logic or an algebra of 2 values, true or false that is why it is so. So, it is; so the variables that we have in any algebra when you say about any algebra. So, there are certain things that we have to talk about like what are the values that variables on this algebra can take up and what are the operations that we can do on those variables ok.

So, in case of like when he was normal algebra that we are familiar with so the variables they can take up some values from some domain, may be integer, real etcetera. And then we can do some operation the addition, subtraction, multiplication, division type of operations on those values.

Similarly, here in case of Boolean algebra; it if the values are only true or false there are only two values possible true or false. Here the term logic means the statement having binary decision, it is, it may be true or yes and false or no. So, either the statement may be true or false or similarly some cases we may call the true as yes and false as no. So, this is the, this is the basic idea behind Boolean algebra.

(Refer Slide Time: 02:27)

The slide has a yellow background and a blue header bar. The title 'Application of Boolean algebra' is centered in a blue font. Below the title is a bulleted list of four points. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

- It is used to perform the logical operations in digital computer.
- In digital computer **True** represent by '**1**' (**high voltage**) and **False** represent by '**0**' (**low voltage**)
- Logical operations are performed by logical operators. The fundamental logical operators are:
 1. **AND** (**conjunction**)
 2. **OR** (**disjunction**)
 3. **NOT** (**negation/complement**)

So, where it is the application of Boolean algebra, so it used to perform logical operation in digital computers and in fact, in digital circuits we are doing. In digital circuits we are doing it. And this digital computer in case of digital computer digital circuits a true is, true is represented by a 1 often it is a high voltage or false is represented by a 0 which is low voltage. Of course, this is high and low voltage. So, these terms are not very much correct, because as we have seen previously that it may be that logic high is a negative voltage also may be taken as logic high, and positive voltage may be taken as logic low. But for the sake of understanding we can say that when it is high voltage, so this is 1 and low voltage is 0. So, it is logic high and logic low.

So, logical operations that perform that on this by some logical operators and there are some fundamental logical operators that are there in Boolean algebra: one is called AND or conjunction, OR or disjunction and NOT which is negation or complement. So, these are the three fundamental operations that you have in Boolean algebra. Of course, there are many other derived operations like say; NAND, NOR, XOR etcetera XNOR and all. And also you can think about any operation that that can be built around these basic operations, and that way we can get some newer operations in the Boolean algebra.

(Refer Slide Time: 04:00)

AND operator

- It performs logical multiplication and denoted by (.) dot.

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES NPTEL

So, the first simple operation is the AND operation. So, it is often denoted by this dot symbol. So, if X and Y are two Boolean variables their AND is denoted by $X \cdot Y$. Many times this dot is not written explicitly where it is implied that the operator is AND. So, we just write $X Y$ meaning that it is $X \cdot Y$ or X AND Y.

Now, if you say that in case of AND, the operation is defined like this; so if you take X equal to 0 and Y equal to 0 then there AND operation is also, ANDed value is also 0. If X is 0 and Y is 1, then this is 0 the result will be 0. The 1 and 0 if X equal to one and Y equal to 0, then the result will be 0, when both are 1 X and Y both are 1, then the result will be 1. So, this is the, this is a binary operator because it takes two variables and does the operation on that ok. So, that two operands and two operand operator.

So, it takes two operands X and Y and depending upon the operands value, the result is produced. And this is the rule: if both the operands are 1 in that case only the output is 1 otherwise the output is 0.

(Refer Slide Time: 05:24)

OR operator

- It performs logical addition and denoted by (+) plus.

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

AND $\Rightarrow \&, \wedge, ;,$
OR $\Rightarrow +, \vee, /$
Disjunction

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next, the other another fundamental operator which is known as the OR operator; so this is like this- that if we have got both the operands as 0, then only the result is 0 and otherwise the result is 1. So, this is 0 OR 1 is 1, 1 OR 0 is 1, 1 OR 1 is 1. So, this logical OR operator is often denoted by a + symbol. Of course, there are many other notations like for this AND for this AND we have got for this AND we have got notations like this, then we have got notation like this Λ ok. So, these are the alternate notations that we have.

Similarly, for OR we have got notations like say + that we have seen and the dot is definitely there, and another one is there is a no symbol. So, the no additional symbols. So, the variables coming just one after the other so that is also AND operation. Now OR for OR we have got +, we have got this particular symbol, which is known as disjunction. So, this is called conjunction and this is called disjunction. So, this is disjunction and this is called conjunction, conjunction. So, this is there and sometimes we represent it by a vertical bar. So, that is also OR. So, there are many notations. So, you may find different notations at different places all of them in the same thing all of them are equally useful and equally valid.

(Refer Slide Time: 07:16)

The slide has a yellow background with a dark blue header bar at the top containing various icons. The title 'NOT operator' is centered in a large, bold, dark red font. Below the title is a bulleted list:

- It performs logical negation and denoted by (-) bar. It operates on single variable.

On the left, there is a table:

X	\bar{X}	(means complement of x)
0	1	$\bar{x}, \sim x, !x, \neg x$
1	0	

On the right side of the slide, there is a small video window showing a man speaking. Below the video window, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

Now, so, this is the OR operator. For the NOT operator, so, it is like this that if it is a unary operator. So, it takes only one operand and does the operation on that. So, this is denoted by a by a bar over the symbol. So, this is X if X is the variable its compliment is denoted as \bar{X} , and if X is 0 then \bar{X} is 1. If X is 1, \bar{X} is 0. So, this is just the complement of X .

So, again there are other notations also like we have got the notation say. So, \bar{X} is one notation that you have already seen here. So, we sometimes we write it as $\sim X$, sometimes we write it as NOT of x . So, sometimes we write it as X' . So, these type different notations are there. So, all these notations in our lecture also many a times will be interchangeably be using these notations. So, meaning all of them in the same thing. So, that is the NOT operator. So, AND OR NOT; so these are the 3 fundamental operations that we have in Boolean algebra.

(Refer Slide Time: 08:30)

Truth Table

- Truth table is a table that contains all possible values of logical variables/statements in a Boolean expression.

No. of possible combinations = 2^n , where n=number of variables used in a Boolean expression.



Next we introduce a concept called Truth Table. So, truth table it is a table that contains all possible values of logical variables or statements in a Boolean expression. So, how does it look like? So, number of. So, if there are n number of variables then there are 2^n possibilities like.

(Refer Slide Time: 08:54)

Truth Table

- The truth table for $XY + Z$ is as follows:

Dec	X	Y	Z	$X:Y$	$XY+Z$
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	0
3	0	1	1	0	1
4	1	0	0	0	0
5	1	0	1	0	1
6	1	1	0	1	1
7	1	1	1	1	1

2^n possibilities



So, like here suppose we have got a Boolean function $X Y + Z$. So, it is like this; so, so this x y and z. So, these are the inputs to the function now if you enumerate the alternatives

that you can have here. So, it can go from 0 0 0 to 1 1 1. So, that way since this is a 3 variable, I have got 8 different alternatives. So, 000 to 111.

Now, if we look into the term $X Y$ or $X \cdot Y$. So, it is an AND of X and Y. So, I can write down this part. So, this is 0, y is 0, similarly this is also 0. So, only when x and y both are one this xy part becomes 1 and $xy + z$. So, this z is also there. So, it is whenever x y is 1 or z is 1 the $xy + z$ is 1. So, I can say. So, here z is 1. So, this is also 1, at this point xy is 1, but z is 0, but we have got this is $xy + z$ to be equal to 1. Similarly here both xy and z both are 1. So, we have got $xy + z$ equal to 1.

So, this way in a truth table, so you can. So, this is slightly extended version of the truth table, in normally we will be, we will be writing say this part and we will be writing the this particular columns only these 2 columns will be rewritten. So, this x y z and the output part which is $xy + z$ in our case. So, normally we write it like this and there are the there are 3 variables here.

So, I will have 2^3 that is 8 such rows there, in general if there are n variables then there will be 2^n such rows. So, this is the truth table. So, truth table as the name suggests. So, these tells what when the function assumes a true value and when the function assumes a false value, when the input variables are assigned different combinations of values ok.

(Refer Slide Time: 11:15)

Tautology & Fallacy

- If the output of Boolean expression is always True or 1 is called Tautology.
- If the output of Boolean expression is always False or 0 is called Fallacy.

P	P'	output ($P + P'$)	output ($P \cdot P'$)
0	1	1	0
1	0	1	0

$P+P'$ is Tautology and $P \cdot P'$ is Fallacy

$\begin{array}{c} f \rightarrow [+] \\ \bar{f} \rightarrow [\cdot] \\ P \rightarrow [\wedge] \\ \bar{P} \rightarrow [\vee] \end{array}$



IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES



So, next we will be coming across 2 terms one is called tautology, another is called fallacy. So, sometimes what happens is that, this out outcome of a Boolean expression is always true it is called a tautology. So, for example, if I say that we are going through the digital circuits course now. So, that is always true. So, we are definitely going through it, but if we say that we at the, we are not talking about say tautology now or say Boolean algebra now so, that is always false statement because at now at present we are talking about it.

So, in this way the sum of the Boolean expressions may be, may be always true or always false. So, when it is always true it is called a tautology, when it is always false it is called a fallacy. So, if the output of the Boolean expression is always false or 0 we call it a fallacy. So, like this say. So, if I have got this P and \bar{P} then output. So, P OR \bar{P} means when this P is 0, \bar{P} will be 1 and when this P is 1 \bar{P} is 0. So, so these 2 are input like if I if I have got a block, if I have got a block which is computing say $P + \bar{P}$ which is computing say OR function ok.

Now if I give it this one side I give P , another I give \bar{P} then what will be the output of this function. So, as I am; so this P as. So, so when I am giving P as 0. So, definitely \bar{P} is equal to 1. So, whenever I am feeding a 0 here I am feeding a one at this point, similarly when you are feeding a 1 here, I am feeding a 0 at this point.

Now, since this is a OR function, so, for both the combinations it will produce a 1. So, we can say that this particular module that I have. So, this always produces 1 irrespective of the value of P whether the P is 0 or 1. So, this module will always give you 1. Similarly if we have got another module that computes this P AND \bar{P} . So, that is. So, this is a AND operation. So, here also I give P AND \bar{P} as input and then I see what is the output. Now whatever be the value of this P , whatever be the value of P this P and \bar{P} is always going to be equal to 0.

So, in this way in one case we have got always 1, in the other case we have got always 0 it does not depend on the value of P . So, this $P + \bar{P}$. So, this is a tautology, because this is always true and $P + \bar{P}$ is a fallacy, because this is always false ok.

(Refer Slide Time: 14:20)

Exercise

1. Evaluate the following Boolean expression using Truth Table.

(a) $X'Y'+X'Y$ (b) $X'YZ'+XY'$
(c) $XY'(Z+YZ')+Z'$

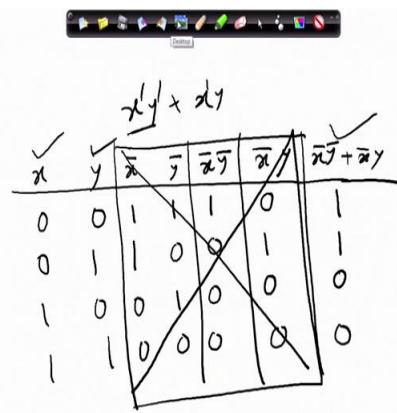
2. Verify that $P+(PQ)'$ is a Tautology.
3. Verify that $(X+Y)'=X'Y'$



So, we will see some we try to evaluate some Boolean expression using this using the truth table like say $\bar{x}\bar{y} + \bar{x} y$. So, what is the value? So, if we want to get the corresponding truth table. So, you have to proceed like this.

Say let us take a new page and do that.

(Refer Slide Time: 14:42)



A hand-drawn truth table for the Boolean expression $x'y + x'y$. The table has columns for inputs x and y , and outputs $\bar{x}\bar{y}$ and $\bar{x}y$. The final column shows the result of the OR operation between the two terms. The table is as follows:

x	y	$\bar{x}\bar{y}$	$\bar{x}y$	$\bar{x}\bar{y} + \bar{x}y$
0	0	1	1	0
0	1	1	0	1
1	0	0	1	0
1	1	0	0	0



So, it is $\bar{x}\bar{y} + \bar{x} y$, I think whatever that $\bar{x} y$ fine. So, if you want to get that truth table then what do you do? You take this x and y . So, it can take up the values 0 0, 0 1, 1 0 and 1 1.

Now we take up this 1. So, I will need \bar{x} and \bar{y} . So, I write down the \bar{x} and \bar{y} . So, \bar{x} value is a 1 1 0 0 because when x is 1, \bar{x} is 0 and when x is 0, \bar{x} is 1 and \bar{y} is 1 0 1 0.

Now, if we have to compute say I am already using this bar and this dash interchangeably. So, if I am have to if I want to compute $\bar{x} \bar{y}$; so for the first term. So, I have to. So, this $\bar{x} \bar{y}$ is 1 because both of them are 1. So, AND is 1. So, this is 0, this is 0 and this is 0, then I have got $\bar{x} y$; so we have to consider this column and this column ok. So, this $\bar{x} y$. So, this is 0, this is 1, this is 0 and this is also 0, $\bar{x} y$. Now finally, so, I have got the column $\bar{x} \bar{y} + \bar{x} y$. So, it is the OR of these 2 columns. So, this is 1, this is 1, this is 0 this is 0.

So, ultimately now, you can forego this part of the, this part of the truth table. So, you can forego this part and you can say that my truth table is this column, this column and this column. So, these 3 columns; so you need not we have done it for our understanding ok, but so, that does not constitute the truth table. So, truth table you will have some an input part and an output part the input part will have all the variable combinations and the output part will have the corresponding value of the function ok. So, we can find out so, that is the function that we are talking about.

So, similarly you can do the other part. So, this thing; so this is one problem it says that verify that $P + P\bar{Q}$ is a tautology.

(Refer Slide Time: 17:24)

		$P + (PQ)'$		
P	Q	PQ	$(PQ)'$	$P + (PQ)'$
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

$\rightarrow [P + (PQ)'] \rightarrow 1$

So, let us see how to do this. So, $P + P\bar{Q}$ is a tautology. So, how do we do this? So, there are 2 variables in this expression P and Q. So, accordingly we take 2 columns P and Q. So, they can take up the value 0 0, 0 1, 1 0 and 1 1. After that we have got P Q here as if as a component function. So, P Q is 0 this is a AND function. So, these are all 0 only when both P Q are 1, then the output is 1. After that we have got this $P\bar{Q}$. So, this is 1, this is 1. So, this is the complement of P Q. So, this is like this. So, what is $P + P\bar{Q}$; so 0 1. So, this is 1, this is 0 this is 1 OR of them. So, this is also 1. So, this is a one and one this is 1 and here P is 1 and $P\bar{Q}$ is 0, but there is an OR function. So, that is 1.

So, you see that this is a tautology because at the output column we have. So, it is always 1, it does not depend on the values of P and Q. So, if I have got a functional block, which has got 2 inputs P and Q and it computes this $P + P\bar{Q}$ then whatever value you give to this P and Q inputs, so, it will always output a 1 or true. So, this is a tautology. So, that is the proof. So, when a whenever if it is required to show that some expression is a tautology or a fallacy, what you need to do is, you have to draw the corresponding truth table and in the truth table you show that the output column is always 1 for a tautology and always 0 for a fallacy.

So, next we take the other another example. So, it says that we verify $x + \bar{y} = \bar{x}\bar{y}$. So, let us see how can we do this using this truth table. So, our problem is to show that $x + \bar{y} = \bar{x}\bar{y}$ ok.

(Refer Slide Time: 19:41)

				x	y	x'	y'	$x+y$	$(x+y)'$	$x'y$
0	0	1	1	0	0	1	1	1	1	0
0	1	1	0	1	0	0	1	0	0	0
1	0	0	1	1	1	1	0	0	0	0
1	1	0	0	1	0	0	0	1	0	0

$(x+y)' = x'y'$

The circuit diagram on the left shows an OR gate labeled $(x+y)$ with inputs x and y , and an AND gate labeled $x'y'$ with inputs x' and y' . Arrows indicate the flow of signals from the truth table columns to the respective gate inputs.

So, how to do this? So, we can we can as we are using a truth table method. So, there are 2 variables x and y. So, we take we write down the corresponding input possibilities. So, it is 0 0, 0 1, 1 0 and 1 1. Now as a component I have got I need this \bar{x} and \bar{y} also. So, I write down the \bar{x} part the compliments of x, similarly \bar{y} . So, that is not of y then I have got $x + y$. So, what is $x + y$? So, this is OR function of these first 2 columns OR of first 2 columns. So, this is 0 0 1 1 1.

So, next I can write down this $x + \bar{y}$. So, $x + \bar{y}$ is the complement of this. So, it is 1 0 0 0 and what is $\bar{x} \bar{y}$? It is the AND of these 2 columns ok. So, this is 1 0 0 0 now you compare between these 2 columns of the truth table. So, they are always same ok. So, that proves that all possible assignments of x and y.

So, so this is the left hand side function and the right hand side function they will produce the same output. Conceptually, so, if I have got 2 functional blocks one is computing this $x + \bar{y}$ and another is computing; so $\bar{x} \bar{y}$. So, if I feed the same values of x and y to both of them. So, I give the same values of x and y to both of them. Then the output that you will get f_1 here and f_2 there so, they will always be same because these truth table says that if you feed the same values of x and y to both of them, so, they will give the same result.

So, this way by using truth table so, you can verify many of the Boolean formulas and they try to establish equivalency between Boolean expressions and all ok. But of course, if the number of variables are more, then this is not a very good method because you will have to draw a table that has got 2^n rows in it. So, here the number of variable n is equal to 2, but suppose the n becomes equal to 10 or 15 or 20 or 100 like that. So, drawing a table with say 2^{10} rows 1024 rows or if it is a 100 variable function to 2^{100} . So, that is a very huge number. So, that. So, it is that then this method is not very much suitable ok. So, we have to do we have to use some other tricks and in fact, this is a very difficult problem to solve. So, you do not have any straight cut answer to this thing.

So, let us go back and see. So, we have seen how to check a tautology, how to check some relationship etcetera; now in case of now how this Boolean algebra. So, why this thing happened that, it was invented long back, but it is it was not used in the practice ok. So, as a mathematical tool it was there, but it was not being practiced in the in the say scientific community.

The reason is that the implementation was not there. So, we could not implement this AND gate or as this AND logic OR logic NOT logic like that. So, with the advent of these digital circuits, what has happened is, these implementations became possible. So, this AND OR NOT functions you can implement very easily.

(Refer Slide Time: 23:40)

The slide has a yellow background with a dark blue header bar at the top containing various icons. The title 'Implementation' is centered in a large, bold, dark red font. Below the title is a bulleted list in black text. At the bottom of the slide, there is a dark blue footer bar featuring the IIT Kharagpur logo on the left and the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES' on the right.

- Boolean Algebra applied in computers electronic circuits. These circuits perform Boolean operations and these are called **logic circuits** or **logic gates**.

So, that gives that gives the impetus like why the why should we work on this, on Boolean algebra for this logic circuits.

So, Boolean algebra is applied in computers and electronic circuits and these circuits perform Boolean operations using something called logic circuits or logic gates. So, we will see the some elements. So, logic gates are again you can say it is some sort of electronic structure. So, that can that can implement in AND operation, OR operation, NOT operation like that.

(Refer Slide Time: 24:15)

Logic Gate

- A gate is an digital circuit which operates on one or more signals and produce single output.
- Gates are digital circuits because the input and output signals are denoted by either 1(high voltage) or 0(low voltage).
- Three type of gates are as under:
 1. AND gate
 2. OR gate
 3. NOT gate

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, a gate is defined to be a digital circuit, which operates on one or more signals and produce single output. So, it has got a number of inputs and a single output. So, based on the values of input so, it will produce some output. So, gates are digital circuits because the input and output signals are denoted by either high or the either 1 that is often represented by a high voltage and 0 often represented as low voltage. Again the same thing that high voltage, low voltage these terms are a bit confusing so, it may be other way also; so we call it at logic high and logic low. So, 1 is logic high and 0 is logic low.

So, as we are considering the 3 fundamental type of operations. So, in case of these digital circuits also. So, you will find that there are 3 basic types of gates that we have. One is called AND gate, one is called OR gate another is called NOT gate. So, AND, OR and NOT. So, these are the 3 fundamental gates that we have, and then we can. So, using this AND gate. So, we can mimic the behavior of this AND operation, using or gate we can mimic the behavior of OR operation and NOT gate can mimic the behavior of NOT operation.

(Refer Slide Time: 25:36)

The slide has a yellow background with a blue header bar at the top containing various icons. The title 'AND gate' is centered in a large, bold, dark red font. Below the title, there are two bullet points:

- The AND gate is an electronic circuit that gives a **high** output (**1**) only if **all** its inputs are high.
- AND gate takes two or more input signals and produce only one output signal.

On the left side, there is a schematic diagram of an AND gate with two inputs labeled 'A' and 'B' entering from the left, and one output labeled 'AB' exiting to the right. The gate symbol is a rectangle with a small circle on the left side.

On the right side, there is a truth table with three columns:

Input A	Input B	Output AB
0	0	0
0	1	0
1	0	0
1	1	1

At the bottom of the slide, there is a blue footer bar. On the left, it contains the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it contains the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a small video window showing a man speaking.

So, let us see what is an AND gate. So, AND gate is again another one electronic circuit that gives high output only if all its inputs are high ok. So, there can be, it can take 2 or more input signals and produce only one output signal. So, it takes a number of inputs and it produces output. Output is 1 or high, only when all the inputs are 1. So, that was the AND truth table if you remember. So, the truth table output column. So, it had 1 only when all the inputs were equal to 1, any of the inputs being equal to 0 the output was equal to 0.

So, in case of symbolically it is represented like this. So, we one straight line and then 2 straight line in front of it there is an ellipse. So, this is the symbol. So, whenever we are trying to represent one AND gate. So, we will be using this symbol. So, this is a 2 input AND gate. So, there are 2 inputs A and B and one output which is marked as AB here. So, you can have multiple inputs like you can have as a logically there is no limitation on the number of inputs that you can have to an AND gate. So, of course, the minimum is true, but you can have any higher number 3 4 5 6 ok.

So, there is there is no such limit, but practically of course, there will be a limit because there will be some digital circuitry which will be there inside this AND gate and that cannot be infinite. So, logically there were logically the any number of inputs may be there more than 1, but there will be a physical limit.

So, this is the behavior of this and circuitry, whatever circuitry we put here it should behave in this fashion that when this A AND B both are equal to 0, the output should also be 0.

When one when both of them are equal to 1, then the output should be equal to 1 otherwise whenever at least one of the inputs is equal to 0. So, it should be the output should be equal to 0. So, that that is the basic AND operation.

(Refer Slide Time: 27:42)

OR gate

- The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high.
- OR gate also takes two or more input signals and produce only one output signal.

Input A	Input B	Output A+B
0	0	0
0	1	1
1	0	1
1	1	1

OR

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The next gate, next fundamental gate that we will consider, is the OR gate. Again the same thing that OR gate is an electronic circuit that gives high output if one or more of its inputs are equal to high. So, this is again depicted by the truth table that we see, that we have seen for the OR gate for the OR functionality. So, whenever all the whenever at least one of the inputs is high, the output is high and when all the inputs are equal to 0, then only output is 0; so again the same thing. So, you can have at least you should have at least 2 inputs, but theoretically you can have any higher number of inputs, but practically there will be a limit. This is the symbol for the OR gate with the 2 input OR gate. So, of course, you can have more number of inputs.

The OR gate, the OR gate truth table is like this. So, this for this whatever circuitry we put here should behave in this fashion, that only when both A and B inputs are at logic low level 0, the output will output should be equal to 0 or logic low otherwise the output should always be high. So, that is the OR operation of A and B.

(Refer Slide Time: 28:56)



Then the NOT gate: so NOT gate is again another electronic circuit that gives high output if its input is low and it takes only one input signal. So, it is a single input gate unlike an AND gate, AND gate and OR gate. So, this NOT gate is a single input gate. So, this is if a output is the complement of the input. So, if input is 0, output is 1 and if input is 1 output is 0.

So, this is also often known as inverter. So, this is NOT gate another very common term for this is inverter. So, it is symbolically represented like this, one triangle and a bubble at the beginning at the end of it, and then A and output is written as \bar{A} and the truth table will be like this. So, it is input A and output is \bar{A} . So, whenever this is 0 output is one and whenever input is 1 output is 0.

So, in this way we can have basic AND, OR and NOT gates. So, in the digital circuits they can realize the AND, OR and NOT functions of Boolean algebra.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 09
Boolean Algebra (Contd.)

(Refer Slide Time: 00:15)

Principal of Duality

• In Boolean algebras the duality Principle is obtained by interchanging AND and OR operators and replacing 0's by 1's and 1's by 0's. Compare the identities on the left side with the identities on the right.

Example

$A+1 = 1$ then $A \cdot 0 = 0$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, next will be looking into something called principle of duality so, in case of in this, what it says is that in case of a Boolean algebra the duality principle. So, it says that you can interchange this AND and OR operators, and at the same time you have to replace 0's by 1's and 1's by 0's, then you get the relations.

For example, this is a Boolean expression $A + 1 = 1$. So, we will see its proof later. So, then if this is true, then you can replace this 1 by 0 and these OR by AND so, this $A \cdot 0 = 0$. So, if A OR $1 = 1$, then A AND 0 will be $= 0$. So, this is called principle of duality. So, you can interchange the OR and AND operator and accordingly you have to interchange the that 1's and 0's

(Refer Slide Time: 01:08)

Basic Theorem of Boolean Algebra

T1 : Properties of 0

- (a) $0 + A = A$
- (b) $0 \cdot A = 0$

T2 : Properties of 1

- (a) $1 + A = 1$
- (b) $1 \cdot A = A$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we will them again and again. So, some of the properties of Boolean algebra so, if we look into the basic theorems are like this, the first theorem is the properties of 0. Says the $0 + A = A$. So, so if we have a variable A, then you can understand that if you OR it with 0 the result will be equal to A. So, these are known as Boolean identities, these are Boolean theorems like that.

Similarly, 0 AND A equal to 0 then $1 + A = 1$, then 1.1 AND A = 1 ok. So, these are some of the properties of Boolean algebra.

(Refer Slide Time: 01:47)

Basic Theorem of Boolean Algebra

T3 : Commutative Law

- (a) $A + B = B + A$
- (b) $A \cdot B = B \cdot A$

T4 : Associate Law

- (a) $(A + B) + C = A + (B + C)$
- (b) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

T5 : Distributive Law

- (a) $A (B + C) = A B + A C$
- (b) $A + (B C) = (A + B) (A + C)$
- (c) $A + A' B = A + B$

A + A' B
↓
 $\cancel{(A+A')} (A+B)$
↓
1 (A + B)
= A + B

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a commutative law it says that $A + B$, $A \text{ OR } B = B \text{ OR } A$; similarly so, by means of duality so, we can replace this $+$ by dot. So, $A \cdot B$ will be $= B \cdot A$, then there is a associativity law. So, that is I am doing OR of three variables out of that I in the first case I do OR of A AND B first and then with the result I OR C , the second case I OR $B + C$ B AND C first and then we do the OR of A with that. So, the both the cases the result will be same. So, $(A + B) + C = A + (B + C)$.

Similarly, from the AND operation also $(A \text{ AND } B) \text{ AND } C = A \text{ AND } (B \text{ AND } C)$. So, that is known as associativity law. Then there is a distributive law which says that $A \text{ AND } (B \text{ OR } C) = (A \text{ AND } B) \text{ OR } (A \text{ AND } C)$ and by means of duality.

So, you get from this relation A , the relation B says that $A + BC = (A + B)(A + C)$. There is another distributive law which say that $A + \bar{A} = A$. So, you can get a proof of this very easily say this one. So, this is actually $A + \bar{A}B = (A + \bar{A})(A + B) = 1 \cdot (A + B) = (A + B)$. So, this way we can have a number of properties of this Boolean algebra.

(Refer Slide Time: 04:02)

Basic Theorem of Boolean Algebra

T6 : Indempotence (Identity) Law

- (a) $A + A = A$
- (b) $AA = A$

T7 : Absorption (Redundance) Law

- (a) $A + AB = A$
- (b) $A(A + B) = A$

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Then there is idempotence or identity law which says that $A + A = A$, and $AA = A$. So, $x+x$ in our normal arithmetic we have equal to $2x$, but here it will be equal to x only. So, here $A + A = A$ similarly $A \text{ AND } A = A$.

Then there is absorption law or redundancy law it says that $A + AB = A$ and by duality $A(A + B) = A$. So, here also the same thing like you can say. So, the, if you try to analyze

it logically. So, it is $A + AB$. So, A is true or AB is true. So, for this one also A has to be equal to true so, naturally that is equal to A .

(Refer Slide Time: 04:59)

Basic Theorem of Boolean Algebra

T8 : Complementary Law

- (a) $X+X'=1$
- (b) $X.X'=0$

T9 : Involution

- (a) $X'' = X$

T10 : De Morgan's Theorem

- (a) $(X+Y)'=X'.Y'$
- (b) $(X.Y)'=X'+Y'$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this way we can have a number of rules for this Boolean algebra, then this $X + \bar{X} = 1$. So, that is complementary law. So, that is the, that is the tautology then $X\bar{X} = 0$ that is the fallacy that we have seen previously then involution. Involution says if you do complementation twice $\bar{\bar{X}} = X$ and then if you take complement again, then you will get back the original variable X . So, this is involution.

And there is another pair of rules which are known as De Morgan's theorems; it says that $\overline{X + Y} = \bar{X}\bar{Y}$. So, similarly by duality so, $\overline{XY} = \bar{X} + \bar{Y}$. So, these are De Morgan's theorems. So, all these identities that we have written here they are very much useful for the sake of Boolean simplification.

Like if you want to get some, if you have a complex expression and you want to simplify, then you may use some of these rules to do the simplification process ok.

(Refer Slide Time: 06:08)

Exercise

Q 1. State & Verify De Morgan's Law by using truth table and algebraically.

Q 2. State and verify distributive law.

Q 3. Draw a logic diagram for the following expression:

(a) $ab + b'c + c'a'$
(b) $(a+b).(a+b').c$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we will be doing a few exercises before going further ok.

(Refer Slide Time: 06:11)

Representation of Boolean expression

Boolean expression can be represented by either

(i) Sum of Product (SOP) form or
(ii) Product of Sum (POS form)

e.g.

$AB+AC \rightarrow \text{SOP}$
 $(A+B)(A+C) \rightarrow \text{POS}$

In above examples both are in SOP and POS respectively but they are not in Standard SOP and POS.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, let us, let us look into some of the properties that the first property that we will look into it says that the complement of x is unique.

(Refer Slide Time: 06:25)

1) Complement of x is unique
Let x_1, x_2 are two complements of x , if possible

$$\begin{aligned} x+x_1 &= 1, \quad x x_1 = 0 \\ x+x_2 &= 1, \quad x x_2 = 0 \end{aligned}$$

$$\begin{aligned} x_1 &= x_1 \cdot 1 \\ &= x_1(x+x_2) \\ &= x_1 x + x_1 x_2 \\ &= 0 + x_1 x_2 \\ &= x_1 x_2 \end{aligned}$$

$$\begin{aligned} x_2 &= x_2 \cdot 1 \\ &= x_2(x+x_1) \\ &= x_2 x + x_2 x_1 \\ &= 0 + x_2 x_1 \\ &= x_2 x_1 \end{aligned}$$

$\Rightarrow x_1 x_2 = 0$

$\Rightarrow x_1 = x_2$

$\Rightarrow x_1' = x_2'$

$$\begin{aligned} x \cdot x' &= 0 \\ x+x' &= 1 \end{aligned}$$

$$\begin{aligned} x \cdot y &= 0 \\ x+y &= 1 \end{aligned}$$

Video window: A man in a blue shirt is speaking.

The complement of x is unique; that means, we cannot have two different complements. So, complement of x means so, if x is a variable, its complement is x we represent it as \bar{x} , then it says that $x\bar{x} = 0, x + \bar{x} = 1$.

In general instead of writing \bar{x} to make it simple, suppose let us write it as y . So, y will be said to be complement of x . So, if I have got $x.y = 0$ and $x + y = 1$.

Now, the question is, can I have more than one y , more than one y for x ? So, can I have more than one complement for x ? So, if possible let we have two complements one is x_1 and another one is called x_2 are two complements of x . So, since x_1 is a complement of x . So, we have got this property $x + x_1 = 1, x x_1 = 0$. Similarly since x_2 is a complement of x . So, we have got $x + x_2 = 1$ and $x x_2 = 0$ ok.

$$\begin{aligned} \text{Now, } x_1 = 1 \cdot x_1 &= (x + x_2) \cdot x_1 = x x_1 + x_1 x_2 = 0 + x_1 x_2 = (x x_2) + x_1 x_2 = x_2 (x + x_1) = 1 \cdot x_2 = x_2 \Rightarrow x_1 = x_2 \end{aligned}$$

So, it says that if I take two different if possible x_1 and x_2 are two differ two complements of x , then we finally, say that x_1, x_2 they are same. So, that proves that the complement of x must be unique ok.

So, next we will prove, look into another property.

(Refer Slide Time: 11:07)

2) $x + 1 = 1$ → By duality $x \cdot 0 = 0$

$$\begin{aligned} &= 1 \cdot (x + 1) \\ &= (x + x') \cdot (x + 1) \\ &= x + x + x \cdot x' \\ &= x + x' \\ &= 1 \end{aligned}$$

So, second property we will says that $x + 1 = 1$. So, how to do this? So, to prove this so, $(x + 1) \cdot 1 = (x + \bar{x})(x + 1) = x + x\bar{x} = x + \bar{x} = 1$.

So, by duality you can say that if you replace this + by dot that is OR by AND and 1 by 0. So, you can get the other property by duality so, $x \cdot 0 = 0$. So, this is by duality. So, you can also take as you can get a similar proof, but since we have proved the left one the so, right one is necessary. So, we can get this one directly ok.

(Refer Slide Time: 12:58)

$x + x = x$ → $x \cdot x = x$

$$\begin{aligned} &= x \cdot x \cdot 1 \\ &= (x + x) \cdot (x + x') \\ &= x + x + x \cdot x' \\ &= x + x \cdot 0 \\ &= x \end{aligned}$$

So, then we will prove that idempotent law it says that $x + x = x$. So, for proving this, $x + x = (x + x) \cdot 1 = (x + x)(x + \bar{x}) = x + 0 = x$.

So, this proves the idempotent property.

Next we will try to prove another result which says that the involution property.

(Refer Slide Time: 14:13)

Involution:

$$\begin{aligned}
 (x')' &= x \\
 (x')' &= (x')' + 0 \\
 &= (x')' + x\bar{x} \\
 &= [(x')' + x] \overbrace{[(x')' + x']}^{\cancel{x} + \cancel{x}' = 1} \\
 &= [x + (x')]'. 1 \\
 &= [x + (x')]'. [x + \cancel{x}] \cancel{x} + \cancel{(x')'}. x' \\
 &= x + x\cancel{x} + \cancel{(x')'}. x \\
 &= x + x(1 + \cancel{\bar{x}}) \\
 &= x
 \end{aligned}$$

So, next we will prove the involution property, which says that $\bar{\bar{x}} = x$. So, we do it like this, $\bar{\bar{x}} = \bar{x} + 0 = \bar{x} + x\bar{x} = (\bar{x} + x)(\bar{x} + \bar{x}) = (\bar{x} + x) \cdot 1 = (\bar{x} + x)(x + \bar{x}) = x + x\bar{x} + \bar{x}x + \bar{x}\bar{x} = x + \bar{x}x = x(1 + \bar{x}) = x$

So, this proves the involution property ok.

(Refer Slide Time: 17:42)

A handwritten proof of the absorption law $x + xy = x$. The proof starts with the equation $x + xy = x$, which is crossed out with a large red X. To the right of the X is the result $x \cdot (x+y) = x$. Below the X, the steps of the proof are written:
 $x + xy$
= $x \cdot 1 + xy$
= $x[1 + y]$
= $x \cdot 1$
= x



So, you can now prove the absorption property which says that $x + xy = x$. Now $x + xy = x$ can be proved easily because $x + xy = x \cdot 1 + xy = x(1 + y) = x \cdot 1 = x$

So, if you follow the duality, then from here I can write that $x \cdot (x + y) = x$. This is the duality of the absorption law.

Then prove another result says that $x + \bar{x}y = x + y$

(Refer Slide Time: 18:44)

A handwritten proof of the result $x + \bar{x}y = x + y$. The proof starts with the equation $x + \bar{x}y = x + y$, which is crossed out with a large red X. To the right of the X is the result $x \cdot (\bar{x} + y) = y$. Below the X, the steps of the proof are written:
 $x + \bar{x}y$
= $\cancel{x} + \cancel{x}y$
= $(\cancel{x} + x)(\bar{x} + y)$
= $x \cdot (\bar{x} + y)$
= y



So, if you do distribution. So, this is $x + \bar{x} = (x + \bar{x})(x + y) = 1 \cdot (x + y) = x + y$. So, this way we can prove it and again by duality you can say that $x(\bar{x} + y) = xy$.

Then associativity if you want to show associativity then we can do it like this associativity means.

(Refer Slide Time: 20:01)

Associativity:

$$x + (y + z) = (x + y) + z$$

$$\Downarrow$$

$$xA \quad xB$$

$$= x[x + (y + z)] \quad = x[x + y] + xz$$

$$= x[x + x(y + z)] \quad = x[x + y] + xz$$

$$= x[x + yx + xz] \quad = x[x + y] + xz$$

$$= x[x + y] + xz \quad = x[x + y] + xz$$

$$= x[x + y] + xz \quad = x[x + y] + xz$$

$$= x[x + y] + xz \quad = x[x + y] + xz$$

$$= xA = xB = x \quad = x[x + y] + xz$$

$$= xA = xB = x \quad = x[x + y] + xz$$

$$x'B = x'(y + z) \quad \Downarrow$$

$$x'B = x'(y + z) \quad = x'[x + (y + z)]$$

$$= x'[x + y] + x'z \quad = x'[x + y] + x'z$$

$$= x'[x + y] + x'z \quad = x'[x + y] + x'z$$

$$= x'[x + y] + x'z \quad = x'[x + y] + x'z$$

$$= x'[x + y] + x'z \quad = x'[x + y] + x'z$$

$$= x'[x + y] + x'z \quad = x'[x + y]$$

$$= x'(y + z) \quad = x'(y + z)$$

So, associativity of OR, say OR operation associativity suppose we have. So, we need to show that $x + (y + z) = (x + y) + z$. So, in Boolean algebra how will you show that what we will do. So, $A = x + (y + z)$; $B = (x + y) + z$;

So, we will take x AND A . So, $xA = x[x + (y + z)] = x + xy + xz = x(1 + y) + xz = x + xz = x \Rightarrow xA = x$.

So, similarly this $xB = x$. So, this can be similarly it can be shown now we take $\bar{x}A = \bar{x}(y + z)$. So, $\bar{x}A = \bar{x}[x + (y + z)] = x\bar{x} + \bar{x}y + \bar{x}z = \bar{x}(y + z)$. Similarly, you can see you can say that $\bar{x}B = \bar{x}[(x + y) + z] = x\bar{x} + \bar{x}y + \bar{x}z = \bar{x}(y + z)$.

So, we have got this $xA, xB, \bar{x}A, \bar{x}B$. So, what is happening is that $xA = xB = x$. So, this is one observation. So, from here $\bar{x}A = \bar{x}B = \bar{x}(y + z)$. So, with these observations so, we will be proceeding like this.

(Refer Slide Time: 24:42)

$$\begin{aligned}
 A &= A \cdot 1 \\
 &= A(x + x\bar{x}) \\
 &= Ax + Ax\bar{x} \\
 &= xA + x\bar{A} \\
 &= xB + x\bar{B} \\
 &= Bx + B\bar{x} \\
 &= B(x + \bar{x}) \\
 &= B \\
 &= B
 \end{aligned}$$

$$\begin{aligned}
 A &= B \\
 x + (y + z) &= (x + y) + z \\
 x \cdot (yz) &= (xy) \cdot z
 \end{aligned}$$

Now, we will take this the variable A. So, A can be written as A AND 1. So, it is $A(x + \bar{x}) = Ax + A\bar{x} = Bx + B\bar{x} = B(x + \bar{x}) = B \cdot 1 = B$.

So, you have got A = B. So, that proves our result that this OR is associative, associativity of the OR is proved. So, you can you can use the duality. So, we have got $x + (y + z) = (x + y) + z$. So, if you use duality, then it says that $x \cdot (yz) = (xy) \cdot z$ So, that way you can have the duality.

(Refer Slide Time: 26:13)

$$\begin{aligned}
 (xy)' &= (x'y')' \\
 x + x\bar{z} + x\bar{z} &= 0 \\
 &= (x + z)(x + z\bar{z}) \\
 &= (x + z)x + x\bar{z} \\
 &= 0 \neq 0 \Rightarrow p\phi = 0 \\
 (xy)' &= (x'y')' \\
 &= (x + z)(x + z\bar{z}) \\
 &= (x + z)x + x\bar{z} \\
 &= 1 \quad p\phi = 1 \\
 &= x\bar{y} \quad \downarrow \quad \downarrow \\
 &= p \quad \phi \\
 &= p' \quad q \\
 &= x\bar{y} = (x + y)'
 \end{aligned}$$

Next we will prove the De Morgan's theorem the first law. So, the De Morgan's law so, it says that $\overline{x + y} = \bar{x}\bar{y}$ ok. So, we have got this properties known to us like $x + \bar{x} = 1$; $\bar{x} = 0$. So, this properties are known to us similarly we can. So, what we do is that we take this sum $x + y + \bar{x}\bar{y} = [(x + y) + \bar{x}][(x + y) + \bar{y}] =$

$$[y + x + \bar{x}][x + y + \bar{y}] = [y + 1][x + 1] = 1.$$

Also, if I take this product ok so, $(x + y)(\bar{x}\bar{y})$. So, if I take this product then what will happen? So, this can be rewritten as $x\bar{x}\bar{y} + y\bar{x}\bar{y} = 0 + 0 = 0$. Now, you compare this terms so, this term $x + y$ another term $\bar{x}\bar{y}$. So, what was happened? If I call it say P and let us call it Q. So, from the first property from this first proof I can say that $P + Q = 1$ and from this proof I can say that PQ is $= 0$; that means, Q is actually the invert of P, Q is P' .

So, now, if I substitute so, Q as $\bar{x}\bar{y}$. So, $\bar{x}\bar{y} = P'$. So, P is $x+y$. So, we have got this De Morgan's theorem here ok.

So, $\overline{x + y} = \bar{x}\bar{y}$ so, this way we can prove the De Morgan's theorem first law and by means of duality, you can claim the second law that is $x \cdot y$. So, if you replace this $+$ by dots. So, you get the dual law. So, $\overline{x \cdot y} = \bar{x}\bar{y}$ so, this is from here by means of duality this is by duality ok.

So, in this way so, we can have proof of this Boolean identities and of course, the other way is to draw the truth table and the show that these are all two expressions are equivalence. So, that can be done, but if it if we want to show by Boolean identities, then also we can prove it like this fashion.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 10
Boolean Algebra (Contd.)

So, in case of Boolean expression so, we can represent it in 2 different forms.

(Refer Slide Time: 00:20)

Representation of Boolean expression

Boolean expression can be represented by either

- (i) Sum of Product(SOP) form or
- (ii) Product of Sum (POS form)

e.g.

$AB+AC \rightarrow \text{SOP}$
 $(A+B)(A+C) \rightarrow \text{POS}$

In above examples both are in SOP and POS respectively but they are not in Standard SOP and POS.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

One is known as sum of product form, another is called product of sum form. So, as the name suggests sum of product means it is a summation of product terms. For example, this $AB + AC$ so, AB is A AND B. So, as we said that it is obvious so, we do not write the dot in between to represent the AND so, AB is the first product term, AC is the second product term. So, overall expression is the summation of the terms AB and AC so, that it is in sum of product form.

Other alternative for this Boolean expression representation is product of sum, where we take the sum as individual components like $A + B$ or $A + C$. So, these are individual component and then we take the product of those components to come to the Boolean expression.

Now, instead of simply being the variables so, it can be the complemented forms of that also like, this is also a correct form. So, $\bar{A} B + \bar{B} C$ so, that is also sum sort of expression

where it is so, this is also a sum of product form, but here this $\bar{A}B$. So, these individual terms they are slightly they are the bit. So, individual terms so, they are in some complemented forms so, that can happen. So, whatever it is so, based on that we can we can have different forms for the, for the functions.

(Refer Slide Time: 01:59)

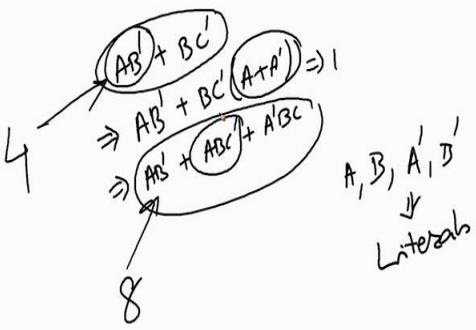
Canonical form of Boolean Expression (Standard form)

- In standard **SOP** and **POS** each term of Boolean expression must contain all the literals (with and without bar) that has been used in Boolean expression.
- If the above condition is satisfied by the Boolean expression, that expression is called **Canonical form of Boolean expression**.

 IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES


Now, but the problem with these forms is that it is not unique. For example, for the same expression so, we can have different forms like I will take an example and try to explain. Like if I have the term say $A\bar{B} + B\bar{C}$ so, this is one form.

(Refer Slide Time: 02:16)


<div style="

Now, if you expand this $A\bar{B}$ so, it can be written as say a say, we can write it as say somebody may write it as $A\bar{B} + B\bar{C}(A + \bar{A})$. So, if you expand this you get like $A\bar{B} + B\bar{C}A + BCA\bar{A}$. Because is $A + \bar{A}$ is equal to 1. So, we can always write in this form.

So, what is happening is that you see this is also sum of product form; this is also sum of product form. So, the problem with these forms is that one of them may be some sort of minimum form and some other one is not that minimized. So, it has got 3 terms in it the first one has got 2 terms in it. So, it is not mandatory that this minimized form they will be unique ok.

So, that is the problem with Boolean expression. So, if the minimum sum minimum term expression may not be unique, there will multiple expressions with the similar type of cost, in terms of number of product terms, number of variables. So, this AB etcetera so, these or the compliment like $\bar{A}\bar{B}$. So, they are called literals.

So, this number of literals in this expressions may be so, in this case in this in this expression we have got 4 literals, in this expression we have got one 2 3 4 5, 6 7 8 literals. So, number of literals may vary so, literals means the variable or it is complemented form. Then we have got this term. So, this $A\bar{B}$ is a term, similarly $AB\bar{C}$ is a term, ok. So, we have got literals term, and then the expression the whole expression.

(Refer Slide Time: 04:20)

**Canonical form of Boolean Expression
(Standard form)**

- In standard **SOP** and **POS** each term of Boolean expression must contain all the literals (with and without bar) that has been used in Boolean expression.
- If the above condition is satisfied by the Boolean expression, that expression is called **Canonical form of Boolean expression**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, coming back to our discussion. So, we have got so, how to check like two expression whether they are same or not. So, for doing that there is a canonical representation. So, anything that is canonical means so, all expressions that are equivalent they will boil down to the same canonical form, ok. So, that is the beauty of this canonical form of expression. So, this standard sum of product or product of sum each term of Boolean expression must contain all the literals, with and without bar that has been used in the Boolean expression.

So, it says that when you are taking the sum of product or product of sum. So, individual terms they should so, all the literals should appear in the individual terms. If which is not, then it is not a not in the canonical form. So, in, if the condition is satisfied, then the Boolean expression is set to be in canonical form of Boolean expression. So, it is called canonical, because now it becomes unique.

So, for a particular Boolean expression, so, you can have only one canonical form canonical sum of product form and another canonical product of sum form. So, you cannot have more than one canonical form for the same expression. So, if you are given two expression and to check whether they are equivalent or not they are same or not. So, what you need to do is to convert them into their canonical forms, and then check whether the terms are same in both the expressions.

(Refer Slide Time: 05:50)

Canonical form of Boolean Expression (Standard form)
Contd..

➤ In Boolean expression **AB+AC** the literal **C** is mission in the 1st term **AB** and **B** is mission in 2nd term **AC**. That is why AB+AC is not a Canonical SOP.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, so, if $AB + AC$. So, here then the first term the literal C is not there. Similarly, in the second term if this B is missing here. So, it is A and C will this to literals are present so, B is missing. So, this is why this $AB + AC$ is not a canonical sum of product expression.

(Refer Slide Time: 06:12)

Canonical form of Boolean Expression (Standard form)
Contd..
Convert $AB+AC$ in Canonical SOP (Standard SOP)

Sol. $\text{AB} + \text{AC} \Rightarrow$

$$\begin{aligned} &\Rightarrow \text{AB}(\text{C} + \text{C}') + \text{AC}(\text{B} + \text{B}') \\ &\Rightarrow \text{ABC} + \text{ABC}' + \text{ABC} + \text{AB'C} \\ &\Rightarrow \text{ABC} + \text{ABC}' + \text{AB'C} \end{aligned}$$

Distributive law

NPTEL ONLINE CERTIFICATION COURSES

So, how to convert this $AB + AC$ non canonical form into a canonical sum of product form? So, this is simple, by using the Boolean identities. So, what we do? We write this AB as $AB.1 + A.1.C$. Then what happens is this AB is so, AB is that this 1 can be re written as $C + \bar{C}$, and this can be written as $B + \bar{B}C$ so, that is what is done here. So, we substitute this in place of one we write here $C + \bar{C}$ in place of one here we write $B + \bar{B}$.

Then if we expand so, you will get this form. So, if these are all equivalent so, from here we are getting this, from this we are getting this. Now you see that sum of the terms are appearing more than once the term ABC has appeared more than once. So, you can just remove the duplicate and write it as $ABC + AB\bar{C} + A\bar{B}C$ so, this is the canonical form. So, whatever be the different forms in which you write this, like somebody may write it as $A(B + C)$, somebody may write it as say $AB\bar{C} + ABC + AC$.

So, in this way different term different people can write the same first expression in different form. But if you convert it into canonical form then all of them will become same. That is the beauty of the canonical form.

(Refer Slide Time: 07:58)

Canonical form of Boolean Expression (Standard form)

Convert $(A+B)(A+C)$ in Canonical ~~SOP~~ (Standard POS)

Sol. $(A+B)(A+C)$

$\Rightarrow ((A+B)(C)) . ((A+C)(B))$

$\Rightarrow (A+B+C) . (A+B+C') . (A+B+C)(A+B'+C)$

$\Rightarrow (A+B+C) . (A+B+C')(A+B'+C)$

Distributive law

Remove duplicates

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, another expression a so, this is this AB so, AB + AC. So, this is converting into canonical POS form. So, this is not SOP so, this is POS so, this is POS form product of sum form from, which is also known as the standard product of sum form ok. So, we have got the POS form. So, how do you do it? So, AB + AC so, this can be written as; so, A + (A + B + 0)(A + C + 0).

Now, this 0's can be written as $C\bar{C}$ here, and $B\bar{B}$ here then we apply the distributive law, where this is the first term this is the second term. So, you remember the distributive law $A + BC$ is can be written as $(A + B)(A + C)$. So, this is if this whole thing is consider as A, and this is consider as say BC , then it is A + B, ok.

So, that is $A + B + C$ in this case and you can write it as X and Y Z better. So, this is say X and this is say Y Z. So, this is this is written as so, this part is my X, this is Y. So, this part is my X so, this is Z, so you can distribute the thing. Similarly for the second term also, so, we can distribute this $B\bar{B}$ into the 2 constituents.

So, that way finally, then if you remove the duplicates. So, you get so, this $A + B + C$ is coming twice. So, you can remove them, and you can get it once. So, this way we can convert the product of sum expression into their canonical forms.

(Refer Slide Time: 09:52)

Next so, how to convert from product of sum to sum of product or sum of product to product of sum is like this. Product of sum to sum of products so, these expression is very simple so, you just go on the multiplying the thing like say yeah I have got an expression like $(A + \bar{B})(A + C)(A + D)$. So, what you do you just multiply $(A + \bar{B})(A + C)$. So, these gives us this term $A + A\bar{B} + AC + \bar{B}C$, then that is that is equivalent to $A + \bar{B}C$, why? Like say, this one sorry like say this one so, $A + A\bar{B}$. So, it can be written as $A(1 + \bar{B})$ that is equal to A.

So, the first part $A + A\bar{B}$ can be converted into A. Now this $A + AC$ so, it can be again be converted into $A(1 + C)$, that is equal to A. So, the first 3 terms of the expression here. So, it boils down to A so, ultimately you have got $A + \bar{B}C$. And then we can multiply it by A + D as a result we get this expression $AA + A\bar{B}C + AD + \bar{B}CD$. Again you have to take common and so, AA is A, then this $A + A\bar{B}C$.

So, by a similar logic $A + A\bar{B}C = A(1 + \bar{B}C)$. So, this one will be so, that gives rise to A. So, a so, this part $A + A\bar{B}C$ gives me A. And with that if you do $A + AD$ so, if you do A + AD, then you will be getting again A common. So, $1 + D$ so, that way I will get A finally, again from here I will get an A. So, it is $A + \bar{B}CD$

So, that is the whole multiplication that we have done. So, product of sum to sum of product conversion we can do it like this. Similarly sum of product to product of sum

conversion. So, you have to factor out. So, this is $A + \bar{B} CD$. So, it can be written as $(A + \bar{B})(A + CD)$. And again if you factor out so, $(A + \bar{B})(A + C)(A+D)$.

So, in this way we can convert from sum of product to product of sum or from product of sum to sum of product, of course, they are not canonical in nature. So, to convert to canonical so, you have to again do it you have to, you have to convert to canonical form like say $A + \bar{B} CD$.

(Refer Slide Time: 12:44)

If you have to convert so, you have to write it as $A(B + \bar{B})(C + \bar{C})(D + \bar{D})$. Now, if you just go on multiplying this so, you will get the canonical sum of product form and similarly for the second expression like say $(A + \bar{B})(A + C)(A + D)$. So, you have to write it as like $(A + \bar{B} + C\bar{C})(A + D\bar{D} + C)(A + D + B\bar{B})$. Then that is not the end, because after this, after you have factored out this.

So, again these as to be factored out, this is this C so, you have to write the first term. Again the same thing will be happen for others. And then it will be re written as $(A + \bar{B} + C + D)(A + \bar{B} + C + \bar{D})$ so, etcetera. So, this way you can convert first you convert from sum of product to product of sum or product of sum to sum of product. And then if you want to convert to canonical from so, you have to expand like that.

(Refer Slide Time: 14:51)

Canonical form of Boolean Expression (Standard form)
Contd..

Minterm and Maxterm *CSOP*

Individual term of Canonical Sum of Products (SOP) is called Minterm. In otherwords minterm is a product of all the literals (with or without bar) within the Boolean expression.

CPOS

Individual term of Canonical Products of Sum (POS) is called Maxterm. In otherwords maxterm is a sum of all the literals (with or without bar) within the Boolean expression.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, when you have converted an expression to this canonical form product of sum or sum of product. So, we can come across 2 new terminologies, one is called minterm another is called maxterm. So, individual term of this canonical sum of product expression. So, they are also known as CSOP this canonical sum of product we often write tell it as CSOP canonical sum of products. Similarly here we will be canonical product of sum we write as CPOS. So, this was a CSOP form. So, it is so, individual terms of this canonical SOP form so, they are called minterms.

So, minterm is a product of all literals within the Boolean expression, with or without the complementation. And in case of this product of sum expression the canonical product of sum expression. So, individual terms so, they will be called individual sum terms so, they will be called maxterms. So, maxterm is a sum of all the literals with or without bar within the Boolean expression.

So, in this way we define minterm and maxterm for Boolean expressions.

(Refer Slide Time: 16:02)

Minterms & Maxterms for 2 variables (Derivation of Boolean function from Truth Table)

x	y	Index	Minterm	Maxterm
0	0	0	$m_0 = x' y'$	$M_0 = x + y$
0	1	1	$m_1 = x' y$	$M_1 = x + y'$
1	0	2	$m_2 = x y'$	$M_2 = x' + y$
1	1	3	$m_3 = x y$	$M_3 = x' + y'$

The minterm m_i should evaluate to 1 for each combination of x and y.
$$(x'y')' = x + y$$

The maxterm is the complement of the minterm

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this is a thing like say 0 0. So, this is a, this the corresponding minterm is $\bar{X}\bar{Y}$. So, 0 1 corresponding minterm is $\bar{X}Y$ like that 1 0 and 1 1, so, $X\bar{Y}$ and XY . As far as max term is concerned, so, they are written as M_0 so, this is written as $X + Y$. So, you note here that for while writing the max term. So, for if the value of variable is 0, we are taking it as uncomplemented form. So, that is the convention that you have to follow. And whenever the value of variable is one so, you have taking it in the complemented form. Whereas, for writing minterm when the variable is in true form so, we are writing it in true form, and when it is in complemented form like say 0 0 so, you are writing it as $\bar{X}\bar{Y}$.

So, minterm was so, all the minterms m_i should evaluate to 1 for each combination of X and Y. And the maxterm is the complement of the minterm. So, this is the complement of this. So, basically $\bar{X}\bar{Y}$ if you take a compliment so, this $\bar{X}\bar{Y}$ so, if you take compliment then by De Morgan's law it becomes equal to $X + Y$. So, maxterm is the complement of minterm.

(Refer Slide Time: 17:25)

Solved Problem

Prob. Find the minterm designation of $XY'Z'$

Sol. Substitute 1's for non barred and 0's for barred letters

Binary equivalent = 100

Decimal equivalent = 4

Thus $XY'Z' = m_4$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, next we will see some problem. So, find the minterm designation of $X\bar{Y}\bar{Z}$. So, we have took so, it is binary equivalent is 1 0 0. So, decimal equivalent is 4 so, it is the m_4 . So, this is this is the standard notation that we are using. So, this is say if this value evaluate to 0 so, will write it as m_0 . So, if this value evaluate is 2 ah. So, 1 0 so, that evaluates 2 so, we write it as m_2 .

So, by giving this number or index, we can directly understand what is the corresponding minterm. So, that is why this indexing is done. Similarly, here the indexing is done, we know that whatever be the values so, it is say a say a one. So, that is 0 1. So, X should be taken into true form and Y should be taken into complemented form. For the max term for the minterm, should be taken in this X should be taken in complemented form and y be taken in true form.

So, this $X \bar{Y}Z$ is 1 0 0 so, decimal equivalent is 4 and so, this is the, this is m_4 .

(Refer Slide Time: 18:32)

The slide has a yellow background and a blue header bar. The title 'Purpose of the Index' is at the top center. Below it is a bulleted list of points. To the right of the list, there is handwritten mathematical notation for a function f. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a professor.

- Minterms and Maxterms are designated with an index
- The index number corresponds to a binary pattern
- The index for the minterm or maxterm, expressed as a binary number, is used to determine whether the variable is shown in the true or complemented form
- For Minterms:
 - '1' means the variable is "Not Complemented" and
 - '0' means the variable is "Complemented".
- For Maxterms:
 - '0' means the variable is "Not Complemented" and
 - '1' means the variable is "Complemented".

$f = \Sigma(0, 5, 9, 11)$

$f = \Pi(1, 5, 7)$

So, why do we need to do this indexing? So, minterms and max and maxterms are designated with an index, and this index number corresponds to a binary pattern. So, if you are talking about a particular function, so, one possibility is that you write down the sum of product or product of sum expression. Otherwise you can just still tell what are the indexes like for the minterms or maxterms. So, that way you can also tell what is the function. So, that is why this index number is associated.

So, index number for minterm or maxterm expressed as a binary number, it is used to determine whether the variable is in true form or in complemented form. So, for minterms 1 means the variable is not complemented and 0 means the variable is complemented. And from max term so, 0 means the variable is not complemented and 1 means the variable is complemented.

So, this is the convention that is followed for this minterm and maxterm. So, indexing actually helps us to tell the function like you can simply say, that my function f is the summation of the minterm so, 0 5 9 and 11. So, this immediately says that since I am going up to 11 so, I will need 4 bits to represent it. And out of that the minterm 0 5 9 and 11 for them, the function value is one, whereas others the function value is 0.

So, this way this is very standard notation. Similarly, I can have the max term representation which is written as this π or product. So, 1 5 7. So, this means that for this

product terms 1 5 and 7 so, this I have to take the sum of product form sorry product of sum form to express the function.

So, talking about so, sum of product form for a Boolean function represented by the truth table.

(Refer Slide Time: 20:27)

Solved Problem

Write SOP form of a Boolean Function F, Which is represented by the following truth table.

Sum of minterms of entries that evaluate to '1'

X	y	z	F	Minterm
0	0	0	0	
0	0	1	1	$m_1 = x' y' z$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$m_6 = x y z'$
1	1	1	1	$m_7 = x y z$

$F = m_1 + m_6 + m_7 = \sum (1, 6, 7) = \bar{x} \bar{y} z + x y \bar{z} + x y z$

Focus on the
‘1’ entries



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

So, this is a so, a truth table means I have already said. So, you have to tell what are the input assignment and corresponding to that what is the function, the output column. So, what will be the value of the function for this one? Like say this is the truth table for this function. So, 0 0 0 is 0, 0 0 1 is 1 so, like that.

So, if you consider only this values, where the function f evaluates to 1, ok. So, this is for this is for this 0 0 1, 1 0 1 sorry 1 1 0 and 1 1 1. So, 0 0 1 the corresponding index is one 1 1 0 index is 6 and 1 1 1 index is 7. So, by overall function is $m_1 + m_6 + m_7$ ok, in that sum of product form. So, this is see these are standard notation where put a Σ , and within bracket we give the terms or you can write it explicitly. So, it is $\bar{X} \bar{Y} Z + X Y \bar{Z} + XYZ$ so, this way also you can write it.

So, this is the, from truth table to minterm to sum of product expression.

(Refer Slide Time: 21:42)

Exercise

1. Write POS form of a Boolean Function F, Which is represented by the following truth table

$$F = \prod_{i=1}^3 (x_i + y_i + z_i) = (x_1 + y_1 + z_1)(x_2 + y_2 + z_2)(x_3 + y_3 + z_3)$$

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

2. Write equivalent canonical Sum of Product expression for the following Product of Sum Expression: $F(X,Y,Z) = \prod_{i=1}^3 (x_i + y_i + z_i) = (x_1 + y_1 + z_1)(x_2 + y_2 + z_2)(x_3 + y_3 + z_3)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, POS form for this one. So, the so, so if you if you want to write the POS form, then we have to concentrate on the 0's. So, this is basically the where the function value is becoming 0, I have to take those functions. So, those terms so, this is ah. So, this is 0 1 0. So, this is 0 1 0. So, I have got so, for 0 1 0 the value is 1 so, I have to the function f. So, the function f in this case is function f in this case is the product is the product of sum form of this one.

So, this is 0 1 0; that is, value is 2, then we have got 0 1 1 that is 3, then we have got 1 1 0; that is 6, 2 3 6. So, which essentially means that the function when I am writing it. So, for 0 while I will write in true form. So, $(X + \bar{Y} + Z)(X + \bar{Y} + \bar{Z})(\bar{X} + \bar{Y} + Z)$.

So, this is the sum of product this is this is the product of sum expression. So, this is represented by max term like this. And then we can we can representative in pi from, and then we can write down this thing. So, similarly the second problem that it says that write down the sum of product of sum expression for this. So, we can understand what we can do this here. So, this is was one is 0 0 1 so, I have to write like $X + Y + \bar{Z}$. So, 3 is 0 1 1 so, $X + \bar{Y} + \bar{Z}$ into 6 that is 1 1 0. So, $\bar{X} + \bar{Y} + Z$ into 7. So, that is 1 1 1 $\bar{X} + \bar{Y} + \bar{Z}$.

So, see apparently it seems that it will how will it give the function for different forms. So, this is this can be obtained like say this product of sum so, if you expand it. So, you will get the vary, the if you convert it into sum of product form, then you will get the original terms, the for where the values are values of the variable is 1.

(Refer Slide Time: 24:38)

Minimization of Boolean Expression

- Canonical SOP (Sum of Minterms) and POS (Product of Maxterm) is the derivation/expansion of Boolean Expression.
- Canonical forms are not usually minimal.
- Minimization of Boolean expression is needed to simplify the Boolean expression and thus reduce the circuitry complexity as it uses less number of gates to produce same output that can be taken by long canonical expression.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Next will see how to minimize this Boolean expression. So, canonical sum of product or canonical product of sum is the derivation or expansion, expansion of Boolean function. So, it essentially we have got less number of very literals per term, and that we have expanding to have more number of literals and that converts into canonical form. But apart from the application in which you need to establish equivalence between Boolean expression. So, this is not much in not much useful because while realizing the function. So, what will look for is the minimum sized realization in terms of less number of gates or less number of connections like that.

So, canonical forms are usually not minimal. So, they are may be cases where the canonical form itself is a minimal form, but it is a, in general that is not the case. So, minimization of this Boolean expression so you needed to it is required to simplify the Boolean expression, and thus reduce the circuit complexity so, that I can use less number of gates logic gates to produce the same output, that can be obtained by taking the canonical expression.

So, this minimization of Boolean function is by itself a very important topic. And we will see some technics for that, but keeping in mind that this a minimization process is not very simple. So, in our course so, will be consists considering Boolean expression with less number of variables only 4 variable 5 variable, at most 6 variable. And we will see that

they are exist exact methods for getting the exact minimum form corresponding to those function.

However as the those methods that will discuss. So, they are not applicable for number of variables say more than 6. So, if you are looking for larger number of variables then exact method do exist, but the problem is that they take enormous amount of time, even if you think of writing a computer program for those minimization process, it will take enormous amount of time.

So, keeping that in mind so, it is so many a many a time what we do is we do some heuristic minimization and be satisfied with that any way so, that is not the purview of this course. So, will be restricting our self to Boolean expression of less number of variables up to 6 variable, and will see some technic by which we can do the exact minimization.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Boolean Algebra (Contd.)

(Refer Slide Time: 00:20)

Minimization of Boolean Expression (Contd...)

➤ Two method can be applied to reduce the Boolean expression –

i) Algebraic
ii) Using Karnaugh Map (K-Map).

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So minimization of Boolean expression; so, there can be two method that can be applied for doing this minimization. One method is known as algebraic minimization and other one is using a mapping, map based method which is known as Karnaugh map, or K-map in short. So, this algebraic method so, we have already seen some Boolean expression conversion using algebraic method; that uses the Boolean identities for reducing the, or converting one Boolean expression to another Boolean expression.

And this Karnaugh map is another a map based method which is much simpler compared to algebraic method. So, both of them can give us minimum from, but this algebraic method so, there is no as hard and fast rule that you have to, you have to apply the algebraic manipulations in this order to get the minimum so that makes it difficult. Whereas, in Karnaugh map so, there are some rules if you follow those rules, then you will definitely get a minimum form expression.

(Refer Slide Time: 01:18)

Minimization of Boolean Expression (Contd...)

➤ **Algebraic Method**

- The different Boolean rules and theorems are used to simplify the Boolean expression in this method.

So, this algebraic method uses the different Boolean rules and theorems that we have discussed in our previous classes. So, they can be applied for simplifying the Boolean expression in this method so, this is the algebraic method.

(Refer Slide Time: 01:32)

Minimization of Boolean Expression (Contd...)

Solved Problem

Minimize the following Boolean Expression:

1. $a'b'c + ab'c' + ab'c + abc' + abc$
= $a'b'c + ab' + ab$
= $a'b'c + a$
2. $AB'CD' + AB'CD + ABCD' + ABCD$
= $AB'C + ABC$
= AC

So, we will take some example; like say, this is a Boolean expression $a'b'c + ab'c' + ab'c + abc$. So, this is in fact, in canonical sum or product form. So, if you take ab common from these 2 terms so, you will get $c' + c = 1$. So, as a result these 2 terms if you combine them so, you will get like ab' .

Similarly, between these 2 terms; so, if you take ab common you will again get $c' + c$ evaluates to 1 so, you get ab. And then between these 2 terms so, if you take a common. So, you will get $b' + b$ that that is equal to 1. So, this is also get cancelled so, you will get a. So, in this way we can convert this Boolean expression into a corresponding simplified form $a'bc+a$. Of course, there is no guarantee that if you if you are applying this Boolean rules you. So, there is no proof that this is the exact minimum form, ok.

In this in this particular example that maybe the case, but in general there is no such guarantee. Similarly if you take say another example. So, $AB'CD'+AB'CD+ABCD'+$. So, here also if you take $AB'C$ common from the first 2 term you will get like $AB'C(D+D')$. So, $D+D'$ is equal to 1 so, you get $AB'C$. Similarly between these 2 terms so, you take ABC common so, it is $D+D'$. So, this, that is equal to 1. So, after getting this so, if you take AC common from these 2 terms so, you will get $B+B'$ evaluates to 1 so, you get AC .

So, in this form, you can apply these Boolean identities to convert this Boolean expressions into some minimized form. Of course, as I said that it is not proved to be absolute minimum ok. So, we can do some exercise this is say this $X'Y'Z'+X'YX'+XY'Z'+XYZ'$.

(Refer Slide Time: 02:36)

Minimization of Boolean Expression (Contd...)

Exercise

A. Minimize the following Boolean Expression:

- $X'Y'Z' + X'YZ' + XY'Z' + XYZ' = x'z'(y'+y) + xz'(y'+y) = x'z' + xz' = (x+x')z' = 1 \cdot z' = z'$
- $a(b + b'c + b'c') \Rightarrow a(b + b'(c + c')) = a(b + b' \cdot 1) = a(b + b) = a$

B. Prove algebraically that

- $(x+y+z)(x'+y+z) = y+z$
- $A+A'B=A+B'$

Handwritten solution for part B:

$$\begin{aligned} & x + xy + xz + x'y + x'y + yz + yz + z \\ &= xy + xz + x'y + yz + z \\ &= (x+x')y + (x+x')z + z \\ &= y + z + z = y + z \end{aligned}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you want to do this so, you can say. So, you between the first 2 terms so, you can say I can take $X'Y'$ common and inside is $Y'+Y$.

Similarly, from the second 2 terms. So, I can take XZ' common, and I get $Y+Y'$. So, that boils down to $X'Z'+XZ'$. And again between these 2 terms so, we can take Z' common. So, we can write it as $Z'(X+X')$. So, $X+X'$ evaluates to 1. So, that is equal to Z' . so, that way you can do the simplification and this one so, you want to simplify, then a in so, you can do many things; like, you can say that I will take $a(b+b'c+b'c')$. So, this $b'c+b'c'$. So, we can take b' common. So, it take it get inside as $c+c'=1$.

So, $a(b+b'.1) = a(b+b')=a$ So, this way you can simplify the Boolean expression. So, you can try to verify these rules.

$$\begin{aligned} \text{So, this } (x + y + z)(x' + y + z) &= xx' + xy + xz + yx' + yy + z + zx' + zy + zz = \\ xy + xz + yx' + y + z + zx' &= xy + xz + y(x' + 1) + z(x' + 1) = xy + xz + y + \\ z = y(1 + x) + z(1 + x) &= y \cdot 1 + z \cdot 1 = y + z \end{aligned}$$

(Refer Slide Time: 07:41)

Minimization of Boolean Expression (Contd...)

Exercise

A. Minimize the following Boolean Expression:

1. $X'Y'Z' + X'YZ' + XY'Z' + XYZ'$
2. $a(b + b'c + b'c')$

B. Prove algebraically that

1. $(x+y+z)(x'+y+z)=y+z$
2. $A+A'B'=A+B'$

Handwritten steps for simplifying $a(b + b'c + b'c')$:

$$\begin{aligned} A + A'B' &= A + A'B' + A'B' \\ &= A(B + B') + A'B' \\ &= AB + AB' + A'B' \\ &= AB + B' + AB + AB' \\ &= AB + B' \end{aligned}$$

Similarly, $A + A'B' = (A + A')(A + B') = 1 \cdot (A + B') = A + B'$

So, next we will see this Karnaugh map method.

(Refer Slide Time: 09:43)

Minimization of Boolean Expression (Contd...)

Karnaugh Map

The Karnaugh map (K-map for short), [Maurice Karnaugh's](#) 1953 refinement of [Edward Veitch's](#) 1952 Veitch diagram, is a method to simplify Boolean algebra expressions.

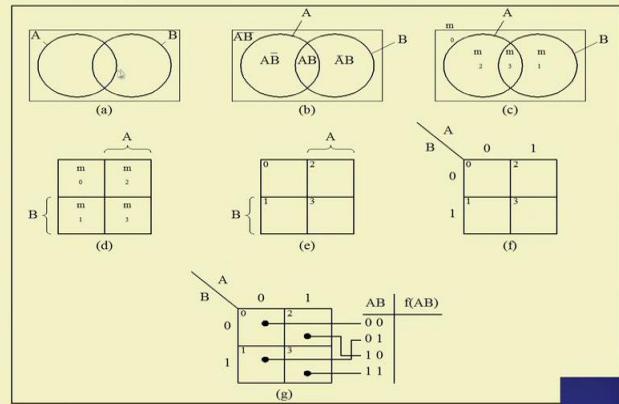
- K-Maps are a convenient way to simplify Boolean Expressions.
- They can be used for up to 4 or 5 variables.
- They are a visual representation of a truth table.



So, this Karnaugh map or K-map so, this was invented by Maurice Karnaugh in 1953. So, this is the refinement of Veitch's 1952 Veitch diagram, which is a method to simplify Boolean algebra expressions. So, K-maps so, they are a very convenient way to express Boolean expressions. They can be used for up to 4 or 5 variables. So, in some cases you can expand to 6 variables as well, and they are the visual representation of a truth table.

(Refer Slide Time: 10:19)

Venn diagram and equivalent K-map for two variables



The figure shows the equivalence between Venn diagrams and K-maps for two variables (A and B). It includes:

- (a) Two overlapping circles A and B.
- (b) A Venn diagram with four regions labeled AB, ĀB, ĀB̄, and AB̄.
- (c) A Venn diagram with four regions labeled m₀, m₁, m₂, and m₃.
- (d) A K-map with four cells labeled m₀, m₁, m₂, and m₃. The row is labeled B and the column is labeled A.
- (e) A K-map with four cells labeled 0, 1, 2, and 3. The row is labeled B and the column is labeled A.
- (f) A K-map with four cells labeled 0, 1, 2, and 3. The row is labeled B and the column is labeled A.
- (g) A K-map with four cells labeled 0, 1, 2, and 3. The row is labeled B and the column is labeled A. Below it is a truth table with columns A, B and rows 00, 01, 10, 11. The output f(AB) is 1 for rows 01 and 10, and 0 for rows 00 and 11.



So, we will see how they look like. So, this Venn diagram, so, this we are all familiar with Venn diagram. So, that is for set you can represent it. So, this is the universe, and suppose

I have got A and B as 2 variables or they are representing 2 sets so, there may be intersections and some portions maybe common. Now outside this region of these 2 sets, so, I have got the region which is represented as $\bar{A}\bar{B}$.

Similarly, this part which is common between A and B so, that we represent it as represent as AB, as if both of them are there both of them are true. This part A is there, B is not there so, it is $A\bar{B}$. And this part A is not there and B is there so, it is $\bar{A}B$. So, you can also tell, if you write down the corresponding index like $\bar{A}\bar{B}$ the index value is 0, $A\bar{B}$ index value is 2, AB index value is 3, and $\bar{A}B$ index value is 1.

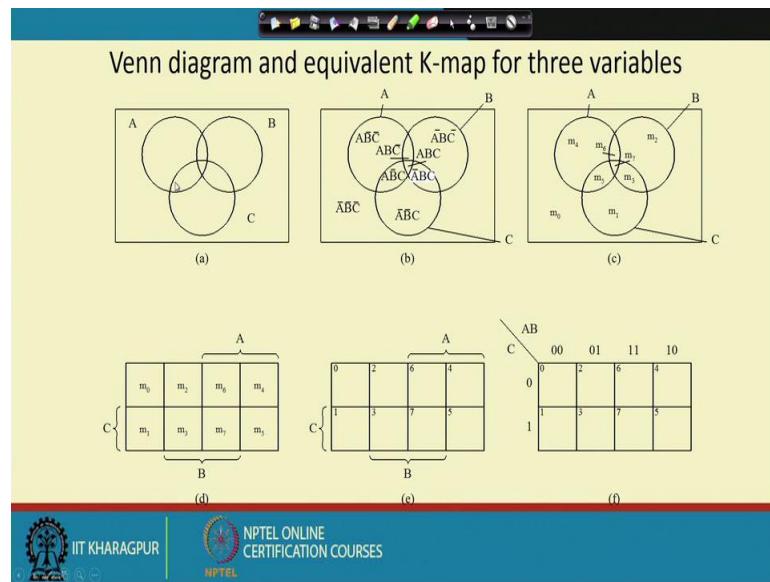
So, this way I can write down the minterm expressions; like, m_0 , m_1 , m_2 and m_3 . So, this is if; this can be represented in the form of a rectangular region, which is divided into 2 in each side, since I am considering only 2 sets or say 2 variables. So, I can represent it as one ah this part this side I represent A and this side I represent B. So, when I am on this part so, this is A both A and B are equal to 0. Here A is equal to 0, but B is equal to 1.

So, this is A equal to 1 and B equal to 0 and here both of them are equal to 1 so, I can represent it like this. So, for the sake of simplicity I can just drop this terms m, and I can write it as 0 2 1 3. So, this this is also this AB. So, this is one form in which I am writing it. So, you can also this is also a standard representation, where we put a diagonal line here, and this side we write A, this side we write B. And then we write like 0 plus the this A, equal to 0 and 1, B equal to 0 and 1.

So, if this is 0 1 0 1 so, like that. So, you can say like, say this is my full so, this is this is another way of representation like say this one. So, first term so, this is 0 0. So, whatever be the value of the function so, we can write here ok. So, this is if this value is say, if this value is say 1 so, this is 0, this is 1 and this is 0. So, we will what we will do? So, at this place we will write a 1, then this 0 1. So, we will write a 0 there. Then 1 0 so, this is this we will write as a 1 here, and 1 1 we will write a 0 there.

So, this way we can fill up this table ok. So, this is known as the Karnaugh map for a function. So, this is, this is a Boolean function, and we can make the corresponding Karnaugh map like this.

(Refer Slide Time: 13:20)



So, this helps in representing the functions like, now what happens to the 3 variables? So, if I have got 3 variables so, in the Venn diagram notation we have got AB and C as 3 circles. And there if you see in that overlapping so, this part is here all the 3 sets are present so, it is ABC, ok.

So, outside is $\bar{A}\bar{B}\bar{C}$. So, here I have got $\bar{A}\bar{B}C$ ok. So, the, because the A and B are not there only C is there. So, this way for every region, you can figure out the corresponding part of the set which is present there. And now in the same fashion so, we convert it into this minterms. So, outside is m 0 so, this is m 1 and this is m 2. So, like that we can write down the corresponding minterms. So, in A tabular notation so, a so, A and B they are put on the on the row side, and C is put on the column side.

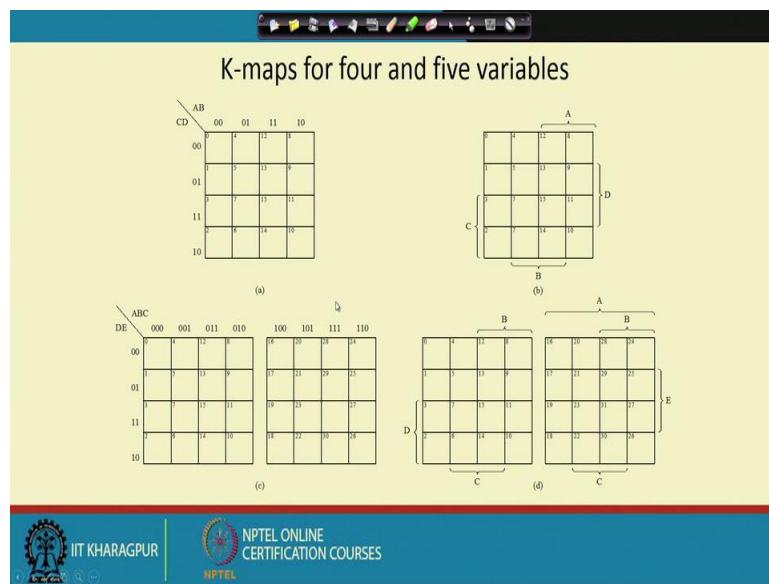
So, 2 variables will be put on the row side and one variable will be taken as the column side. So, that way so, this ab is 0 0 0 1 1 1 and 1 0. So, you note the you take the notation that, this A is represented by these 2 row, these 2 columns and B is B equal to 1 spans over these 2 columns. So, A equal to 1 spans over these 2 columns, B equal to 1 spans over these 2 columns. C equal to 1 spans over this particular row.

So, this part, this particular column A and B both are 0. And here A is 0 B is 1, ok. So, that way, we can have different rows and columns representing different parts of the, of this diagram. And ultimately we have got these rectangular block, rectangular box, where we can this is also a notation a standard notation; that is you write AB here and C here on this

side we enumerate the combinations 0 0, 0 1, 1 1, 1 0. This side we enumerate the combination for C 0 and 1.

Now, given the function so, we will be, we will populate these entries. And accordingly we can fill it up.

(Refer Slide Time: 15:28)



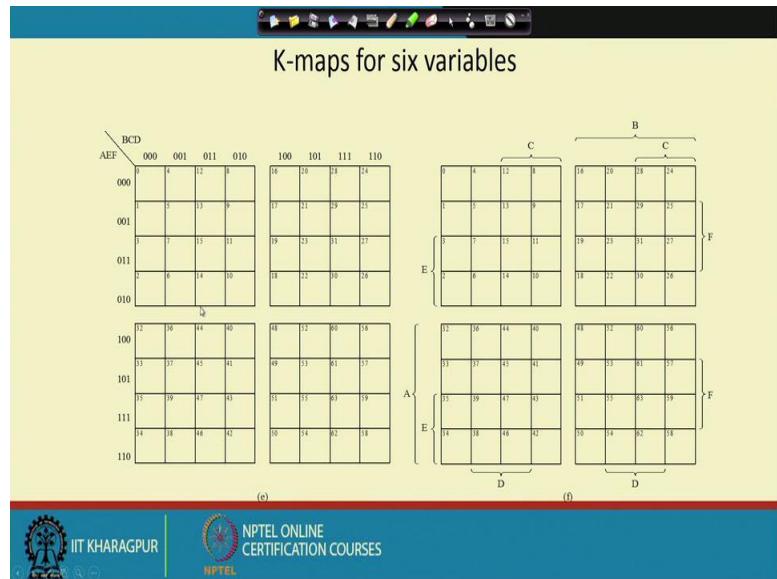
So, 4 variables so, if we are taking 4 variables then this AB so, we were using the row side for representing 2 variables. Now we will be using the column side also for representing 2 variables so, AB and CD. So, 0 0, 0 1, 1 1, 1 0 and CD is 0 0, 0 1, 1 1, 1 0. And so, in a, sometimes we write it in this format also, and for these 2 columns A is 1, for these 2 columns B is 1. So, like that is also a standard notation.

So, generally we will be using this notation for most of the cases. But sometimes we will be going to this notation as well. So, for 5 variable what we do? We take two maps, ok. So, in one of the map A is equal to 0. So, you see all the combinations that we have; here A is equal to 0. So, this is 0 so, here also A is 0, here also A is 0. And on this side I have taken all the maps where all the permutations where all the combinations where A is equal to 1. So, that way, we as if we split the map into 2, 1 for A equal to 0 other for A equal to 1. So, this way we can go for 5 variable maps, ok.

So, though they are not a single map, but we can go for in this way. Or we can also visualize it as if this is a 3D; so, one is tacked over the other. So, here it was a 2D up to 4 variable

the map remains 2D, but when you go to 5 variable so, map becomes 3D. So, you have to expand in another direction on the vertical direction.

(Refer Slide Time: 17:02)



6 variable so, again so, as you can understand, that I have to have 4 maps ok. So, for this map A equal to 0 and B equal to 0. For this map A equal to 0, B equal to 1. Here A equal to 1, B equal to 0. And for this map, I have got A equal to 1 and B equal to 1.

So, this way the minterms get distributed over, the number of maps number of tables. And then we can try to do some manipulation with those numbers and get the minimized form. And similarly so, this is also another notation of the same map, 6 variable map.

(Refer Slide Time: 17:45)

Truth table to K-Map (2 variable minterm)

A	B	P
0	0	1
0	1	1
1	0	0
1	1	1

The expression is:
$$\overline{A} \cdot \overline{B} + A \cdot \overline{B} + A \cdot B$$

	B	\overline{B}	B
A	0	1	1
\overline{A}	0	1	1
A	1		1

minterms are represented by a 1 in the corresponding location in the K map.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how the truth table gets represented in the form of a Karnaugh map, ok. So, 2 variable suppose this is the function so, AB 0 0, 0 1, 1 0 and 1 1 for 0 0, 0 1 and 1 1, the value of the variable the value of the function is 1, otherwise it is 0.

So, in the Karnaugh map so, we represent the minterms at by putting a one at the corresponding location of the Karnaugh map. So, at so, these 3 ones were there so, these 3 ones we are put here, ok. So, this is the, this is the representation of this truth table in the form of Karnaugh map. So, minterms are represented by a one in the corresponding location in the Karnaugh map.

(Refer Slide Time: 18:34)

K-Maps (2 Variables k-map contd...)

- Adjacent 1's can be "paired off"
- Any variable which is both a 1 and a zero in this pairing can be eliminated
- Pairs may be adjacent horizontally or vertically

$$\begin{aligned} \bar{A}\bar{B} + \bar{A}B &= \bar{A}(\bar{B} + B) \\ &= \bar{A} \end{aligned}$$

The expression is:
 $A'B' + A'B + AB$

B is eliminated, leaving \bar{A} as the term

After reduction the expression becomes $\bar{A} + B$

A is eliminated, leaving B as the term

NPTEL ONLINE CERTIFICATION COURSES

So, 2 variable Karnaugh map so, it is adjacent ones can be paired off so, we can do it like this.

So, this is one and this is one. So, these 2 can be paired together, and we can get a pairing of them. So, this is. So, in this so, the term that we have is so, if we look into this so, it is $\bar{A}\bar{B} + \bar{A}B = \bar{A}(B + \bar{B}) = \bar{A}$. So, do circle them, combine them together then, one of the variables get eliminated.

So, in this case B is changing from 0 to 1. So, whichever variable change it is polarity so, that variable gets eliminated. So, this is the variable B has got eliminated. Similarly if you look into this term, then what is happening is; so, this is nothing but $\bar{A}B + AB = B(A + \bar{A}) = B$. So, here this the term A has got eliminated.

So, in this way, once we have represented the function in Karnaugh map so, we automatically do this type of grouping and eliminate variables for getting the minimum form. So, this is the beauty of Karnaugh map so, once you have drawn it. So, you can just go on grouping them and getting the minimum form. And there is a proof that these are going to be the minimum one. So, there are certain rules so, if you follow that. So, there is a proof that you will be definitely getting a minimum sum of product sum.

(Refer Slide Time: 20:33)

• Three Variable K-Map

A	B	C	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Notice the code sequence:
00 01 11 10 – a Gray code.

$\bar{A} \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot C$

NPTEL ONLINE
CERTIFICATION COURSES

Let us take a 3 variable function. So, this ABC combination varying from 0 0 0 to 1 1 1, and this is the output column that the function the function P. Now in a so, this is one at 0 1 0, 1 0 0 and 1 1 0. So, this is 0 1 0 so, 0 1 0 is this one, then we have got 1 0 0 so, 1 0 0 is this one and 1 1 0 is that one.

So, in in this if you see that I can this is the representation. And then I have to I can do a or I can do something called this grouping as you are done doing in the previous example. So, one square is filled in for each minterm. And here you see one thing that the way we are writing these indices. So, this is 0 0, 0 1, 1 1 and 1 0. So, it is you just keep in mind that it is not like 0 0 0 and 1 0 1 1.

So, it is not the binary ascending order, but it is in some sort of gray code, ok. So, this we will see later that this is something called this is a, this type of coding where only one bit changes between successive terms so, that is called a gray code. So, it is 0 0 to 0 1 only the LSB is changing. Similarly, 0 1 to 1 1 only the second bit is changing, and 1 1 to 1 0 again only the LSB is changing. So, when only one bit is changing so, we get a gray code.

So, this gray coded form so, this is useful because when you are grouping. So, if I have got 2 successive terms. So, if you write here so, so, instead of this row B this column B in 1 1 so, if this was say 1 0. Then if you are trying to combine these 2, then the problem is that the more than one variable is changing, but in that case it will be a wrong one. So, we

cannot take more than one variable changing between the successive terms. So, that is that is the rule of this Karnaugh map; so, you have to take it in the gray coded format.

(Refer Slide Time: 22:46)

The figure shows a three-variable Karnaugh map for variables A, B, and C. The columns represent BC values (00, 01, 11) and the rows represent A values (0, 1). The map has four minterms: m₀₀ (0,0), m₁₀ (0,1), m₁₁ (1,1), and m₀₁ (1,0). A blue box highlights m₁₀. Two pairs of adjacent cells are circled: one pair (m₀₀, m₁₀) is labeled 'BC' with a value of 00, and another pair (m₁₀, m₁₁) is labeled '(1)' with a value of 1. A third pair (m₀₀, m₀₁) is also circled and labeled '(1)' with a value of 1. A large oval on the left groups m₀₀ and m₀₁. A bracket on the right indicates that the pair (m₁₀, m₁₁) equates to $B\bar{C}$ as A is eliminated. The expression $A'B'C' + A'BC'$ is simplified to $(A'+A)BC'$, which further simplifies to BC' .

Our truth table simplifies to $A\bar{C} + B\bar{C}$ as before.

equates to $B\bar{C}$ as A is eliminated.

Here, we can "wrap around" and this pair equates to $A\bar{C}$ as B is eliminated.

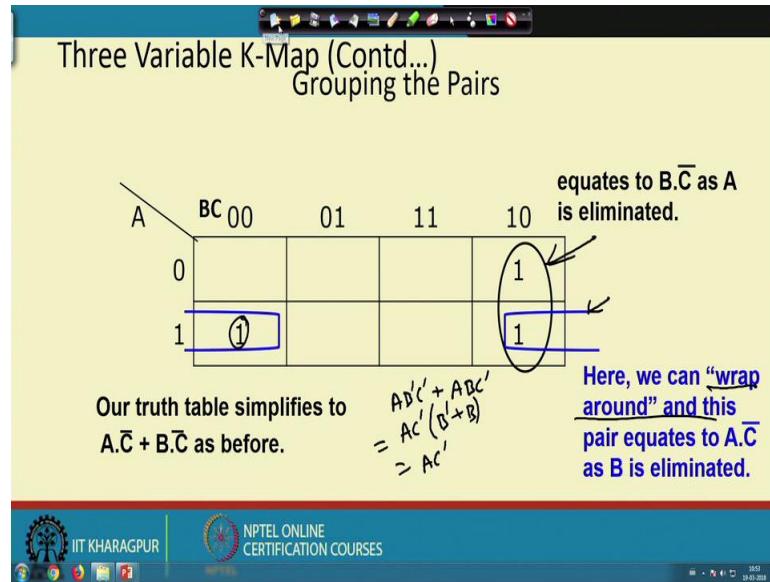
Now the grouping part; so, this first the example that we have. So, say for that so, you can do a grouping like this. So, where I have got a ones at 2 successive places in the vertical direction. And the term that gets eliminated is given by so, this is this can be written as say this is $\bar{A}B\bar{C} + AB\bar{C} = B\bar{C}(A + \bar{A}) = B\bar{C}$.

. Also while you can think this Karnaugh map to be some sort of folded one. So, this last row or the last column you can take it as if the whole table has been folded. So, you get this last column adjacent to the first column. Similarly if you have got more the higher sized Karnaugh map say 4 by 4 Karnaugh map, then also you can take this folding.

So, you can take as if this part is at this row is adjacent to this row, and again this column is adjacent to this column. Because it is folded in the both in the row wise and the column wise direction. Because of very simple like if you look into these 2 patterns. So, this is 0 0 and so, this is 1 0. So, only one bit is changing between them so, whenever only one bit is changing the columns are considered to be adjacent. Similarly, between the rows if only one bit is changing. So, we will consider the rows to be adjacent.

So, here the so, for by following that rule so, this 0 0 and 1 0. So, these 2 columns are adjacent to each other so, we can do a grouping taking the corresponding ones. So, if you do that then you will be getting like.

(Refer Slide Time: 25:02)



So, this is so, this term is $A\bar{B}\bar{C} + AB\bar{C} = A\bar{C}(B + \bar{B}) = A\bar{C}$.

So, ultimately so, we have got this $A\bar{C}$ term from the blue square, and we have got this $B\bar{C}$ term from the first one. So, the overall function is so, this is that is what I was telling, but it is a wraparound connection. So, wrap around so, we can wrap around, and we can say that we can get more number of minimizations for this part, ok.

(Refer Slide Time: 25:52)

Three Variable K-Map (Contd...)

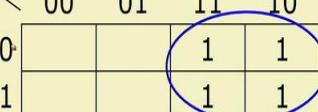
Expression is $\overbrace{ABC+ABC'+A'BC+A'BC'} = \Sigma(7,6,3,2)$

Groups of 4 in a block can be used to eliminate two variables:

ABC	Y	BC			
000	0	00	01	11	10
001	0	0	0	1	1
010	1	0	0	1	1
011	1	1	0	1	1
100	0	0	0	0	0
101	0	0	0	0	0
110	1	0	0	0	0
111	1	0	0	0	0

QUAD = $A'BC + A'BC' + ABC + ABC' = A'B + AB$ = B

Groups of 4



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, next we will see another example; suppose, you have got an expression like say $ABC + ABC' + A'BC + A'BC' = \Sigma(7,6,3,2)$. So, you have got 4 terms like this.

Now, so, this is the function so, we have got so, if you if you look into the corresponding Karnaugh map; so, say 3 variable map and we will have situation like this. So, this $A'BC'$. So, A' so, it will be on this row BC' . So, this one so, this is this is 1, then 0 1 1. So, $A'BC$. So, this is also equal to 1, then we have got ABC' . So this one. So, this is one and then we have got ABC so, this is one.

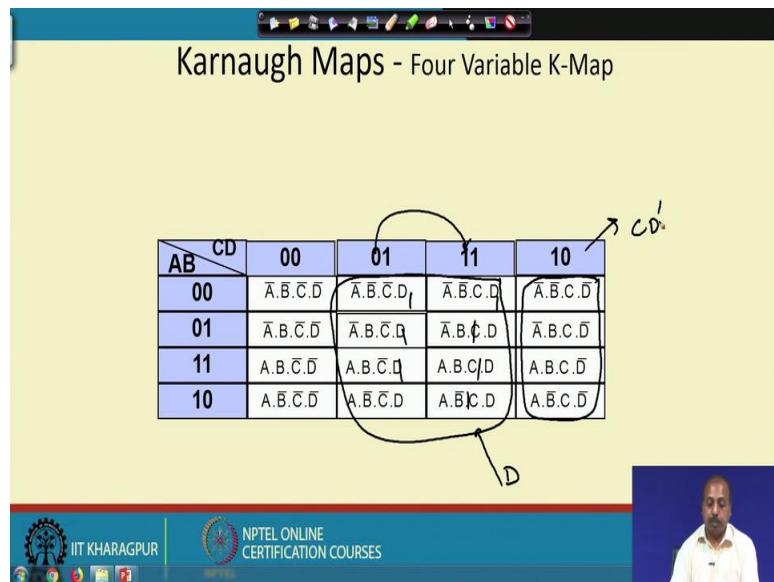
So, ultimately what you get is a 4 ones which are close to each other. So, they are all one hop away from each other. So, you can see that this expression. So, you can whenever we have got more 4 1. So, you can club them together into a group of 4 1's. And then if you do a minimization by using Boolean identities, then you see that this whole term it reduces to B. So, in terms of Karnaugh map you can what you can say in this case, you see A is changing it is polarity from 0 to 1, if you consider this 4 terms A is changing it is polarity. And your C is also changing it is polarity so, it is becoming 1 to 0.

So, what is not changing is the polarity of B. So, this whole quad. So, it can be written as the term the variable which is not changing it is polarity, that is B. So, without doing this algebraic manipulations so, we can directly write that, the term corresponding to this quad will be B. So, we just note down the variables which are not changing it is polarity.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 12
Boolean Algebra (Contd.)

(Refer Slide Time: 00:18)



So, Karnaugh map for 4 variables. So, you can understand what are we going to do. So, this is a in the row side and the column side we have got 4 rows and 4 columns, unlike 3 variable map where we have 4 columns, but only 2 rows. So, since here we have to represent for 4 variables. So, we have got 4 rows and 4 columns in the row. We have got say we write it like this, so this diagonal line.

(Refer Slide Time: 00:51)

K-Map
Reduction Rule

To reduce the Boolean expression, first we have to mark pairs, quads and octets.

Pair – remove one variable
Quad – remove two variables
Octet – remove three variables
Imp – To get the optimum reduction, priority is given to octet first, then quad and then pair.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, below we write the variables that will be represented by the variable that will be represented by the individual values of this rows. So, we have got this AB equal to AB vary. So, AB equal to 0 0 corresponds to the first row, AB equal to 0 1 corresponds to the second row like that. Or similarly on the column side so, CD equal to 0 corresponds to the first column CD equal to 0 1 corresponds to the second column so, like that.

So, we can understand the minterms of the 4 variable functions, they are represented by different boxes in this 4 variable map. So, once we have represented the Boolean function by means of the Karnaugh map. So, we can follow some reduction rule some of that the rules, I have already said like we have to make grouping of this one's, ok. So, to reduce Boolean expression first we have to mark pairs, quads and octets. Octet means the group of 8 ones, quad means group of 4 ones, and pair means group of 2 ones.

So, if we make pair so, we can remove only a single variable, because between a pair only one variable can change it is a polarity. So, if the original function was a 4 variable function then if you are taking a pair. So, you will getting a you will be getting a 3 variable term out of that. Whereas, if you can make a quad. So, 2 variables will change it is polarity one in the row direction another in the column direction so, that way you can remove 2 variables. And if you are if you can make octet, ok. So, you can you can remove 3 variables because in a octet if you it is a 4 variable function then in the octet the 3 variables will

change their polarity only one variable will remain unaltered. So, that way you can do the grouping.

So, if you look into this one like if I can make an octet for example. So, this octet may be like this; so, if all these values are one then I so, all these values are one in my function, then I can make an octet. Now if you look into this octet what is happening is in the row direction, this A is changing it is polarity. So, it is going from \bar{A} to A, B is also changing it is polarity going from \bar{B} to B and all. In the column direction so, \bar{C} C is changing it is polarity it is going from 0 to 1. But only D is not changing it is polarity so, that way this entire octet will be represented by a single literal D if you can make quad say this one.

So, if I make a quad like this as I was telling that this A and B, both of them are changing their polarity. So, they will get eliminated what will remain is this $C\bar{D}$. So, $C\bar{D}$ will remain as the term. So, if I can make so, it is beneficial so, if we can make quads we can make octets, for 4 variable function. So, if the octet is not possible you try to make this type of quads, and quads are also not possible then you try to make pairs. And the pairs are also not possible in that case we have to take as simple single variable.

Like it may so, happen that your ones are distributed like this. So, your ones are distributed like this. So now, you cannot do any pairing also, because you do not get one variable change over between the terms. So, you have to group like these individual terms only. And then there is no minimization that is possible, ok.

So, another important thing that we have is that while getting this octet or this quad or pair like that. So, you have to go in a particular row or a particular column only. So, you cannot make something like this. So, you cannot make a quad like this so, that is not allowed, ok. So, you can do you can make a quad in this format you can make a quad like this or you can make a quad like this ok, but they cannot be like say 2 ones here.

So, as I was showing previously. So, that type of grouping is not allowed. So, you have to group where only variables are changing; so in a row and column fashion. So, if you can, if you cover one you can cover one full row or a full column or multiple rows and multiple columns, but not part of if row and part of another part of one row and part of another row, or part of one column and part of another column to make say quads and all. So that is not allowed because that does not lead to minimization.

(Refer Slide Time: 05:38)

K-Map
Reduction Rule

To reduce the Boolean expression, first we have to mark pairs, quads and octets.

Pair – remove one variable
Quad – remove two variables
Octet – remove three variables
Imp – To get the optimum reduction, priority is given to octet first, then quad and then pair.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, it is important that to get the optimum reduction priority is given to the octet first then quad and then pair. So, if you follow this rules so, it can be proved that you will definitely get a minimum some of product form, ok. Of course, there can be many groupings possible many alternative groupings possible, but all of them leading to some minimal form. So, you can be you can be assured that you will get a minimum size representation of the function.

(Refer Slide Time: 06:11)

Karnaugh Maps - Four Variable K-Map

Octet Reduction

AB \ CD	C'D'[00]	C'D'[01]	CD[11]	CD'[10]
A'B'[00]	1	1	1	1
A'B[01]	1	1	1	1
AB[11]	1	1	1	1
AB'[10]	\	\	\	\

$f = 1$

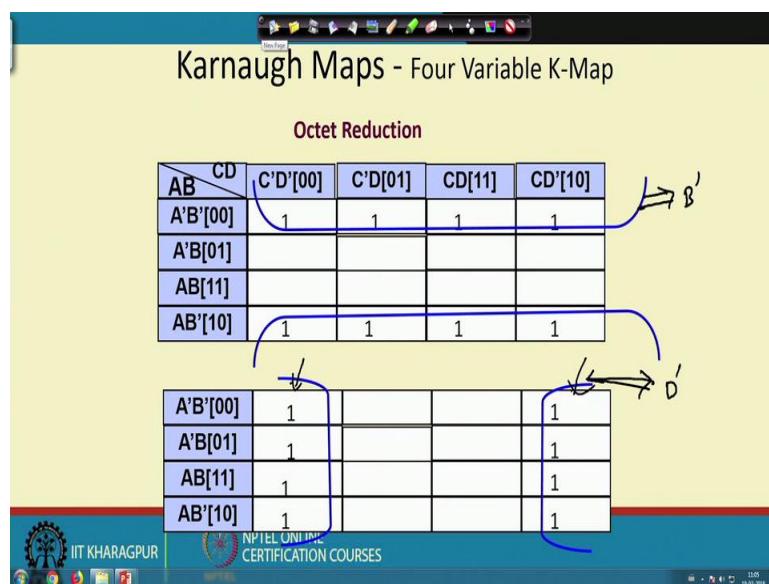
$f = 1$

AB \ CD	C'D'[00]	C'D'[01]	CD[11]	CD'[10]
A'B'[00]	1	1		
A'B[01]	1	1		
AB[11]	1	1		
AB'[10]	1	1		

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the octet reduction so, this is one possibility this is another possibility, ok. So, we can make groups like this or groups like this. So, of course, what will happen if all this all the values are one like, if I have the Boolean function f equal to true. So, f equal to 1. That means, so, in that case all these are 1's. So, you can make this whole thing is a as a group. And now you see that all the variables are changing their polarities. So, naturally the minimized form is f equal to 1, because now we no variable remains which does not change its polarity. So, that is there, but that is obviously, not a not a case to be minimized we normally do not come across such functions.

(Refer Slide Time: 07:07)



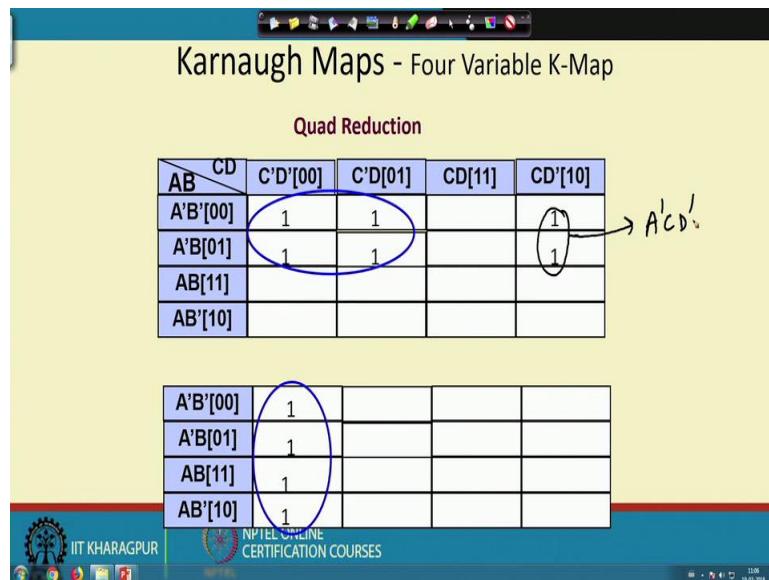
Now, for another possibility of octet reduction as I was telling that the maps are considered to be folded maps. So, you can take this row adjacent to this row, because between these 2 rows only one bit is changing. So, only the A value A variable is changing its polarity from A' to A .

Similarly, between the columns also so, this first column and last column; so they are close to each other they are adjacent to each other so, you can fold your table, that way also. So, you can create this type of octet. So, what will be the minimized form? For this octet so, this is here you see that between these 2. So, A is changing its polarity from \bar{A} to A so, \bar{B} is not changing its polarity. But C and D they are changing its polarity. So, this whole expression for this one is \bar{B} .

Similarly, here when we are taking these 2; so you see that this A is changing its polarity from A, A to one to 0 B is also changing its polarity, between C and D C is changing its polarity because for this for this column. So, this is \bar{C} and for this column this is C, so, what is not changing its polarity is the \bar{D} . So, D remains complemented, here and D remains complemented there also.

So, this one so, this one the term that will get is \bar{D} so, this is way we can do the minimization. So, we should try to maximize the number of octets and when we fail so, no more octet can be formed. So, we try to make a quads, and if quads are not also not possible, then we will try to go to pairs that should be the rule that we should follow so, this is the quad.

(Refer Slide Time: 08:54)



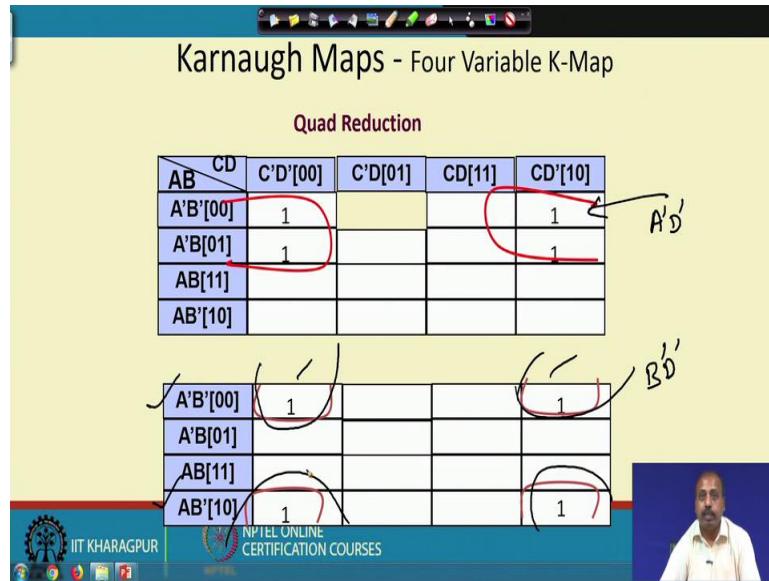
So, this is one possibility so, this is another possibility. So, quad reduction and of course, we can also do this type of grouping like in the previous example you see that these one. So, if you take it in an isolated fashion then you so, not get. So, you will get a pair out of that. So, pair when we do so, you are you are actually taking more number of variables. Like this pair if you represent then this is basically \bar{A} is not changing its polarity. So, $\bar{A} C \bar{D}$ so, that way only B is changing its polarity. So, you can do it like this.

But the point is that you need not be doing it like that. So, you can you can make quads like this also. So, this one and this one because these 2 rows these 2 columns are adjacent

to each other. So, you can take them together, and you can do it in this way. So, you can make a quad out of that.

So, we will take one example.

(Refer Slide Time: 10:03)



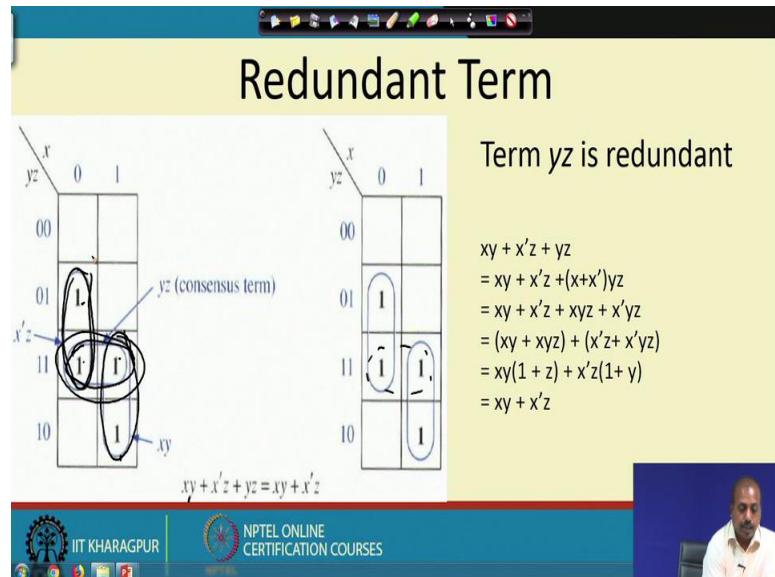
So, here you see that we have got this one this is \bar{A} So, this is a \bar{C} so, this is a $C\bar{D}$ and $\bar{C}\bar{D}$. So, these between these 2 so, you see that your \bar{A} is not changing it is polarity, B is changing it is polarity, and between these two C, C is changing it is polarity, but D is not changing it is polarity. So, you get so, if you take this a red one red quad that we are talking about. So, this quad boils down to $\bar{A}\bar{D}$.

So, you can also make a quad like this. So, you take so, this one this one this one and this one. So, these 4 because these 2 rows are adjacent these 2 rows are adjacent to each other so, I can fold the table that way. Similarly these 2 columns are adjacent to each other so, I can also fold the table this way. So, ultimately after these 2 times folding this 4 ones come close to each other.

So, they are within they are adjacent rows and columns. So, I can do the; I can say that the minimization will be done and this is going to be. So, between these A is changing, it is polarity \bar{B} is not changing it is polarity. So, this is \bar{B} and between these two your \bar{C} C is changing it is polarity, but D is not changing it is polarity so, it is $\bar{B}\bar{D}$. So, if you group

like this if you grouping like this fashion 4 ones like this. So, you can get a quad represented by $\bar{B}\bar{D}$.

(Refer Slide Time: 11:48)



So, this is possible. So, sometimes we introduce some redundant some terms become redundant like see here what is happening is that you see that ok. So, this is a 3 variable Karnaugh map so, where we have got ones like this at this position so, we have got ones. So, this is this one this one and this one.

Now, you may group it somebody while doing since I cannot from quads. So, what I do I form pairs so, I form a pair like this I form a pair like this, somebody also makes a pair like this. So, and in another case somebody says that, I have to cover the ones. So, I have cover the ones like this so, I do not take this pair ok. So, what is the difference between these 2 cases?

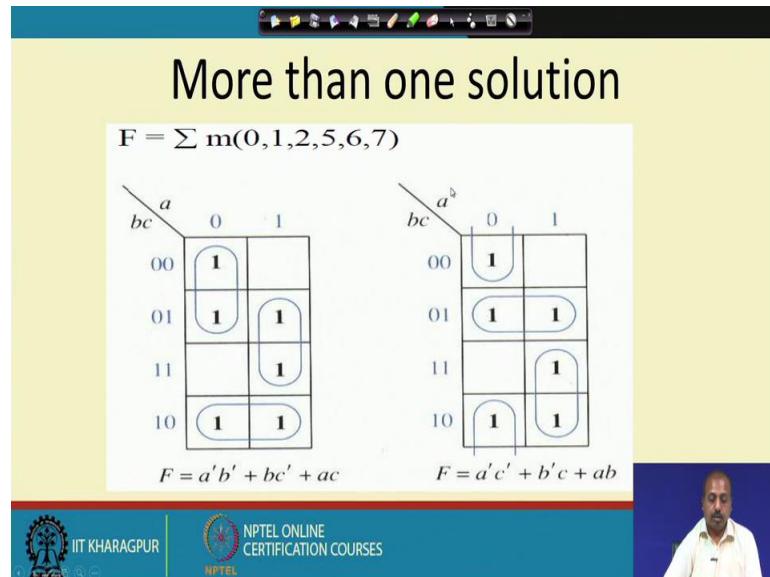
Now, if you do it like this, if you take this third one also this third one also. So, between so, the first term that you get is this one this pair. So, it gives me the term xy , then this pair gives me the term $\bar{x}z$. So, this one this vertical one; so this gives me $\bar{x}z$ and this one gives me yz . So, by taking that I can get then the expression like this.

Now, this is not the minimum form, because between this you can you can show that this is equivalent to $xy + \bar{x}z$. So, how to do? How to do this? So, this yz ; so we can expand it as $x + \bar{x}$, and then do multiplication and then just redo the grouping ultimately it turns out

that these 2 are same. Whereas, the person who has not taken this consensus term yz will make grouping like this. So, which one is correct? So, you see that after when we do this so, we are not getting the minimum form. So, we are getting some extra terms. So, naturally we should not do this type of grouping.

So, here what is happening is that this ones that are being covered by this particular group are already covered by other groups ok. So, they are already covered by other pairs so, this is a redundant term. So, at if one of them was not covered then of course, it is fine. So, this is. So, I can.

(Refer Slide Time: 14:39)



So, that way I can say that, if one of them was not covered. So, if I remove this so, if one of them gets uncovered then of course, you have to cover, but in this particular case this ones covered by the term yz are already covered by the other 2 terms. So, that brings redundancy. So, that should be so, you to get the minimum forms so, that should be avoided.

So, this is so, I can have more than one solution also like as I have said that the Karnaugh map solution is not unique. So, like here you see that for the same function consisting of the mean term 0 1 2 5 6 7 3 variable Karnaugh map. So, this is the 1 placement of ones that I have.

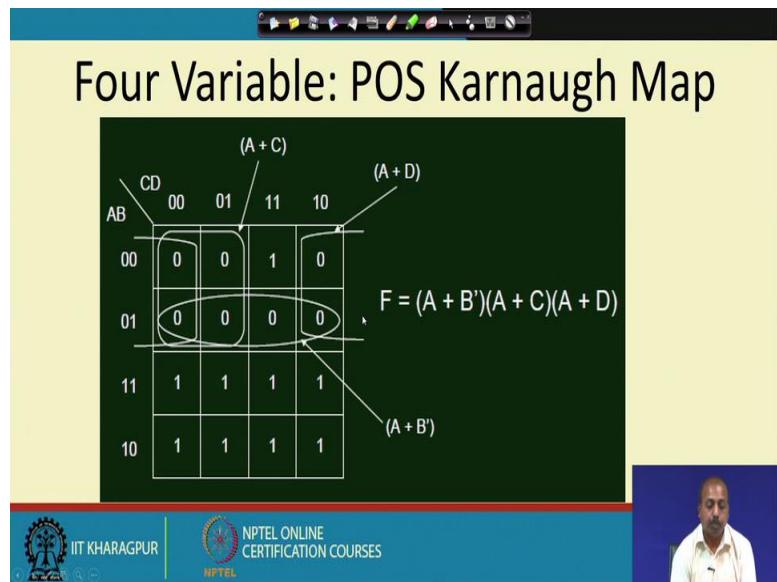
So, now, I can do grouping in different format like one person does the; so, no quad formation is possible so, only pairs are possible. So, while pairing so, I will do it like this. So, this is made one pair this is made one pair this is made another pair. So, that way I get an expression like this so, $\bar{a}\bar{b} + b\bar{c} + ac$

Somebody does a grouping in this format; so, instead of covering like this covers say these, these 2 ones into one group into one pair, these 2 ones into one pair and these 2 ones in another pair. So, that way if we will do the covering, then you will get a different type of cover. So, $\bar{a}\bar{c} + \bar{b}c + ab$.

Now, which one is better; so as far as the implementation is concern so, both of them are equally costly. So, here also the number of literals that we have is 1 2 3 4 5 6, here also number of literals that we have got is 1 2 3 4 5 6 numbers of terms in both the expressions are also same. So, this is there in 3 terms here and here also there are 3 terms; so these 2 expressions there, perfectly similar ok, and if you take it to their equivalent also because they are if you take into the canonical some of product form. So, they will boil down to all these ones that we have in the Karnaugh map that is 0 1 2 5 6 7 so, they are definitely same.

So, that is why we say that the Karnaugh map it gives us the minimal form. So, there can be multiple such minimum results, ok. So, it does there is no discrepancy between them. So, all of them are correct so, you can you can produce any of these grouping and say that the result is correct, ok. So, we will get the correct result?

(Refer Slide Time: 16:40)



Now, how do you so, for product of sum expression. So, in case of product of sum what we do we make groups of 0's ok; so here we in case of to get sum of product representation. So, we were grouping the 1's in case of product of sum will be grouping the 0's. Like here after putting the Karnaugh map, the ones and 0's on to the table. So, we are grouping this 4 0's into one quad this 4 0's into one quad, and this 4 0 that is this 2, and these 2 into one quad. You see if you look into it carefully you see that say this quad, and this quad, they are covering this 4 0's twice. But in both the cases so, there is something new that is happening like if you remove this quad. So, this 0 will become uncovered. Similarly if you remove this quad, this 0 will become uncovered. So, at least 1 0 is there which is which will be uncovered, if we remove the quad. So, it does not.

So, it ensures that we are not introducing redundancy in the process like the example that I so, to some time back where the consensus term. So, that 1's even if we remove the grouping of that consensus term. So, ones will remain covered by some other pairs, but here if we remove any of this quads. So, they will be the at least one, one of 1 0 will become uncovered. So, we have got this type of grouping, and then while writing the expression. So, I we know that between these 2 in case of a say this particular quad, the symbol C is not changing, it is polarity the variable C is not changing it is polarity and variable A is not changing it is polarity.

So, for pos I will be writing in the complemented form so, 0's will be written as true and ones will be written as false as complemented one. So, this one gives me $A + C$, so, this quad is A plus C . Similarly this quad is $A + \bar{B}$ A plus B bar so, this is $A + \bar{B}$ and this is $A+D$ ok. So, this way we can write down the consensus we can write down the product of sum expression from the Karnaugh map. So, that is a good technique by which to get the product of sum expression for Boolean functions.

(Refer Slide Time: 19:10)

Karnaugh maps: Don't cares

- In some cases, outputs are undefined
- We "don't care" if the logic produces a 0 or a 1
- This knowledge can be used to simplify functions.

If SW1 is on, L1 is on
If SW2 is on, L2 is on

SW1	SW2	L1	L2
X	1	1	1
1	1	1	1
1	0	1	0
0	X	0	0

- Treat X's like either 1's or 0's
- Very useful
- OK to leave some X's uncovered

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Sometimes we have got a Boolean function that has some of the terms as don't cares. So, like so, it may be the case that outputs are undefined. So, we don't care whether the logic value produced is 0 or 1. So, it may so happen that we may be interested in a say. So, we may say it like this that if say switch one is on the switch one is on, then light L1 is on, if switch 2 is on, then light L2 is on. So, suppose we tell only these 2 terms. So, if you are drawing a truth table then it is switch 1, switch 2, light 1 light 2. From this expression it says that if switch one is on whatever be the condition of switch 2 light one should be on. So, I am not bothered about the condition of switch 2. Similarly, if switch 2 is on whatever be the condition of switch one the light 2 should be on.

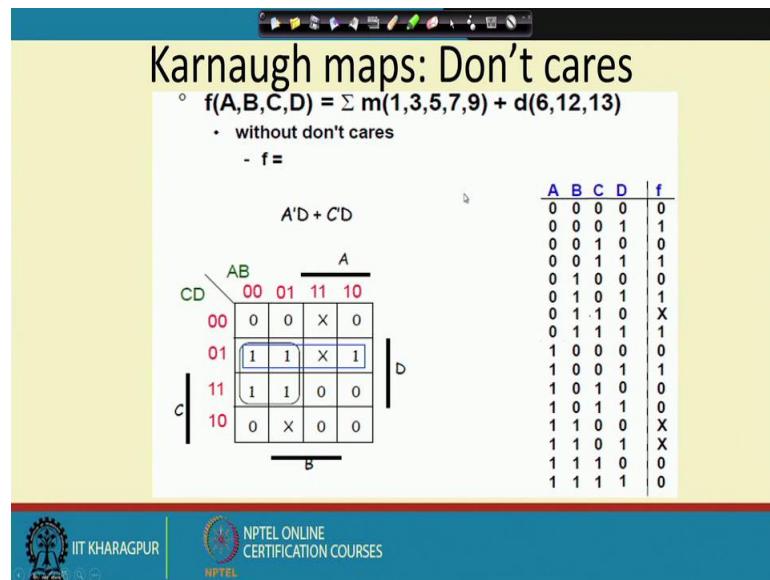
So, that way I say that so, this is the don't care condition. So, you don't care the condition of the switch 2 for controlling light one, similarly we do not consider the status of switch one for getting the status of light 2. So, this way we can have some of the terms as don't care. So, outputs are undefined. So, in those cases; so in some cases; so we can take it as

any value. So, we can take so this value so whenever drawing the truth table. So, for 1 0 so, we can say this is light 2 value is not known. So, light 2 may be on, light 2 may be off. So, like that so, that way. So this, don't care so, this is actually for those combinations. So, they I can treated treat them as 0 or 1 arbitrarily. So, this is the function where we have said this AB and CD so, this AB. So, this AB is 00, 01, 11, 10, CD is 00, 01, 11, 10, and the function value. So, this so, if the AB is 0 0 and CD is 01, then it is one AB 01 CD 01 is also one. But if AB equal to 1 1 and CD equal to 0 0, then we are not bothered about what is the value of the function. So, that is represented as X in the truth table, ok.

Similarly, for AB 1 1 CD 01 we are not bothered about the output of the function. So, this is also X while considering while doing the minimization. So, we can treat this X's either as ones or as 0's; like, say if this X so, between these 2 X's. So, if I take these X as 0 and this this X as 1, then I can I not be bothered about realizing this 0. So, I can just I can make a quad out of that, but if I take this one as 0 I cannot make a quad. So, in that case I have to if I take this as 0, then I have to do a pairing like this or say this 2. So, I can do a I have to so, this is 0. So, I have to do a pairing in this fashion.

But if this is one, if I take this X as one then I can do make a quad formation like this. Similarly, for this X so, if I take a 0, I do not need to do anything for this row, but if I take this as one then of course, this has to be covered in some sense I have to introduce some term them. So, that way this X's to the facility of this to facilitate this minimization process, this X's can be taken as 1's or 0 it depending upon the situation.

(Refer Slide Time: 23:29)



So, you will take some examples and see, say this function ABCD so, these are the minterms 1 3 5 7 9, and this, this 6 12 and 13 so, they are don't cares. So, on the output column we have got this 6 12 and 13. So, this values has don't cares, and then we can say that the corresponding truth table. So, it will be looking something like this.

Now, while making the groups what we do so, this X is treated as 1. So, I make a quad out of that similarly this 4 are ones. So, this quad will remain, but I can take this X as one and make a quad out of that. Similarly, this X is take it as 0 so, that I do not have to do a covering. So, I can get the minimization form like this quad gives me like $\bar{A}D$ the first one and this is the blue square. So, this is the blue quad it gives me $\bar{C}D$. So, this way I can realize the functions by fruitfully exploiting the do not cares to be either equal to 0 or 1.

(Refer Slide Time: 24:37)

Don't Care Conditions

- In some situations, we don't care about the value of a function for certain combinations of the variables.
 - these combinations may be impossible in certain contexts
 - or the value of the function may not matter in when the combinations occur

hexadecimal value $\geq 0 \rightarrow F$

$d_4 \ d_3 \ d_2 \ d_1 \ L$

$0 \ 0 \ 0 \ 0 \ | \ L$

$0 \ 0 \ 0 \ 1 \ | \ 0$

$0 \ 0 \ 1 \ 0 \ | \ 0$

$0 \ 0 \ 1 \ 1 \ | \ 0$

$0 \ 1 \ 0 \ 0 \ | \ 0$

$0 \ 1 \ 0 \ 1 \ | \ 0$

$0 \ 1 \ 1 \ 0 \ | \ 0$

$0 \ 1 \ 1 \ 1 \ | \ 0$

$1 \ 0 \ 1 \ 0 \ | \ 1$

$1 \ 0 \ 1 \ 1 \ | \ 0$

$1 \ 1 \ 0 \ 0 \ | \ 0$

$1 \ 1 \ 0 \ 1 \ | \ 0$

$1 \ 1 \ 1 \ 0 \ | \ 0$

$1 \ 1 \ 1 \ 1 \ | \ 0$

$10-12$

< 10

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, don't care condition in some situation we don't care about the value of a function at certain combinations of the variables, this combinations may be impossible in certain contexts, or the value of the function may not matter in when we combinations occur when the value the value comes is does not matter. For example, if we are say, if we are say doing a say conversion, from say one code to another code, ok. So, say I have I am doing a conversion like if I have if I. So, this hexadecimal number system so, hexadecimal value; so this hexadecimal value can go from 0 to f.

So, to represent this 0 to f, I need 4 bit value. So, this input is 4 bit. So, it is so, d1, d2, d3 and d4. And I say that whenever the value is less than 10 whenever the value is less than 10 one light will glow. And whenever the value is more than 10 so, whenever the value is more than 10, then the light will not glow light will be off. So, for the combinational like 0 0 0 0 to 10 so that is. So, 1010, up to this combination the light value is equal to 1. So, all these are equal to 1, then for from 1011. So, the light value is 0 to 1 1 1 1 the light value will be 0. So, it is less than 10.

So, may this so, this in this case it is completely specified, but suppose it says that for light value less than 10 the light. So, if the light value is less than 10 then the light should be on if the input value is less than 10 the light value should be on. And also if the input value is input value is a between 10 to 12, between 10 to 12; then also then the light value should be off.

(Refer Slide Time: 27:00)

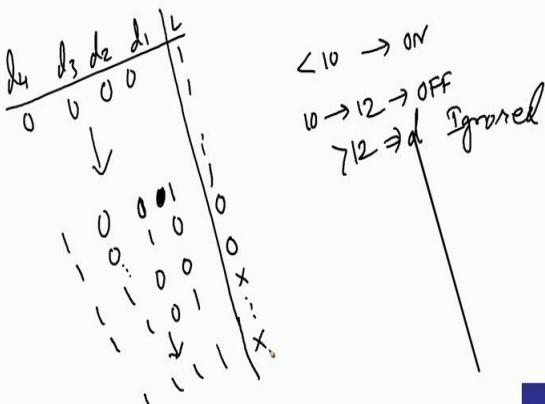
Don't Care Conditions

- In some situations, we don't care about the value of a function for certain combinations of the variables.
 - these combinations may be impossible in certain contexts
 - or the value of the function may not matter in when the combinations occur
- In such situations we say the function is *incompletely specified* and there are multiple (completely specified) logic functions that can be used in the design.
 - so we can select a function that gives the simplest circuit
- When constructing the terms in the simplification procedure, we can choose to either cover or not cover the don't care conditions.



So, between 10 to 12 the value should be off and so, if the light value is less than 10, then it is it must be on.

(Refer Slide Time: 27:07)



And if light value is between 10 to 12, then the light value then the light should be off if the decimal value is between 10 to 12, the light should be off. However, for the value greater than 12, it does not matter. So, it is it can be ignored, it does not matter.

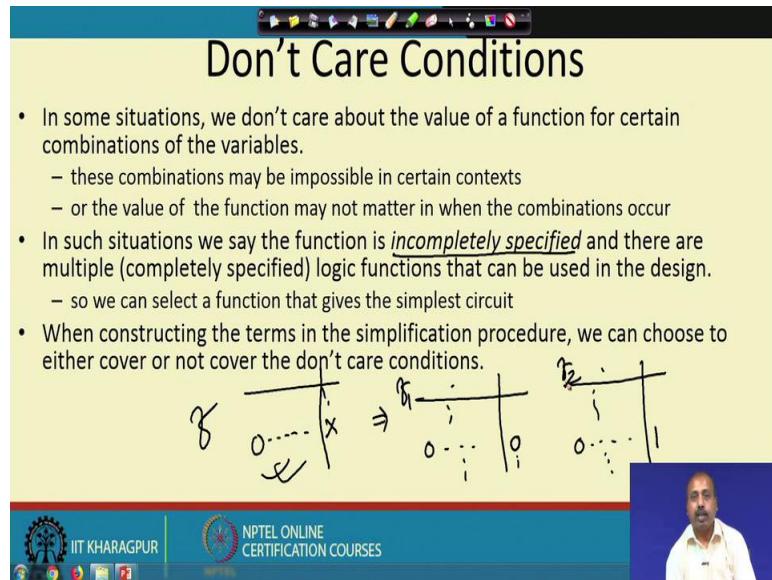
Then the truth table that we were drawing previously, so, it is d1 d2 d3 d4, and this is the light. So, for 0 0 0 0 to 0 0 so, 1010 so, up to this much the light value is one then for 10 to 12 for less than 10. So, it is not so, it is not this is 0 up to 9 it should be one, from 1010 the light value should be off to 12. So, that is 1100, from 101 0 to 1100 the light value should be off, but from 1101 to 1111 so, in this region the light value is undefined so, they are all do not cares.

So, this way we can introduce my problem statement be such that there are do not cares, and if do not cares are there. So, we can exploit it in the minimization process.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 13
Boolean Algebra (Contd.)

(Refer Slide Time: 00:18)



So, in case of this don't care conditions so, what is happening is that; the original function that we are given is incompletely specified function. So, this is an incompletely specified function, because for some of the combinations of input so, we did not mention what is the output. So, if the output can be anything so, the designer is not bothered about the output for those cases.

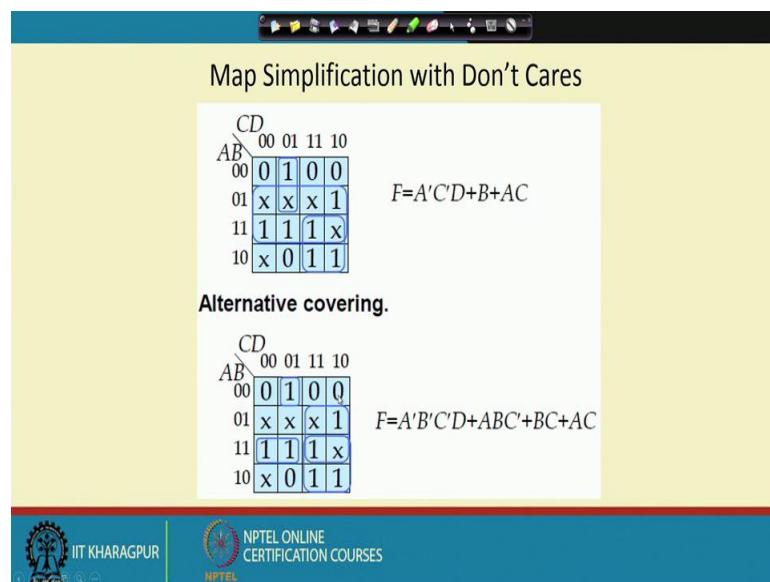
So, in that sense I can get multiple functions. So, truly speaking so, if this is the truth table, then suppose for say this particular combination the output value is don't care. Then it essentially says that I can have 2 different truth tables. So, in one case the rest of the row column so, they get the values from here, but for this particular column. So, if the value was x. So, I make it 0, rest of the thing remain same as this truth table. And in another case so, I take this one and for this particular combination so, I take the value as one.

So, essentially this is one function this is another function. So, this is the original incomplete function f , from their I can derive 2 completely specified function f_1 and f_2 . And we can minimize them, and see whichever gives as the minimum result. So, while we

are taking this x is in our grouping so, it will automatically take the representation that we will have the minimum simplest represent, simplest representation in terms of number of literals and number of terms.

So, we can have we can select a function that gives the way the simplest form. And when constructing the terms in the simplification process, we can choose to either cover or not cover the don't care conditions. So, that way we can choose between the alternate functions.

(Refer Slide Time: 02:05)



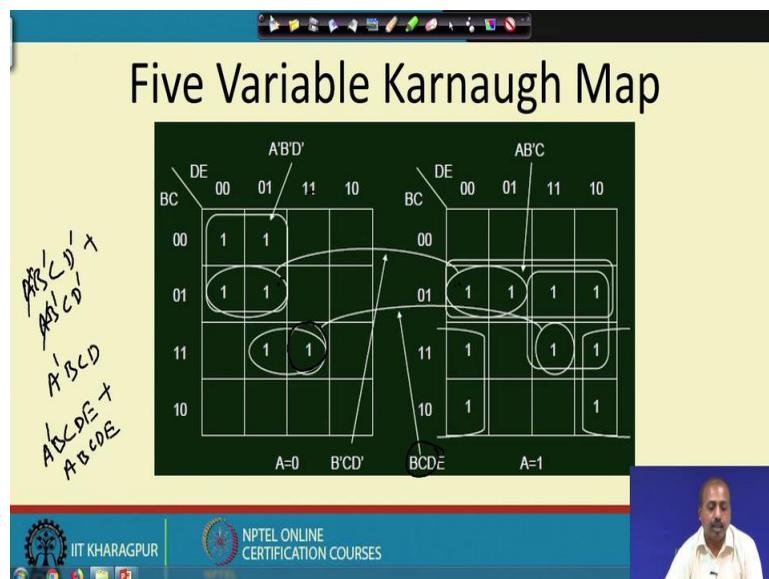
So, this is map with some don't care. So, you see that in this case so, so, this don't cares we have fruitfully exploited to make a quad here ok. So if so, that way this quad gives me like a only B is not changing it is polarity all the other variable they are changing their polarity so, that is B.

Similarly, this quad so, this sorry, sorry this so, this is an octet so, containing 8 1s. So, this is an octet so, that gives me B and this is a quad. So, that here A is change not changing it is polarity, and C is not changing it is polarity so, this is AC. So, B +AC and this pair so, A is not changing it is polarity. And your with A is not changing it is polarity, and this C and D, they are not changing their polarity. So, $\bar{A}\bar{C}D$ is for this pair so, accordingly we can do this thing.

Another possible or covering may be like this; so, somewhat it has a covering in this fashion ok. So, this is this is one singleton, singleton one so, that is $\bar{A}\bar{B}\bar{C}D$ like this, then we have got this $\bar{A}\bar{B}\bar{C}$. So, we take this pair so, it is $\bar{A}\bar{B}\bar{C}$ then we have say BC like say this if I take this term. So, A is changing it is polarity B is not changing so, B here and then between these 2 columns C is not changing it is polarity. So, you get a quad BC, and for this one you get a quad AC. So, this way we can write down a set of quads and singletons and pairs so, that way we get it like this.

So, definitely we can understand that this grouping is not good, because it does not fruitfully exploit the don't cares here, ok. So, that way the resulting solution is not very good.

(Refer Slide Time: 04:08)



Next we will look into 5 variable Karnaugh map. So, for we have look seen up to 4 variables. So, for 5 variable Karnaugh map what is done? We take groups of we take groups of 2 Karnaugh maps, one is for A equal to 0 another is called A equal to 1.

So, for A equal to 0 so, we have got a this particular map or A equal to 1 we have got a map like this. And then we have this one so, this is map is $\bar{A}\bar{B}D$, because for all these terms here A is equal to 0. So, they will come as A bar A bar terms $\bar{A}\bar{B}\bar{D}$, and this will come as $\bar{A}BC$. So, this will come as so, A is the A is remaining unchanged so, A is remaining as \bar{A} . So, between these 2 columns E is not changing it is polarity. So, this term E remains and

between a also between these 2 between these 2 rows our B B is changing it is polarity, but C is not changing.

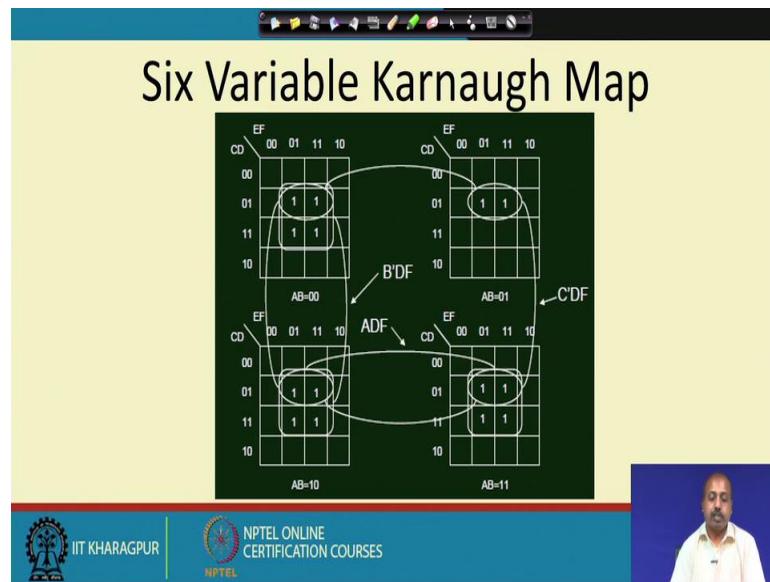
So, if we consider only this map. So, you get this term this pair leading to the term \bar{A} , then it is the between these from here you will be getting say $\bar{A}BC$, and from here you will be getting D. So, \bar{A} is definitely their then this particular it is BC and between these 2 columns. So, we have got so, D is not changing it is polarity so, D is coming as D. So, you were getting BCD, $\bar{A}BCD$.

Now, now this $\bar{A}BCD$ so, then you can find that from this map. So, this BCD is also their so, if we if you take this term. So, if you take. So, if you for example, if you take say this singleton. So, this is your BCDE, now this BCDE, and this BCDE so, here also in for a equal to 1 map we have got BCDE. So, this from the first map so, I am getting it as A bar BCDE. For the second map I am getting it as ABCDE. So, when they are combined so this, $A + \bar{A}$ will make it one so, ultimately we will get this BCDE. So, in this way by combining these ones from 2 different map. So, if the ones are at same position so, you can combine them to make pairs, ok.

So, similarly say this so, this term is this pair this quad is way of itself. So, it is A bar BD A bar B bar D bar, similarly so, this these pair these pair and these pair. So, they are located at the same place so, they are from the first map so, we were getting the term as $\bar{B}C$ and here the D is not changing it is sign. So, $\bar{B}C\bar{D}$ so, \bar{A} in front of that because it is from the first map, and then from the second from this one. So, for this term you are getting the thing like $\bar{B}C\bar{D}$ and A in front of that.

So, if you combine these 2, if you combine these 2 then this \bar{A} will cancel out. So, you will get the term $\bar{B}CD$. So, by combining this 2 pairs, we can form a quad and you can get the term $\bar{B}C\bar{D}$. So, this is slightly cumbersome, but it is still we can used it, we can use it for combination across the tables.

(Refer Slide Time: 08:35)



But it is not that elegant you can say, but it will be useful. Going to 6 variable map, so, you can; so, we have got 4 such maps, where AB equal to 0 0, 0 1, 1 0, and 1 1 they are 4 maps. And again after making the individual grouping of pairs quads etcetera. So, you have to see whether there exist a similar pair or quad or singleton, or octet in the other maps also. So, if it there then you can combine them together to get a minimized form ok. So, that way we can use this 6 variable map for doing the minimization.

So, here you see that these 2 these 2 pairs. So, 1 1 here and 1 1 here so, they are at the same place. So, we can combine them, similarly say this octet and this quad this quad. So, this quad is present in a here as well as their so, we can combine these 2 quads together. Similarly, this quad is present at in this map and this map, and between these 2 maps only one variable is changing its polarity. So, you can combine this, but you cannot combine say this quad with this quad, because here AB is 0 0, and here AB is 1 1. So, the both the variables are changing their polarity. So, you cannot do this combination.

But you can do that as long as they are within the so, within similar within the quads are with the same locations of 2 maps which are varying only in one bit, ok. So, that way we can do that.

(Refer Slide Time: 10:08)

Simplification using Map-Entered Variables

- Extend K-Map for more variables
- When variable 'E' appears in a square, if E=1, then the corresponding minterm is present in G.
- $G(A,B,C,D,E,F) = m_0 + m_2 + m_3 + Em_5 + Em_7 + Fm_9 + m_{11} + m_{15} + (\text{don't care})$

$\begin{array}{c ccccc} & AB \\ \backslash CD & 00 & 01 & 11 & 10 \\ \hline 00 & & & & \\ 01 & X & E & X & F \\ 11 & & E & 1 & 1 \\ 10 & & & X \end{array}$	$\begin{array}{c ccccc} & AB \\ \backslash CD & 00 & 01 & 11 & 10 \\ \hline 00 & & & & \\ 01 & X & & X & \\ 11 & & 1 & (1) & 1 \\ 10 & & 1 & & X \end{array}$	$\begin{array}{c ccccc} & AB \\ \backslash CD & 00 & 01 & 11 & 10 \\ \hline 00 & X & & & \\ 01 & X & 1 & X & \\ 11 & X & 1 & X & X \\ 10 & X & & X \end{array}$	$\begin{array}{c ccccc} & AB \\ \backslash CD & 00 & 01 & 11 & 10 \\ \hline 00 & X & & & \\ 01 & X & X & 1 & \\ 11 & X & X & X & X \\ 10 & X & & & X \end{array}$
G	$E = F = 0$ $MS_0 = A'B' + ACD$	$E = 1, F = 0$ $MS_1 = A'D$	$E = 0, F = 1$ $MS_2 = AD$



IIT KHARAGPUR



NPTEL
NPTEL ONLINE
CERTIFICATION COURSES

Sometimes, we can do this simplification using some map entered variable. So, if we have got more number of variables, you can do that using some map entered variable concept. So, so this is a situation like I have got a 6 variable map, where the terms are expressed like this. So, $m_0 + m_2 + m_3 + Em_5 + Em_7 + Fm_9 + m_{11} + m_{15}$, that is E, E equal to 1, then m 5 will be coming if E equal to 7, then if E equal to 1 then m7 will come. If F equal to 1 then m9 will come, then m 11 m 15.

So, this is represented as 4 variable terms like m0 to m15, but here I have got the other variables introduce the E F etcetera, so, this is actually a 6 variable function, but while writing the function, so, we are writing it in we can we can write it in next less number of variable so, Em5 Em7 so, in the map we write it like this. So, this don't cares are not written here explicitly the don't cares are there. So, these 2 these 3 are the don't cares that we have. So, they are retained as it is, but this E the in place of m5 we entered E in place of Fm9 so, entered F and in place of m7 also we entered E.

Now, we consider the case where E and F both are equal to 0. If E and F both are equal to 0 so, you get so, you get a map and we do the grouping like this. So, you get $\bar{A}\bar{B} + ACD$. Now if E equal to 1, if E equal to 1 and F equal to 0 so, we get a map now here, what we are doing is that we are not considering these 1's. So, we are taking it as don't care, because this one is already covered by these situations. So, this m0 term so it does not depend on

the value of E and F; so that has already been covered here, so, they are taken as don't cares.

So, as soon as some ones are covered so, we can take them as don't cares in the successive formulation.

(Refer Slide Time: 12:20)

An Example

- $F(A, B, C, D) = A'B'C + A'BC + A'BC'D + ABCD + \text{don't care}(AB'C)$
- Consider D as a map-entered variable
- When $D = 0$, $F = A'C$
- When $D = 1$, $F = C + A'B$
- $F = A'C + D(C + A'B) = A'C + CD + A'BD$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is another example. So, ABCD is say this one, $\bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C}D + ABCD + \text{don't care}(A\bar{B}C)$ so, this is if we consider D as the map entered variable. So, then when D equal to 0 so, if put D equal to 0 here so, you will get the function $\bar{A}B$ is a so, you get the function as $\bar{A}C$ like if we just take this variable if we take this function and put D equal to 0.

So, these 2 terms will cancel out so, you are getting $\bar{A}\bar{B}C + \bar{A}BC$. So, this $B + \bar{B}$ will cancel out so, get $\bar{A}C$, and when D equal to 1, so, we are getting F equal to $C + \bar{A}B$ so, this is also obtained from this one, say when we take in a map entered variable. So, when D equal to 1 so, these 2 terms are coming into picture the first 2 term $\bar{A}\bar{B}C$. So, $\bar{A}\bar{B}C$ is coming here so, this is \bar{A} , and this is $\bar{B}C$ so, this is one.

Similarly, $\bar{A}BC$ so, \bar{A} and BC so, these 2 are ones. So, we have got $A\bar{B}C$ as don't care that is there, but for the other 2 terms so, we call them map entered terms. So, $\bar{A}B\bar{C}D$ so, when D equal to 1, then this is coming. So, this \bar{A} so, this D actually so, $\bar{A}B\bar{C}$ so, that is coming here and for D equal to 1 it is ABC. So, we get the map like this and then we do a

minimization so, first in this map so, we take this 2 as ones because they are always present. So, we take them out so, it is independent of so, that will give us the term $\bar{A}C$. So, $\bar{A}C$ will definitely come, then for the second term so, we get this one it is a map entered variable and D. So, these ones are to be converted to don't cares, because they are already covered by the map where D is not D is not considered at all.

So, we don't need to do it again so, D so, for these ones are taken as don't cares here. So, you get now D equal to 1 so, you get these two D as ones ok. So now, if do a grouping so, you get one term as this one, which is the nothing but C ok. So, that is nothing but C, and from this pair you will get $\bar{A}B$. So, $C + \bar{A}B$, and then that is. So, this map is coming when D equal to 1 so, it is multiplied by D.

So, then if we expand it you will get $\bar{A}C + CD + \bar{A}BD$. So, this way we can. So now, if we consider the 4 variable map for the original function, fine? So, this one so, if you take the 4 variable map so, you see that $\bar{A}\bar{B}C$. So, it is irrespective of D when it is 0 0, AB is A is 0 and B is 0, then it is independent of this as D value the corresponding bits will be turned on. So, this is the corresponding Karnaugh map. And here, if you do the grouping you will see that we will be getting back with this original term so, original these 3 terms always.

So, this 4 variable k maps so, we can solve it using a series of 3 variable, k maps is it taking help of this map entered variable concept. Though of course, with the increasing number of variables so this will also become cumbersome. But up to certain lengths so, up to certain number of variables so, you can try it out.

(Refer Slide Time: 16:07)

Implicants

- Must cover all 1s of the function on K-map
 - Each 1 can be used multiple times in generating the terms of the expression
- When we group 1's on a K-map, generating a term, that term is an *implicant* of the function
 - Prime Implicants
 - Essential Prime Implicants

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, another view of k map so, we can we can view this k map as collection of implicants. So, an implicant so, whenever we are doing this Karnaugh map covering so, we must cover all ones of the function on the Karnaugh map. So, you while doing the minimization or writing down the corresponding Boolean expression for a Karnaugh map. So, you cannot say that some of the ones will remain uncovered I cannot take the term such that some of the ones are uncovered.

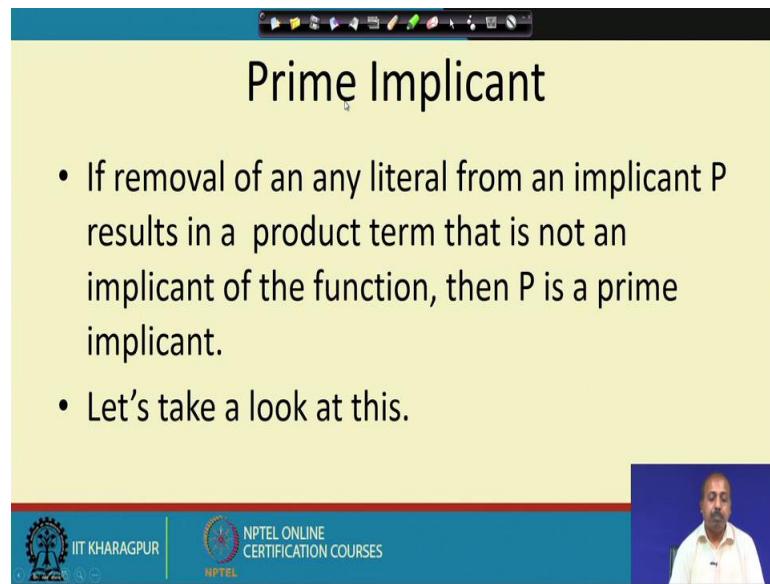
So, each one can be used multiple times in generating the terms of the expression. So, that can happen because were one. So, if you look into a Karnaugh map, then if this is a Karnaugh map and if this particular empty is one so, it may be covered by several pairs. So, it may be covered like this also it may be covered like this it may be covered in a bigger quad or like that. So, that may be that way. So, this particular one may be generated by all the terms. So, here it is covered by 3 terms so, whenever is so these all the 3 terms can generate this particular one, that is why it said that each one can be used multiple times in generating the terms of the expression.

Now, when we group ones on a Karnaugh map generating a term that term is called an implicant of the function. So, what is an implicant? So, implicant means that so, it is that term can generate some one in the Karnaugh map. And this implicants may be we can classified them into prime implicants and essential prime implicants so, we will see what do they mean.

(Refer Slide Time: 17:46)

Prime Implicant

- If removal of an any literal from an implicant P results in a product term that is not an implicant of the function, then P is a prime implicant.
- Let's take a look at this.



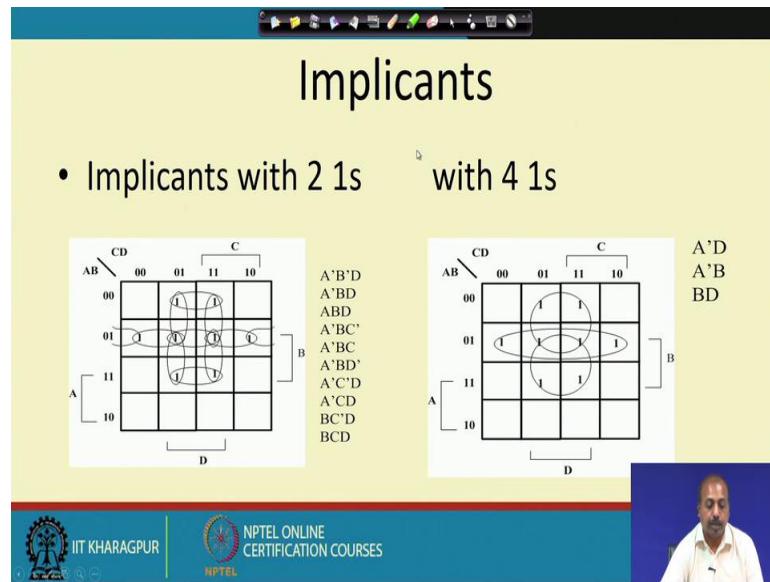
IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, a prime implicant is one if we remove any literal from the implicant then it will result in a product term that is not an implicant of the function so, in that case p is called a prime implicant. So, implicant means something that generate some one now if you remove some terms.

(Refer Slide Time: 18:08)

Implicants

- Implicants with 2 1s
- with 4 1s



IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if it leave some of the so, ones uncovered then it will be called a prime implicants. Like see these are the implicant like say suppose this is a Karnaugh map. So, with implicants with 2 ones these are all the implicants like that, ok. And if I so, here I have

purposefully group them into pairs. So, that I can all the implicants are covering exactly 2 ones here, ok. So, these are the implicants so, these are all implicants.

Now, if you take groups of 4 then we will be getting like this. So, these are also implicant because they are covering some of the ones ok, but whether they are essential or not that we will see. So, consider this term $\bar{A}\bar{B}D$ the first one. So, $\bar{A}\bar{B}D$ so, this pair we are talking about. Now, so, if we remove \bar{A} we get $\bar{B}D$. So, if I take this $\bar{B}D$, if I take this $\bar{B}D$, then what will happen is that so, $\bar{B}D$ so, \bar{B} is this row and while so \bar{B} is this one and this one and D is this one. So, $\bar{B}D$ means these ones will also come into picture so, they are actually not there in the function, fine.

(Refer Slide Time: 19:32)

Which are prime implicants?

- Consider
 - $A'B'D$
 - Remove $A' \rightarrow B'D$ THUS meets previous statement
- However, on an n-variable map, the set of prime implicants corresponds to the set of rectangles made up of 2^m squares containing 1s with each rectangle containing as many squares as possible.
- Thus, $A'B'D$ is not a prime implicant. It is an implicant, just not a prime implicant.

So, it says that if we are removing this $\bar{A}\bar{B}D$. So, $\bar{B}D$ will cover the original mean terms like $\bar{B}D$ also covers these 2 ones. So, that way it covers ones so, it can be an implicant. However, on a n variable map the set of prime implicants corresponds to the set of rectangles made up to **2 power m** squares containing ones with each rectangle containing as many squares as possible. So, $\bar{A}\bar{B}D$ is not a prime implicant. So, because if we remove this, then the it will be it will be generating a bigger sized bigger sized quad in fact, we can say like here it was a pair $\bar{A}\bar{B}D$ so, if I remove \bar{A} so, I am getting $\bar{B}D$. So, this is this corresponds to a quad like say this means that these 2 ones and so, I am talking about these 2 1's and these 2 1's.

So, that way so, that will be giving rise to a quad so, the size increases. So, as the number of literals reduces the size of the covering increases, ok. Size of the rectangles increases so, it says that this is not a prime implicant, because if you take it as if you remove, if you remove this then some of the implicants are some of the ones are getting introduced so, which is not there in the original function. So, this is an implicant, but not a prime implicant. So, $\bar{A}\bar{B}D$ is an implicant, but not a prime implicant so, if you do this if you take it is for a prime implicant.

So, we should have the set of rectangles made up to 2^m squares, containing ones with each rectangle containing as many squares as possible. So, you cannot maximize it further that will be called a prime implicant.

(Refer Slide Time: 21:28)

Prime implicants of the function

- Have
 - $A'D$ $A'B$ and BD
 - Removal of any literal from any of these terms results in an implicant that is not implicant of the function.
 - They also cover all 1s of the function and are not contained in some larger implicant
 - These are the prime implicants of the function

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

So, this $\bar{A}D$, $\bar{A}B$, and BD ; so, if you remove any literal from this terms it will results into an implicant that is not an implicant of the function, ok. So, they cover all ones of the function and are not contained in some larger implicant so, these are the prime implicant so, we will go back to the example and see. Like you see that these are implicants so, these are also implicants, but this particular implicant. So, these $\bar{A}\bar{B}D$ is contained in the larger implicant $\bar{A}D$. So, this is that $\bar{A}D$ implicant so, this is covering this lower sized implicant. So, for an implicant to be a prime implicant, it should not be covered by; it is it should not be possible that we cover it by a larger sized implicant.

So, that is why this $\bar{A}\bar{B}D$ is not a prime implicant, but $\bar{A}D$ is a prime implicant. So, if you remove any of the literal from $\bar{A}D$. So, it will no more remain an implicant of the function, because some 0s will be taken as one so, that is not correct. How and also we cannot reduce it you cannot reduce it further like this if the size of this $\bar{A}D$ cannot be reduced further without undefined the function, but this $\bar{A}\bar{B}D$ so, it is contained in this $\bar{A}D$ in this rectangle, and we have got this we so, this is this is a this is an implicant $\bar{A}\bar{B}D$ is an implicant, but is not a prime implicant, but $\bar{A}D$ is also a prime implicant.

So, this $\bar{A}D$, $\bar{A}B$, and BD so, they are prime implicant because removal of any literal from any of these terms results in an implicant, that is not implicant of the function, and they cover all ones of the function and are not contained in some larger implicant, ok. So, they these A bar D A bar B and BD so, they are covering all ones, but none of the ones are remaining uncovered and they are not contained in some large sized implicant so, these are called prime implicant of the function.

(Refer Slide Time: 23:41)

Some general statements on PI

- A single 1 on a map is a prime implicant if it is not adjacent to any other 1 of the function.
- Two adjacent 1s on a map represent a prime implicant, provided that they are not within a rectangle of 4 or more squares containing 1s.
- Four 1's that are an implicant are a prime implicant if they are not within a group of 8.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this way I can say that Karnaugh map is nothing but a collection of implicants, out of them are prime implicants and some of them are essential prime implicant. So, what you have to do is that while after drawing the Karnaugh map; so the ones that we are putting in the Karnaugh map. So, they are actually the implicants, now you have to try to do grouping. So, such that you can get some large sized implicants, ok. And then you will learn up into a set of prime implicants, and this prime implicant means from they are if

you cannot remove anything. So, ultimately we are looking for getting the prime implicants.

So, 2 adjacent ones on a map represent a prime implicant provided, they are not contained within a rectangle of 4 or more squares of squares containing ones. Similarly, 4 ones that are an implicant are a prime implicant, if they are not covered within a group of 8. So, you can just go back to this example so, these are all implicants. So, but it is not a prime implicant, because it is covered in some larger sized implicant.

So, similarly this 4 1's so, this is forming a quad so, this is a prime implicant, because it is not covered into some some sort of octet. So, if it is for example, if the in the is original function if these 2 bits were also one. So, these 2 combinations are also one then these whole thing would have formed a formed an octet.

So, in that case $\bar{A}D$ and so this one and this one so they will not remain your prime implicant anymore, but they will be simply implicants. So, that is said here that 2 adjacent ones on a map they will represent a prime implicant provided they are not within a rectangle of 4 or more squares containing ones. And similarly 4 ones that are implicant, so that can that can be a prime implicant only if it is not containing within a group of 8. So, this way it goes on this prime implicant will go on.

(Refer Slide Time: 25:50)

The slide has a yellow header bar with a toolbar icon. The main title 'Essential Prime Implicant' is centered in a large black font. Below the title is a bulleted definition: '• A prime implicant that contains a 1 that is not covered by any other prime implicant of the function is an essential prime implicant. IT MUST BE INCLUDED IN ANY MINIMAL REPRESENTATION OF THE FUNCTION.' At the bottom, there is a blue footer bar with the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' next to the NPTEL logo.

Essential prime implicant a prime implicant that contains a 1, that is not covered by any other prime implicant of the function is an essential prime implicant. And it must be included in any minimal representation of the function. So, this prime implicants they must be included in any minimal, so, you cannot you cannot replace them. So, you cannot avoid them in the minimization process so, they are called essential prime implicants.

(Refer Slide Time: 26:18)

Example

- Consider the example
- Prime Implicants
 - $A'D$
 - BD'
 - $\bar{A}B$
- Essential Prime Implicants
 - $A'D$ ✓
 - BD' ✓
- So $F = A'D + BD'$ ✓

IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES

So, like in this case we have got prime implicant $\bar{A}D$, $B\bar{D}$, and $\bar{A}B$. So, these are all prime implicants out of that say this $\bar{A}D$ and $B\bar{D}$ so, these are they are essential prime implicant. So, you see that while doing this grouping where, this is somebody has done a grouping like this a grouping like this as well this grouping. So, from that map itself you can understand that there is some redundancy here like, so for example, the ones have been covered by more than so these 2 ones. So, they have been covered by this as well as by this.

So, similarly these ones have been covered by, sorry. So, these ones have been this one has been covered by this as well as by this. So, it may be possible that we can get rid of some of these groupings and we can still get the minimized form. So, that is what is told that these are all prime implicants. So, $\bar{A}D$, $B\bar{D}$, and $\bar{A}B$, because they are containing ones in them, but they are not essential, because you can get rid of some of them like. So, it is like if you say this $\bar{A}D$. So, $\bar{A}D$ is this one. So, $\bar{A}D$ is so, this grouping is $\bar{A}D$ in the first one. So, this grouping is $\bar{A}D$ so, if this $\bar{A}D$ and this $B\bar{D}$ if you take then; so this is B and if

you take this 2 so, they actually this particular grouping. So, this is $B\bar{D}$ so, if you take this 2 it is only. So, this all the ones are getting covered, but in that case we don't need this particular we don't need this particular grouping of $\bar{A}B$. So, $\bar{A}B$ is not essential so, you can take A bar D and BD bar.

But if you take say $\bar{A}D$ and this $\bar{A}B$ you take $\bar{A}D$ and $\bar{A}B$ then also it is fine that may be another possibility so, that way. So, this is also not unique so, I can have I can also take another one like say $\bar{A}D$ and $\bar{A}B$. Because $\bar{A}D$ will give me this 1, ok, and $\bar{A}B$ will give me $\bar{A}B$ now $\bar{A}D$. So, this is not essential this will not give me full covering. So, this is my $\bar{A}D$ so, these ones will remain uncovered. So, these are not these cannot be taken as essential prime implicant. So, $\bar{A}B$ is not essential so, we have got $\bar{A}D$ and $B\bar{D}$ are essential prime implicant so, the minimum form is $\bar{A}D + B\bar{D}$.

(Refer Slide Time: 29:31)

Using non-essential prime implicants

- Consider the example
- Having Prime Implicants

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, essential prime implicants we cannot remove. So, non-essential prime implicants sometimes if you take this example you see; that if you take only prime implicants so, you will get a grouping like this ok. But you see that these ones it is; so, this particular this one is covered by more than ones like that. So, that way it is we are taking some non-essential prime implicants.

(Refer Slide Time: 29:59)

continue

- Include those that are essential
- Leaves just a single 1 uncovered
- Have the 2 choices to use for covering it, both of equal size (i.e. same # of literals)
- Choose either
- Choose as shown getting
- $F = A'B'C'D' + BC'D + ABC' + AB'C + ABD$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, include those that are essential so, if I just so, in the previous case. So, this is covering this is showing all the implicants that I can make, ok. So, these are all the implicant that I can make from here I have to choose the essential set.

Now, how to we choose the essential set? For we take though that the take ones that are essential. So, these are essential so, I take this thing. So, this was not essential, because this is already covered by that. But the problem is that after taking the essential one. So, this one is left this one is left uncovered. So, in that case we have a choice. So, we can we can take say either this one or that one. So, we have got 2 choices of covering it both of equal size so, we can check either of them and ultimately come to this representation.

So, by taking after choosing the essential one; so we have to check we have to check some others which if some ones are still left, and then we can we can have a choice. So, I can group this one with this one or this one. So, this one can be group like this or so, I can group it in this fashion or I can group it in this fashion. So, both are possible, but in whatever you do ultimately you get the same type of cost so, that way it is same.

(Refer Slide Time: 31:25)

Selecting non-essential

- Selection Rule: Minimize the overlap among prime implicants as much as possible.
- Make sure each prime implicant selected includes at least one minterm not included in any other prime implicant selected.

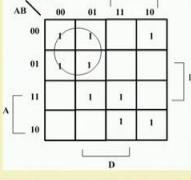
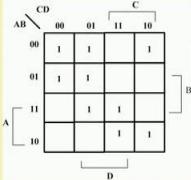
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, we want to minimize the overlapping so, that way we can we can try to do as much minimization as possible by doing this selecting in such a fashion.

(Refer Slide Time: 31:36)

Another problem

- Problem:
- Cover largest group of 1s



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, this is another example we have got this 1's, and then we have we cover the largest group of one's so, this is the there.

(Refer Slide Time: 31:48)

Select 1s not covered

- First

A Karnaugh map for four variables (A, B, C, D) with minterms 0000, 0001, 0011, 0100, 0111, 1000, 1011, and 1110 marked as 1s. The map shows two groups of two adjacent 1s each, circled with dashed ellipses. The first group is at (0000, 0001), and the second is at (0111, 1011). The remaining cells (0010, 0101, 1001, 1100, 1110) are marked with 0s.

- Second

A Karnaugh map for four variables (A, B, C, D) with minterms 0000, 0001, 0011, 0100, 0111, 1000, 1011, and 1110 marked as 1s. The map shows three groups of two adjacent 1s each, circled with dashed ellipses. The first group is at (0000, 0001), the second at (0111, 1011), and the third at (1000, 1011). The remaining cells (0010, 0101, 1100, 1110) are marked with 0s.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And after that, we can after selecting this select the ones that are not covered yet. So, this is the one that is not yet covered you select that one. Then we select another group that is not covered yet. So, we have so, essential we have essential part we have done.

(Refer Slide Time: 32:01)

Just 1 more to go

- have a choice
 - Both are equal
 - Choose 1
- Get
- $F = A'C' + ABD + AB'C + A'B'D'$
- Equal in cost to
- $F = A'C' + ABD + AB'C + B'CD'$

A Karnaugh map for four variables (A, B, C, D) with minterms 0000, 0001, 0011, 0100, 0111, 1000, 1011, and 1110 marked as 1s. The map shows four groups of two adjacent 1s each, circled with dashed ellipses. The first group is at (0000, 0001), the second at (0111, 1011), the third at (1000, 1011), and the fourth at (1110, 1111). The remaining cells (0010, 0101, 1100) are marked with 0s.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we have got a choice so, these one may be covered with this or it may be covered with this both are having equal cost. So, if we take the example that is shown here so, it is giving me this particular expression whereas, if I do a grouping with say this one and this one, ok.

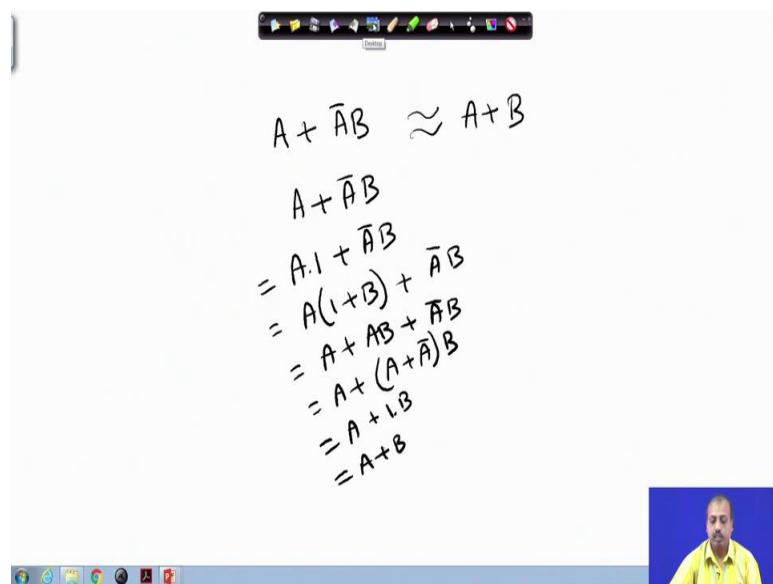
So, these two ones a group so, we will get the second configuration, but both of them are having same number of literals so, they are just same getting the same; so that way in Boolean expression minimization so, we can use Karnaugh map, and we can have different views it may be viewed from a set theory approach it may be viewed from prime the implicants approach. So, whatever you do ultimately it is a matter of covering. So, if you in the covering process, if you are not taking the redundant 1's, then you will definitely get the minimum form with minimum number of literals and minimum number of terms.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 14
Logic Gates

So, next we shall be starting with Logic Gates. So, gates are used for realizing digital circuits. Now, before we do that we will just clear one confusion, which was there regarding this.

(Refer Slide Time: 00:38)


$$\begin{aligned} A + \bar{A}B &\approx A + B \\ A + \bar{A}B &= A \cdot 1 + \bar{A}B \\ &= A(1 + B) + \bar{A}B \\ &= A + AB + \bar{A}B \\ &= A + (A + \bar{A})B \\ &= A + 1 \cdot B \\ &= A + B \end{aligned}$$

So, there was a confusion in Boolean algebra, where we had an expression like $A + \bar{A}B$ and which we wanted to equate to $A + B$ so, that simplification process there are some problems.

So, let we will do the simplification like this like $A + \bar{A}B$ it can be written as $A \cdot 1 + \bar{A}B$, where this one can be written as $A(1 + B) + \bar{A}B$. So, if you multiply you get $A + AB + \bar{A}B$ and so, it gives $A + (A + \bar{A})B$. So, this $A + \bar{A}$ is again 1 so, this gives $A + B$.

Since in one class there are some confusion about it. So, this is the derivation So, with this we will be starting with this logic gates. So, as I said that logic gates are circuit elements digital circuit elements by which we can realize the digital circuits. So, far we have seen

how to minimize the Boolean function and this class onwards we will see how to realize a Boolean function in terms of components.

(Refer Slide Time: 02:03)

• A **binary quantity** is one that can take only 2 states

Battery Switch Lamp
A simple binary arrangement

Total Security
There are many suspicious file entries in your Quarantine folder, that are sent to be submitted to research lab. Suspecting the administrator ensures the detailed analysis of the file and sends it to the research lab. This helps in maintaining the integrity of the virus signature database which will be provided to updates to all the users.
To submit quarantined file, provide your email address below and click OK.
Email :
 Don't display this message again
 OK Cancel

S	L
OPEN	OFF
CLOSED	ON

A truth table

S	L
0	0
1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so, if you, the binary quantities and variables so, if you look into a binary quantity is one that can take only two states, so either it is high or low if you can think of it, as if there is a battery and, there is a switch and a lamp. And then when this switch is closed then, only the lamp will glow.

So, this switch was got two positions open position, when it is now like this at the closed position, when this bar connects to this point. So, that way we have got two positions for the switch open position and closed position. So, when this is open then the lamp does not get the current flow. So, lamp is off and when this is closed the lamp is on.

So, that we can think of it as a variable the switch can be considered as a variable that has got two states in it open and closed and, accordingly the lamp has got two states off and on. So, this S and L that this switch and lamp both are binary quantities, or binary variables, you can just represent in the form of truth table, where we say that this open is represented by 0 and closed is represented by 1 for the switch.

So, switch can be either at state 0, or at state 1, on the other hand the lamp it can be off which is represented by say 0 and on which is say represented by 1.

You see that this open represented by 0, or this on represented by 1 so, these are arbitrary. So, this is done by the designer ok. So, for the design the simplicity so, you can follow any of the logic however, one thing is true that whatever convention we follow so, that has to be followed uniformly. So, you cannot change from one part of implementation to another part while doing this thing.

(Refer Slide Time: 03:51)

- A binary arrangement with two switches in series

(a) Circuit

S1	S2	L
0 = open	0 = open	0
1 = closed	1 = closed	1
		0 = off
		1 = on

$L = S1 \text{ AND } S2$

(b) Truth table

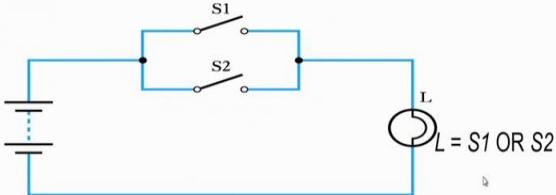
So, if we consider two switches like this suppose I have got S1 and S2 two switch switches they are connected in series. So, the S1 can be either open, or closed and S2 can also be open or closed. Now, this lamp it get is the current flow only when both S1 and S2 are in closed state ok.

So, if I represent 0 open by 0 and closed by 1. So, only when both the switches S1 and S2 are closed. So, they are getting the value 1 and 1, then only the lamp is on so, this L equal to 1 otherwise the lamp is 0.

So, this is actually the condition that the lamp gets on only when both S1 and S2 are on, both are closed. So, this S1 and S2 so, please mind this term AND so, this AND gate is coming into picture. So, here I say that L the logic expression for L S is given by S1 AND S2. So, when S1 and S2 both this variables get the value 1, then only L will get the value 1, otherwise L will get the value 0. So, this is the AND operation of S1 and S2.

(Refer Slide Time: 05:00)

- A binary arrangement with two switches in parallel



(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

Similarly we can connect the two switches in parallel. So, apparently it seems why should we connect two switches in parallel, but for the sake of examples. So, let us take it like this that we have got two switches connected in parallel S_1 and S_2 . And either of the switches being closed the light gets the lamp L gets current as a result it turns on. So, we can say that if again the same thing, if 0 for switches represent they are open and one represent they are closed.

So, whenever either S_1 or S_2 is 1, then L value becomes equal to 1. So, this is the truth table and, here the operation that we are doing say L is on if S_1 is 1, or S_2 is one again mind this term or, because this is the or operation of 2 binary quantity. So, this L binary quantity, or binary variable L , it get is the value of S_1 OR S_2 where S_1 and S_2 are the again 2 binary variables.

(Refer Slide Time: 06:00)

• Three switches in series

S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$L = S1 \text{ AND } S2 \text{ AND } S3$

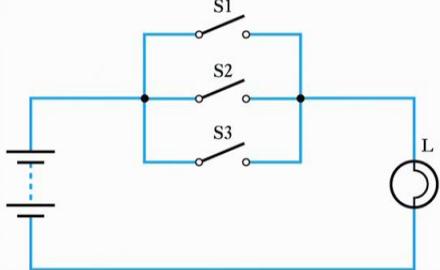
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We can connect three switches in series like S₁, S₂, S₃. So, what happens is that here this lamp L get current, only when all the three switches are closed. So, this can be represented in the form of a truth table, where this S₁, S₂, S₃ is so, if I just write down all the possibility. So, S₁, S₂, S₃ these are three variables so, it can start from 0 0 0, 0 0 1 and go up to 1 1 1, where 0 means the switch is open and one means the switch is closed.

So, as I said that since this is a series switch S₁, S₂, S₃ so, this L will be equal to 1, only when S₁, S₂, S₃ all of them are equal to 1, So, in the truth table also you see that this L equal to 1 only when S₁, S₂, S₃ all are equal to 1.

(Refer Slide Time: 06:48)

- Three switches in parallel



$L = S1 \text{ OR } S2 \text{ OR } S3$

S1	S2	S3	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Similarly, we can have OR operation. So, we can have in parallel the switches may be connected in parallel $S1, S2, S3$, they are connected in parallel so, whenever any of these switches are closed the lamp is turned on and, when all the switches are open and then the lamp is off. So, whenever this lamp L equal to 0 only when $S1, S2, S3$ all of them are equal to 0. Otherwise whenever the any of them is equal to 1, the L value is equal to 1.

So, we can say that as if this lamp L is equal to $S1 \text{ OR } S2 \text{ OR } S3$. So, it is the OR of 3 variables. So, 3 all the $S1, S2, S3$ variables are ORed to get the value of L , but OR so, for whatever we have seen OR is a 2 variable operand it is a 2 variable operation OR 2 operand operation. So, we have to write 2 OR's in between so, $S1 \text{ OR } S2 \text{ OR } S3$ ok.

(Refer Slide Time: 06:50)

• A series/parallel arrangement

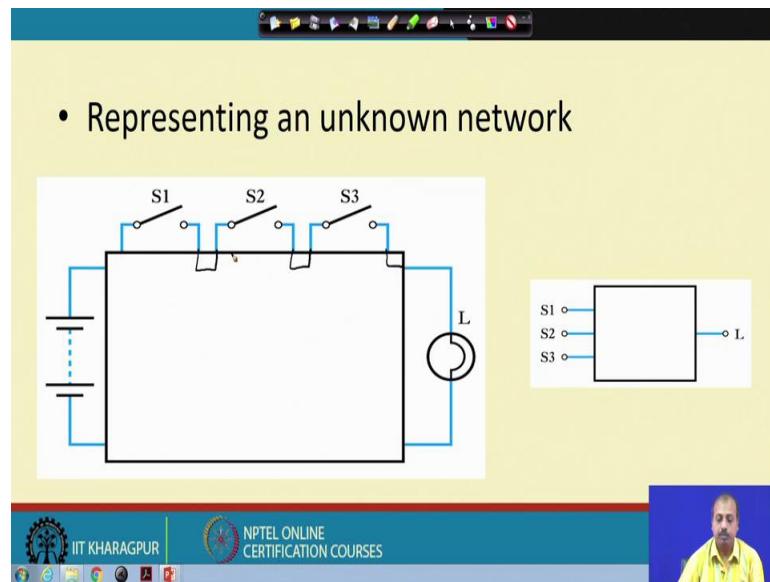
$L = S1 \text{ AND } (S2 \text{ OR } S3)$

S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

So, this way we can represent 3 switches in parallel slightly more complicated one. So, let us say that we have got the switch S1 connected in series with a parallel combination of S2 and S3. So, for this lamp to glow S1 must be closed and at least one of the S2 and S3 should be closed. So, this is basically S2 OR S3 AND S1 ok. So, that is the logic for L so, here also if you look into the truth table, this L is on for L to be on S1 must be equal to 1 and at least one of S2 and S3 should be equal to 1 so, then only L is equal to 1.

So, this over all operation of this a lamp can be written as L equal to S1 AND S2 OR S3. So, that captures the status of all the three switches S1, S2, S3 and we get the combination ok. So, this way we can represent this operation on Boolean quantities by means of switches and some output.

(Refer Slide Time: 08:50)



So, now, if we suppose we have got an unknown network ok. So, this is a black box so, this is a black box I do not know what type of connection we have between these points. So, this say from this output of S1, how is it connected to S2 so, S2 S3 may be connected in series maybe connected in parallel maybe all of them are in parallel or maybe all of them are in series.

So, they are may be many possibilities. So, it maybe the connection pattern is the like this, that from this point sorry from this point we have got so, from this point, we have got a connection that takes it to this as well as this. And then from this and this they are connected to the output point.

So, what happens is that so, we are getting S2 OR S3, because S1 has to be closed and then at least S2 OR S3 has to be closed so, that we can get that so, that so that we can reach this point ok. So, this way or it may be that we have got a connection like this that S1 is connected like this and S2 is S2, S3 it is connected like this and, this is the connection.

So, that way what we have is that S1, S2, S3, they are connected in series ok. So, this is AND operation so, if this is an unknown network, if this is an unknown network, then we can we may try to represent. So, in a black box form so, we can say that as if there are three input S1, S2, S3 for this black box and there is one output L Now, how do you we find out the operation so, exact operation.

(Refer Slide Time: 10:28)

The building blocks used to create digital circuits are **logic gates**

- There are three elementary logic gates and a range of other simple gates
- Each gate has its own **logic symbol** which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a **truth table** or using **Boolean notation**

So, for that what we need to do is that we need to apply different combinations and see whether the light glows there or not. Suppose only when we connect S1, S2, S3 all of them in closed position, then only the light glows; that means, only when S1, S2, S3 all the variables are 1 L will be equal to 1.

So, it is a AND of three variables so, that way you can identify the logic function that is implemented by this box by applying different switch combinations here, and checking the status of this light.

So, this logic gates so, they are building blocks for the creating digital circuits and, there are three elementary logic gates and the range of other simple gates. So, we will see that there are three basic gates and some and there from and from there, there are many other derived gates, each gate has its own logic symbol which is universally accepted. So, it is expected that we use those symbols to represent those logic gates.

And so, we can represent, So, since each of them is a symbol so, you can use those symbol for AND operation OR operation etcetera and, then connect between the lines by means of some lines connect between the inputs and outputs by means of lines, to get the overall logic diagram.

So, that we will call a logic diagram. So, the complex function can be represented by logic diagram and, the function of each gate can be represented by a truth table, or using the

Boolean notation. So, you can either use a truth table, or a Boolean notation in terms of some Boolean function you can represent it.

(Refer Slide Time: 12:00)

The slide has a yellow background with a black header bar at the top containing various icons. The title '• The AND gate' is in blue at the top left. Below it, there are three parts labeled (a), (b), and (c):
(a) Circuit symbol: A logic gate symbol with two inputs (A and B) and one output (C).
(b) Truth table:

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

(c) Boolean expression: $C = A \cdot B$
At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

So, we will start with the first basic logic gate which is the AND gate. So, this is the symbol of the AND gate ok. So, this is universally accepted so, it is expected that whenever we are drawing AND gate we use this symbol. Now, why this symbol there is no reason like that and, in some special cases you may use some other notations, but it is expected that in general you use this notation. So, this is the symbol so, whenever in the digital circuit you see this symbol we understand that it is an AND gate.

Now, there are 2 inputs coming to it and 1 output going out of it. So, AND gate always has got 1 output, but the number of inputs maybe more so, in this particular example so, you have got 2 input AND gate. So, if there are more number of input. So, more number of lines will be connected to input side. So, this is the input side this is the output side, so, if there are more number of inputs. So, there will be connected to the more number of lines in the input side.

So, this same circuit symbol so, it is it can be represented in the form of truth table. So, truth table actually tells what is the functionality implemented by the corresponding logical element. So, as the name suggest this is an AND gate so, only when both the inputs A and B are equal to 1 we will be having output C equal to 1. So, so, 0 0 when the inputs are 0 and 0 output is 0, 0 1, then also 0, 1 0, then also 0 and 1 1 is 1.

So, this is the truth table and the in terms of Boolean expression. So, this is an AND of A and B, two variable. So, $C = A \text{ AND } B$ so, in different situation so, we will be using different notation like when you are trying to represent the circuit, we will draw this type of gates this type of circuit symbols, when we are interested about the functionality in the form of some truth table. So, we can write it like this detail functionality. Sometimes, we need a compact notation. So, there will be writing in terms of the Boolean expression. So, depending upon our requirement we will do that.

(Refer Slide Time: 14:00)

• The OR gate

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

$C = A + B$

(a) Circuit symbol (b) Truth table (c) Boolean expression

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

The next fundamental gate that we have digital circuit is the OR gate and, again thus this symbol is the universally accepted symbol and, it is again expected that for OR gate we will represent it like this. So, this is the circuit symbol, this is the truth table. So, or means whenever at least one of the inputs is equal to 1 output will be equal to 1.

So, this is happening here so, you see whenever A or B at least 1 of them is equal to 1 output C is equal to 1. So, we get $C = A + B$ so, this is the Boolean expression and, this is the truth table.

Again the same thing that this is a 2 input OR gate. So, if you want to more number of input so, they can be added here. So, theoretically there is no limit on the number of inputs that you can have to a gate, but of course, ultimately this is the these will be represented by digital IC's. So, they will have their own restrictions. So, and you will find different number of inputs for different IC chips.

So, you can have say 2 input OR gate or 4 input OR gate like that. So, normally you do not have do not have odd number of inputs to the gate and 2 and 4 these are the common ones. So, we do not have more because that will make the gate more complex.

(Refer Slide Time: 15:17)

The NOT gate (or inverter)

(a) Circuit symbol

(b) Truth table

(c) Boolean expression

A | B
0 | 1
1 | 0
 $B = \bar{A}$

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another fundamental gate that is there is the NOT gate, or the invertor. So, its symbol is like this a triangle and on top of the there is a bubble, ok. So, this is a single input so, OR NOT gate cannot have more than 1 input. So, it has got a single input and it has got a single output.

So, as the name suggest so, this is the invert of the input output is the invert of the input. So, in terms of truth table whenever A is 0, B is equal to 1 and, whenever A is 1, B is equal to 0 and, in terms of Boolean expression it is written as $B = \bar{A}$ ok. So, this is \bar{A} so, this is the complement of A so, that is the NOT gate ok.

(Refer Slide Time: 16:00)

• A logic buffer gate

(a) Circuit symbol (b) Truth table (c) Boolean expression

A	B
0	0
1	1

$$B = A$$

So, sometimes we use a logic or buffer gate. So, this is buffer means that it just copies the input to the output. So, whenever A is 0 B is also 0, whenever A is 1, B is also 1 logic A Boolean expression is B equal to A, apparently it is seems that why do we need such buffer gate so, ok.

So, sometimes what happens is that we need to drive some large load, in terms of the, in terms of circuit elements like it may be it may so, happen that one gate output it drives a number of gate out gate inputs in the next stage. So, that way it has to drive a large current and these buffers, they have the capacity to drive large current ok.

So, they can be connected. So, that way we can use this type of buffers so, this otherwise as far as logic is concerned, it does not add anything to the circuits. So, it just copies the input to the output.

(Refer Slide Time: 16:53)

The slide contains the following information:

- The NAND gate** = AND \times INV
- (a) Circuit symbol**: A standard logic gate symbol with two inputs (A and B) and one output (C). The output has a small circle (bubble) indicating it is the inverse of the AND function.
- (b) Truth table**:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0
- (c) Boolean expression**: $C = \overline{A \cdot B}$ or $= \overline{AB}$

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video thumbnail of a professor.

Next we come to some derived gates like the NAND gate first one is a NAND gate. So, this is actually you can say that NAND is equal to a NAND is equal to AND plus inverter ok. So, as so, this is a AND plus inverter you see that here the symbol is also like that. So, this is the AND gate and after that there is an inverter bubble ok. So, this A B so, as the name suggest. So, this will be output will be equal to 1 ok.

Whenever the AND gate output will be 0. So, AND gate output is 0 whenever any of these input is equal to 0. So, in terms of NAND gate we can say that whenever any of the inputs is 0 output will be equal to 1 and, only when both the inputs are equal to 1 the output will be equal to 0. So, this is the truth table. So, you see that whenever when both the inputs are equal to 1 output is equal to 0, otherwise output is equal to 1.

So, symbolically so, in a Boolean expression. So, it is represented like this $C = A\bar{B}$ so, this is also written as this dot is often ignored so we write it like this so $A\bar{B}$ so this is also there. So, this NAND gate so, you see that NAND gate is called a derived gate, because it can its can be derived from the AND gate and inverters. So, AND, OR inverter, they are fundamental gates and, from them we can derived this gate. So, NAND gate is one such derivation.

(Refer Slide Time: 18:30)

• The NOR gate

Handwritten note: OR X

(a) Circuit symbol (b) Truth table (c) Boolean expression

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

$$C = \overline{A + B}$$

Next we will look into NOR gate so, just like NAND gate was equal to AND plus NOT so, similarly if you do OR plus NOT what you get is a NOR gate. So, what we have got this NOR equal to this NOR is equal to so, OR plus inverter. So, OR plus inverter and the symbol is also like that. So, we have got here the OR and then AND there is an inverting bubble. So, OR plus inverting bubble. So, that is the circuit symbol for NOR.

Of course the other thing remain same like I can have more number of inputs here and all and then the truth table is simple like OR gate was whenever, any of the inputs is equal to 1 output was equal to 1, so since this is NOR so, whenever any of the inputs is equal to 1 output will be equal to 0 and only when both the inputs are equal to 0 this NOR gate output is equal to 1.

So, this is the $C = \overline{A + B}$. So, that is the Boolean expression for NOR ok. So, of course, you can apply De Morgan's on this to simply to simplify the circuit that we will see later.

(Refer Slide Time: 19:52)

The slide is titled 'The Exclusive OR gate'. It features a truth table with columns for inputs A and B, and output C. The table shows the following values:

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

To the right of the table, the word 'or' is written above 'Inclusive or'. Below the table, the Boolean expression $C = A \oplus B$ is given.

Below the truth table, there are three sub-labels: (a) Circuit symbol, (b) Truth table, and (c) Boolean expression. The circuit symbol is a standard logic gate icon with two inputs (A and B) and one output (C).

The footer of the slide includes the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

So, next we will look into the another derived gate which is known as exclusive OR gate. So, if you remember the OR gate. So, it was like this so, the truth table for OR gate was the A B and C, whenever the whenever any of the inputs was equal to 1. So, this was equal to 1 so, 1 0 is the also 1 and 1 1 was also equal to 1. So, this was the truth table for OR gate.

Now, you see that in case of XOR gate we first compare at the truth table levels. So, truth table wise you see that when both the inputs are equal to 1 XOR gates gives output 0 whereas, OR gate gives the output 1 so, the so, it so as the name suggest this exclusive OR gate it requires that exclusively only one of the inputs will be equal to 1. So, if both the inputs are equal to 1, then the output will be equal to 0. And as a result this OR function. So, this is also sometimes called inclusive OR so, this is also called inclusive OR, just to separate it out from exclusive OR. So, this is also known as inclusive OR.

So, sometimes we will see that exclusive OR gate detail later, it has got many fundamental usage in digital circuit design and, you see that here what we are having we are having 2 inputs A B and the C is output and C can be written as this is the symbol for XOR.

So, so this plus is the symbol for OR and, if you put a circle around it. So, this is will be an XOR operation. So, this is an XOR operation. So, this is if both the inputs are equal to 1 in the out will be 0 output is 1 only when exactly one of the inputs is equal to 1.

(Refer Slide Time: 21:45)

• The Exclusive NOR gate

(a) Circuit symbol (b) Truth table (c) Boolean expression

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

$$C = \overline{A \oplus B}$$

So, next we will look into the inverted version of XOR which is known as XNOR gate exclusive NOR gate. So, this is NOR but exclusive. So, NOR gate was that whenever we have got any of the inputs equal to 1 output was equal to 0. This is the difference in XNOR gate when both the inputs are equal to 1. So, output is equal to 1 so, this is not equal to 0.

So, in a NOR gate this output is equal to 0, but in an exclusive NOR gate so, these output is equal to 1. So, for your X for NOR operation. So, the truth table is was like this. So, this was the NOR operation where when both the inputs are equal to 1 output was equal to 0, now it will be equal to 1. So, symbolically we represent it as $C = \overline{A \oplus B}$, or it is written as it is read as A XNOR B so, it is read as A XNOR B.

(Refer Slide Time: 22:50)

Combinational Logic

- Digital systems may be divided into two broad categories:
 - **combinational logic**
 - where the outputs are determined solely by the current states of the inputs
 - **sequential logic**
 - where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state

A → **Dig. Circ.** → P

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will look into some design of this combinational logic. So, we are using this basic gates that we have discussed so far. So, we can realize some combinational function, any digital circuit can be represented by in terms of these gates, so we will slowly go towards that.

Now, if you look into all digital systems. So, they can broadly be classified into two categories one is known as combinational logic, another is known as sequential logic. Combinational logic is like this. So, output is determined fully by the current states of input. So, if I, if I represent the, if I represent this is suppose is digital circuit, suppose this is a digital circuit.

Now, it has got a number of inputs say A B C and, it has got say 2 outputs P and Q. Now, if it happens like this that the values of at any point of time, the values of P and Q it is dependent on the values of A B C at current instant only ok. So, it is does not depend on the previous values of A B C. So, if that is the situation then will tell that this is combinational circuit, because it does not depend on the on a it does not depend on the history of the inputs in previous times.

On the other hand, there will be an another class of circuits where you will see that this output P and Q it will depend not only on the current value of A B C, but on the previous values of A B C also like. If you observe the system from time t equal to 0 and at present you are at time t equal to 10, then the output P Q at time 10 will be dependent not only on

the values of A B C at time 10, but also on the values of A B C at time instants 9, 8, 7, 6 up till 0.

So, at all the previous values of A B C so, it need not be exactly at this bound that is like 9, 8, 7. So, you can say that it depends on the total history of this inputs from the beginning of the system. So, that type of logic will be known as sequential logic. So, in our course we will be dealing with the both the types of logics. So, both combinational and sequential, but initially we will be discussing on combinational logic because, that that is the simpler to understand and then we will proceed towards the sequential logic.

(Refer Slide Time: 25:22).

• Implementing a function from a Boolean expression

Implement the function $X = A + B\bar{C}$

A ——————
B ——————
C —————— \bar{C}

$X = A + B\bar{C}$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, suppose we have got Boolean expression $X = A + B\bar{C}$ ok. So, to how can we realize in terms of logic gates. So, you see that this OR so, fine at the highest level we have got this OR operation ok. So, we so, at that at the output at the top most level. So, we will have this OR gate so, OR gate will have 2 inputs. So, this A input and this $B\bar{C}$ inputs. So, input is connected to first input of OR gates and this $B\bar{C}$ has to be connected to the second input of OR gate.

Now, how do you get $B\bar{C}$ for getting $B\bar{C}$? So, you have to do B AND \bar{C} . So, you need an AND gate so, where 1 input is B another input has to be \bar{C} . So, this here you get $B\bar{C}$ now how do you so, B we have already got because, B is a primary input to the circuit and, then the \bar{C} to get to get \bar{C} what we do is we start with C put an inverter on here, and then from

the C we get a \bar{C} so, that way this the circuit. So, it represents the, it can implement the Boolean expression $A + B\bar{C}$.

(Refer Slide Time: 26:37)

So, this way these Boolean expressions are implemented like let us say another, take another example slightly more complex $Y = \bar{A}B + C\bar{D}$ ok. So, you see that this if you look into the highest level. So, this is a NOR operation so, this plus so this is OR and there is a inversion at the top so; that means, this is a NOR operation. So, I take a NOR gate and, then to the NOR gate I should have the inputs $\bar{A}B$ and $C\bar{D}$. So, this NOR gate it is getting two lines. So, in 1 line I have to realize $\bar{A}B$ another line I have to realize $C\bar{D}$.

Now, how do I realize $\bar{A}B$ I have to do and of \bar{A} AND B ok. So, this is a AND gate and we have got \bar{A} AND B here. So, B is coming directly from the primary input, but to get \bar{A} I have to take an inverter and, then this will be applied here and \bar{A} will be obtained from there.

Similarly, for getting $C\bar{D}$ so, I need another AND gate and in this AND gate C will be one of the input and \bar{D} will be the other input and for getting \bar{D} from D I have to have an inverter in between. So, this way starting from the top level of your expression so, you can just go back, realizing till the simplest element that is till you reach the primary input. So, until the expression refines to the primary input we have to go on doing this thing.

(Refer Slide Time: 28:00)

• Generating a Boolean expression from a logic diagram

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

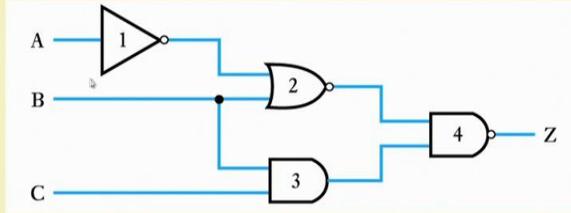
So, we can also do something for generating the Boolean expression from a logic diagram. So, suppose this is a Boolean this is a logic diagram. So, we can trace through this logic diagram to see how the corresponding Boolean expression can be derived.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 15
Logic Gates (Contd.)

So, for generating the Boolean expression from a logic diagram so, we have to proceed from the primary input site and go towards the primary output.

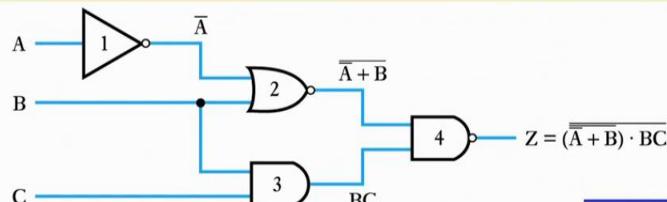
(Refer Slide Time: 00:18)



• Generating a Boolean expression from a logic diagram

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 00:22)



Example (continued)

- work progressively from the inputs to the output adding logic expressions to the output of each gate in turn

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the procedure that work progressively from the inputs to the output adding logic expressions to the output of each gate in turn. So, this is my input A B C now, I find that at the first level there is an inverter here. So, inverter will transform A to \bar{A} ; at the next level if you see there are 2 gates, gate number 2 and gate number 2 where gate number 2 is a NOR gate. So, in the NOR gate one input is A bar another input is B so, at this point I am getting the expression \bar{A} NOR B ok. So, $\bar{A} + \bar{B}$ whatever way you read it and then this gate number 3 it has got B and C add has two inputs and this is an AND operation. So, you get a BC as the output.

Then finally, at gate number 4 this is a NAND gate so, you gate the NAND of these two inputs. So, $(\bar{A} + \bar{B}) \cdot BC$ so, you get the NAND of them so, Z equal to this expression. So, of course this can be simplified, but it is not done here. So, you can break down this expression and see what it is it is turning out to be, but it can be see that later.

(Refer Slide Time: 01:33)

• Implementing a logic function from a description

The operation of the Exclusive OR gate can be stated as:

*"The output should be true if either of its inputs are true
and but not if both inputs are true"*

This can be rephrased as:

*"The output is true if A OR B is true,
AND if A AND B are NOT true."*

We can write this in Boolean notation as $X = (A + B) \cdot (AB)'$

$OR(A, B) \cdot (NAND(A, B))'$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, sometimes it is necessary that we are given some logical statement and from there we need to get the corresponding Boolean circuitry. So, this can be done like this a typical example has been taken here. So, this exclusive OR gate so, we can write it like this that output should be true if either of its inputs are true, but not if both inputs are true; as you know that is the functionality of the XOR gate. So, its output is a one only when only one of the input is equal to 1.

So, you can say we can say the way we have put this statement so, it is true if either of its inputs are true. So, this is the first part of the statement so, either of its either of its inputs are true. So, this is the first part of the statement, but so, but we can read it like this; so if I replace this, but by and so, we can say and not if both inputs are true ok.

So, so this can be represented like this so you now just put the statement like this that this either of its inputs are true; so this boils down to A OR B ok. So, this A this either of its inputs are true so, this boils down to A OR B is true and not if both inputs are true. So, so, both inputs are true so, this part so, this part is basically A AND B ok. So, this is A AND B and this not so, this NOT is the it is not true so, this NOT.

So, this has to be say A OR B should be there and A AND B NOT should be there so, A AND B NOT is basically the NAND. So, I can say this if I if I join this two I get a NAND. So, from this I get a NAND of I get a NAND of A AND B I get a NAND of A AND B. So, that should be there and then it should be ANDed with it should be ANDed with A OR B. So, this is OR of AB. So, this is the whole expression that I am expecting. So, we can write it like this so, here we have what I have written. So, this is first part is A OR B then this AND so, AND is this and now I have got A AND B NOT so, A AND B NOT is basically this one. So, this is the NAND gate A AND B NOT.

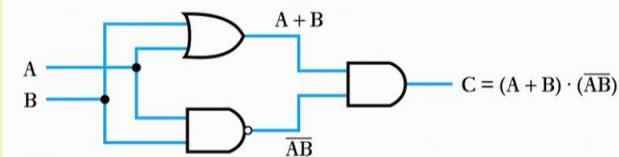
So, this way we can from Boolean expression Boolean from logic statements so, we can try to come to the corresponding Boolean expression so that may be done. And if it is difficult, we can try to draw the corresponding truth table from the Boolean from the logic expression from the logic statement. And from there we try to come to the Boolean expression.

So, we will see those techniques slowly has you proceed through the course.

(Refer Slide Time: 04:48)

Example (continued)

The logic function $X = (A + B) \cdot (\overline{AB})$



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now once we have got the logic function so, we can represent it in the form of gates like say so, this first of all I need an OR gate. So, this OR gate it has got A AND B as input so I get A OR B then I need a NAND gate. So, to get \overline{AB} so, this is the NAND gate getting \overline{AB} from A AND B and then finally, these two are to be ANDed so, this is ANDed. So, you get $C = (A + B) \cdot \overline{AB}$

So, this way from the logic statements we can, so we start with the logic statement and from there we finally come to the corresponding logic circuit. So, that can be done.

(Refer Slide Time: 05:33)

• Implementing a logic function from a truth table

Implement the function of the following truth table

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

– first write down a Boolean expression for the output
– then implement as before
– in this case

$$X = \overline{A} \overline{B} C + A \overline{B} C + A B \overline{C}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

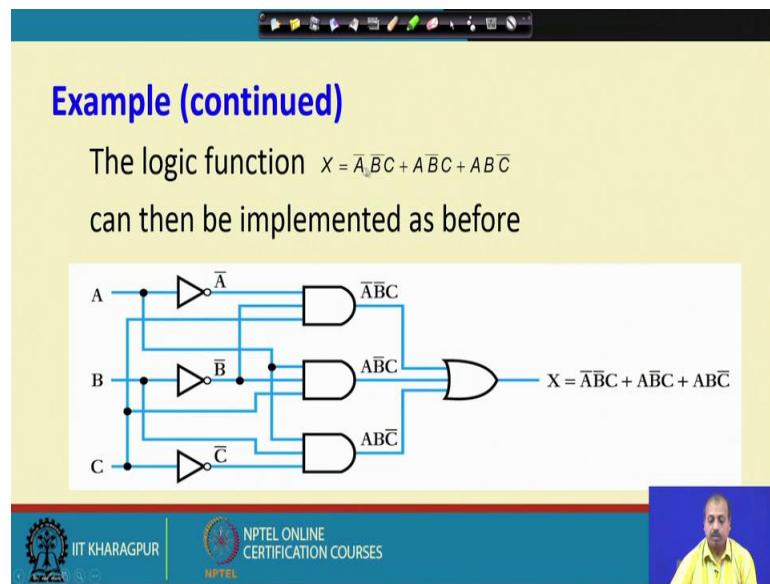


So, as I was telling that sometimes it is easier to get the logic, logic function through some truth table. So, so we start with the logic statement from there we try to draw the truth table and then the truth table gives us the functionality and then we try to implement it.

So, suppose we have got some logical statement and from where we get this as the truth table that is whenever the C being equal to 1 this is they so, I we write down all possible combinations of values combination of values of A B and C. So, 0 0 0 to 1 1 1 out of that we see that in these three cases the output is equal to 1 ok. So, when this $\bar{A}\bar{B}C$ A AND B and 0 and C is 1 then output is 1 then this $A\bar{B}C$ in that case it is equal to 1 and $AB\bar{C}$ this case this is equal to 1.

So, if we are trying to get the corresponding logic circuit then what we should do from the truth table we should write down the Boolean expression first and then we try to we realize this Boolean expression by means of logic gate. So, corresponding to this we can say that $X = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$ of course, I have not done any simplification of the Boolean expression here that may be done, but I am not going into that I will come to that later.

(Refer Slide Time: 07:03)



So, once we have got this Boolean expression now the process is straight forward. So, first of all at the top level I will need an OR gate and since there are three such terms in this in sum of product expression. So, I will need a three input OR gate so, these OR gate is three input OR gate. So, it will be the first input should be corresponding to $\bar{A}\bar{B}C$, second input to $A\bar{B}C$ and third input to $AB\bar{C}$. Now, how to get this $\bar{A}\bar{B}C$? For getting $\bar{A}\bar{B}C$ I need a

three input AND gate and where \bar{A} should be the first input, \bar{B} should be the second input and C should be the third input. And for getting \bar{A} so, we compliment A so, getting \bar{A} here similarly, we compliment B we get \bar{B} here. So, this $\bar{A}\bar{B}C$ they are connected to the first gate first AND gate.

Similarly, second and gate I will realize $A\bar{B}C$ so, this A is connected this B inverted output \bar{B} is connected and this C is connected. So, so, this way we get $A\bar{B}C$ and the third gate third gate, third AND gate is $AB\bar{C}$. So, the similarly we get $AB\bar{C}$ and finally, when they are ORed you get this Boolean expression fine.

So, in this way so, you can start with the truth table and from the truth table you can realize the corresponding function.

(Refer Slide Time: 08:25)

- In some cases it is possible to *simplify* logic expressions using the rules of Boolean algebra

$$X = ABC + \bar{A}BC + AC + A\bar{C} = BC(A + \bar{A}) + A(C + \bar{C}) = BC + A$$

can be simplified to $X = BC + A$

hence the following circuits are equivalent

So, it may be possible to simplify the logic at that we have seen in our discussion in on Boolean algebra that sometimes it is possible to simplify this sum of product expression or product of sum expression, to have lesser number of literals and lesser number of operands. Like it may be that suppose we have got a Boolean expression like this so, X equal to $ABC + \bar{A}BC + AC + A\bar{C}$.

Now, if you try to realize it directly then what will happen is that you will need a three input or sorry four input OR gate like we have it here and then it should this four inputs

correspond to this sum this product $ABC, \bar{A}BC, AC, A\bar{C}$. Now for so, you since your needing both A and \bar{A} so, you will be needing and inversion of A to get \bar{A} .

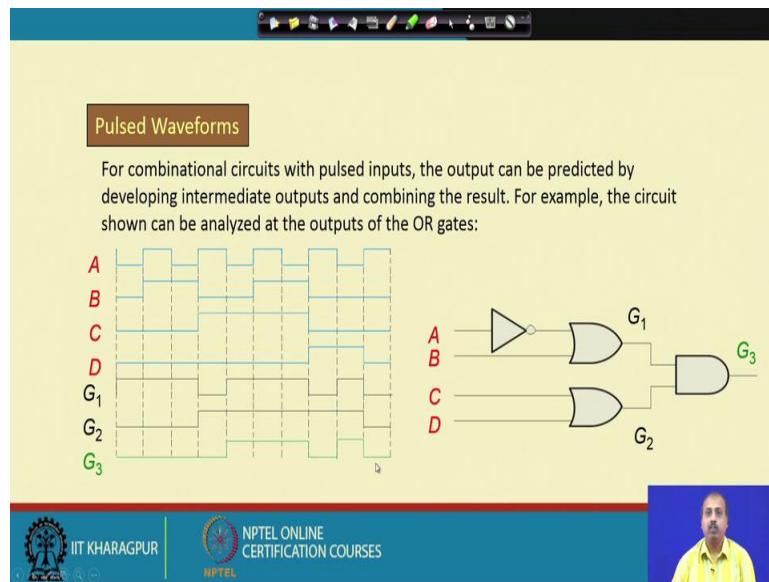
So, this line \bar{A} line can be connect should be connected to this point whereas, the points where you are getting A directly say can be can taken from this point and connected to A . So, you can the so, this for the sake of simplicity the connections are not shown, but what we mean is this A this line is connected here. Similarly, this line is connected to this A , then this A ok, then this A it is connected to all this.

So, it is not shown a explicitly because that will complicate the diagram so, it is they are not shown. So, but this whole expression can be simplify to this $BC + A$ so, you can just do this simplification. So, you can just from the first two terms so, you can take BC common and this is $A + \bar{A}$. Similarly from the next two terms you can take A common so, it is $C + \bar{C}$. So, this $BC(A + \bar{A})$ equal to 1 and $C + \bar{C}$ equal to 1 so, this simplified form is $BC + A$.

So, once we have done this simplification so, you can realize the circuit with much less number of gates. So, you see that I needed two input OR gate for or in this A AND BC and a two input AND gate for getting this product BC from the input B and C . So, that way this Boolean expressions can be simplified and then the simplified Boolean expression may have much less number of logic gate. So, that is one of the motivation why you should do this logic minimization because in terms of logic gates so, it will require less number of gates and less number of connections.

And complexity of the circuit becomes less and naturally the since the number of gates needed is less, so, the area then power requirement everything will go down for the resulting circuit.

(Refer Slide Time: 11:17)



So, next we will be looking into another important concept, like many times what happens is that if say this is suppose this is an example circuit where we have got a few gates G₁ G₂ G₃ and then inverter and is A B C D these are the inputs. So, many a times what happens is we this A B C D so, they do not have fixed value over the lifetime ok. So, they have got different values. So, we just draw some sort of timing diagram so, they are also known has pulsed waveform.

So, this we just see try to see when this inputs are changing like initially A was equal to 1, at time 0 it has become equal to 0. It continue still this time then it becomes 1 and it continue to be high for this much time, then it again comes down to 0, then it goes like this. So, this way suppose this is the wave form for A that is every alternate time instant so, it just toggles its state. Similarly, suppose the input B so, it was initially 0 initially low then here it goes 1 so, it remains high for this much time then it goes low, then again it remains low for this time.

So, it toggles every 2 time units you can say so, it is toggling like that. Similarly, C is toggling every 4 times units so, this is this was initially 0, it becomes 1 here for 4 times units it remain equal 1 then it comes down and it goes like this. Similarly, D is on D is 0 for this much time then it goes high and remains high for 2 time units then comes down and then goes to 0. So, these are the inputs A B C D these are the inputs so, these waveforms are given to us.

So, based on that we try to predict what will be the output of the circuit at different time instants. So, to do that what do we do is we first see how G1 will look like. So, for getting G1 so, you see whenever A is equal to 0 so, \bar{A} so, this is \bar{A} so, this is equal to 1. So, OR B so, you can understand since OR of these two whenever B equal to 1 or A equal to 0; the G1 will be equal to 1.

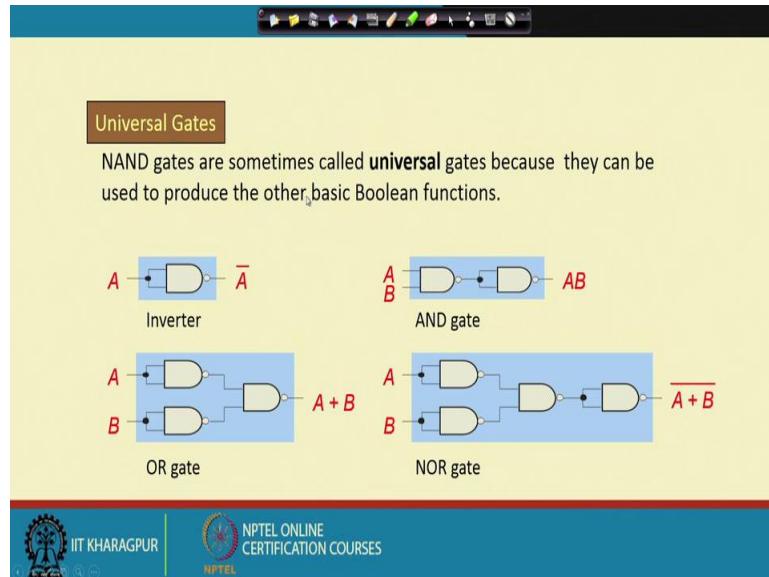
So, you see that whenever B is equal to 1 so, this G1 equal to 1. Similarly, whenever B equal to 1 G1 is equal to 1 and also whenever A is equal to 0 G1 is equal to 1. So, A equal to 0 G1 equal to 1 A equal to 0 G1 equal to 1; similarly A equal to 0 G1 equal to 1. So, whenever A is equal to 0 like here A equal to 0 G1 equal to 1; so in this way from this logic circuits so, we can draw the diagram for G1.

Similarly, for G2 so, G 2 is basically OR of C AND D, so whenever C OR D so, either of them is equal to 1 G2 output will be equal to 1, otherwise it is 0. So, C is 1 for this much time so, accordingly G2 is 1 for this much time and then after that say D has become 1. So, from this point onwards so, this is equal to again 1, but till D is equal to 1 and then at in this part so, both C and D are equal to 0. So, it comes down to 0.

Similarly, at the beginning both C and D are equal to 0 so, G2 remains at 0. So, this way we can find out the status for G2; how it proceeds with time and finally, the G3. G3 is the AND of G1, G 1 and G2. So, whenever G1, G2 both are equal to 1, G3 will be equal to 1 so, G1 G2 both are equal to 1 for this much time. So, this G3 is equal to 1 again for at this time so, G1 and G2 both are equal to 1. So, this is equal to 1 otherwise it remains at 0.

So, this way very often so, we will be drawing this pulse wave forms for combinational circuits. So, given the input so we will proceed step by step drawing these pulses, pulsed output for it of the gates. And finally, will reach the primary output and there we get the overall output for the system.

(Refer Slide Time: 15:25)



Next we will be discussing something called universal gate. So, as the name suggests, universal means you can do anything using those gates ok, you can realize any digital circuits using those gates. So, so, like NAND gates is one of the one such example that can be used for realizing any circuitry and that is why it is called universal gate. So, they can be used to produce other basic Boolean function.

So, basic Boolean functions are if you look into this sum of product or product of sum expression. So, the basic functions are AND OR and invert. So, if I have got a gate by which we can generate these three basic functions AND OR and INVERT so, that gate can be used to realize any other function. So, that is why so, we will be calling this AND OR NOT as the basic gates and the basic Boolean functions.

So, this NAND gate can be used for doing, realizing this basic Boolean function. How? So, if you want to get an inverter so, what you can do take a two input NAND gate and then just short both the inputs and connect the input A to the inverter the corresponding so, we are interested to get this inverter. So, our objective is to get an inverter like this. So, this A will be input and \bar{A} will be output ok.

So, what you do is that you take a NAND gate and then short the two inputs and then apply A at both the, or you can say that you apply A at both the inputs so, and you get the \bar{A} has the output. So, you can realize inverter using NAND gate then you can also you

can also realize this AND gate by using NAND gate, because for the realizing AND gate first of all you do a NAND of A AND B.

So, here the objective is to get the AND function so, I will have this AND gate and this A AND B are the two inputs ok. So, A and B are two inputs and you just do this; first of all you do a NAND so, you get $A\bar{B}$ here and after that use another NAND gate as inverter so, you get AB. So, you can get this AND function from the NAND. And for getting OR gate what we do is that we this OR gate this is slightly tricky. So, we use this De Morgan's theorem actually. So, this A OR B it can be written has $\overline{\overline{A} + \overline{B}}$.

So, that way I am taking to a compliment twice so, has a result this will be complimenting this this will be giving me back this A+B. Now, if I just apply De Morgan's law on this part ok, if I apply De Morgan's law on this part so, this will give me $\overline{A}\cdot\overline{B}$ and this whole bar remains as it is. So, what is happening is that so, this is nothing, but NAND of \overline{A} and \overline{B} . So, you take the NAND of \overline{A} and \overline{B} and for getting \overline{A} from A, you use the one NAND gate has an inverter and for getting \overline{B} from B is another NAND gate has an inverter.

So, that way this whole circuit gives us A+B ok. So, using only NAND gates of course, number of NAND gates needed is more, but that that is not a problem. So, if, if assuming that I have got an infinite supply of NAND gates so, I can realize any function by means of this NAND gates only. So, I do not need any other gate for realizing the functions.

Now, similarly you can also have this you can also realize this NOR gate using this similar technique. Like say so, the up to this much this is this is OR gate and if you need a NOR gate so, you can just put another NAND gate is as an inverter at the end and that will give you the NOR gate. So, this NOR gate can also be obtained from NAND gate. So, all the gates can be obtained.

Similarly, so, we have got NOR gate as another universal gate. So, apart from NAND gate we have got NOR gate as another universal gate. How?

(Refer Slide Time: 19:56)

The slide is titled "Universal Gates" and discusses the universality of NOR gates. It shows three examples:

- Inverter:** A single NOR gate with one input A and one output \bar{A} .
- OR gate:** Two NOR gates in series. The first NOR gate has inputs A and B, and its output is connected to the second NOR gate's input. The second NOR gate's output is labeled $A + B$.
- AND gate:** Two NOR gates in parallel. Both have inputs A and B. Their outputs are connected to the inputs of a third NOR gate, which has its output labeled AB .

Below these diagrams, De Morgan's theorem is derived:

$$\overline{XY} = \overline{X} + \overline{Y}$$
$$AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$$

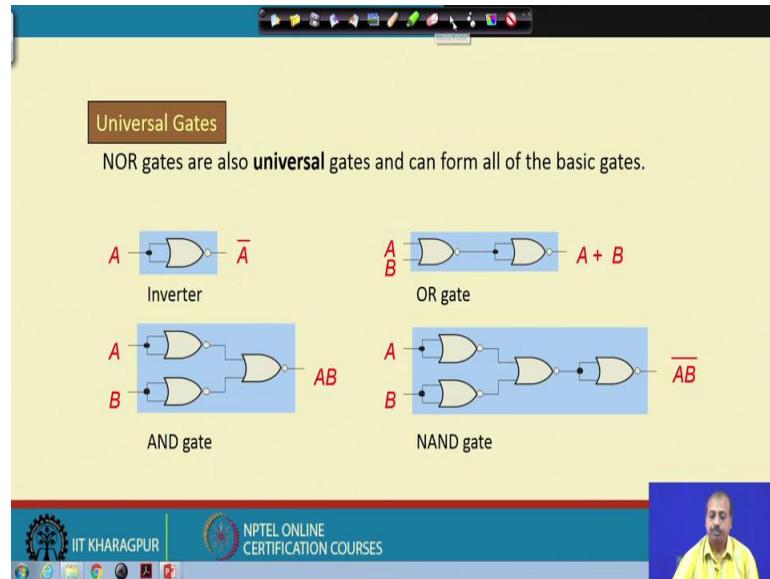
The slide footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a professor.

So, here also the same thing like if you are trying to get this inverter \bar{A} from A. So, what you do apply the same input A to both the inputs of the NOR gate so, you get \bar{A} as the inverter. If you are trying to get a OR gate, it is quite simple first of all you do a NOR of A AND B so, you get A nor B and then use another NOR gate has an inverter. So, you get A OR B.

If you are trying to get an AND gate so, then it is slightly tricky. So, you have to apply that De Morgan's theorem. So, basically this AB can be written has \overline{AB} . So, this is $\overline{\overline{A} + \overline{B}}$ fine. De Morgan's law was this $\overline{XY} = \overline{X} + \overline{Y}$ so, I am just using that formula. So, this \overline{AB} and so, this \overline{AB} I am writing as \overline{A} plus \overline{B} ; the upper bar remains as it is.

So, what we have is that so, here you have got that this is the \overline{A} OR \overline{B} that an NOR gate \overline{A} NOR \overline{B} , that NOR operation and for getting \overline{A} from A. So, use another nor gate as an inverter and for getting \overline{B} from B use another NOR gate as inverter. So, that way we can do this thing.

(Refer Slide Time: 21:28)



So, if you are trying to get a NAND gate then it is a simple so you get this AND gate up to this much and after that you put an inverter ok. So, another NOR gate as an inverter so, you get the $A\bar{B}$.

So, this way we can have this universal NOR gate it can realize any other gates ok, AND OR NOT. Actually, this NAND is not necessary because if you have to prove that a particular gate can act as a universal gate, what you need to show is that it can be realized as an inverter, it can realize the OR function, it can realize the AND functions. So, these are the three things to be shown so, this is shown as an extra ok. So, this is not mandatory to show.

(Refer Slide Time: 22:15)

Other Universal Possibilities ...

- {AND, OR, INVERT}
- {XOR, AND}
- {XOR, OR}
- ...

$$F = AB + C\bar{D}$$
$$F = A \oplus B$$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, some other combinations; so AND, OR, INVERT so, other universal gate combination. So, apart from this NAND and NOR so, we do not have any other gate which alone can act as a universal gate, but if you take a combination then it may be possible. Like say this one say AND, OR, INVERT is definitely AND OR INVERT is definitely one possibility because this here it is direct. So, AND gate can be realized for realizing AND part, OR gate for the OR part and the INVERTER part for the NOT operation.

So, any from any Boolean function that you write say $F = AB + C\bar{D}$ something like that. So, we can very easily realize using AND OR INVERT gate that we have already seen and interesting combination is this XOR and AND ok.

So, what is the, this XOR so, you see suppose I have got XOR gate. So, $F = A \oplus B$. So, what is the truth table for this so, this AB so, when both are 0, 0 1, 1 0 and 1 1. So, this is equal to 0 and this is equal to 0 and these two are equal to 1 fine. Now, you see that if I make this B input forcefully equal to 1 so, this $B = 0$ combination does not come. So, B is made forcefully equal to 1 then what happens is that if you give $A = 0$, you get $A = 1$ at the output, if you give 1 as an input you give 0 as the output.

So, this XOR gate which is like this AB as two inputs and F has the output. Now, if I said this B to be equal to 1 then this is nothing, but I have got this if this is an INVERTER. So, this when A is equal to 1 it behaves as an INVERTER, as if I have given A here and I have

got F which is equal to \bar{A} . So, in XOR gate so, if you tie one of the inputs to high so, you get an INVERTER. So, from the XOR gate I can get INVERTER.

(Refer Slide Time: 24:46)

So, now if I have got an infinite supply of XOR and AND gate so, I can realize OR gate also for example, so, so, I have realized say A OR B ok. For getting A OR B so, I can write it like this so, $\bar{A} \cdot \bar{B}$. Now, I can for getting this \bar{A} I can use one XOR gate ok. So, I can use an XOR gate with one of the input equal to 1. So, A is applied here so, this is your \bar{A} and similarly I take another XOR gate with one input as B, other input tied to 1.

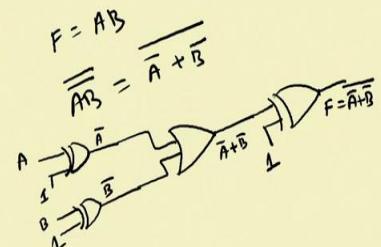
So, this gives me \bar{B} and once I have got this \bar{A} and \bar{B} so, I can take an AND of these two I take AND of these two. So, I gate here $\bar{A} \cdot \bar{B}$ and after that I have to take an inversion. So, I can take to another XOR gate ok. So, I can do it like this with one of the other input of the XOR gate tied to 1 ok. So, here you get the F. So, you see ultimately you are getting F equal to this one $\bar{A} \cdot \bar{B}$ and that is nothing, but A OR B. So, this way if I am given an infinite supply of XOR gate and AND gate I can realize OR function also and once. So, I have shown you the realization of OR; AND is already there in the set and from XOR I can get the inverter also. So, this AND, OR, INVERTER all the three basic gates can be obtained from this set XOR and AND. So, naturally XOR AND is an universal set.

Similarly, this XOR OR so, this is also an universal set because in this case OR is already there. So, what is not there is the AND gates. So, F equal to AB ok.

(Refer Slide Time: 26:40)

Other Universal Possibilities ...

- {AND, OR, INVERT}
- {XOR, AND}
- {XOR, OR}
- ...

$F = AB$
 $\overline{AB} = \overline{\overline{A} + \overline{B}}$


The circuit diagram illustrates the derivation of the expression $\overline{AB} = \overline{\overline{A} + \overline{B}}$. It shows two inputs, A and B, entering a first XOR gate. The output of this gate is connected to one input of a second XOR gate, along with the inverted inputs \overline{A} and \overline{B} . The output of the second XOR gate is the final result $F = \overline{AB}$.

 IIT KHARAGPUR |  NPTEL ONLINE
CERTIFICATION COURSES

So, AB if you want realize so, you can take two bars and then you can simplify it has $\overline{A} + \overline{B}$. And now, the same strategy I will follow I will take one XOR gate to get the INVERT of A ok, this is tied 1. So, you get \overline{A} here and then you take another XOR gate so, this is your B and 1. So, you get \overline{B} at this point and then you take OR of these two.

Since I have got an infinite supply of OR so, I can I can use one OR gate here. So, these gives me \overline{A} OR \overline{B} and then I do another inversion by connecting it to another XOR gate with the other input tied to 1; other input tied to 1. So, here you get F equal to $\overline{A} + \overline{B}$. fine. So, this way we can realize this XOR OR it is also an universal combination. So, you can find any many other universal combination so, which does not include NAND and NOR gate ok. But the fundamental thing like if you are given a only AND OR. So, only these two then it is not an universal set because you cannot do inversion.

Similarly, in the XOR family, so with XOR if AND is not given, OR neither of AND OR OR neither of them are given, then only XOR will not act has an universal set, because in that case you cannot realize all possible basic gates from there.

So whenever we have got a combination of gates that can give us all possible basics function. So, we will call it has an universal set, and then NAND and NOR gates their individually universal gates and we can have other combination like AND, OR, INVERT XOR, AND XOR, OR like that.

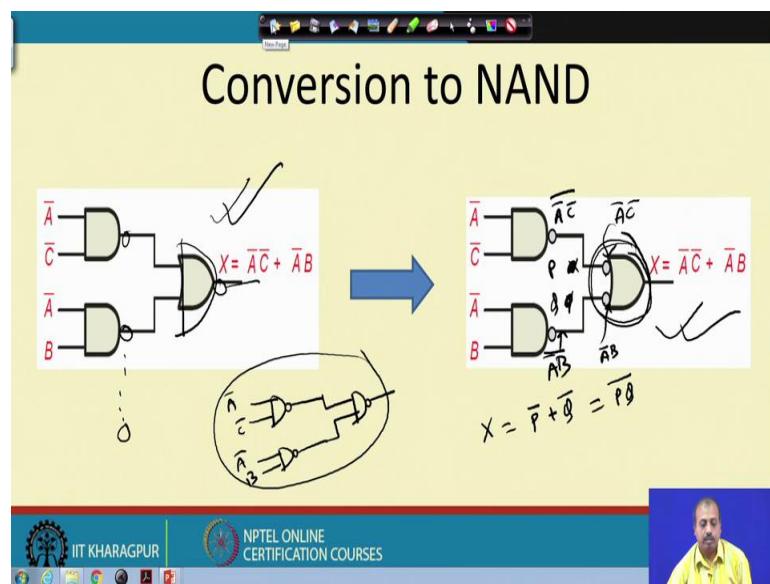
Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 16
Logic Gates (Contd.)

Sometimes we need gates some circuit to be converted into NAND gate circuit, because as I said that NAND is an universal gate so many logic libraries. So, when you are realizing some circuit digital circuit may be only NAND gates are available. So, there are many, **I** integrated circuit chips which has got only NAND gates in them.

So, if we have to realize some circuit using those chips then you have to realize using NAND gates only. So, but normally when you do a Boolean minimization we come up with some sum of product or product of sum expression which consists of a say, AND and OR gates and inverters like for example, in this particular circuit.

(Refer Slide Time: 00:54)



So, ultimate expression that I have got is X equal to $\bar{A}\bar{C} + \bar{A}B$; so it means that we have got this is the OR and then these are the 2 AND gates.

But if I do not have this AND and OR gates. So, I have got only NAND gates then you can do it like this. So, what we do? This $\bar{A}\bar{C}$ so, this AND gate is converted into NAND gate. So, this is the NAND operation, and this is so, this $\bar{A}B$ so, it was a NAND gate here

we convert it into NAND gate. And after that so, there is another bubble here. So, as so, we can understand that this is converting this a here you are getting $\bar{A}\bar{C}$. So, at this point you are getting $\bar{A}\bar{C}$. And after this after this inversion so, you are getting $\bar{A}\bar{C}$.

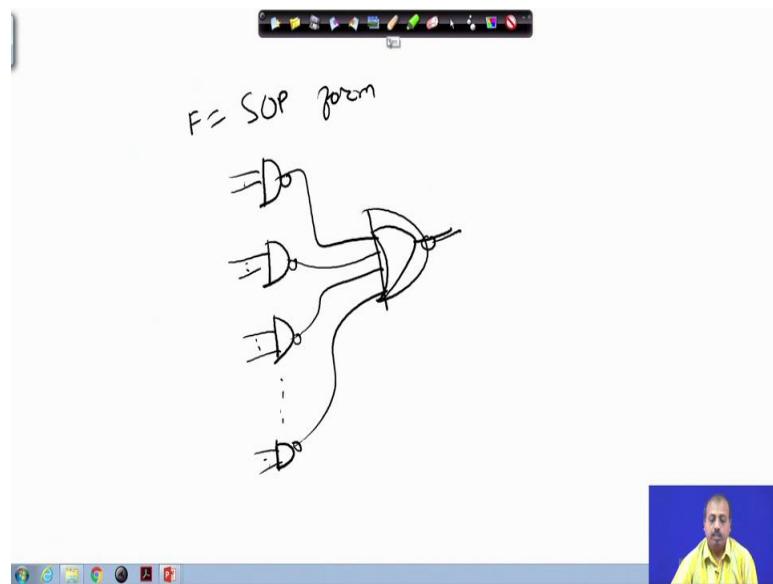
Similarly, at this point at this point you are getting $\bar{A}B$, and after this bubble here second bubble there. So, you are getting so, sorry, evaluating $\bar{A}B$. So, here you are getting $\bar{A}B$ and then as if this OR and that is $\bar{A}\bar{C} + \bar{A}B$. But what is this gate? Ok so this, gate if I say that this input is X and this input is Y. So, what I am realize sorry this in say instead of X and Y let us write something else.

So, let us say this is P and this is Q. So, what we are getting is $X = \bar{P} + \bar{Q}$ because P is inverted here Q is inverted here, and then they are OR'ed. So, if you apply De Morgan's theorem so, this is nothing but $P\bar{Q}$ so, this gate is nothing but a NAND gate. So, we you can simplify that drawing as if that this, you can simplify the whole drawing like this that this is a one NAND gate, where we have got this $\bar{A}\bar{C}$ as inputs and you have got another NAND gate where you have got this $\bar{A}B$ as a input.

And then you have got here another NAND gate, where these 2 are connected ok. So, this diagram that we have here so, this is actually for the sake of our understanding, but we must appreciate that this is equivalent to this, this is equivalent to this which is convert. So, suppose I have got a sum of product expression from there I have got this, this realization of the function in terms of AND and OR gate. So, for converting to NAND gate what you have to do is that whatever AND gates you have here. So, convert or if there are many such AND gates many such product terms.

So, you convert all those ands to NANDs and when this and then finally, convert this OR gate to a NAND gate so, that we will solve our purpose ok. So, I will just so, in general I can say that in a sum of product expression so, f is in sum of SOP form.

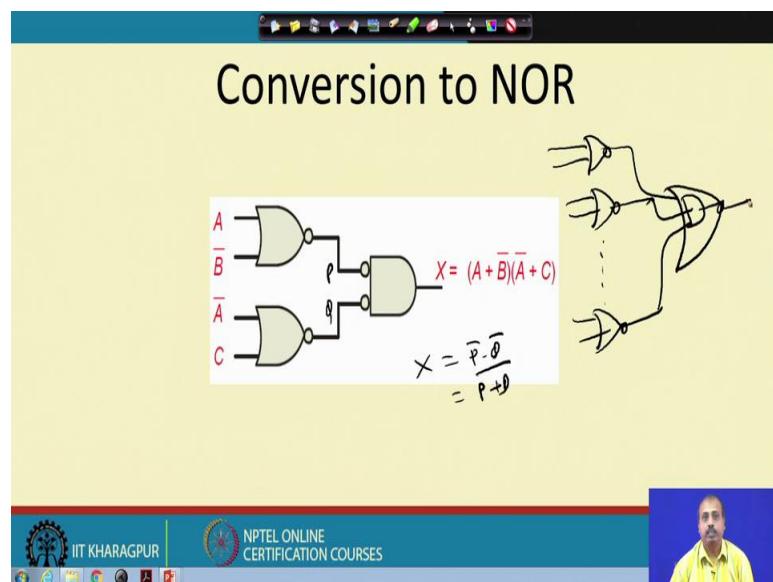
(Refer Slide Time: 04:02)



So, f is in SOP form, then the thus individual sum terms so, they were represented by this AND gates, these were the individual sum terms they were coming there, and then it was feeding the OR gate so, this was the situation.

So, to convert to NAND what you do is you convert each of them to NAND each of them to NAND and finally, convert this OR gate also to a NAND, ok. So, this way we can convert any Boolean circuit into a circuit consisting of NAND gates, ok. So, next we will be looking into how to convert to NOR gate.

(Refer Slide Time: 04:55)



So, NOR gate conversion so, you should start with the product of some expression, and then this A or B bar. So, this is ORed here and then ultimately they were ANDed there; so you just so, this gate is nothing but another NOR gate, ok.

So, if we go by the previous explanation that we had. So, this is P and Q so, $X = \bar{P}\bar{Q} = \bar{P} + \bar{Q}$. So, if you have got a product of sum realization of set of terms, ok. So, you the so, Boolean function realized as a product of sum expression. So, you have got individual inputs there and they are ultimately feeding one AND gate, if this is the situation. Then what you can do you can replace each of this OR gates by NOR gates and then replace this AND gate by another NOR gate. So, that will give you the NOR realization of the function.

So, it starts with the product of sum form and then convert it into, convert the OR gates to NOR gates, and then the AND gate to NOR gate. Or if you and if you have to convert to NAND gate form, you start with the sum of product expression of the function, and then convert all the AND gates to NAND gates and then convert the final OR gate to NAND gate so, they will work.

(Refer Slide Time: 06:30)

Exclusive-OR (XOR) Function

- XOR (also \oplus) : the “not-equal” function
- $\text{XOR}(X,Y) = X \oplus Y = X'Y + XY'$
- Identities:
 - $X \oplus 0 = X$
 - $X \oplus 1 = X'$
 - $X \oplus X = 0$
 - $X \oplus X' = 1$
- Properties:
 - $X \oplus Y = Y \oplus X$
 - $(X \oplus Y) \oplus W = X \oplus (Y \oplus W)$

Handwritten notes and diagrams include:

- $X \oplus 0 = \bar{X} \cdot 0 + X \cdot \bar{0} = 0 + X = X$
- $X \oplus 1 = \bar{X} \cdot 1 + X \cdot \bar{1} = \bar{X} + 0 = \bar{X}$
- $X \oplus X = X \cdot \bar{X} + \bar{X} \cdot X = 0 + 0 = 0$
- $X \oplus \bar{X} = \bar{X} \cdot \bar{X} + X \cdot X = \bar{X} + X = 1$
- A logic circuit diagram showing a three-input XOR gate (labeled X, Y, W) implemented using two NOT gates and one OR gate.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will be looking in more detail this exclusive OR function because this is very interesting function. So, XOR which is represented as this plus within a circle so, these or this is also known as not equal function. Why is it not equal? So, this you see that this Boolean expression for this is X XOR Y written as $\bar{X}Y + X\bar{Y}$. So, whenever X and Y they are not equal output is equal to 1, ok, for the XOR gate. So, that is why it is called a not

equal function so, there are certain identities like $X \text{ XOR } 0$ is $X \text{ XOR }$ one equal to \bar{X} ok. So, we can we can just put this so, $X \text{ XOR } Y$ sorry $X \text{ XOR } 0$, $0 \text{ XOR } 0$, if you put into this expression. So, this is $\bar{X}0 + X\bar{0} = X$ so, you have got the first identity.

Similarly, $X \text{ XOR } 1$ is equal to $\bar{X}1 + X\bar{1} = \bar{X}$. So, that is the second 1. Now $X \text{ XOR } X$. So, $X \text{ XOR } X$ is so, $\bar{X}X + X\bar{X} = X$ ok. So, $X \text{ XOR } X$ equal to 0 and $\bar{X}\bar{X} + XX = X + \bar{X} = 1$ So, these are some identities that involve $X \text{ XOR }$ operation, and this XOR operation is commutative like this $X \text{ XOR } Y$ is same as $Y \text{ XOR } X$. So, that is true for any other Boolean function also and it is also associative like if you have to do an XOR of 3 variables $X \text{ Y}$ and W XY and W . So, this property what it tells is that you can either do this XOR first and then with that result you $\text{XOR } W$. So, that may be one possibility or you first do this $Y \text{ W XOR}$, and then with that result you $\text{XOR } X$. So, both are both will give you the same result so, this is associative in nature.

So, these are some of the properties of this XOR gates that that makes is makes it interesting in various domains or of communication systems Boolean circuits and all circuit design and all ok.

(Refer Slide Time: 09:42)

XOR function implementation

- $\text{XOR}(a,b) = ab' + a'b$
- Straightforward: 5 gates
 - 2 inverters, two 2-input ANDs, one 2-input OR
 - 2 inverters & 3 2-input NANDs
- Nonstraightforward:
 - 4 NAND gates

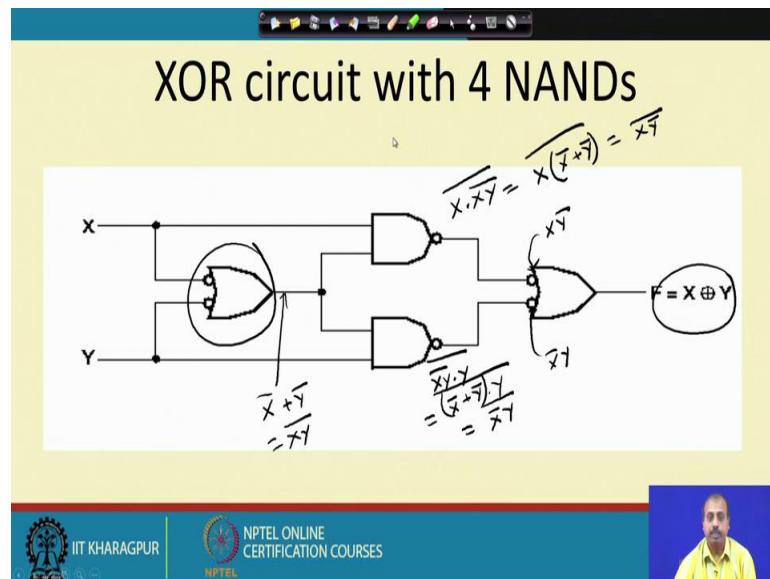
IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES

So, XOR function implementation so, if I have got since the XOR ab is $ab' + a'b$. So, if I do a straightforward implementation in terms of say and OR gates. So, what I will do? I will have this ab' so, I will be taking this a and this is the line a . So, I will take an inverter and this is the line b I will take another inverter so, I will get a' here b' there.

And now I have to do ab' so, ab' means I have to take one AND gate, where this a line has to be connected and this b' line has to be connected and then $a'b$. So, $a'b$ so, this a' line has to be connected and this B line has to be connected, and then they are to be ORed like this. So, as a result number of gates needed is equal to 5 the 2 inverters 2 a to 2 2 input AND gates, and one 2 input OR gate ok.

So, you can also do this by using this part. So, they can be converted to NAND gates each of them can be converted to NAND gates. As a result so, you can do the whole thing using 2 inverters and 3 NAND gates so, that also can be done. However, there exist another clever implementation of this XOR function that includes only 4 NAND gates. So, we will see that.

(Refer Slide Time: 11:18)



So, this is the 4 NAND gate realization of the XOR gate so, we have got this so, here this is basically $x'OR y'$, sorry. So, this point we are getting $x'OR y'$ at this point we are getting $x'OR y'$ so, which is nothing but; so, this is also a NAND gate. So, here I am getting and here it is NAND'ed with x , ok. And then so, you can say, so, this is this is $\bar{x} \cdot \bar{x}\bar{y}$, ok. So, if we simplify so, you will get here as; so $\bar{x}(\bar{x} + \bar{y})$ so, you will get $x\bar{y}$.

Similarly, from this side if you look into so, you will be getting here you will be getting this $\bar{y} \cdot \bar{x}\bar{y} = (\bar{x} + \bar{y}) \cdot \bar{y} = \bar{x}\bar{y}$. Now, here this is another NAND gate so, the after this inversion so, this term is $x\bar{y}$ and this term is $\bar{x}y$, and then I am doing an odd. So, I am getting this or of these 2 so, this is basically the XOR function. So, this way I can do this

XOR realization 2 input XOR gate realization using 4 NAND gates, ok. So, that is say for like we can save some gates, like in the previous implementation in the previous implementation. So, we had here 2 inverters and 3 2 input NAND gates, but in the next implementation we have got only 4 NAND gates. So, we are saving some gates so, this may be useful in some situation.

(Refer Slide Time: 14:07)

- XNOR: the “equality” function
- $\text{XNOR}(a,b) = ab + a'b'$
- Observe that $\text{XNOR}(a,b) = (\text{XOR}(a,b))'$ $x \oplus y = \bar{x}y + x\bar{y}$
 - $(a \oplus b)' = (a'b + ab')'$
 - $= (a'b)'(ab')'$
 - $= (a + b')(a' + b)$
 - $= ab + a'b'$
- $a \oplus b' = (a \oplus b)' = a' \oplus b$

So, just the opposite of XOR is XNOR so, XNOR is known as the equality function so, XOR was non equality function XNOR is equality function, and XNOR of ab is defined as $ab + a'b'$ ok. So, whenever both the inputs are equal so, if a and b both are equal both are equal to 1 then ab is equal to 1, if both of them are equal to 0, then $ab + a'b'$ is equal to 1 as a result this XNOR function will be giving us 1, when either both a and b are equal to each other.

So, we can see that X a in the XNOR of ab is the invert of XOR of ab. So, how do you prove it so, this a XOR b the, this XNOR is nothing but $\overline{a \text{ XOR } b}$, ok, now this a XOR b is $a'b + ab'$. So, this can be you can apply De Morgan's theorem. So, we get $(a'b)'(ab')' = (a + b')(a' + b) = ab + a'b'$

So, we so, if you take the invert of XOR so, you come to $ab + a'b'$ which is nothing but XNOR of ab, ok. Now if you if you just take one of the variables as a complemented one so, $\overline{a \text{ XOR } b}$, that will give us the XNOR of a and b. So, let us check that; so, let us say $a \text{ XOR } \bar{b} = a\bar{b} + \bar{a}\bar{b} = ab + \bar{a}\bar{b}$ so, this is the XNOR.

So, this $a \text{ XOR } \bar{b}$ so, this will give us this XNOR of a and b. And similarly by the similar logic this $b \text{ XOR } \bar{a}$ will also give us the XNOR of a and b. So, these are the relations between XOR and XNOR.

(Refer Slide Time: 16:36)

Odd Function

- $x \oplus y = x'y + xy'$
- $x \oplus y \oplus z = xy'z' + x'yz' + x'y'z + xyz$
- $x \oplus y \oplus z \oplus w = x'yzw + xy'zw + xyz'w + xyzw' + x'y'z'w + x'yz'w' + x'y'zw' + xy'z'w'$
- ... Observe a pattern here?
- An n-input XOR function is implied ($=1$) by all the minterms that have an odd # of 1s
- Thus, XOR is also known as the ***odd function***

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

So, this XOR is also known as odd function; so, why a odd function? So, you see that if you if you take a 2 input XOR gate, then this is the logic expression. So, $x \text{ XOR } y$ is $x'y + xy'$.

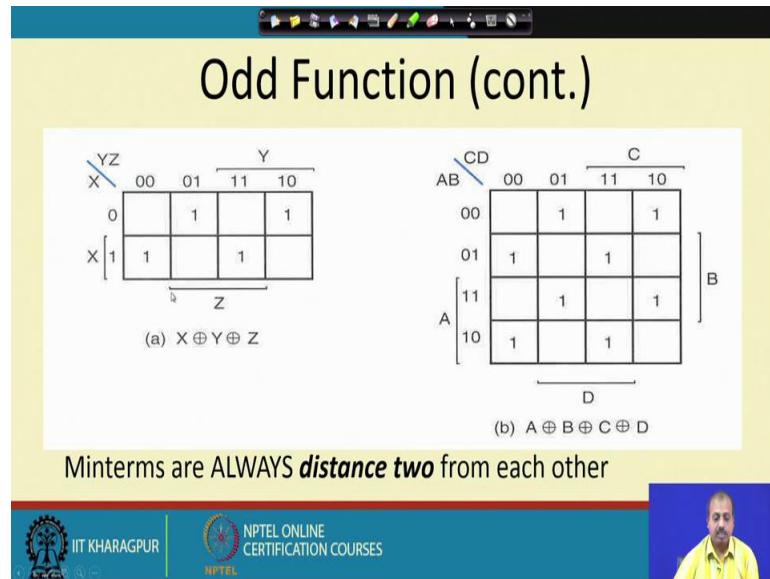
So, if you take a 3 input XOR gate, 3 input XOR function $x \text{ XOR } y \text{ XOR } z$, then this if you expand so, you will get an expression like this $xy'z' + x'yz' + x'y'z + xyz$. If you take a 4 input XOR function, then the expression will be like this. So, can we observe a pattern here, of course, there is a pattern. So, in all of them so, the terms where odd number of inputs are equal to 1, it is surviving, like for the 2 variables so, here you see that these 2 terms they have got only one input equal to 1, that here only y equal to 1 here only x equal to 1.

So, if you look into say this term so, here only x equal to 1 y and z are equal to 0. Similarly in this term only, y is equal to 1 $x'x$ and z are equal to 0. If you like take a 4 variable term, this $xyz'w$ so, here x y and w they must be equal to 1, and z must be equal to 0. So, whatever you do? So, you see that here 3 variable, the 3 variable should get the value one and one of them should get value 0.

Similarly here so, only x variable should get the value 1, and 3 of them should get value 0. So, either one variable is getting in a 4 variable XOR function, either only one of the input is equal to 1 or 3 of the inputs are equal to 1, then only output is equal to 1. So, in a general case we can say that when odd number of inputs are equal to 1, then we will have the XOR gate output equal to 1.

So, an n input XOR function is 1, by all minterms that have an odd number of ones in it. So, that is why XOR is also known as odd function. Because it will have the odd number of minterm, odd number of ones equal to 1 in those minterms.

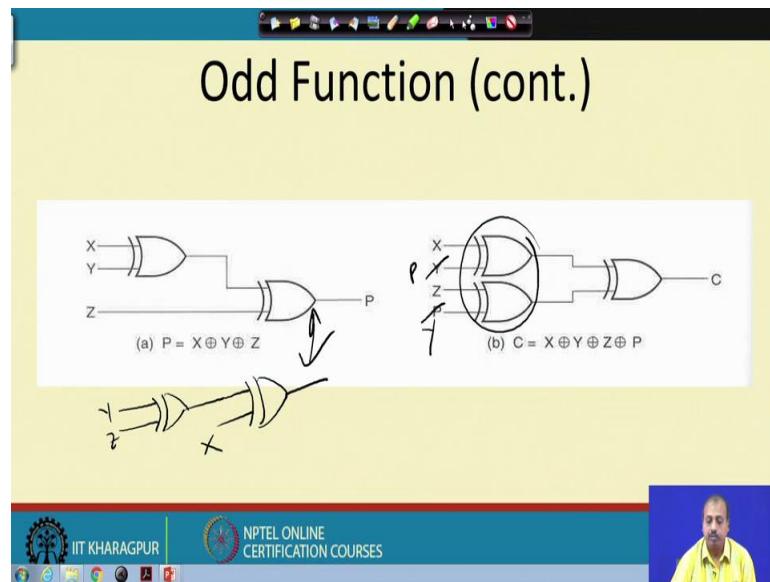
(Refer Slide Time: 18:52)



Now, if we just look into that truth table the Karnaugh map; so, this is for the 3 variable so, X Y and Z. So, you see that they will be distributed like this. So, this so, they do not group in a quad or pair or octet like that so, they get distributed like this

Similarly, a 4 variable Karnaugh map so, this ones are getting distributed in such a fashion, that they are not forming any pair, ok. So, minterms are always distance 2 from each other so, this is the property of odd function.

(Refer Slide Time: 19:29)



So, you can realize these odd functions like this. So, this X XOR Y coming here and X then Z so, X XOR Y XOR Z so, we can realize by means of 2 XOR gates.

If I have take 4 variable XYZ and P, then this X and Y can be XOR'ed first then Z and P can be XORed, and then this one can be XOR. And it does not matter in whatever order we do it. So, somebody may realize the same function like this also somebody may say that I will do this Y and Z XOR first, I will do this Y and Z XOR first, and then I will XOR this output with the X, ok. So, that is because of this associativity property of this XOR operation. So, it does not matter the order in which we do the XOR'ing so, these 2 are exactly same.

Similarly here also so, you can change the pair, like in instead of taking Y here you can take P here, and instead of P you put the Y here. So, that can be done ok. So, that will not change the expression at all it will remain all the same, due to this associativity property of this XOR.

(Refer Slide Time: 20:47)

Even Function

- How would you implement an even *function*?

The complement of XOR → XNOR

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can also have something called an even function. So, even function if you want to realize. So, how can we do this? Like complement of XOR which is the XNOR.

So, XNOR gives us the even function.

(Refer Slide Time: 21:00)

Parity Generation and Checking

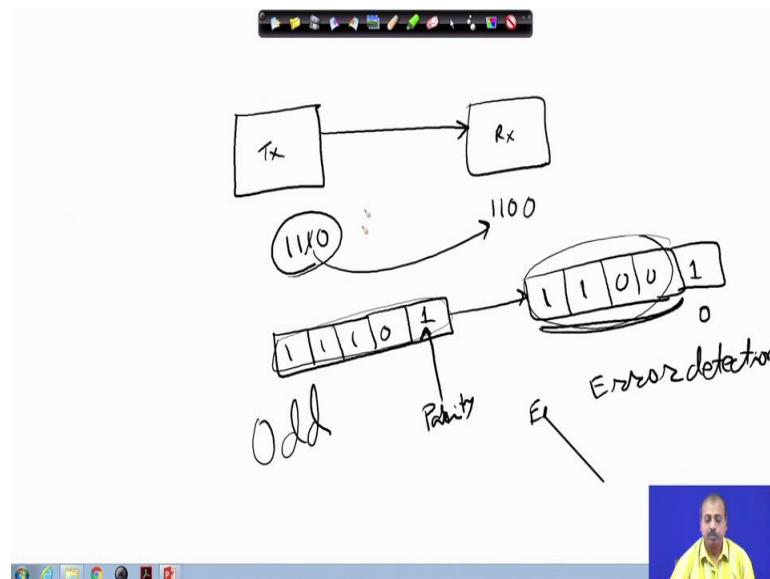
- Odd and even functions can be used to implement parity checking circuits used for error detection and correction.
- Use even parity as example.
- *Parity generator*: the circuit that generates the parity bit before transmitting.
- *Parity checker*: the circuit that checks the parity in the receiver.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, many a times we use this XOR to generate the parity. So, whenever we have got an odd and even function, they can be used for implementing parity checking circuits in error detection and correction. So, what is parity? So, parity is something like this, that suppose,

you are you have got a communication system or a where there is a transmitter and a receiver.

(Refer Slide Time: 21:26)



So, this is the transmitter and this is the receiver so, you are transmitting some bits from here to here. Now suppose I transmitted the bit pattern 1 1 1 0, now at the receiving end, I should receive I get I should get this 1 1 1 0, but due to this communication channel, the some error may into the keep into the transfer. So, it may so happen that this one at the receiving end it has become 0 so, at the receiving end it receives 1 1 0 0. Now at the receiver side how do we know that we have received a possibly correct pattern.

So, for that what is done is; apart from this 4 bits that we are transmitting. So, this 1 1 1 0, this 1 1 1 0 another bit is added. So, this bit is called parity, this bit is called parity. So, parity is actually it keeps a watch on the number of ones that you have sent onto the bit stream. Like, if you say that this parity so, we can say that whenever we transmit we always transmit an even number of ones. So that means, in this case we are transmitting 3 ones so, that is odd number so, this bit we make it one.

So, that at the receiving end so, if you receive a pattern like this so, assuming that this parity bit has been transmitted properly; so, the received bit pattern is something like this so, you have got 1 1 0 0 and then parity bit is 1. But from this if we calculate the parity bit, since, we have taken the protocol that we will always be transmitting even number of 1's. So, based on the received bits so, since these 2 bits are 1 this bit should actually should is

expected to be 0. But the receiver finds that it has received the bit one. So, it understands that there must be something wrong in this transmission so, receiver can detect the situation ok.

So, this is called error detection so, this is called error detection. So, the receiver can detect that there is some error in the transmission so this parity is actually used for this part. So, apart from the bits that you are transmitting; so you transmit the another extra bit now you may follow even parity, where you check whether you make the number of ones even, or you can make you can follow the other option other alternative also you can say it is a odd parity that is the total number of 1's, that you will transmit you will make it odd, ok.

So, both of them can be done using this um using this XOR gate. So, this parity generator so, this circuit generates the parity bits before transmitting, and this parity check so, this circuit is present in the transmitter. So, this will generate the parity bit, and that the receiver side you will have a parity checker circuit. So, it will check the parity after the bit pattern has been received.

(Refer Slide Time: 24:36)

Even Parity Generation

Three-Bit Message			Parity Bit
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- $P(X,Y,Z)$ must produce a 1 for all the input combinations that contain an odd number of 1s
- Thus, it is a 3-input odd function $P = X \oplus Y \oplus Z$

IIT Kharagpur **NPTEL ONLINE CERTIFICATION COURSES** **NPTEL**

So, like this so, suppose I have got this situation so, even parity. So, I will make the number of ones transmitted as even.

So, if it is the bits transmitted at 0 0 0, then the parity bit is also 0; because number of ones is equal to 0, that is even if the message bits are 0 0 1 then the parity bit will be equal to 1

so, that number of ones in the whole message plus parity is even. Similarly, 0 1 0 also parity bit is 1, 0 1 1 the parity bit is 0. So, if you so, this way for the for a 3 bit message so, we can write down what is the parity bit.

So, $P(X,Y,Z)$ the parity must produce one for all input combination that contain an odd number of 1's. And whenever we get this statement, we understand that without doing any Boolean minimization anything, this statement straightway tells that this is an odd function so, this is basically XOR of X, Y and Z. So, this parity so, even parity generation can be done by means of the XOR gates. So, that is why XOR is a very popular gate in the communication system.

(Refer Slide Time: 25:54)

Even Parity Checking

How would you implement a parity checker for the previous example?

Use a 4-input XOR circuit (odd function)

$$C = X \oplus Y \oplus Z \oplus P \rightarrow 1 \text{ indicates an error}$$

OR

A 4-input XNOR circuit (even function)

$$C = (X \oplus Y \oplus Z \oplus P)' \rightarrow 1 \text{ indicates a pass}$$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, how do we check the parity? So, you can do the operation like this. So, you can have a 4 input XOR circuit so, you can have this since you are getting 4 bit XYZ and P and if you do an XOR operation. So, if you get an one; that means, there is an error because even parity means if you XOR with the parity bit also, if you get always get a 0 so, if you get a one; that means, there is an error.

So, for if you are on the other hand, if it is an odd parity, then if you are doing this if you can use a 4 input XNOR gate, and so, in this case if you get a one; that means, it is it is all right so, if it is 0 then there is a problem, so this way using this XOR and OR gate. So, we can use this even parity checking. So, for even parity checking either you can use a 4 input XOR gate or in that case say one will indicate an error. Or you can you can use a 4 input

XNOR gate. In that case, the output one will indicate that it is correct, 0 will indicate that there is a problem, ok.

(Refer Slide Time: 27:03)

Logic Families

- DTL
- TTL
- ECL
- NMOS
- PMOS
- CMOS
- Etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you can also use a, similarly you can also take care of the odd parities. Next we will be looking into logic families. Like this different this so, for we are talking about logic gates and logic symbols. Now it is the same logic gate how do you realize like, one implementation we have shown in terms of switches and lamps at the very beginning of this discussion, now there are many circuit elements that can be utilized in like say diodes transistors etcetera.

(Refer Slide Time: 27:48)

Even Parity Checking

How would you implement a parity checker for the previous example?

Use a 4-input XOR circuit (odd function)
 $C = X \oplus Y \oplus Z \oplus P \rightarrow 1$ indicates an error

OR

A 4-input XNOR circuit (even function)
 $C = (X \oplus Y \oplus Z \oplus P)' \rightarrow 1$ indicates a pass



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



That can be utilized for realizing different logic functions for getting different circuits. For example this DTL so, this stands for this stands for this diode transistor logic, and then that, sorry, this diode transistor logic. So, this is this consists of diodes and transistors, then we have got TTL which is transistor logic. So, here all transistors are used for getting the circuit, and for getting this basic logic gates. Then there is a ECL logic. So, this is again another family, then do you have got NMOS which is metal oxide semiconductor family. And this n stands for n channel metal oxide semiconductor. Then PMOS P channel metal oxide semiconductor, then CMOS this is a complementary metal oxide semiconductor.

So, this way there are many such logic families that has been used for developing this basic logic gates realizing this basic logic gates over the years. So, DTL is one of the first logic families that we had involving diodes and transistors, TTL came after that. And at present so, we are working with the CMOS family, ok. This CMOS family is the most popular now, because of several reasons, because of its compactness, because of its power consumption less power consumption and all. So, this CMOS is going to be, is right now it is the de facto standard that we have for this integrated circuit chips ok.

So, we will look into this CMOS in detail in the successive lectures.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 17
Logic Gates (Contd.)

So, we will look to static CMOS logic family. Why static so, we will come to that later, but as I have said that CMOS is the de facto standard for this digital circuit, digital gate realization. Now, we will see that in successive slides.

(Refer Slide Time: 00:35)

Dopants

- Silicon is a semiconductor
- Pure silicon has no free carriers and conducts poorly
- Adding dopants increases the conductivity
- Group V: extra electron (n-type)
- Group III: missing electron, called hole (p-type)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now, as far as the CMOS realization is concerned so, this is done on Silicon. Silicon as we know that it is a semiconductor and pure Silicon is in a Silicon crystals. So, it consists of this Silicon atoms and they form bonds between them so, there is no free carrier. So, neither electron nor hole nothing is free there. So, it is just forming a crystal like this so, it does not conduct.

However, if we add some dopant into it so, we can either add dopants like Arsenic or this Boron ok. So, this type of dopants group 5 elements or group 3 elements. So, Silicon is a group 4 element in the periodic table. Now, if we it has got 4 electrons at the outer valence, now if you are having so, if you are adding say this is a arsenic then arsenic is a group 5 is a group 5 element. So, that is it has got an extra electron and said that extra electron does not bond with anybody. So, that is why if you this extra electrons they are they can move

freely ok. So, if you apply some potential then this electrons can be made to flow very easily.

So, by adding this extra electron in the so, or doping the pure Silicon with arsenic so, we get something called n-type semiconductor. In an n-type semiconductor we have got plenty of electrons available which can be made to flow if we apply some potential difference. On the other hand, so instead of applying a group 5 dopant; so if we apply a group 3 dopant so, group 3 dopant we have got 3 electrons at the outer valency. So, there it has got a tendency to capture one more electron. So, in some sense it is like a hole that is there in the outer valency.

So, that is why this type of dopants so, they are called donor and they are called acceptor and these this has got a hole ok. So, group 3 they have go missing electron as if as if it has got a hole and this type of semiconductors that you get so, they are called they are they are p-type dopant.

And if you put this Boron type of impurity into this Silicon crystal, Silicon wafer then we will get a p-type semiconductor ok. So, this is both n-type and p-type semiconductors are there in realizing different circuits like starting your diode to transistors to MOS and all. So, we will using them particularly in the context of this MOS transistors.

(Refer Slide Time: 03:10)

nMOS Operation

- Body is commonly tied to ground (0 V)
- When the gate is at a low voltage:
 - P-type body is at low voltage
 - Source-body and drain-body diodes are OFF
 - No current flows, transistor is OFF

The diagram illustrates the nMOS transistor in its off-state. The gate electrode is at a low voltage, creating a reverse bias across the source-body and drain-body junctions. This results in no current flowing through the channel between the source and drain, despite the presence of majority carriers (electrons) in the source and drain regions. The body of the transistor is connected to ground.

IIT Kharagpur | NPTEL Online Certification Courses

So, a basic nMOS transistor the operation is like this so, we have got this bulk Silicon. So, this is the bulk Silicon so, this bulk Silicon so, we can say it is an wafer, wafer of Silicon and this is a p-type Silicon. So, this is a doped with Boron so, that this is a p this is the this rectangular region that we have drawn so, you can take it as a p-type semiconductor.

Now, on that we have got two heavily doped region of of n-type semiconductors ok. So, there that is done by means of this diffusion process that you can learn in some course on VLSI design. So, this is the some diffusion is done so, that you get this n-plus regions two such n-plus regions. Between the two n-plus region so, we have got a Silicon dioxide layer which is an insulating material and on top of the Silicon dioxide layer so, we have got a diffusion of this poly Silicon layer. So, we have got a deposition of poly Silicon.

So, what happens is that if you if you do not apply any extra potential ok. So, this p-type voltage p type so, this is the body or the bulk Silicon so, this is kept at a low voltage. Now, if this source body and drain body diodes are OFF. So, how can this be OFF? So, if you do not apply any voltage to this source and drain all these diodes are off so, so there will be no current flow.

However, if you apply some positive voltage in the gate; so if I apply some positive voltage in the gate then what will happen; electrons that are there in this substrate this p-type substrate. So, they will be drawn toward this region, they will be drawn towards this region. So, we have got the electrons coming here.

So, so far I have applied only some gate voltage I have not applied any voltage at source and drain. Now, if you apply some a voltage difference between source and drain ok, then this electrons may be made to flow from one side to the other; like if we make this drain voltage high and this source voltage low then the electrons will be drawn towards the drain as a result as a current will flow from drain to source.

So, when I do not apply any gate voltage. So, this transistor is OFF, but as we are applying some voltage gate voltage the transistor is this on and if we have a potential difference between drain and source then the current will flow through the device. So, pictorially, it is represented by a by a symbol like this. So, this is so, this particular symbol we have got two terminal source and drain and then the other terminal. So, this is called the gate or G ok. So, this is called the gate or G. So, when G equal to 0 then this source drain there is no current flow between source and drain, otherwise there will be a current flow.

(Refer Slide Time: 06:26)

The slide is titled "Transistors as Switches". It contains two sections: "nMOS" and "pMOS". Each section has three diagrams corresponding to gate voltages $g = 0$, $g = 1$, and $g = 0$ again. In the first diagram ($g = 0$), the drain terminal (d) is connected to the source terminal (s) by a solid line, indicating the switch is OFF. In the second diagram ($g = 1$), there is a vertical arrow pointing from the gate (g) to the drain (d), and the drain (d) is connected to the source (s) by a dashed line, indicating the switch is ON. The third diagram ($g = 0$) is identical to the first. The symbols for nMOS and pMOS are standard electronic symbols with a triangle and a circle respectively.

So, so, these are the different combination like say I can have a as I was telling that this is the symbol we have got three terminal, one drain terminal, one source terminal and one more gate terminal.

So, this gate is something like a switch ok. So, gate when we apply a voltage here when you when we apply a voltage here then the gate as if the electrons are drawn into this region; as a result the channel gates established or the channel. So, the this the becomes ON.

Similarly, when like if now if you have a potential difference between d and s that is if g equal to 1 then there can be a current flow between d and s. So, this g acts as a switch between d and s. So, when g equal to 0 so, in that case this switch is OFF and when d equal to 1 then the switch when a g equal to 1 then the switch is ON. So, when g equal to 0 the switch is OFF, when g equal to 1 the switch is ON that is for nMOS type of transistor.

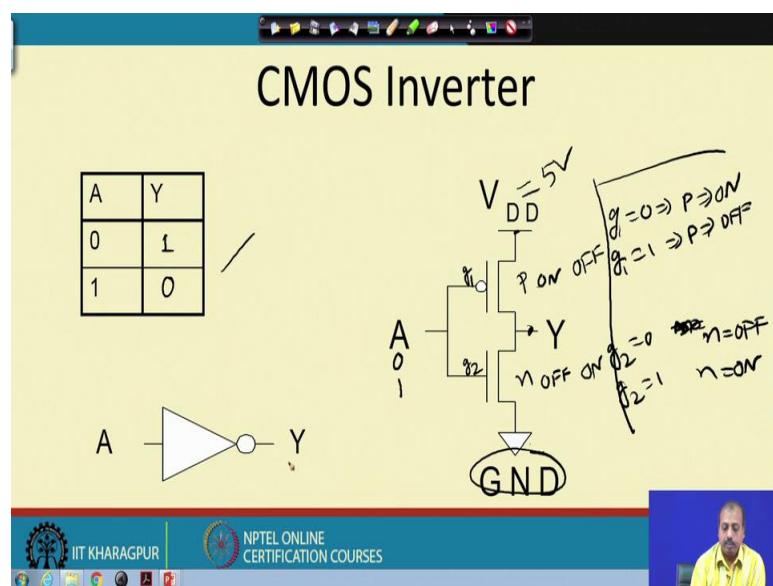
In pMOS type of transistor what will happen is that so, this is the channel is made by holes ok. So, these are made by holes and for making hole so, we have to apply a negative voltage so, or we can say I have to apply a 0 here. So, if I apply a 0 then the pMOS transistor is ON and if I apply a 1 then the pMOS transistor is OFF. So, this way this by controlling this gate voltage so, we can consider the transistor to be ON or transistor to be OFF. For nMOS transistor it is ON if g equal to 0 and it is OFF sorry it is OFF if g equal to 0 and it

is ON if g equal to 1. For pMOS transistor it is ON if gate equal to 0, g equal to 0 and it is OFF if g equal to 1.

So, in so, we can use either of this two types of transistors nMOS and pMOS transistors and different logic families have been built, like say nMOS family that I have said. So, that actually uses this nMOS type of transistors to get a different types of gates like a AND gate, OR gate, NAND gate like that; pMOS also can be used for that.

And in a CMOS so, we have got both nMOS and pMOS transistors for realizing the logic circuit logic gates ok.

(Refer Slide Time: 08:57)



So, a CMOS inverter looks like this so, we have got this is the symbol of a CMOS inverter. So, upper transistor so, whenever we have got a bubble here so, this means this is a p-type transistor. So, this is a p-type transistor and this is an n-type transistor. So, p-type transistor means if g if I say this is g so, this is say g1 and so, this is say g2 when g1 is equal to 0 then p is ON, when g1 equal to 1, p is OFF ok. Similarly, for n-type transistor if g2 equal to 0, then the n-type transistor is OFF, then n is OFF and if g is equal to, g2 equal to 1 then n is ON. So, this is the situation.

Now, let us consider the case where this 2 gate terminals of this p-type transistor and n-type transistor they have shorted and the input A is applied to both of them. Now, what happens? Now, if A equal to 0 so, if A equal to 0 in that case this transistor is as per this

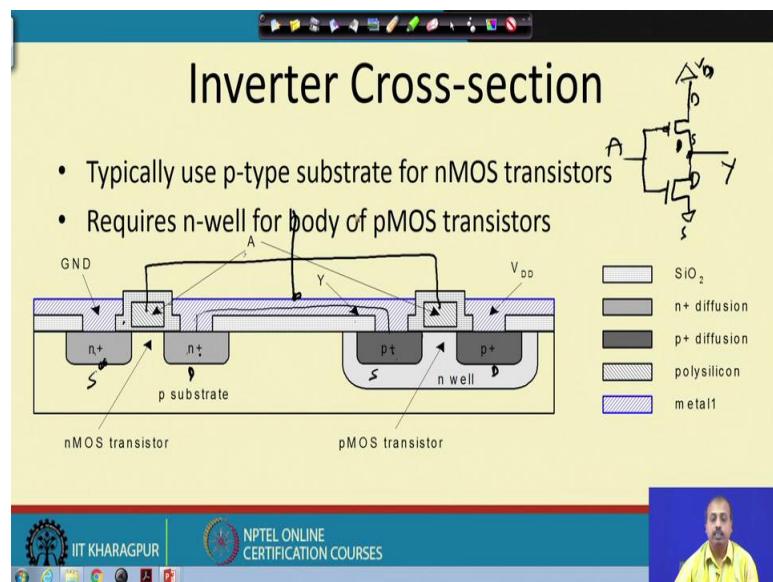
writing. So, as per as this transistor is ON and this transistor is OFF fine; so this upper transistor is ON and lower transistor is OFF.

Now, if I measure the voltage here how much what is the value I will get. So, I will get a value close to V_{DD} that a V_{DD} is say high so, this is this is say some 5 volt then I will get a 5 volt here. So, I can say if A is equal to 0 then I am getting Y as 1 logic high which is say equal to 5 volt. Now, if A equal to 1 in that case what I will get I so, in that case this p is OFF. So, this is OFF now and this is ON now, the lower transistor is ON now. Now, if I measure the voltage here I will get a value close to this ground potential which is 0. So, I can say if it is 1 then I will get a 0 there.

So, what is this functionality this is nothing, but an inverter. So, it is so, I can say this is an inverter when A equal to 1, Y is equal to 0 when A equal to 0, y equal to 1. So, this basic circuit so, this basic circuit acts as an inverter and it has got 2 transistors of two different type nMOS transistor and pMOS transistor. So, we call it; that is why it is called as CMOS inverter so, it has got complementary metal oxide semiconductor inverter.

So, complementary means it has got both p and n components in it and the input is feeding to both the n transistor as well as the p transistor. So, this way I can have this CMOS transistor CMOS inverter.

(Refer Slide Time: 12:13)



So, next we will be looking into a cross section like how this CMOS inverter looks like if I just take such a device which is nMOS inverter and take a cross section of it. So, normally what is done is that so, for as we have said you seen in the first slide. So, we have got a p type substrate on to which we have got the n n type regions.

So, we have got p type substrate on to which we have got this n^+ regions for the drain and source and then we in between we have got Silicon, this is a Silicon dioxide and here we have got the gate ok. So, this is the poly Silicon gate that we have so, these are the ligands that we have used.

Now, you see that these inputs are so, if this is the source, this is the drain so, this is the source and this is. So, for the for the p type transistor what we have to do is I cannot use a p type substrate I have to use an n type substrate. So, for that what is done within this p type substrate one n type well is found ok, region where I do some n diffusion. So, it creates an n type well and within this n type well so, we do this p^+ type of diffusion to create the p^+ regions.

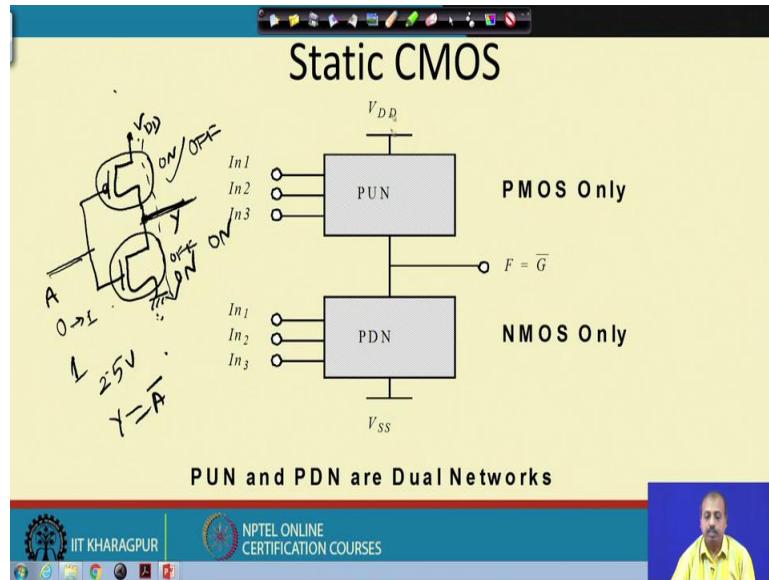
Now, using this metal using this metal line so, this is a blue colored regions. So, this region you see that these two are shorted so; that means, the there is drain of the **p** n transistor and the source of p transistor so, they are shorted. So, if I draw in terms of diagram like if the lower transistor is the lower transistor is the n type transistor so, this the source. So, this end this point is that S that I have.

So, this gate is there then this drain is connected to the source of the p transistor. So, this is nothing, but something like this so, this is the p transistor ok. So, this the drain and this is the source of this is the drain of p and this is the so, sorry. So, this is the source of p and this is the drain of n. So, they are shorted. Similarly, the drain of p so, it is connected to V_{DD} .

So, this is connected to V_{DD} as it is shown here and we have got these 2 terminals these 2 gates are available so, these two are to be shorted ok. So, these two are to be shorted and the A input has to be connected here and I have to take output from this point ok. So, this is the shorting out of from the two shorting so, this is the Y output which is going out. So, we can I have I can put some additional metal lines to connect short the 2 gate inputs and apply A here. So, as it is shown here that the A is applied to both of them and Y is taken from here.

So, this way we can have a n type sorry we can have the CMOS inverter implemented using this CMOS technology. So, this A is going to both the gates and Y is taken from the shorted point of this source this drain and source, drain of the p type transistor and the source of n type transistor. So, this way we can realize this CMOS inverter.

(Refer Slide Time: 15:56)



Now, in general so, if you look into static CMOS family then what happens is that we so, it can be the whole circuit can be divided into two regions. One is called the pull up network, another is called the pull down network. So, PUN is the pull up network and PDN is the pull down network and these inputs are connected to both pull up and pull down network. So, this In_1 , In_2 , In_3 so, these are the 3 input they are connected to pull up network and they are also connected to pull down network and then output.

So, V_{DD} is connected to the pull up network and this V_{SS} or ground is connected to the pull down network. And the output is taken from the middle point of this pull out and pull down network at from the shorted point of them and you get the output as F. So, if we if we take the example of this nMOS inverter sorry CMOS inverter you see the way we did it is was like this that this was this was a pMOS transistor and this was an nMOS transistor. So, this was grounded and the input A was connected to both and we were taking the output from here. So, this was connected to V_{DD} so, this was the Y point.

Now, you try to correlate. So, you see that input is so, this is my pull up network consisting of only one transistor and this is my pull down network consisting of again only one

transistor. Now, you see the input is going to both pull up network and pull down network. **VDD** is connected to the pull up network and ground is connected to the pull down network and the point of shorting between pull up and pull down network from that middle I have taken out, the output ok.

Now, how does it operate? So, you see that whenever the this so, in this in the case of inverter what happens is when A was equal to 0, when A is equal to 0 then this transistor was ON and this transistor was OFF. As a result if you look into the current flow through the device so, there is no current flow from V_{DD} to ground because this transistor is OFF. So, there is no current flow path from V_{DD} to ground.

Similarly, when A is equal to 1 then this transistor is OFF and this transistor is ON, as a result there is no current flow path from V_{DD} to ground again. So, the in case of static CMOS whenever the input is stable so, whenever the input is 0 or input is 1 either the pull up network or the pull down network. So, either of them will be OFF; as a result there is no current flow path from this V_{DD} to the ground and that is why it consumes very little power.

So, when this current flow path will exist only when this A makes a transition from 0 to 1. So, it is somewhat intermediary we say around 2.5 volt so, around 2.5 volt what will happen is that this transistor will be ON and this will be also be ON and this will also be ON. So, for that time there will be some current flow through this device, but that will last for a very small amount of time because soon A will be going beyond 2.5 volt and as a result the upper transistor will be turned OFF.

So, whenever this A makes a transition from 0 to 1 or 1 to 0 the there will be some current flow through the device and it will consume some power; otherwise it will not consume any power ok, that is the advantage of CMOS. So, that it consumes very less power because there is no static current flow through the device.

Now, this pull up network it is implemented using pMOS transistors and pull down network is implemented using nMOS transistor. So, they are called dual network because here where the same functionality is implemented by pull up network and pull down network ok. One and another important thing that you get here I am getting here Y as a function where, $Y = \bar{A}$. So, in general so, you will normal you will get the complement of a function like this we will see some example that will clarify it.

(Refer Slide Time: 20:23)

Complementary CMOS Logic Style Construction

- PUP is the **DUAL** of PDN
(can be shown using DeMorgan's Theorem's)
 $\overline{A + B} = \bar{A}\bar{B}$
 $\overline{AB} = \bar{A} + \bar{B}$
- The complementary gate is inverting

$Y = \overline{AB}$

$Y = A \cdot B$

AND = NAND + INV

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

See suppose I have to get this pull up network is the dual of pull down network. So, for example, if it is a what is done is that in case of pull up network. So, if I am doing using say so, if I have connected the say 2 transistors like this 2 nMOS transistors A and B ok. So, I have so, I am not drawing the pull up network I am just assuming that there is some resistance here and then from here I have taken it to some output Y and this is connected to V_{DD} .

Then what happens is that when this A and B both are equal to 1 then only you get a 0 fine; either of them being equal to 0 output will be equal to 1. So, what is this logic? This is nothing, but the NOR gate a either of them being equal to 0, the output will be equal to 1. So, either of them being equal to 1 output is 0. So, you get a Y equal to A NOR B, any of them sorry I am extremely sorry.

So, this will be equal to output will be 0 when both of them are equal to both the A and B are equal to 1 and it will be equal to it will be equal to 1 whenever at least one of them equal to 0. So, Y equal to so, if any of them is equal to 1 then if, both of them are equal to 1 then the output will be equal to 0. So, you get an AND operation sorry NAND operation of A and B. So, any of them when both of them are equal to 1 then only output will be equal to 0; if any of them equal to 0 naturally there is no current flow path there is there is no discharge path for this Y to ground. So, in that case at this point if you measure so, you will get equal to V_{DD} so, you get a 1.

So, this way I can see that if I put this 2 transistors in series then I get the 2 nMOS transistors in series I get the NAND function. So, for the pMOS part what we will have to do is we have to take these two in parallel ok. So, let us see why? So, suppose this is my V_{DD} and here I have got this A and B. This is A, this is B and I have connected this two and there is a resistance here which is grounded and here it is the output Y.

Now, what happens here is that so, here if I will get a 1 for a getting a 1 so, either of this A or B should be equal to 0 ok. So, if any of them is equal to 0 then at this point if I measure a voltage so, I will get a equal to V_{DD} . But if both of them are equal to 1 then there is no current flow path so, as a result I will get a 0 there.

So, this is also realized this is the function A equal to \overline{AB} . So, the same NAND function whenever putting them in nMOS transistor so, you are getting them connected in series. And if you are connecting if you are putting in cm pMOS transistor so, we have to connect them in parallel. So, that is the duality between this pull up network and pull down network.

Since, these pMOS transistors are implementing the pull up network and these nMOS transistors are implementing the pull down network. So, whatever you connect in series in case of pull down or nMOS network. So, you connect them in parallel in the pMOS network or pull up network and vice versa. So, whatever is connected in parallel in the nMOS network or pull down network so, you connect them in series in the pull up network so that that goes vice versa.

And another point is that you will never get a true output in the in the Y like you cannot get say Y equal to AB. So, this is not possible using this CMOS family so, you cannot get Y equal to AB. So, you will always get the NAND function ok. So, you will get the NOR function or you can get the inverter function, but you cannot get AND, OR things like that. So, that type of functions cannot be obtained.

However, that does not matter much because if you need an AND gate so, you can have you can first realize one NAND gate and follow it by an inverter. So, that can be done so, that way we can realize in CMOS family oh. So, we can have this NAND gate followed by inverter to get AND gate.

(Refer Slide Time: 25:36)

Example Gate: NAND

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table of a 2 input NAND gate

$PDN: G = A \cdot B \Rightarrow$ Conduction to GND
 $PUN: F = \overline{A} + \overline{B} = \overline{AB} \Rightarrow$ Conduction to V_{DD}

$G(I_{n_1}, I_{n_2}, I_{n_3}, \dots) \equiv F(\overline{I_{n_1}}, \overline{I_{n_2}}, \overline{I_{n_3}}, \dots)$

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

So, how do we realize the NAND gate? So, here is a here is an example. So, you see suppose I have got this is a NAND gate truth table 0 0 1 0 1 1 1 0 0 and 1 1 0 first. So, this is the corresponding circuitry as I was telling so, there will be a pull down network consisting of nMOS transistors and there will be a pull up network consisting of the pMOS transistors. And if you look into these two networks so, they are dual up each other, like whenever I have got this nMOS transistors in series. So, corresponding pMOS transistors are in parallel and we will see vice versa. So, if you put these nMOS transistors in parallel for some circuit requirement so, you have to put the nMOS transistors in series.

So, let us see the combination that when this A equal to 0 and B equal to 0 output is equal to 1. How does it happen? So, both of them are equal to 0 means both the p transistors are ON and both the n transistors are OFF. As a result if you measure the voltage so, you will see the voltage value equal to V_{DD} .

Similarly, if it is 0 1 so, A transistor so, this is 0 and this is 1, as a result this transistor is ON and this transistor is OFF. Similarly, this transistor is OFF and B transistor is ON; however, since these two are in series so, it does not matter. So, there is path from this output to the ground, but there is a path from V_{DD} to OUT via this A transistor; because A is equal to 0. So, you get the output as 1.

Similarly, when both the inputs are equal to 1 then both the upper p transistors are OFF and both the n transistors are ON; as a result there is a current flow path from this output

to these things. So, you get a voltage equal to 0 at this point. So, that way this particular circuit it can realize it can realize the NAND operation. So, you can so, in general we can say that this pull up network pull down network is AB and pull up network is $\bar{A} + \bar{B}$ which is \overline{AB} . So, that is conduction to V_{DD} and this is conduction to ground.

(Refer Slide Time: 27:49)

Example Gate: NOR

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table of a 2 input NOR gate

$OUT = \overline{A+B}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can also realize NOR gate in a similar fashion like for NOR gate we have to do this A and B in the pull down network. This A and B they should be connected in parallel and in the pull up network this is A and B they should be connected in series. So, whenever it is this is a lower network is in parallel, upper network is in series and vice versa. So, here we can quickly check that when A and B both are equal to 0 output is equal to 1.

If A and B both are equal to 0, then both the lower transistors are OFF and both the upper transistors are ON. As a result at this point you get the voltage value equal to V_{DD} . And if any of, if both of them are equal to 1 then both transistors lower transistors are ON and both the pMOS transistors are OFF; as a result at this point you will get a 0 volt, ok.

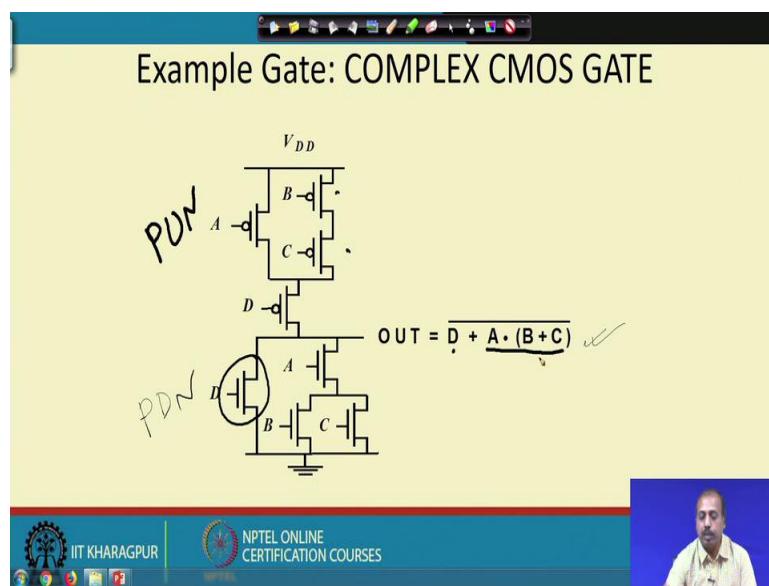
So, this way we can realize this NAND gate and NOR gate in CMOS family.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 18
Logic Gates (Contd.)

Next, we will look into a more complex example of CMOS gate based combination circuit realization.

(Refer Slide Time: 00:27)



So, let us consider the example where it is given by output is $D+A.(B+C)$; say this example. Now what is said is that when you are doing it; so, we should have a N network and a P network. So, N network consists of the N type transistors which constitutes the pull down network and we have got this P type network which is corresponding to the pull up network. So, this is the pull up network; so, this part upper part is the pull up network lower part is the pull down network.

Now how do we realize the circuit? Now as a rule what we do is that from the expression, we find out the some of the parallel terms like say in these expressions if you look into. So, D is one term and this $A.(B+C)$ is another term. So, these 2 terms are in parallel.

So, in the N network we will have this D and this $A.(B+C)$ as 2 parallel branches. Like here, you see the D is one branch and that $A.(B+C)$ is in parallel with that and in $A.(B+C)$;

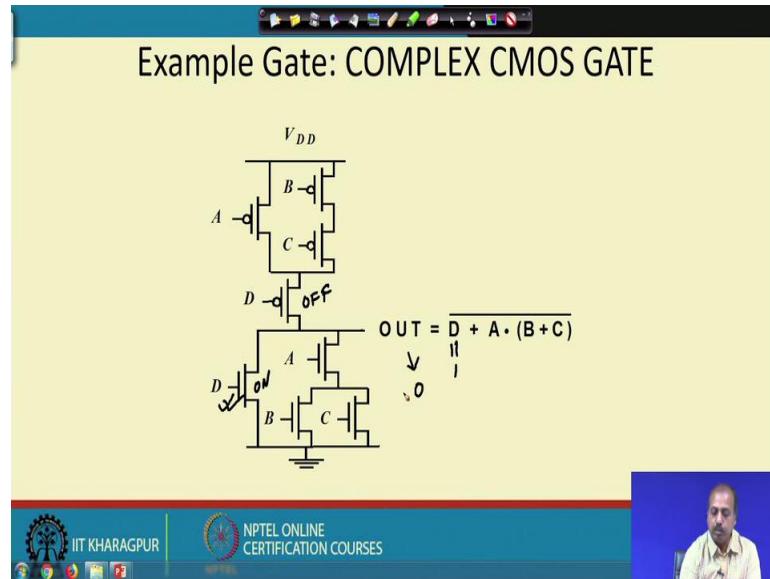
A and this dot is there. So, after A the remaining part B + C will be in series with A. So, that is what is happening; so, A is there and after that this B + C. So, that is in series with A and B since there is a plus between B and C; so, this B and C these 2 transistors will be in parallel.

So, given any logic expression; so, you can directly draw the pull down network. For every plus symbol, so, you can think that the corresponding portion of the circuit will be in parallel to each other and whenever we have got the dot or AND, so those portions will be in series with each others. Now once we have completed the pull down network now we have to draw the pull up network and for the pull up network; so, it is just the complement of the pull down network. So, whatever is in series in pull up pull down network will be in parallel in pull up network and whatever is in parallel in the pull down network will be in series in pull up network.

So, if you look into the example; so, D is in parallel with this part in the pull down network. So, in the pull up network D will be in series with the remaining part. So, that was that is what is happening similarly B and C are in parallel in the pull down network; so, in pull up network they are in series. Similarly with this B and C; A is in series in the pull up network. So, in the pull down network I will have this A in parallel with this B and C portion. So, this way we can very easily draw the CMOS transistor level diagram of any combination and logic circuit combination logic function.

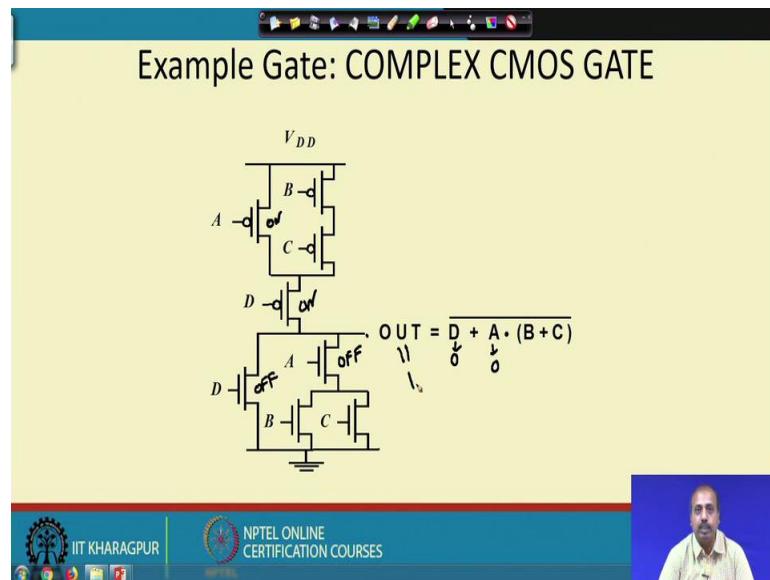
So, what is required of course? The output should be in complemented form. So, output is not in a complemented form; so, you can put another inverter after this to get the function. Now to just understand that just to get the confidence that this circuit really realizes the function that we are looking for example, in this case if D is equal to 1 then the output will be equal to 0.

(Refer Slide Time: 03:36)



So, let us see whether this happens or not. So, if D is equal to 1; so, this transistor is off. So, irrespective of other transistors and since D is equal to 1 this transistor is off. So, this transistor is on and this transistor is off; as a result we do not have a current flow path from V_{DD} to the supply voltage to the output; whereas, from output to ground there is a connection via this transistor. So, the output will become equal to 0.

(Refer Slide Time: 04:16)



So, if D is equal to 1 we will get a 0 here; for getting output equal to 1 what is required is this term should be equal to 0 and this whole term should be equal to 0 and for that purpose

it is sufficient to put a equal to 0. So, if we put D equal to 0 and A equal to 0 output should be equal to 1.

Now, you see look into this circuit if D equal to 0 and A equal to 0; so, this transistor is off and this transistor is also off this transistor is also off. Now naturally there is no current flow path from the output to the ground; so, that part it is connected. Upper side; so this A is getting a 0. So, this transistor is on and D is 0; so, this transistor is also on. So, we have got a current flow path from VDD to the output. So, naturally you will get a high at the output point. So, this will give output equal to 1.

So, in this way using complex CMOS gates; so, we can realize combinational functions very easily.

(Refer Slide Time: 05:17)

Properties of Complementary CMOS Gates

- High noise margin. V_{OH} and V_{OL} are at V_{DD} and GND respectively
- No static power consumption
- Comparable rise and fall times
- Highly compact structure

So, next we will look in to the properties of this complementary CMOS gates why is it so popular? So, it has got high noise margin we will explain this term in the next few slides V_{OH} . So, V_{OH} means what is the output voltage level when we have which we are considering to be high.

So, we have got V_{DD} that is the supply voltage which may be close to say 5 volts and V_{SS} or ground which is say equal to 0 volt. Now you may say that anything above 2.5 is considered as logic high and any anything below set 2.5 is the logic low. So, if we put a boundary like that; so, this is 2.5 volt above this is high and above below this is low. So,

the problem is that there may be noise coming into the signal lines; as a result some low may be misinterpreted as high or high may be misinterpreted as low.

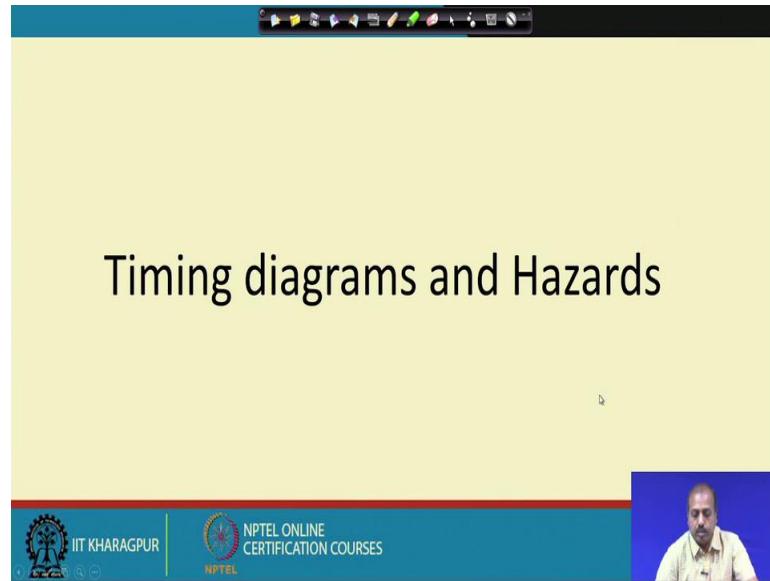
So, normally what we have is this V_{OH} is the voltage level above which we will consider the output to be high. And V_{OL} is the voltage level below which we will consider the output to be low and in case of CMOS. This V_{OH} is at V_{DD} and V_{OL} is at ground. So, they are at they are maximum values, ok. So, V_{OH} can at most go up to V_{DD} and V_{OL} can be a minimum be ground. So, both the things are satisfied; so, this has got a very high noise margin.

So, we will explain noise margin in the next slide; there is no static power consumption there is no static power consumption as we have said that when this CMOS gates are in work, then there is never a current flowing path from V_{DD} to ground. So, at any point of time either the pull up network is on either the pull up network is off or the pull down network is off. So, it is never simultaneously both the networks are on; so, naturally there is never a current flow path from V_{DD} to ground in the static condition. So, there is no static power consumption CMOS consumes very less power.

Comparable rise and fall times so rise time means that if you apply if you are input changes from say 0 to 1 how much time the outputs takes to rise from 0 to 1; so, that is the rise time. Similarly fall time means when the input changes from say input makes a change and the output should go from 1 to 0. So, how much time it takes to go to go from 1 to 0? So, these times are pretty small in case of CMOS gates. So, that is why this is also one of the important feature and highly compact structures.

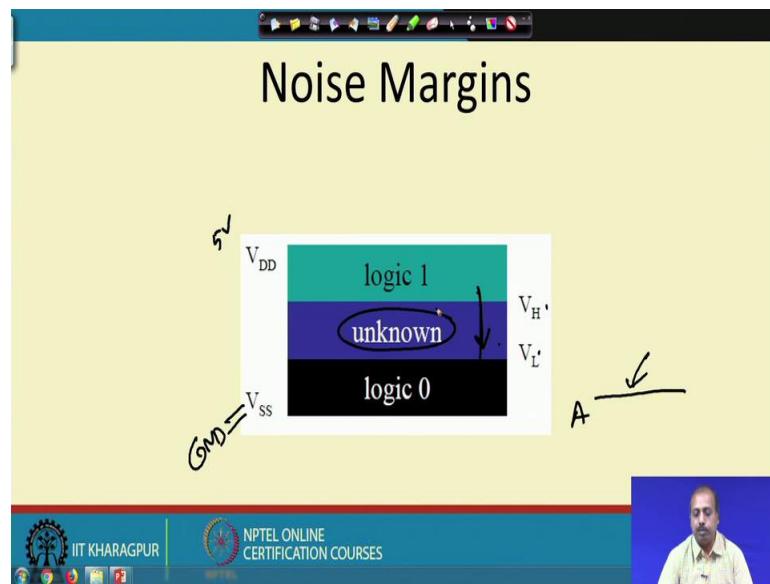
So, you can make in a less area or less Silicon area; so, you can put large number of CMOS gates that makes it area efficient.

(Refer Slide Time: 08:08)



So, next we will be looking into some timing diagrams and hazards. So, these are the two important concept that will be required for understanding the operation of this combinational circuits.

(Refer Slide Time: 08:18)



So, as I was talking about noise margins see suppose this is my total range ok. So, for any gate whatever be the technology say in particular say CMOS. So, V_{ss} ; so, V_{ss} is generally equal to ground and V_{dd} is the maximum supply voltage that we can have DC supply voltage that we have.

So, V_{DD} may be say equal to say 5 volt or depending upon the technology. So, this voltage will be coming down. So, for example, if we are going to say 90 nanometer technology then this will be around 1 volt and so; so that way the volt V_{DD} value will be coming down and this is continually coming down. So, as technology is progressing; so, V_{DD} value is continually coming down. Now my total range of voltage level that a gate output can have is from V_{SS} to V_{DD} .

So, out of that to keep some margin in the operation of it; so, what we do? So, we mark this 2 voltages V_L and V_H ok. So, anything below V_L is taken as logic 0 and anything above V_H is taken as logic high. If a single state is between V_L and V_H ; so, that is taken as an unknown level. The idea is very simple it is because of the fact that due to noise. So, if suppose a signal line A is at logic 0, but due to some noise coming into it; so it increases slightly.

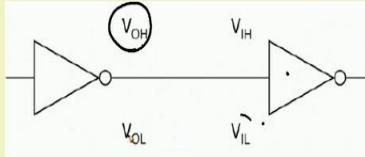
Now if it goes beyond V_L ; then it will be difficult to understand like what it is. Similarly if something is V_H ; so, if due to noise margin it decreases further. So, if it crosses this barrier and comes to this V_L region then it will be misinterpreted as a low ok.

So, this barrier or is this margin is kept so that noise cannot take a take a V_H value down to V_L or cannot take a V_L value up to V_H ok. So, that way it is it will help us in understanding the logic level sorry.

(Refer Slide Time: 10:37)

Logic Level Matching

- Levels at output of one gate must be sufficient to drive another gate



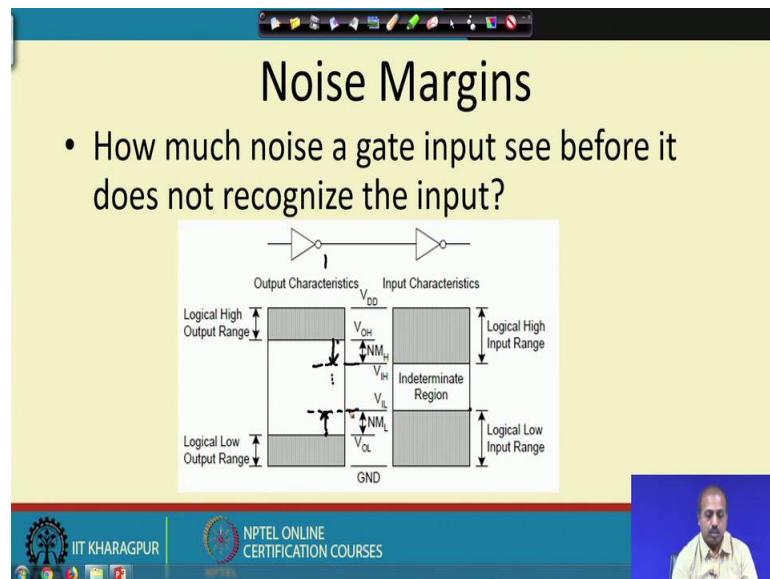


IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES

So, consider this situation like we have we have got 2 gates ok. So, both are inverters in this case, but it need not be inverter. So, output of one gate is feeding input of another gate. Now, when this gate output say high ok; this gate output say high and their voltage level is V_{OH} this V_{OH} should be sufficient to for the second gate to take it as high. So, if the second gate takes anything above V_{IH} as high; so V_{OH} must be above that.

Similarly if this second gate takes anything below V_{IL} as low this V_{OL} value of the first gate should be lesser then that other though otherwise there will be a misinterpretation. So, this is explained in the diagram here.

(Refer Slide Time: 11:32)



So, say for this second gate; so this is the situation. So, it takes from ground to it takes from ground to this voltage level as the logic low input range. So, that is the V_{IL} ; so, this much is the V_{IL} for the; so, any signal coming between whose value is between 0 and V_{IL} is taken as low by the second gate. And anything between this V_{DD} and this V_{IH} ; so, this voltage level, so anything between these 2 voltage level that signal value is taken as logic high by the second gate.

Now, what is the condition on the first gate? So, first gate if it wants to a make an output low ok. So, it should produce a value less than V_{IL} ; similarly if it wants to produce something high it should produce voltage level which is at least equal to which is which is at least equal which is more than this V_{IH} value; so to be on the safe site. So, what we do for the output high and low levels of the first gates. So, they are kept appropriately; so, this

output low level is kept to be lesser than this input low level of the next gate and this output high level of the first gate is kept as the as the more than the input high level of the second gate.

So, if there is a noise suppose this first gate has a output at a 1 ok. So, first gate has a output at a 1 and the voltage value is some voltage value is equal to V_{OH} . Now if there is a noise and that noise takes this value lower than say this. So, it takes the value lower than this; so, in this region in that case in that case it will be the second gate will not be able to understand the that it has a logic high.

Similarly, if the first gate has output at a low; so, it has output at this and due to some noise if it goes beyond this level if it goes beyond this level, but then the second gate will not be able to understand it has logic low fine.

(Refer Slide Time: 13:50)

The slide has a yellow background with a black header bar containing icons. The title 'Noise Margin' is centered in a large, bold, black font. Below the title is a bulleted list in black font:

- Noise Margin = Voltage difference between the output of one gate and input of next.
Noise must exceed noise margin to make the second gate produce wrong output.

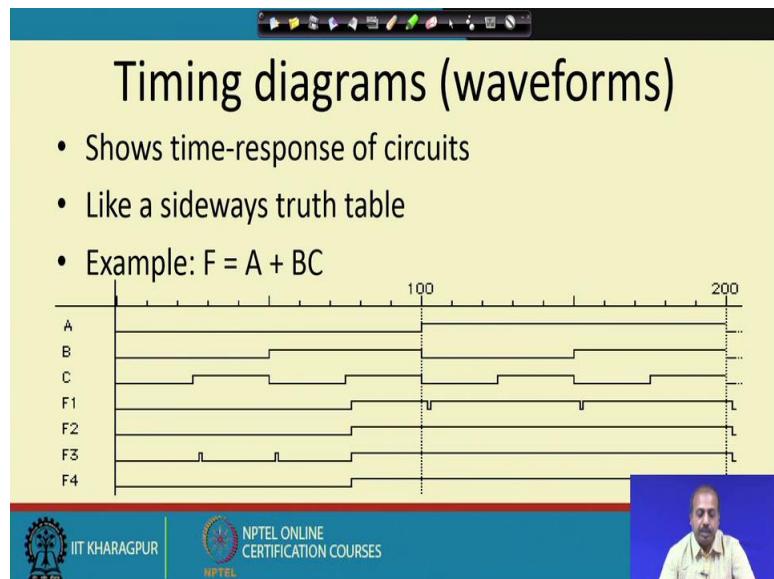
At the bottom, there is a blue footer bar with three sections: 1) IIT Kharagpur logo and text 'IIT KHARAGPUR'. 2) NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. 3) A video window showing a person's face.

So, we have got this noise margin which of voltage difference between output of one gate and input of next and so, in this case; so, noise margin this noise margin high is equal to $V_{OH} - V_{IH}$. Similarly noise margin low is $V_{IL} - V_{OL}$. So, this way we can define this noise margins low and high and this if this noise margin is high; that means, will be more protected against the noise noisy in the noisy environment.

On the other hand if some logic family has got this noise margin low then there is a high chance of corruption in the operation of the circuit realized by in that logic family. So,

noise must exceed the noise margin to make the second gate produce wrong output. So, that is the condition.

(Refer Slide Time: 14:46)



Next we look into timing diagrams; so, timing diagram this refers to the time response of the circuit. So, if you take if you take the circuit over a period of time and this input symbol, the input signals like say in this case if the function $F = A + BC$; suppose this A B C they are changing their values in some form ok. So, in this diagram; so we have drawn the time instance and these signals A B C they are changing they are values at different times; then what happens to the output ok.

So, here this F_1, F_2, F_3, F_4 ; so, these are corresponding to 4 different realizations of the function that we will see shortly. So, ideally whenever this A is equal to 1; the output should be 1 ok. So, this, but you see that it if this one of the realization. So, output is temporarily showing some 0 ok; so, these are known as the hazards ok.

So, there is there is a glitch in the output. So, output is otherwise 1, but at this point it is coming down similarly as this point it is coming down. So, it can happen how can it happen; so that we will see in successive slides; why does it happen? Is that the real gates they have delays.

(Refer Slide Time: 16:00)

Timing diagrams

- Real gates have real delays
- Example: A' . $A = 0$?

time

width of 3 gate delays

- Delays cause transient $F=1$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if I have got an AND gate then the AND gate had certain delay. So, as soon as this inputs its input has a inputs have changed. So, it cannot be the output changes immediately; so, it takes some time to change. And because of that the circuit may show some intermediate behavior which is not expected. So, say in this case what we have done? We have taken an input A and it is inverted by 3 stages by 3 inverters where producing output B C and D and finally, they are ANDed.

So, this A; so here B equal to \bar{A} , C equal to A, D equal to \bar{A} and A. So, we are supposed to get F equal to 0, but due to this delay of this inverters. So, it may so happen that you get some other behavior how? See this A; suppose A is changing like this. So, A was initially low and now it is; now it is changing to high at this point it is being high this much and then it is again going low.

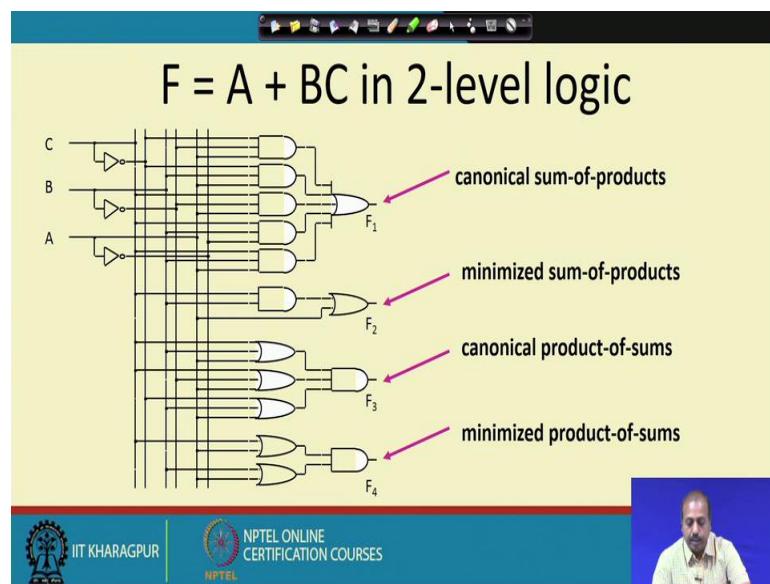
So, B. So, if this inverter has got some delay; so this status change of A will be reflected at output B after sometimes. So, this is the delay of one inverter if you assume like that. So, this much is the delay of one inverter; so after that time your B will be going low. Similarly at this point A is changing so after that inverter delay B will be high. Then the C output. So, this we will again this is a delayed version of delayed inverted version of B. So, this again shows the similar thing; so, as if B is delayed and inverted ok. So, we have got this symbol; this timing diagram.

D is again another delayed version like this; now, we are F is equal to D and A. So, D and A; so, in this part first part of the graph. So, A and D A is low and D is high. So, we get a 0 at this point after some time; so, after some time so, A has change the value to 1 and D is 1. So, it is expected that my output should become equal to 1.

So, it will happen after the delay of 1 AND gate; so, again this is taken as the gate delay. So, when A has become 1; so, after sometime D becomes F becomes equal to 1. So, the this is reflected at this point after the delay and then it remains high for this much of time and then D became low at this point; so after a gate delay after a gate delay; so where this F has come down.

So, we were expecting to get a constant 0 at the output F, but what you see is a in between F has become 1. So, this is called the hazard that we will see in more detail in successive slides. So, here this width of this pulse will be equal to 3 gate delays and delays they will cause a transient F equal to 1.

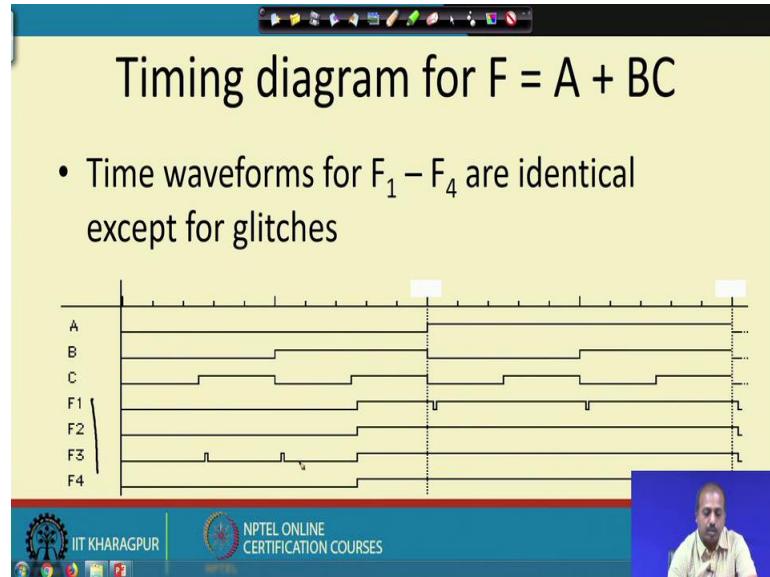
(Refer Slide Time: 19:11)



Now, if we consider say $F = A + B C$ that function and we have got 4 different realizations of the function in the in F 1 is a canonical sum of products realization, F 2 is the minimum sum of products. So, here this A is coming; so, from this A line. So, A is connected to this OR gate and B C they are ANDed and that is fed to this OR gate.

So, F_2 is the minimized sum of products, F_3 is the canonical product of some form and F_4 is the minimized product of. So, these are various realizations of the function $F = A + B C$.

(Refer Slide Time: 19:47)



Now, ideally we expect that for all the realization F_1 to F_4 they should have identical waveform, because they are ultimately realizing the same function. So, we expect that they will give as the same waveform, but in reality it does not happen. So there are some glitches; so, if we look into this F_1, F_2, F_3, F_4 ; so, there are some there are some glitches in this F_1, F_2, F_3, F_4 ; there are some glitches otherwise they are same ok, otherwise all the waveforms are same.

(Refer Slide Time: 20:24)

Hazards and glitches

- **glitch:** unwanted output
- A circuit with the potential for a glitch has a **hazard**.
- Glitches occur when different pathways have different delays
 - Causes circuit noise
 - Dangerous if logic makes a decision while output is unstable

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player shows a man speaking.

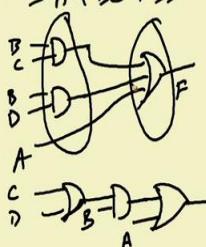
So, we will go to the topic called hazards and glitches. So, glitch is unwanted output; so, output should be say steady one in between it becomes a 0 for a very small amount of time and again it goes to 1. So, a circuit that has got the potential to have potential for a glitch said to have a hazard; so, it is ideal it is expected that my circuit should be hazard free. Glitches will occur when different paths have different delays to the output if different paths have got different delays from the input; then the glitches may occur. And it causes circuit noise and dangerous if logic makes a decision while output is unstable. So, your say combinational circuit has got a hazard.

So, the output is not stable in between and at that time if we take a decision based on the output of the circuit so that is the incorrect. So, that creates some that creates the problem. So, that has to be; so, this glitches and are to be avoided as far as practical way. So, how do we do it?

(Refer Slide Time: 21:27)

Hazards and glitches

- Solutions
 - Design hazard-free circuits
 - Difficult when logic is multilevel
 - Wait until signals are stable

$$\begin{aligned}f &= A + B(C+D) \\&= A + BC + BD\end{aligned}$$


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, we design hazard free circuits ok; so, this is one possibility somehow we do modify the circuit. So, that it becomes hazard free; however, it is difficult when we are doing a multi level realization. So, any combinational circuit they can be realized in 2 form one is called 2 level another is called multi level. In 2 level realization we have got only 2 level one AND level and one OR level or and one OR level and one AND level like that. So, I will whereas, for multi level circuits like say if I have got a function $F=A+B.(C+D)$ ok. So, this is nothing, but the circuit a function $A+BC+BD$.

Now, if I realize it like this then what I do? I have got one AND gate which is B C another AND gate which is B D and then we have got an OR gate where we have got this line this line and A connected; so, this is F. So, this realization has got 2 level the first level consist of the AND gate gates second level consist of the OR gate. Now if you think about say this realization then it can also be realized like this. So, first we do an OR of C and D. So, this is C or D; so, this is ANDed with B and then it is finally, ORed with A.

So, this circuit has got 3 levels ok; so, if you go by level by level. So, this circuit has got 3 levels perhaps the first circuit has got 2 level. So, these are multi level circuit; so, whatever we will discuss regard with respect to this hazard and glitches so, they are valid for 2 level circuits.

(Refer Slide Time: 23:34)

The slide is titled "Types of hazards". It lists three types of hazards:

- Static 1-hazard
 - Output should stay logic 1
 - Gate delays cause brief glitch to logic 0
- Static 0-hazard
 - Output should stay logic 0
 - Gate delays cause brief glitch to logic 1
- Dynamic hazards
 - Output should toggle cleanly
 - Gate delays cause multiple transitions

Waveform diagrams are shown for each type:

- Static 1-hazard: A waveform starting at 1, dropping to 0, and then returning to 1.
- Static 0-hazard: A waveform starting at 0, rising to 1, and then returning to 0.
- Dynamic hazards: Two waveforms. The first starts at 0, rises to 1, falls to 0, and then rises to 1 again. The second starts at 1, falls to 0, rises to 1, and then falls back to 0.

The footer of the slide includes the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES".

For multi level circuits, it is very difficult to have this hazard problem solved.

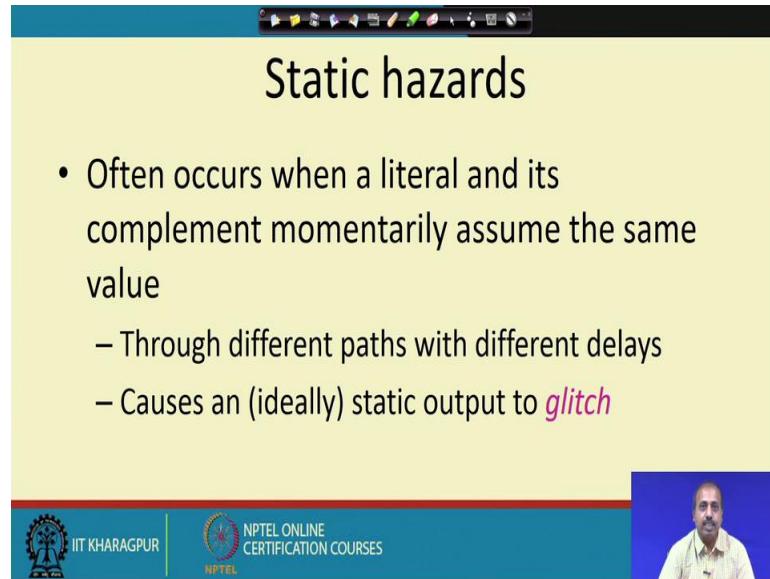
So, another possibility is that we wait till the signals have becomes stable. So, there are 2 types of hazard one is known as static 1 hazard. Static 1 hazard means that output ideally should remain 1, but in between there is a glitch and it goes to logic 0; for some small amount of time; so this is called static one hazard. Similarly, we can have a static 0 hazards; so, static 0 hazard means the output will remain output should ideally remain 0, but in between it goes to logic high and then again comes backs to 0. So, this is the static 0 hazard and there can be a dynamic hazard.

So, dynamic hazard means the output should toggle like it should go from 0 to 1, but what happens is that instead of going from 0 to 1 directly; it makes this types of toggles, it first goes to 0 to 1 then again comes down to 0 and then again goes to 1. Or it is one the circuit should go from 1 to 0, but in between it go make some toggle, so 1 to 0 then 0 to 1 then again 1 to 0. Finally, it settles to 0, but it in between it shows some transition. So, gate delays multiple transitions are created; so, this type of hazard, so they are known as dynamic hazards.

(Refer Slide Time: 24:44)

Static hazards

- Often occurs when a literal and its complement momentarily assume the same value
 - Through different paths with different delays
 - Causes an (ideally) static output to *glitch*

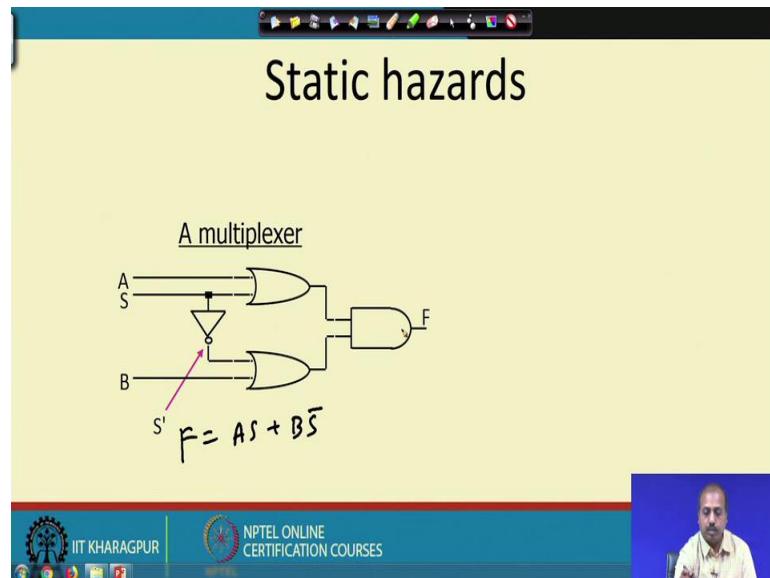


So, static hazards; so it often occurs when a literal and its complement momentarily assume the same value. This can happen due to the gate delays that circuit a one literal and its complement both of them are having the same value and through different paths with different delays and it cause the static output to glitch ok.

(Refer Slide Time: 25:00)

Static hazards

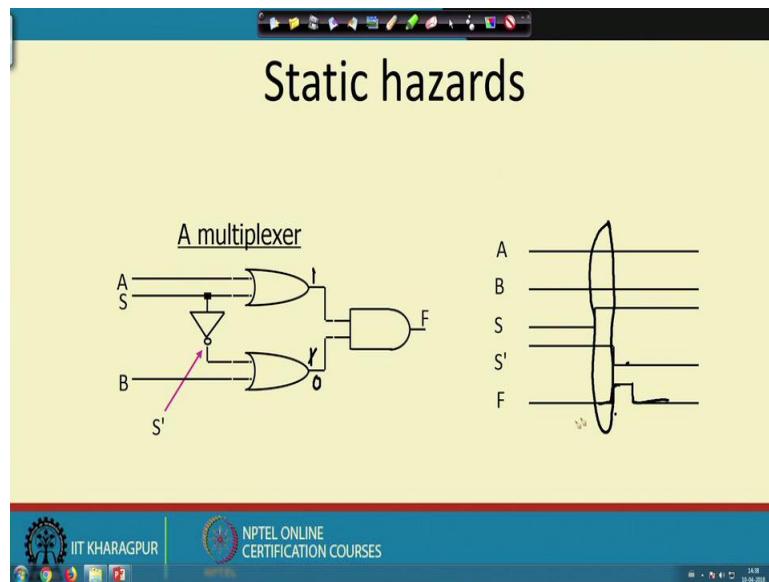
A multiplexer


$$F = AS + BS'$$

So, we will take an example suppose this is a circuit which is realizing some multiplexer.

So, here this function F is the function F is here the function $F = AS + B\bar{S}$. So, if S equal to 0 then this A will go to F and if S equal to 1, then this B will be then this B will be passing to F ok.

(Refer Slide Time: 25:50)



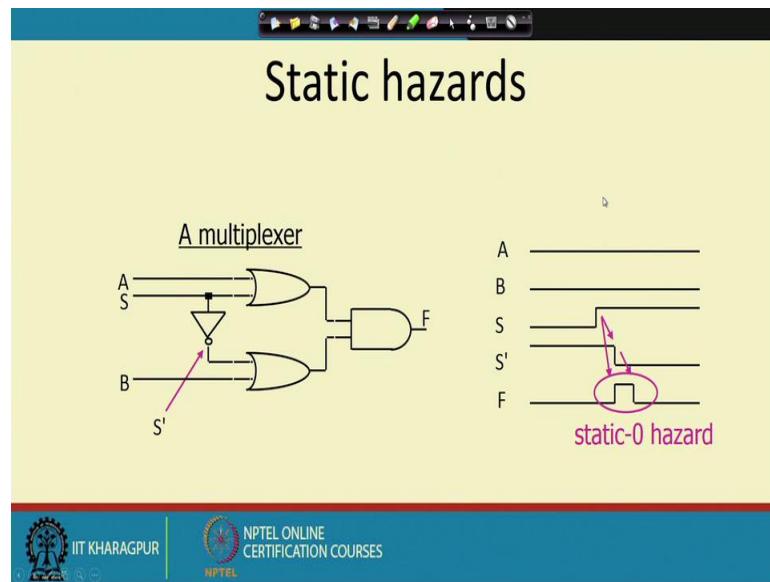
So, so, if the if S, so the realization will be like this, that; so, this is the situation. But because of this because of the presence of inverter suppose A is A and B both are continually at ground and S changes its state from low to high.

Since A and B both are both are low then even if S changes its value output F should not change its status ok. Because the OR gates they should always produce 0 because A and B they are they are both at 0 and this S is selecting one of them. So, one of the OR gate output will be 0 always that that is the that is expected, but what happens you see due to the delay of this NOT gate. So, this S this \bar{S} does not get the value immediately ok. So, it takes some delay and after that S; S does become one.

Now, for say this much for this range of time what is happening is that for the upper OR gate, we have got A is equal to 0 and S is equal to 1. So, this OR gate output is 1 in this region. So, in this region in this region A is 0 and S is 1. So, as a result it output say 1 and the lower OR gate; so it has got B equal to 0 and sorry not in this region sorry. So, if you if you take say this region, this region of operation. So, here A is 0 and S equal to 1; so, as a result output of this OR gate is 1 and for the lower OR gate B is 0 and S' is also 1; so, here also you get a 1.

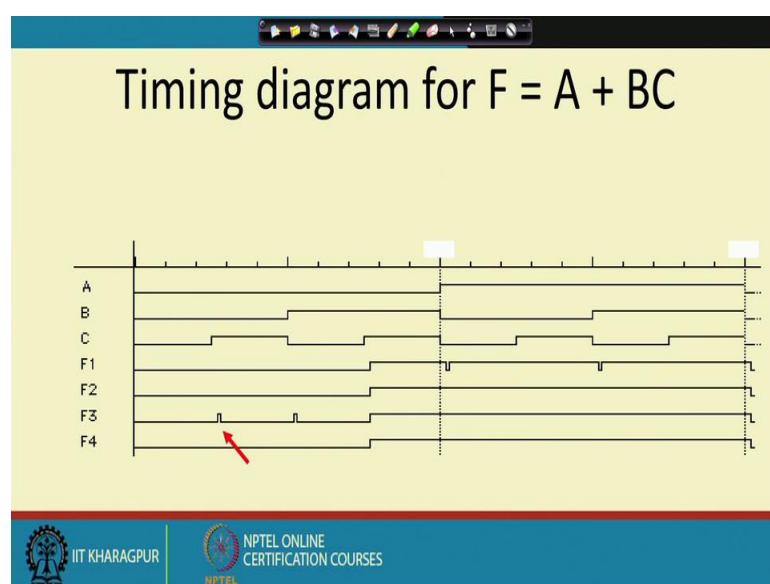
So, this AND gate inputs are 1; so, after the gate delay of this AND gate. So, you will get this signal will be going to high; after sometime this S will be becoming equal to 0. So, this lower OR gate will become equal to 0 and then after that AND gate delay. So, this will be coming down to 0; so, you see that ideally you should not have this glitch, but here in due to the presence of this glitch is appearing; so, this is a static hazard.

(Refer Slide Time: 28:13)



So, this is the static hazard that we have.

(Refer Slide Time: 28:17)



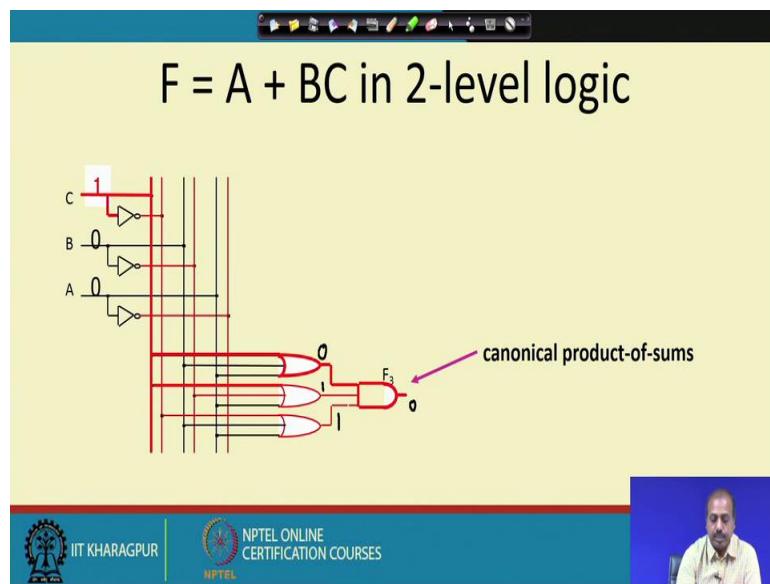
So, on the A + B C function; so, there can be multiple hazard that can occur. So, we will see them for few cases.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 19
Logic Gates (Contd.)

So consider the function $F = A + BC$ when it is realized in canonical product of sum form; and this is the realization, ok.

(Refer Slide Time: 00:19)

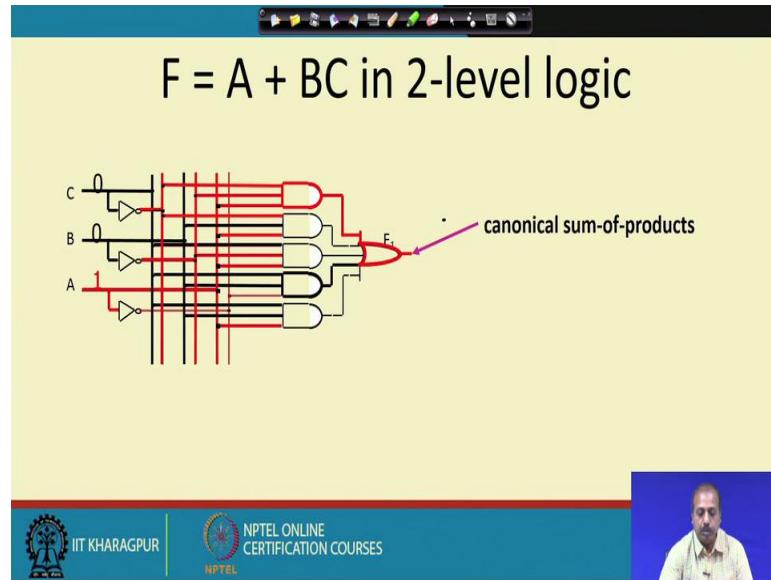


Now suppose this ABC all of them are at 0. So, what has happened is that all of them being at 0. So, here this OR gate, all the 3 inputs are all the 3 inputs are 0 so, this output is 0. And this OR gate, this has got some of the inputs as 1, like when B is 0 so, this input is 1. So, this is 1 and similarly, this is also 1, because this input is coming here as a 1 so, this AND gate output is equal to 0.

Now, now consider the case when this C value it has changed from 0 to 1. When it has changed from 0 to 1 so, we have, when this C value has changed from 0 to 1 so, this output is 1. I have this C, C input has become equal to 1, as a result this OR gate will now be equal to 1. So, this output becomes equal to 1. And after sometime, after sometime, after sometime this inverter output will be activated. So, inverter has got some delay. So, before the inverter delay was activated so, this OR gate was remaining at 1 so, you are getting a, some output.

But now after sometime this inverter will be equal to 0, as a result this OR gate value will be become equal to 0 so, this F3 will be will be equal to 0, ok. So, this way due to the presence of these delays of the inverters, ok, so, this circuit will produce a glitch at the F3 output, ok.

(Refer Slide Time: 02:20)



So, this, similarly this F1. So, F1 suppose this ABC the A is 0 B and C are 1. So, this is the situation the F1 output is equal to 1.

And now if there is a change that B and C become equal to 0 and A changes to 1. Then what will happen is that due to the delay, so, the effect will not come immediately. So, F 1 will be remaining as equal to 1, and now it will become equal to 0. And then after sometime B suppose this inverter of B, it changes it's state. So, it is get you 0 so, it will be getting as the it will affect the circuit after that. So, this way so, if you look into the gate delays into if you take this gate delays into consideration, you may find some behavior of this circuits in some intermediary form, so, that is not a that is not stable actually so, in between it shows some glitch, so, that can happen.

So, it is just you can just trace through this circuit to see that this is changing the value, ok. So, whenever it is red means it is going to logic 1, and wherever I am showing it by black line means it is at logic 0, ok. So, you follow that convention and trace through this diagrams, then you will find the implementation, this creation of the glitches.

(Refer Slide Time: 03:37)

Dynamic hazards

- Often occurs when a literal assumes multiple values
 - Through different paths with different delays
 - Causes an output to toggle multiple times

Dynamic hazards

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A video player window showing a man speaking is visible on the right side.

On the other hand this dynamic hazard so, they occur when a literal assumes multiple values, ok. So, through a different through different paths, different delays, they have got different values, and causes an output to toggle.

So, as we have seen the shown the output should go from 0 to 1, but in between it makes some toggle, or it goes from 1 to 0 in between it makes some toggle.

(Refer Slide Time: 03:59)

Dynamic hazards

The slide shows a logic circuit with inputs A, B₁, and C, and output F. The circuit consists of several logic gates. Below the circuit, waveforms for inputs A, C, B₁, B₂, B₃, and output F are plotted. The waveform for output F shows a transition where it toggles between 0 and 1 during the time interval where B₂ and B₃ are both high, which is highlighted with a pink oval and labeled 'Dynamic hazard'.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A video player window showing a man speaking is visible on the right side.

So, consider this circuit that has got this A, B, C as input, and this suppose this B value. So, the so, suppose situation is like this, that A is at logic low C is at logic high. And this B, B point at B 1, at B 1 it changes state from a 0 to 1; so, that B 1 it changes the state.

Then what will happen? So B 2 so, at this point so, there are 2 inverters from B 1 to B 2 so, after 2 inverter delays so, B 2 will show this waveform, at this point. Then at B 3 so, B 3 has got so, B so, A is at logic low so, for this \neg NOR gate so, it is nothing but an inverter so, this B 3 is at so, B 1 sees the delay of this NOR gate plus this inverter to produce B 3. So, that is a, if I assume that this NOR gate has got delay equal to 2 inverters. So, after 3 delays so, you or after so, they are they are having some same delay. So, NOR gate and this inverter they are having same delay.

So, after 2 gate delays so, B 3 gets the value of B 3 gets the value of B 1, so, B so, B 1 has changed from 0 to 1. And B 3 has changed from 0 to 1 after 2 gate delays. Now when it comes to this F and this C is always 1 ok. So, C is always one means this, this NAND gate it is it is always giving B 1 bar, and then this is giving me the whatever is coming here depending upon this B 2 value so, this output of the NOR gate is coming.

Now, if you trace this circuit, again you can find that the situation, will be like this that initially the F will be high, it will show be low for one gate delay, then it will be high for 2 gate delays, and then again it will be low for the remaining part of the system. So, that way so, there is a glitch so, this is the dynamic hazard here. So, it shows a dynamic hazard at this point, so, this there is a dynamic hazard at this point. So, we can just trace through this circuit. And see that this type of timing diagram this waveforms are appearing, and that creates the dynamic hazard.

So, this has to be avoided, now how do we do that? So, this is the dynamic hazard.

(Refer Slide Time: 06:25)

• Key idea: Glitches happen when a changing input spans separate K-map encirclements
– Example: 1101 to 0101 change can cause a static-1 glitch

	AB\CD	00	01	11	10
00	0	0	1	1	
01	1	0	0	1	
11	1	1	0	0	
10	0	0	0	0	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how can we; actually this dynamic hazards are difficult to remove, ok. So, so we will be talking about static hazard first. So, the key idea for eliminating a static hazard is that, the glitches happen when a changing input spans separate K map encirclements. So, if some so, if you are going from say this box to this box, ok. So, we are changing some inputs so, that it goes from this circle to this.

So, in between it may go through some 0 state, as a result a circuit output may become 0 so, that is the key point, ok. For example, so, when you are going from 1 1 0 1 to 0 1 0 1. So, 1 1 0 1 is this one 1 1 0 1 is this one to 0 1 0 1 so, 0 1 0 1 is this one, ok. So, whenever you are going from here to here. So, it may so happen that this delay comes so, A is delayed a bit and then this a 1 to 0 so, that delay takes some time, and in between it goes to this 0 state or this 0 state and that way it creates a it creates a glitch, ok.

(Refer Slide Time: 07:48)

Eliminating static hazards

- ABCD: 1101 → 0101

$F = AC' + A'D$

Inputs: A, B, C, D

Output: F

Karnaugh Map:

AB		00		01		11		10	
CD		00	0	0	1	1			
		01	1	1	1	1			
		11	1	1	0	0			
		10	0	0	0	0			
C	B	00	01	11	10				

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, how to solve this? Sorry, so, how to solve this is something like this, that so, this is the situation like, say we have got the function so, this square it represents $A\bar{C}$, and this other quad, it represents $\bar{C}\bar{D}$, so, it is $A\bar{C} + \bar{A}\bar{D}$. Now if I have got the situation like ABCD is 1 1 0 1 so, you get this 1 1 0 1. So, A and \bar{C} so, A and C bar are 1 1 and A bar and D. So, that is 0 1 so, it is applied here.

So, you get 1 and 0 here accordingly the output is 1. Now it is changing to 0 1 0 1, but this A changes, and it takes some time for \bar{A} to change, as a result you may find that it will produce a 0 in between.

(Refer Slide Time: 08:35)

Eliminating static hazards

- Solution: Add redundant K-map encirclements
 - Ensure that all single-bit changes are covered by same block
 - First eliminate static-1 hazards: Use SOP form
 - If need to eliminate static-0 hazards, use POS form
- Technique only works for 2-level logic

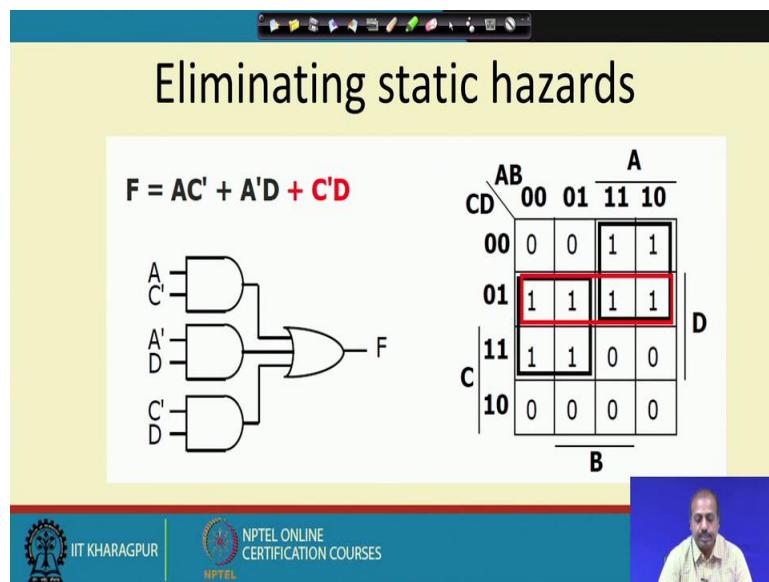
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, that 0 has to be avoided, how to solve this is that, we add some redundant K map encirclements ok. So, we add some extra encirclements. So, like this so, that this single bit changes can be covered in the same block.

So, it will eliminate static 1 hazard, using sum of product form, and this if there is static 0 hazard is also there, then we have to use the POS form product of sum form also. So, use this so, first of all so, the process becomes complex, first of all you do encirclement and make the SOP form, and then if you find that this static 0 hazard is also there. Then you have to use this product of sum form and again do the encirclement in that fashion.

But unfortunately this technique works only for 2 level logic. So, multilevel logic there is no solution to this static hazard problem. So, this is the so, that if it occurs so, we have to be careful there.

(Refer Slide Time: 09:35)



So, this is an example so, how do we do this. So, the same circuit that $A\bar{C} + \bar{A}D$, ok. So, what we have done, we have done some redundant encirclement ok. So, this C so, this was this red one, this is this red quad was not necessary, ok, because we have the all the ones are already covered by the remaining 2 quads.

But we do not, but we take this redundant one, what will happen is that even if this A is changing the value. Ok this $\bar{C}D$ so, this will hold the output ok; so, this is the problem occurred because for some amount of time this A and \bar{A} both of them were equal to 0. As

a result this OR gate input though the both the inputs were 0. But this $\bar{C}D$ term has been added. So, so $\bar{C}D$ does not make a change so, $\bar{C}D$ holds the value at 1. So, it will be we will be getting a one if you are changing the input combination like that, ok. So, this way we can eliminate the static hazard by doing some redundancy.

So, we are going a bit counter intuitive like for Boolean function minimization. So, we do not allow this type of extra, extra quad or square formations. But in case of for eliminating this static hazard so, we are doing the other way. So, we are allowing this extra terms so that this hazards are not there.

(Refer Slide Time: 11:01)

Summary of hazards

- We can eliminate static hazards in 2-level logic for **single-bit changes**
 - Eliminating static hazards also eliminates dynamic hazards
- Hazards are a difficult problem
 - Multiple-bit changes in 2-level logic are hard
 - Static hazards in multilevel logic are harder
 - Dynamic hazards in multilevel logic are harder yet

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, we can eliminate static hazards in 2 level logic for single bit changes, and eliminating static hazards also eliminates dynamic hazards. So, that is one good thing that if we eliminate static hazards the dynamic hazards will also not be there; so, that way it is useful.

Hazards are a difficult problem so; multiple bit changes in 2 level logic are hard. So, we are whatever example we are considering. So, we are considering only a single bit change, ok. So, if the multiple bits are changing so, it is difficult to solve the, that type of a hazards, and static hazard in multi-level logic is difficult, and dynamic hazards in multilevel logic are we can it can so, it is yet more harder, ok. So, that way it is difficult to solve this for multilevel circuit this dynamic hazard problem.

(Refer Slide Time: 11:52)

Example: Calendar System

- Determine number of days in a month (to control watch display)
 - Used in controlling the display of a wrist-watch LCD screen
 - Inputs: month, leap year flag
 - Outputs: number of days

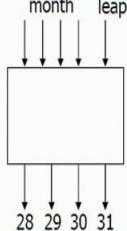
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Next we will be looking into a few examples of this combinational logic design. So, suppose we have the first example we are determining number of days in a month to control some watch display. And it is used in controlling the display of a wrist watch LCD screen, inputs are month leap year a month and leap year flag, and outputs are the number of days. So, you give the month and the leap year flag it will output the number of days in it. So, you I have a combinational we I have to design a combinational circuit, a month will be one input and whether the year is a leap year or not. So, that will be coming as input and it will be outputting the number of days in the month.

(Refer Slide Time: 12:31)

Formalize the Problem

- Encoding:
 - Binary number for month: 4 bits
 - 4 wires for 28, 29, 30, and 31 one-hot – only one true at any time
- Block diagram



month	leap	28	29	30	31
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	0	1
1011	-	0	0	1	0
1100	-	0	0	0	1
1101	-	-	-	-	-
111-	-	-	-	-	-

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this is a typical example, see we have got this month can go from January to December, ok. So, we this month is a 4 bit input because there are 12 months. So, I can have I need 4 bits for encoding this 12 months ok. So, 4 bit month bits are coming. And leap year is a flag so, this leap year if it is it may be 0 it may be 1, ok.

So, this is the final block that we are designing ok. So, here you see that we have got as input this 4 bit month and leap as input. And there are 4 outputs 28, 29, 30 and 31. So, if it is 28 day in that month, then this output will be 1, rest will be 0 similarly the number of days in the month is 29, then this output will be 1, rest will be 0, that way similarly the 30 and 31.

So, if we look into the truth table so, it is like this. So, here the month is 4 bit so, if it is so, our month will start with January. So, we are assuming that we will be entering 1. So, 0 0 0 0 is never given as month so, for that I do not care the output. So, for 0 0 0 0 whatever be the leap year value so, output need not be considered, ok. So, that is do not care. When 0 0 0 1 so, if it is the again the leap year need not be considered leap year is do not care. So, it is 0 0 0 1 so, it is a January so, 31 day month.

Then this 0 0 1 0 is the February, now I have got the leap year, this is without leap year and this is with leap year. So, in with leap year 29 will be a high, and without leap year 28 will be high so, it is goes like this. So, this way I can draw this table, now this one sorry 8 so, this is my month 12 December. So, this is accordingly it is 31 month then. So, this is 1 1 0 1 and 1 1 1 0, 1 1 1 1 so, they are all do not care, because we do not have 13 14 15 months, ok; so, they are all do not cares, this way I can make the truth table.

(Refer Slide Time: 14:45)

Implementation

$$28 = m8' m4' m2 m1' \text{leap}'$$
$$29 = m8' m4' m2 m1' \text{leap}$$
$$30 = m8' m4 m1' + m8 m1$$
$$31 = m8' m1 + m8 m1'$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Once you have made the truth table so, we can write down the corresponding combinational function, ok. So, 28 is $m8' m4' m2 m1' \text{leap}'$. So, 28 you see that, a 28 is one only for this combination. So, a this all this bits are the $m8'$ so, this is $m8'$, $m4'$, $m2$, $m1'$, and leap bar similarly 29 is for this 30 is; so, this is the these are the corresponding logic expression. So, what you can do, for getting this you can draw the corresponding truth table is or from this truth table you can make the corresponding Karnaugh maps and do minimization and get the logic function.

And then in terms of logic gates so, you can realize this combinational function.

(Refer Slide Time: 15:30)

BCD to 7-Segment Display Controller

- Input: 4-bit BCD digit (A, B, C, D)
- Output: 7 control signals for the display (C0 to C6)

The diagram illustrates the internal logic of a BCD to 7-segment display controller. On the left, a 4-bit BCD input (A, B, C, D) is fed into a 'BCD to 7-segment control signal decoder'. This decoder generates seven control signals (C0 to C6), which are then connected to a 7-segment display. The display is shown in two rows, each containing four digits. The top row displays '00239' in red, and the bottom row displays '58888' in red. The segments are labeled C0 through C6, corresponding to the segments of a standard 7-segment display.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



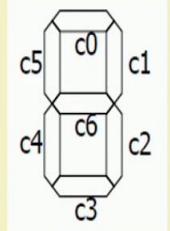
Next we look into another example which is BCD to 7 segment display controllers. A 7 segment display is quite common so, you can find many such displays on the roads and all. So, here we have got some; so, actually this 7 segments are there. So, each segment is a light emitting diode so, you can control it to be a high or low. So, if you can turn it on or off so, for that I have to give a one to that segments so, that the segment will be turned on. And for giving it off so, 0 should be given.

Now, normally we want to display the digits 0 to 9. So, we have got a 4 bit binary coded decimal digit, say and they are the this value is coming here this ABCD is the 4 bit binary BCD value that we are having. Now based on this inputs so, I have to select the appropriate segments. So, these segments are named as C0, C1, C2, C3, C4, C5, C6 so, 7 segments. So, for example, if you give 0, then except in C6 all other segment should be turned on similarly if you are giving 8 as input then all the segments should be turned on.

(Refer Slide Time: 16:40).

Truth Table

A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	-	-	-	-	-	-	-	-
1	1	-	-	-	-	-	-	-	-	-



IIT Kharagpur
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

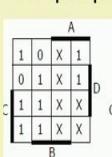
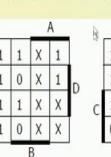
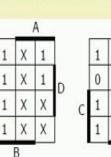
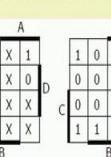
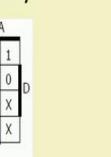
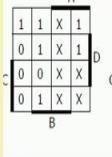
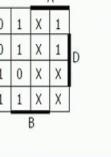
So, I can draw the corresponding truth table like this. So, for ABCD equal to 0 0 0 0. So, C0 to C5 all are 1 and C6 equal to 0. And similarly whenever you have the outputting say for 8, 1 1 0 0 0. So, we have got for 1 0 0 0 we have got all inputs or all the segments to be turned on so, we have got all one.

So, for 9 we have got this, for 10 11 12 13 14 15 so, they are do not care so, do not have to do anything.

(Refer Slide Time: 17:11)

Implementation

- 15 unique product terms when minimized individually

				
		$C0 = A + BD + C + B'D'$ $C1 = C'D' + CD + B'$ $C2 = B + C + D$ $C3 = B'D' + CD' + BC'D + B'C$ $C4 = B'D' + CD'$ $C5 = A + C'D + BD' + BC$ $C6 = A + C'D + BC' + B'C$		

IIT Kharagpur
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

So, for getting the corresponding implementation, you have to draw the corresponding Karnaugh maps so, these are the Karnaugh maps for the different conditions, ok. And then so, these are the corresponding Boolean expression; so, C0, C1, C2, C3, C4, C5, C6 so, these are the Boolean expressions.

And once we have this Boolean expressions so, you can realize them using the logic gates, ok, so, that can be done.

(Refer Slide Time: 17:38)

Better Implementation

- 9 unique product terms
- Share terms among outputs
- Each output not necessarily in minimized form

$C_2 = A + B D + C + B' D'$ $C_1 = C' D' + C D + B'$ $C_2 = B + C' + D$ $C_3 = B' D' + C D' + B C' D + B' C$ $C_4 = B' D' + C D'$ $C_5 = A + C' D' + B D' + B C'$ $C_6 = A + C D' + B C' + B' C$	$C_0 = B C' D + C D + B' D' + B C D' + A$ $C_1 = B' D + C' D' + C D + B' D'$ $C_2 = B' D + B C D + C' D' + C D + B C D'$ $C_3 = B C' D + B' D + B' D' + B C D'$ $C_4 = B' D' + B C D'$ $C_5 = B C' D + C D' + A + B C D'$ $C_6 = B' C + B C' + B C D' + A$
--	--



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The better implementation can be like this that there are 9 unique terms ok. So, if you are looking into this functions; see this A is coming here as well as there.

So, this then this say this $\bar{B}C$. So, $\bar{B}C$ is there in C3 as well as in C6, ok. So, in this way if you look into the if you look into the patterns, then many of this product terms are common between the functions.

So, to do so, this is the situation; so, this $\bar{B}C$ is common and all. So, what we do, we can modify it so that this is the, this commonality can be used, ok. So, this commonality can be used so, that way I will have a less number of terms ok. So, less number of terms will be realized. So, this so this they will share the number of outputs and then the of course, the each output is not in minimized form like you see that here we have got so, this B in this C0, it was $A+BD+C+\bar{B}\bar{D}$. But we have modified this A so the C0 so that the expression becomes like this.

That is we do a grouping so that it is not the optimal one, but what has happened is that this $B\bar{C}D$ is shared between C0 and C3. Similarly, this $\bar{B}\bar{D}$ is shared between say this C0 and this C3. So, that way we can make these terms common between these multiple outputs so that total number of unique terms becomes less ok. So, this is actually a combinational logic minimization problem and for multiple outputs this is a very difficult problem in fact. And there are many CAD tools that are available which does this type of optimizations, where it tries to share terms between the logic functions to get it the to get the minimum number of product terms transfer the whole function.

So, whatever minimization technique we have learnt using Karnaugh map and all, so, they are for single output function. So, whenever we have got multi output function, then there is a goal to share between the share the product terms between outputs. And this is a one such example where we see that if we group in a different fashion, then for example, this C2 instead of grouping like this, where we are getting say $B + \bar{C} + D$.

So, in this type of grouping so, you will get C2 as a term like this. But the grouping is that you do not need any new term so, for example, this $\bar{B}D$ is there in C1 also \bar{C} so, the $B\bar{C}D$ is there in C0 also, $\bar{C}\bar{D}$ is there here also CD is also there, and then $BC\bar{D}$ is also there in C0. So, as a result more number of terms are getting shared so, using only 9 unique product terms so, you can realize all the functions C 0 to C 6 ok; so, that can be done.

(Refer Slide Time: 20:51).

The screenshot shows a presentation slide with a yellow header bar containing various icons. The main title is "Production Line Control". Below the title is a bulleted list of requirements and observations:

- Rods of varying length (+/-10%) travel on conveyor belt
 - Mechanical arm pushes rods within spec (+/-5%) to one side
 - Second arm pushes rods too long to other side
 - Rods that are too short stay on belt
 - 3 light barriers (light source + photocell) as sensors
 - Design combinational logic to activate the arms
- Understanding the problem
 - Inputs are three sensors
 - Outputs are two arm control signals
 - Assume sensor reads "1" when tripped, "0" otherwise
 - Call sensors A, B, C

At the bottom left, there are logos for IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side, there is a small video window showing a man speaking.

Next we will consider another example which is production line control so, we have got some rods with varying length that travel on a conveyer belt, ok. So, the length may vary the length of the rod at the production side so, it can vary between $\pm 10\%$. And mechanical arm pushes rods with specification $\pm 5\%$ to one side, and the second arm pushes rods too long to the other to the other side.

So, what is so, the rods are produced with an accuracy of $\pm 10\%$, but we want the rods so, we want the rods to be on the, to be or of proper specification.

(Refer Slide Time: 21:36)

Sketch of the problem

- Position of Sensors
 - A to B distance = specification
– 5%
 - A to C distance = specification
+ 5%

IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

So, we will have these things. So, this is the specification. So, this is within specification the rods are within specification then it will pass.

If the rods are shorter so, that is too short so, they will be taken out by this on this line. And similarly if the rods are larger like here so, it is too long so, then also it will be taken out. So, A to B distance is the specification minus 5 percent so, A to B distance is specification minus 5 percent. So, any rod that passes through this AB so, that is actually not acceptable. Similarly A to C distance is specification plus 5 percent so, that way that is also not acceptable.

So, anything between this plus minus 5 percent is acceptable or, but otherwise it is not.

(Refer Slide Time: 22:31)

Problem formulation

- Truth Table
 - Show don't cares

A	B	C	Function
0	0	0	do nothing
0	0	1	do nothing
0	1	0	do nothing
0	1	1	do nothing
1	0	0	too short
1	0	1	don't care
1	1	0	in spec
1	1	1	too long

logic implementation now straightforward
just use three 3-input AND gates

"too short" = $AB'C'$
(only first sensor tripped)

"in spec" = $A BC'$
(first two sensors tripped)

"too long" = $A BC$
(all three sensors tripped)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for this one if ABC is 0 0 0 so, that is basically this ABC so, they are so, it is not violating any of the conditions, then we do nothing. If it is 0 0 1 that is it is not way it this A and B are 0 0 and C is 1 so, then also it is do nothing. So, only when this A is if A is equal to 1 in that case the rod is too short so, output is too short. And then you if it is 1 0 1 is again do not care or do nothing, because that is do not care condition. Then 1 1 0 this is the valid within specification and 1 1 1 is too long.

So, I can write down the corresponding logic expressions for too short in within specification and too long, and accordingly we can get the corresponding function. So, I can have a combinational logic so, based on this ABC outputs from this rail, so, from this rail so, we can identify whether the rod is within specification too long or too short.

(Refer Slide Time: 23:41)

Logical Function Unit

- Multi-purpose Function Block
 - 3 control inputs to specify operation to perform on operands
 - 2 data inputs for operands
 - 1 output of the same bit-width as operands

C0	C1	C2	Function	Comments
0	0	0	1	always 1
0	0	1	A + B	logical OR
0	1	0	(A • B)'	logical NAND
0	1	1	A xor B	logical xor
1	0	0	A xnor B	logical xnor
1	0	1	A • B	logical AND
1	1	0	(A + B)'	logical NOR
1	1	1	0	always 0



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Another example so, we have got a multipurpose function block. So, this is that is a C0, C1, C2 so, these are the control inputs, and there are 2 data inputs A and B. So, based on these C0, C1, C2 so, it will perform some function.

So, for example, if C0, C1, C2 is all of them are 0 so, it is a constant function. So, irrespective of the values of A and B o, it will output A 1. If C0, C1, C2 is 0 0 1 then it will compute A + B. So, it is a logical OR of A and B so, 0 1 0 is logical NAND, 0 1 1 is logical XOR. So, like that we have got different functions to be implemented if the C0, C1, C2 are having some specific value.

(Refer Slide Time: 24:25)

C0	C1	C2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Now, so, this is the truth table so, this C0, C1, C2 so, this is 0 0 0, then this A and B they are having so, this is a constant function. So, irrespective of the values of A and B output will be 1. Similarly this for 0 0 1 so, this is the OR of A and B so, this is the OR function is realized here. Then this 0 1 0 so, this is the NAND function so, this doing a NAND here. So, this way you can draw the truth table.

And after you have made this truth table you can do a logic minimization to get an expression for F in terms of C0, C1, C2, A and B ok, and that can give us the that can give us the expression. So, that can give so that the a final circuit is not shown here, but you can always do some logic minimization and get the function the logic the digital circuit implemented in terms of C0, C1, C2, A and B.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 20
Arithmetic Circuits

So, next we will be looking into Arithmetic Circuits. So, the circuits there are many digital circuits that perform arithmetic operation like addition, subtraction, multiplication, division etcetera and they by themselves form one special class of circuits ok. So, this is arithmetic circuits are looking into that.

(Refer Slide Time: 00:40)

Adder

- Great example of Iterative design
- Design a 1-bit adder circuit, then expand to n-bit adder
- Look at
 - Half adder – which is a 2-bit adder
 - Inputs are bits to be added
 - Outputs: result and possible carry

$$\begin{array}{r} & 1 \\ & | \\ \overline{1} & 0 \\ + & \overline{0} \\ \hline & 0 \end{array}$$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, the first example of this arithmetic circuit that we will consider is that of an adder. This is an example of iterative design. So, iterative design means so, you have got a basic block and that basic block is repeatedly used for getting the more complex designs ok. So, that way this adder is an example of this iterative design. So, we design a 2-bit adder and using the 2-bit adder. So, we can expand it to 3 bit, 4 bit, 5 bit like that.

So, we design a 1-bit adder circuit and then expand to n-bit adder circuit. So, this 1-bit adder; so it has got 2 1-bit inputs and it produces the sum of them sum and carry bit and accordingly and then this 1-bit adder can be expanded to have any arbitrary n-bit adder.

So, we will look into half adder first; so which is a 2-bit adder and inputs are bits to be added and the outputs are result and possible carry. So, when you are adding 2 1-bit numbers say; so, if the bits are say 1 and 1 then the sum is 0 and there is a carry of 1. So, this has to be generated; whereas if it is 1 and 0 then the sum is 1 and the carry there is no carry; so, carry is 0.

So, we can have this type of situation or say 0 0; sum is 0 and the carry is also 0. So, this way we have got half adder; so, this is a 2-bit adder and then output and this sum and carry; so these are the 2 outputs.

(Refer Slide Time: 02:17)

Adder

- Great example of Iterative design
- Design a 1-bit adder circuit, then expand to n-bit adder
- Look at
 - Half adder – which is a 2-bit adder
 - Inputs are bits to be added
 - Outputs: result and possible carry
 - Full adder – includes carry in, really a 3-bit adder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

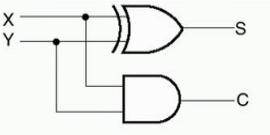


So, on the other hand we have got full adder which includes a carry in also. So, it has got these 2 bits to be added and it has got a carry input. So, that is this is actually all this 3 bits will be added to produce the sum and the carry.

(Refer Slide Time: 02:33)

Half Adder

- $S = X \oplus Y$
- $C = XY$



Logic Diagram of Half Adder

Truth Table of Half Adder

Inputs		Outputs	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, the half adder is something like this; so, this is the logic diagram of half adder. So, this is the; so, truth table is like this. So, as I have said that it has got inputs X and Y and it has got 2 outputs sum and carry. So, how; so, if 0 and 0 are added these 2 bits are added then sum equal to 0 and carry is also equal to 0. If it is 0 and 1 sum is equal to 1 and carry is equal to 0, 1 0 sum equal to 1 carry equal to 0; 1 1 sum equal to 0 and carry is equal to 1.

So, if I look into the corresponding circuit then this sum output it is nothing, but the XOR function you see whenever X is 0 1 or Y is 1 then only the output is 1 and that is in the exclusive OR fashion. So, this S output is the XOR of X and Y and the carry output is the AND of X and Y. So, C is equal to 1 when X and Y both the inputs are at 1. So, we have got the logic diagram of half adder consisting of one XOR gate and one AND gate. So, S equal to X XOR Y and C equal to XY.

(Refer Slide Time: 03:45)

Full Adder

- Three inputs. Two are operand bits, third is C_{in}
- Two outputs: sum and carry

Inputs			Outputs	
x	y	z	c	s
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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The other one full adder; so, in case of full adder we have got this inputs like this. We have got this X Y and Z as input and then when this 0 0 0 are added. So, sum is 0 and carry is 0 when 0 0 1 in that case sum is 1 and carry is 0. So, for 0 1 1 sum is 0 and carry is 1 and when it is 1 1 1 both sum and carry are 1. So, in this way we have got the truth table of the full adder.

Now, it has got 3 inputs the full adder has got 3 inputs 2 are the operand bit say X and Y may be the operand bit and the third is the carry input C in.

(Refer Slide Time: 04:36)

K Map for S

YZ \ X	00	01	11	10
0	0	1	0	1
1	1	0	1	0

In a half adder the sum bit:
 $S = X \oplus Y$

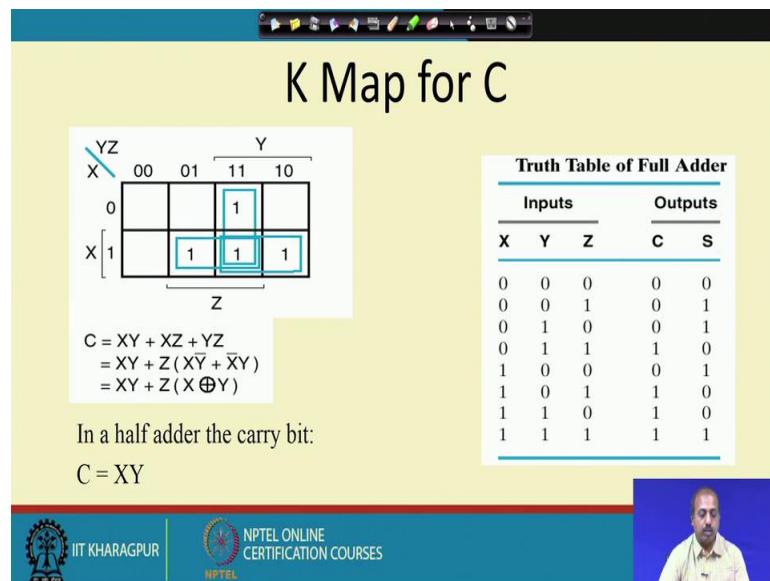
Inputs			Outputs	
x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

It has got 2 outputs sum and carry and so, for the Karnaugh map for sum is like this. So, if you if you draw the Karnaugh map then it will be like this that we have got this. So, this is basically the XOR ok; so, this is the XOR of XY and Z. So, this is basically this a sum equal to $\bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$ which is nothing, but $X \text{ XOR } Y \text{ XOR } Z$. So, the sum is equal to XOR of X Y and Z and in case of half adder you remember that the sum is equal to $X \text{ XOR } Y$.

So; that means, if I use this half adder and we can make a full adder out of that how? We will see that later.

(Refer Slide Time: 05:20)



But this K map for Karnaugh map K map for the carry output is like this. So, carry output is 1 whenever this Y and Z equal to 1 then or this X Z equal to 1 XY equal to 1 or XYZ equal to 1. So, I if I draw the truth table and then make the grouping.

So, it will be like this and so, I can say that is C equal to $XY + XZ + YZ$. So, it can be written as $XY + Z(X + Y)$ and this $X + Y$ can be written in this in this XOR form also because if X and Y both are equal to 1; then this term actually takes care of this thing. So, of this C output but if this only one of them is equal to 1, then this Z; Z being equal to 1; so output should be equal to 1.

So, I can change this XY term X + Y term into this XOR form. And then it terms out to be XY + Z.(X XOR Y); so, Z AND X XOR Y. So, in a half adder the carry output was XY and in case of full adder; so, you have got this as the carry output.

(Refer Slide Time: 06:37)

Using two half Adders to Build a Full Adder

- Full adder functions

$$S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

$$= X \oplus Y \oplus Z$$

$$C = XY + XZ + YZ$$

$$= XY + Z(\bar{X}\bar{Y} + \bar{X}Y)$$

$$= XY + Z(X \oplus Y)$$

- Half adder functions

$$S = X \oplus Y$$

$$C = XY$$


IIT KHARAGPUR

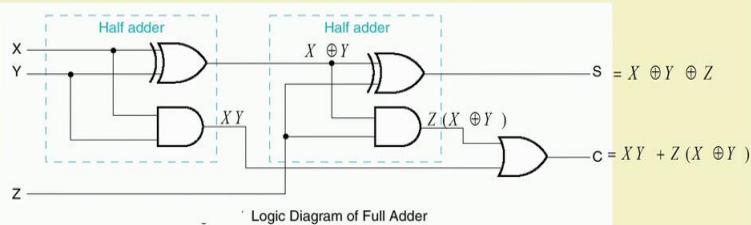
NPTEL ONLINE
CERTIFICATION COURSES



So, if we consider the full adder function; so, sum equal to X XOR Y X OR Z and carry equal to C equal to XY + Z.(X XOR Y). So, and the half adder function was S equal to X XOR Y and C equal to XY.

(Refer Slide Time: 06:53)

Two Half Adders (and an OR)



The diagram illustrates the logic of a full adder using two half adder blocks and an OR gate. The inputs are X, Y, and Z. The first half adder takes X and Y as inputs, producing a sum (S) and a carry (XY). The second half adder takes the carry from the first half adder (XY) and the input Z as inputs, producing a sum (X ⊕ Y) and a carry (Z(X ⊕ Y)). An OR gate then takes the sum from the first half adder (S) and the sum from the second half adder (X ⊕ Y) as inputs, producing the final sum (S = X ⊕ Y ⊕ Z). The final carry output is C = XY + Z(X ⊕ Y).


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES



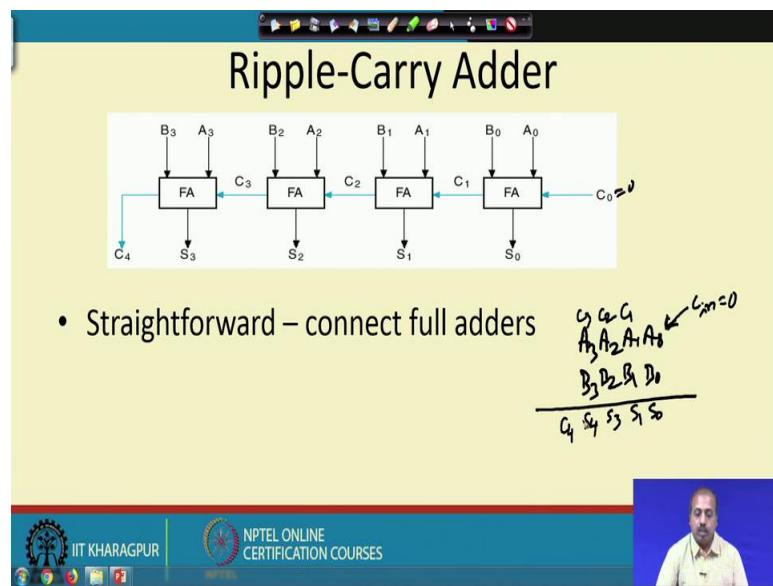
So, what we can do? We can use 2 half adders to get a full adder. So, here you see that the in the first half adder we have given X and Y as input. So, we have got XY as the carry output and X XOR Y as the sum output. So, with that if we put this Z as the second input. So, second half adder has got this sum as one input and the Z as another input; so, this X XOR Y XOR Z.

So, that gives me the sum S equal to X XOR Y XOR Z and this carry is generated so, this carry part. So, what we have done? We have taken a separate OR gate. So, this first this half adder; first half adder; so it gave XY as output in the carry, but what we need is

$$XY + Z.(X \text{ XOR } Y).$$

So, this AND gate will be realizing that $Z.(X \text{ XOR } Y)$ because $X \text{ XOR } Y$ is coming here. So, it is in the half adder form; so, it is connected to the AND gate. So, it is connected here and then this Z input is connected. So, here we get $Z.(X \text{ XOR } Y)$; so, ultimately if I use an OR gate to OR between these 2 outputs. So, you get C equal to $XY + Z.(X \text{ XOR } Y)$. So, that is the full adder logic diagram; so, using this half adder, so we can realize a full adder.

(Refer Slide Time: 08:20)



Next, we will be looking into another extension like how can I have more number of bits in an adder? So, so far wherever we have seen a full adder; so, it can have 2 inputs A and B and a carry input and then it is producing the sum output and carry output just in the

previous slide we have seen that a full adder. So, it is having XYZ as the 2 input as Z may be considered as carry input and it is producing a sum output and a carry output.

Now, if you have got a number of if you have to if you have got to add a number bits word size of individual numbers are more say 4 bit numbers; then what we can do we can connect this full adders in a cascaded fashion ok. So, this C_0 is the carry in coming from the first adder.

So, it may be if there is no carry for the addition; so, basically I am interested to do this addition this A_3, A_2, A_1, A_0 this bit pattern I want to add with B_3, B_2, B_1, B_0 . So, and there may be some previous carry coming into the thing; so, that is a C input. And if this carry is not there then we can set it to 0 or this C_0 can be set to be equal to 0 if there is no carry coming from the previous stage; so, carry coming from the input side.

So, that way we can; so, what we are doing? For the first full adder; so, we are giving A_0 and B_0 and this C_0 . So, it produces the S_0 bit and the carry 1. So, this a $0 + B_0$; so the C_0 comes here and the carry output goes here. The second full adder will add the C_1, A_1 and B_1 ; it will produce S_1 and it will produce another carry C_2 ok. So, C_2, A_2 and B_2 ; so they will added by the third full adder. So, this will giving S_3 and C_3 and this will be giving me S_4 and C_4 .

So, that is what is happening ok. So, we have got two 4 bit numbers they are getting added and it is producing a 5 bit number consisting of 4 sum bits and 1 carry bit. So, this way in a straight forward fashion; so, we can connect full adders and get the circuit for higher number of bits.

(Refer Slide Time: 10:50)

Ripple-Carry Adder

```
graph LR; FA1[FA] -- B3 --> S3; FA1 -- A3 --> C3; FA2[FA] -- B2 --> S2; FA2 -- A2 --> C2; FA3[FA] -- B1 --> S1; FA3 -- A1 --> C1; FA4[FA] -- B0 --> S0; FA4 -- A0 --> C0; C4 --> C3
```

- Straightforward – connect full adders
- C_4 : Chain carry-out to carry-in of FA of bits A_4 & B_4
 - C_0 in case this is part of larger chain
 - otherwise just set to zero

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, C_4 is the chain carry out to the carry in of a full adder of bits A_4 and B_4 . So, we can if we have got next stage A_4 and B_4 ; so, we can we can chain it to the next stage and C_0 in this case is part of larger design. So, if it is coming from another, another such some adder then C_0 may be the carry chain output of that adder otherwise we can just set it to 0.

(Refer Slide Time: 11:17)

Hierarchical 4-Bit Adder

- We can easily use hierarchy here
- Design half adder
- Use in full adder
- Use full adder in 4-bit adder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can now we can see the hierarchy ok. So, for 4 bit adder; so, we first design the half adder and use that half adder to design full adder and then use this full adders; it will get say 4 bit adder. So, that way we can have a clear cut hierarchy in the adder design.

(Refer Slide Time: 11:37)

The slide has a yellow background and a blue header bar. The title 'Subtractor' is centered in the header. Below the title is a bulleted list:

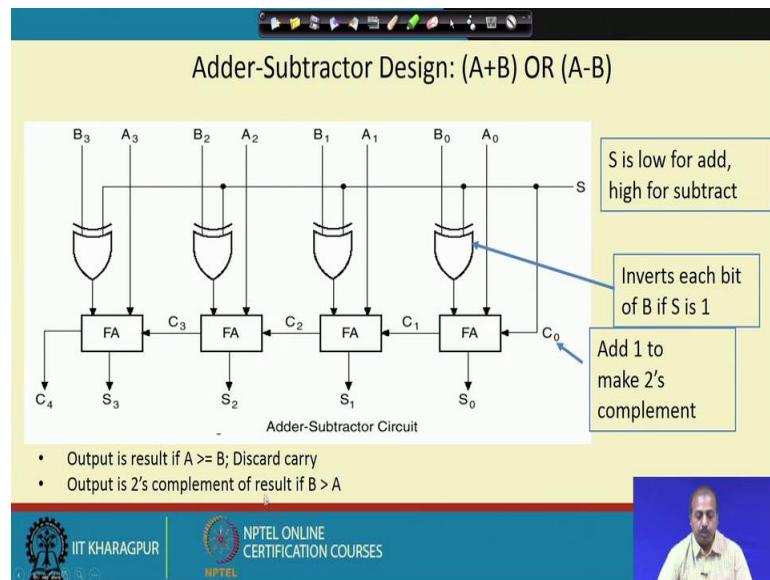
- Compute $M-N$
 - Add 2's complement of N to M :
 $M + (2^n - N)$
 - If $M \geq N$, need only "one adder" and a "complementer of N " to do the subtraction.
 - If $M < N$, we also need to take 2's complement of adder's output to produce magnitude of result
 $2^n - \{M + (2^n - N)\} = N - M$

At the bottom of the slide, there is a red footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man speaking.

On the other hand subtractor: if you want to compute $M - N$ and from our knowledge of this number theory. So, we know that we have to add 2's complement of N to M . So, we should have this so that is $M + (2^n - N)$; so, that is to be done. Now if M is greater or equal N . So, in that case we need only one adder and completer of N to do the subtraction. So, if M is greater or equal N then we just; so, we take the 2's complement of N and do the addition that will give me the subtraction.

However, if M is less than N ok; so, then we need to take 2's complement of adder's output to produce the magnitude of the result. So, if M is less than N then the after doing the; so, the addition part is same or the sub same as the 2's complement subtraction part. So, $M + (2^n - N)$; so, that is done in both the cases, but for getting the 2's complemented result. So, we have to do this $2^n - \{M + (2^n - N)\} = N - M$.

(Refer Slide Time: 12:50)



So, this is the Adder-Subtractor design $A + B$ or $A - B$. So, either of them can be done so this. So, here this S is it is the S is a control. So, this S is low for addition and high for subtraction. So, if S is low; so here I am getting a 0, C_0 is 0. So, it will be simply doing and this X OR gates. So, they will be having this S input has 0. So, as a result they will be passing it as 0 the B_0 will be passed here B_1 will be passed here, B_2 will be passed here and B_3 will be passed here. So, this will do the simple addition.

Now, if you are doing subtraction in that case this S will be equal to 1. So, this B_0, B_1, B_2, B_3 so, they will get complemented via this XOR gate; in XOR gate if one of the input is tied high we know that it acts as an inverter. So, at this point I will get \overline{B}_0 here I will get $\overline{B}_1, \overline{B}_2$ and \overline{B}_3

So, I will get the complemented versions there and then this C_0 is equal to 1. So, that gives us the 2's complement of B and ultimately that results in $A - B$.

So, this Adder-Subtractor design is very simple; so, on top of this basic adder circuit. So, we have got a completer circuit which is consisting, consisting of a few XOR gates and that gives us the solution. Output is a result if A greater or equal B ; so we just discard the carry bit and output is 2's complement of result, if B is greater than A . So, if you want to get back the original value then we have to take another 2's complement at this point.

(Refer Slide Time: 14:43)

Overflow

- Overflow means that **result cannot be represented with the number of bits used**
- Two cases of overflow for addition of signed numbers
 - Two large positive numbers overflow into sign bit
 - Not enough bits for result
 - Two large negative numbers added
 - Same – Not enough bits for result

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Now, over flow may occur; so, what is an overflow? So, when the result cannot be represented within the number of bits used ok. So, doing some 2 additions and then the I am doing 2 4 bit number addition the result can go to 5 bits. So, that way they if I am using only 4 bits for their realization; so there is an overflow. Particularly, for 2's complement number system when I am doing the subtraction; so, it can occur quite often.

So, there can be 2 cases of overflow for addition of signed numbers 2 large positive numbers overflow into sign bit. So, not enough bits for the result or 2 large negative numbers are being added and same that is a not enough bits for the result. So, so, as I said that 4 bit numbers being added it gives 5 bit result and the 5th bit is actually for the sign. So, if I say that way then that will be effect in the sign bit.

(Refer Slide Time: 15:40)

The slide contains the following bullet points:

- Consider $7 + 7$
- Overflow: cannot represent 14 using only 4 bits
- Generates NO CARRY, $C_4 = 0$

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

So, $7 + 7$; so, this is giving me 14. So, if I am. So, in 4 bit; so, if I am representing both positive and negative numbers we know the range is - 8 to + 7. So, the + 14 cannot be realized you cannot be represented in 4 bit; so that will give us an overflow.

(Refer Slide Time: 16:00)

The slide has a title "Example 2" at the top. It contains the following bullet points:

- Consider $-7 - 7$
- Overflow: cannot represent -14 using only 4 bits
- Generates CARRY, $C_4 = 1$

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

So, it generates no carry and C_4 may remains equal to 0; on the other hand say you can consider $-7 - 7$ that is the result is - 14. So, here also it is an overflow because I cannot represent - 14 in 4 bits, but in this case it generates a carry. So, C_4 equal to 1 if you look into that final carry generated; so, this C_4 will be equal to 1.

(Refer Slide Time: 16:25)

The screenshot shows a presentation slide with a yellow background. At the bottom, there is a blue footer bar containing the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the slide, there is a small video window showing a man speaking.

- Consider $4 + 4$
- **Overflow: cannot represent 8 using only 4 bits**
- Generates NO CARRY, $C_4 = 0$

Now, $4 + 4$; so, here the result is 8 here also I cannot represent 8 in 4 bits because the range is - 8 to + 7. However, it does not generate a carry C_4 equal to 0.

(Refer Slide Time: 16:38)

The screenshot shows a presentation slide with a yellow background. At the bottom, there is a blue footer bar containing the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the slide, there is a small video window showing a man speaking.

- Consider $7 - 7$
- **NO Overflow**
- Generates CARRY, $C_4 = 1$

And the $7 - 7$; there is no overflow and, but it generates a carry C_4 equal to 1. So, if you trace through the addition process then you can understand.

(Refer Slide Time: 16:47)

Overflow Detection

- Condition for overflow:
 - either C_{n-1} or C_n is high, but not both
 - $7 + 7$; only $C_{n-1}(C_3)$ is high; **overflow**
 - $-7 - 7$; only $C_n(C_4)$ is high; **overflow**
 - $4 + 4$; only $C_{n-1}(C_3)$ is high; **overflow**
 - $7 - 7$; BOTH C_{n-1} & C_n (C_4 & C_3) are high; **no overflow**

The diagram shows an n-bit Adder/Subtractor block with its carry output C_n connected to one input of an XOR gate. The other input of the XOR gate is connected to the carry output C_{n-1} . The output of the XOR gate is labeled V , which represents the overflow bit.

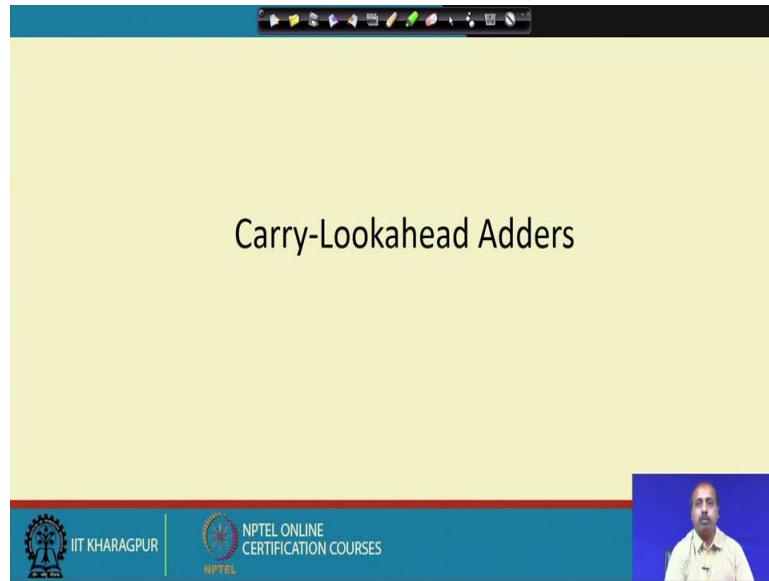
IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

So, this overflow detection is basically if there is an addition, Adder-Subtractor, so, you have to compare between the carries generated at nth stage and $(n - 1)^{\text{th}}$ stage. So, if the if these 2 carry if you if you take an XOR of these 2, then that will be giving us the overflow bit; so, whether it is overflow or not.

So, the condition for overflow is either C_{n-1} or C_n is a high, but not both. So, that way it; it this is the overflow condition. So, all 4, 4, 5 cases that we have seen previously, if you if you make a summary of them; then we will see that it comes to some such conclusion that is a C_{n-1} or C_n should be high, but not both; so, that is the condition for overflow.

So, this $7 + 7$ only C_3 is high; so C_4 is not high. So, that is an overflow. $-7 - 7$. So, C_4 is high, but C_3 is not high; so, that is also an overflow $+4 + 4$. So, that is C_{n-1} is high that is C_3 is high so again there is overflow. And $7 - 7$; so, both the carries are generated, but there is no overflow. So, that way this overflow detections circuit is the XOR of C_n and C_{n-1} .

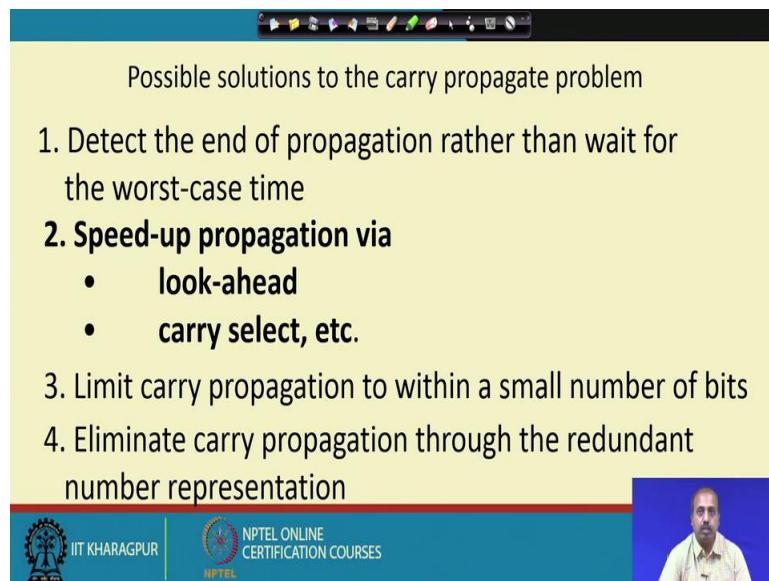
(Refer Slide Time: 18:08)



Now, one difficulty with this type of adder that we have seen is that I have got; I have cascaded number of full adders. Now the second stage of the adder it cannot work till the first stage has completed the addition and the carry bit is available from the first stage.

So, that way if we go ahead then for further and further stages; there will be problem in terms of this carry not yet appear not yet come to the adder stage ok. So, that makes this addition process or these ripple carry addition process pretty slow ok.

(Refer Slide Time: 18:49)



So, another type of adder which is quite fast is known as carry look ahead adder that we will see in this. So, this carry propagation is a problem; so, detect the end of propagation rather than wait for the worst case time to occur. And speed up the propagation while via look ahead carry select etcetera, and it will limit the carry propagation to a small number of bits and eliminate carry propagation through the redundant number redundant number representation. So, these are some of them so we will see them slowly.

(Refer Slide Time: 19:15)

Basic Signals

Generate signal:	$g_i = x_i y_i$
Propagate signal:	$p_i = x_i \oplus y_i$
Anihilate (absorb) signal:	$a_i = \overline{x_i} \overline{y_i} = \overline{x_i + y_i}$
Transfer signal:	$t_i = g_i + p_i = \overline{a_i} = x_i + y_i$
$c_{out} = 1$ given $c_{in} = 1$	

Carry recurrence

$$c_{i+1} = g_i + c_i p_i = g_i + c_i t_i$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, you see that for a particular stage ok; so. So, if I take an example then suppose this is a particular adder stage ok. So, it has got this x_i and y_i as the bits to be added and c_i is the carry. And it produces the sum output and the carry output; now we try to see when a carry will be generated from this stage. So, the carry will be generated only when this x_i and y_i both are equal to 1; so, that is.

So irrespective of whatever be the carry coming from the previous stage; if x_i and y_i both are equal to 1; so, this is x_i and y_i if both are equal to 1 then irrespective of whatever be the carry coming from the previous stage, so it will generate a carry; this carry will be generated. So, if I have got a previous stage; so then this carry will be coming to this stage. So, this carry will be generated from this stage.

And other condition is other possibility of generating a carry at getting a carry at this point is there was a carry coming to the previous stage and that propagated through this stage and when this will propagate? So, this will propagate when only one of this x_i and y_i value

is equal to 1 because in that case that one will be added. So, suppose x_i equal to 1 and this $carry_i$ equal to 1; so, as a result this sum will be 0 and this carry will be generated at this stage. So, I can say that whatever carry was coming as input to the stage that is getting propagated to the next stage.

However, if both of them are equal to 1; so, x_i and y_i both are equal to 1. So, the sum is like this; so, anyway this, so this will be the output and anyway this X this x_i and y_i being equal to 1; so, carry is already generated. So here, the generation and propagation does not have any separate meaning, so they are all the same. So, we can say in summary that this x_i and y_i . So, they are they will generate a carry if both the bits are equal to 1. So, x_i generate is equal to $x_i y_i$ and propagate is equal to $x_i \text{ XOR } y_i$. So, if only one of them is equal to 1 then it will propagate.

Now, when a carry will not propagate? So, it will not propagate or it is absorb if $a_i = \bar{x}_i \bar{y}_i$. So, if both x_i and y_i are 0 then the carry that was coming from previous stage will get absorb there.

(Refer Slide Time: 22:00)

Basic Signals

Generate signal:	$g_i = x_i y_i$
Propagate signal:	$p_i = x_i \oplus y_i$
Anihilate (absorb) signal:	$a_i = \bar{x}_i \bar{y}_i = \bar{x}_i + \bar{y}_i$
Transfer signal:	$t_i = g_i + p_i = \bar{a}_i = x_i + y_i$
$c_{out} = 1$ given $c_{in} = 1$	

Carry recurrence

$$c_{i+1} = g_i + c_i p_i = g_i + c_i t_i$$

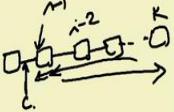
So, what I mean; what I say is if this x_i and y_i both are 0. So, if there is a carry coming from the previous stage then this will be get getting absorbed in it sum and the carry generated is 0 from this stage so that is the absorbed signal.

So, we can say that there is an absorbed signal $a_i = \overline{x_i + y_i}$ and there is a transfer signal which is g_i or p_i . So, it will be transferred the; if it is if the carry is generated or carry is propagated. So, so, this $g_i + p_i$, so it can be written as $\bar{a}_i = x_i + y_i$ and c_{out} is 1 given c_{in} equal to 1.

And the carry recurrence is like this. So, so I can say that c_{i+1} is either the carry is generated at this stage or there was a carry coming from the previous stage and this stage allowed propagation ok. So, this was the expression is $g_i + c_i p_i$ and this expression can be simplified to be written as $g_i + c_i t_i$ instead of $c_i p_i$. So, I can write is a write it as $c_i t_i$ also the transfer signal. So, this way I can say the how the carry prop, recurred carry recurrence occurs.

(Refer Slide Time: 23:30)

Unrolling Carry Recurrence

$$\begin{aligned}
 c_i &= g_{i-1} + c_{i-1} p_{i-1} = \\
 &= g_{i-1} + (g_{i-2} + c_{i-2} p_{i-2}) p_{i-1} = g_{i-1} + g_{i-2} p_{i-1} + c_{i-2} p_{i-2} p_{i-1} = \\
 &= g_{i-1} + g_{i-2} p_{i-1} + (g_{i-3} + c_{i-3} p_{i-3}) p_{i-2} p_{i-1} = \\
 &= g_{i-1} + g_{i-2} p_{i-1} + g_{i-3} p_{i-2} p_{i-1} + c_{i-3} p_{i-3} p_{i-2} p_{i-1} = \\
 &= \dots = \\
 &= g_{i-1} + g_{i-2} p_{i-1} + g_{i-3} p_{i-2} p_{i-1} + g_{i-4} p_{i-3} p_{i-2} p_{i-1} + \dots + \\
 &\quad + g_0 p_1 p_2 \dots p_{i-2} p_{i-1} + c_0 p_0 p_1 p_2 \dots p_{i-2} p_{i-1} = \\
 &= g_{i-1} + \left(\sum_{k=0}^{i-2} g_k \prod_{j=k+1}^{i-1} p_j \right) + c_0 \prod_{j=0}^{i-1} p_j
 \end{aligned}$$




IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, I have got this $c_i = g_{i-1} + c_{i-1} p_{i-1}$. So, if you just expand it; so, you will get the $c_{i-1} = g_{i-2} + c_{i-2} p_{i-2}$; so, it is like this. So, if you go on expanding like this. So ultimately it will be coming to an expression in this format. So, you can you can get the series sum like this. So, it is g_{i-1} ; so $c_i = g_{i-1}$ that is the; $(i-1)^{th}$ stage has generate a carry or this stage k from 0 to $i-2$.

So, k^{th} stage generated the carry and the remaining stages; they propagated the carry ok; so this is the situation. So, this is what we say is that; so, we have got this stages we have got this stages. So, whenever we are talking about so, this is the i^{th} stage; so either at this point I have got the c_i . So, either this, so this is the stage $i-1$. So, from $(i-1)^{th}$ stage I

have got the carry c_i . So, c_i is either at $(i - 1)^{\text{th}}$ stage it was generated; so, that is given by this g_{i-1} .

Or in $i - 2$ stage the carry was generated and this stage just propagated it. So, this is g ; so, $k = i - 2$. So, it is generated here and it is propagated it is generated at $i - 2$ stage and propagated at $i - 1$ stage or it is generated at any stage in this site. So, it generated at any k -th stage; so some k -th stage some stage k at which the carry was generated and then it got propagated through all this remaining stages; so, that is taken care of by this expression.

So, this is telling that it is generated at stage k and the stages. So, $k + 1$ to $i - 1$; they are just propagating the carry. So, that way we have got this carry propagated and then other possibility is that initially c_0 was there. So, it was given the carry was the input carry was 1 and then all the stages 0 to $i - 1$ they propagated that carry. So, these are the 3 alternatives that we can have for the carry generation.

(Refer Slide Time: 25:55)

4-bit Carry-Lookahead Adder

$$C_4 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3$$

$$C_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$$

$$C_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$$

$$C_1 = g_0 + c_0 p_0$$

$$S_0 = x_0 \oplus y_0 \oplus c_0 = p_0 \oplus c_0$$

$$S_2 = p_2 \oplus C_2$$

$$S_1 = p_1 \oplus C_1$$

$$S_3 = p_3 \oplus C_3$$

 IIT KHARAGPUR |
  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now, so, if you have just looking into the 4 bit carry look ahead adder. So, this c_4 will be equal to something like this if you expand that expression then c_3, c_2, c_1, c_0 like this. Now so the sum 0 is given by $x_0 \text{ XOR } y_0 \text{ XOR } c_0$ which is nothing, but $p_0 \text{ XOR } c_0$ similarly

$s_2 = p_2 \text{ XOR } c_2, s_1 = p_1 \text{ XOR } c_1$. So, this way this sum and carries of individual stages can be written.

(Refer Slide Time: 26:36)

4-bit Carry-Lookahead Adder (2)

$$C_4 = g_3 + c_3 p_3 \quad \text{3 gates less}$$
$$c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$$
$$c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$$
$$c_1 = g_0 + c_0 p_0$$

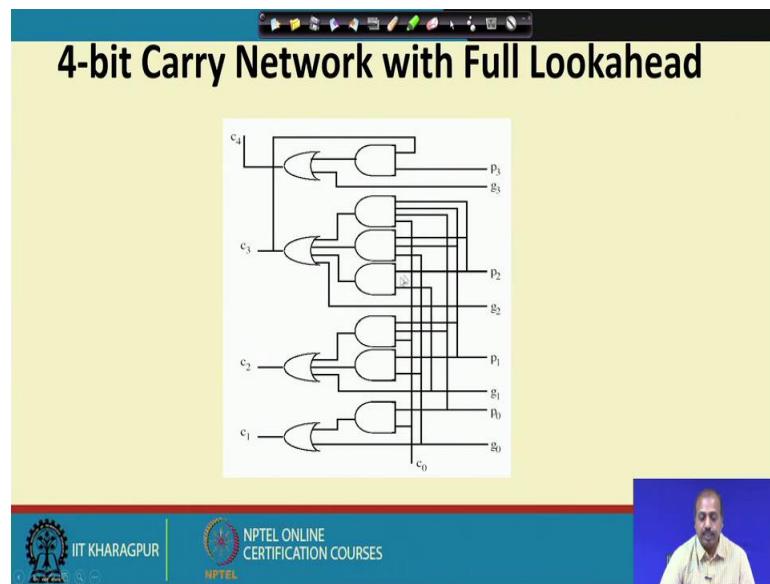
$$S_0 = X_0 \oplus Y_0 \oplus C_0 = p_0 \oplus c_0 \quad S_1 = p_1 \oplus c_1$$
$$S_2 = p_2 \oplus c_2 \quad S_3 = p_3 \oplus c_3$$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Now, we can just optimize it a bit having a requiring 3 less gates where the c_3 expression this $c_4 = g_3 + c_3 p_3$ then $c_3 = g_2 + g_1 p_2$ like this. So, we can just modify this previously written expression optimize it a bit; so, that it will take 3 gates less.

(Refer Slide Time: 26:48)



So, the circuit that we get is something like this c_0 goes to all the stages then this is g_0 . So, this is actually this carry network. So, where this c_1, c_2, c_3 and c_4 they are generated. So, they are there is a based on this generation generate a propagate logic. So, this is a

coming from this expression; so ok. So, this is straight away implementation of these expressions.

(Refer Slide Time: 27:22)

Equations for 4-bit Lookahead Carry Generation

$$C_{i+3} = g_{i+2} + g_{i+1} p_{i+2} + g_i p_{i+1} p_{i+2} + c_i p_i p_{i+1} p_{i+2}$$

$$C_{i+2} = g_{i+1} + g_i p_{i+1} + c_i p_i p_{i+1}$$

$$C_{i+1} = g_i + c_i p_i$$

$$g_{[i..i+3]} = g_{i+3} + g_{i+2} p_{i+3} + g_{i+1} p_{i+2} p_{i+3} + g_i p_{i+1} p_{i+2} p_{i+3}$$

$$p_{[i..i+3]} = p_i p_{i+1} p_{i+2} p_{i+3}$$

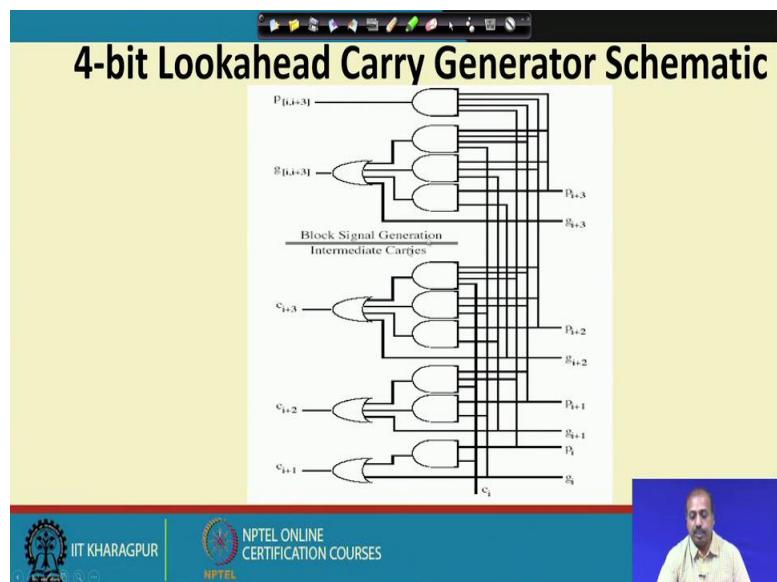

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES



Now, so for carry look ahead generation; so, we have got this expression in terms of i , so, we have go these expressions.

(Refer Slide Time: 27:34)



So, in general I will have this type of situation. Now the difficulty is that you see the number of gates it is increasing significantly. So it is somehow at some point of time I do not want to generate further carries, ok. So, I want to do some ripple carry and that way I

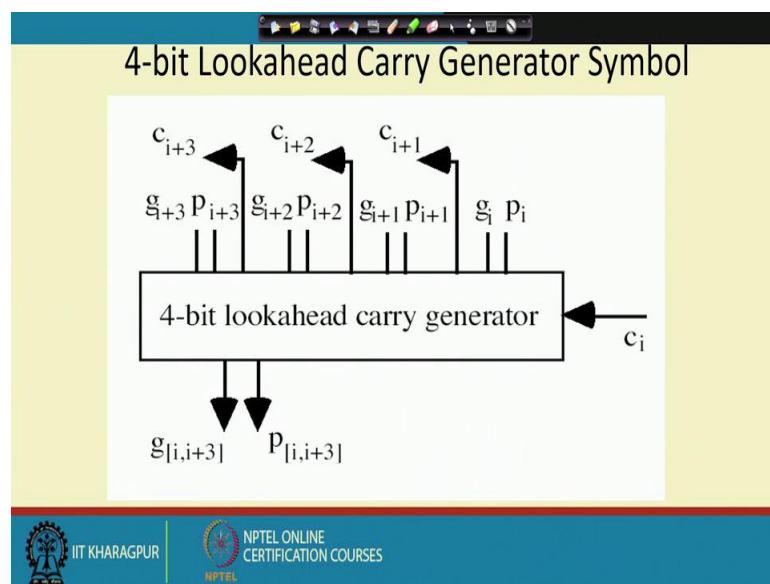
will make it simple so that so many, so much of gates are not required in the carry generation process.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21
Arithmetic Circuits (Contd.)

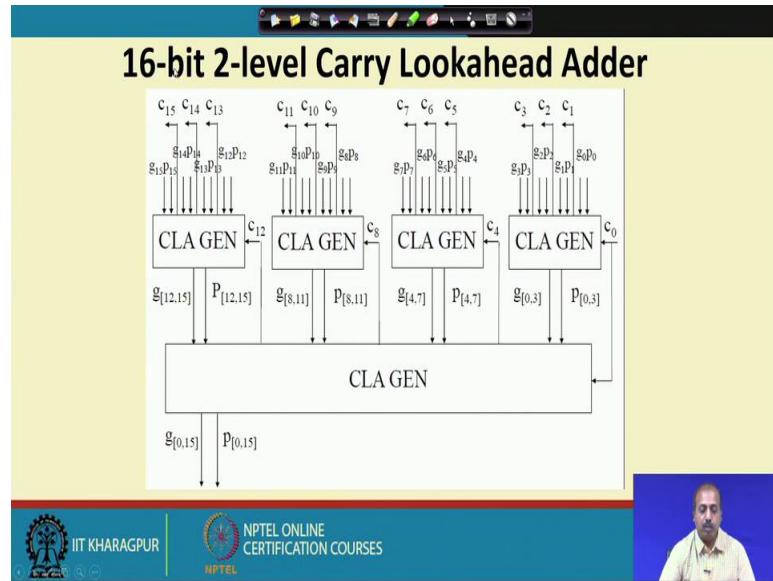
So, this look ahead carry generation part this portion; so, we can make it a design which is again an iterative design.

(Refer Slide Time: 00:18)



Suppose I have got this 4-bit look ahead carry generator circuitry; so, that has got c_i as input and it generates of course, this a_i and b_i those x_i and y_i those bits are there. So, it is not shown here explicitly. So, this g_i and p_i ; so, they are coming here in terms of that x_i and y_i . So, that part is not shown here and it generates this c_{i+1} , c_{i+2} , c_{i+3} and it generates this $p_{[i,i+3]}$ and $g_{[i,i+3]}$; so, those portions.

(Refer Slide Time: 01:00)



Now, if you have got that this carry look ahead generator 4-bit versions. So, now, you can use it for 16 bit carry look ahead circuitry; so, we have got this 4 such carry look ahead blocks connected. And these carry look ahead blocks; so, they are getting say this say this say $g_0, p_0, g_1, p_1, g_2, p_2$ etcetera and it is generating 2 generate one generate signal and one propagate signal. So, this generate signal will be will be fed to this CLA generator circuit.

So, this is actually this block is similar to this block; so, whatever be the number of the inputs here similar number of inputs are there in this CLA generator block also and we are just using some sort of hierarchical structure ok. So, this is using the same CLA generator block, but it is this, this g inputs are coming from this the corresponding CLA generator block of the previous level of hierarchy.

And it is generating $g_{0 \text{ to } 15}$ and $p_{0 \text{ to } 15}$ that is the if you take a 16 bit block then this will be the generate signal from the 15 bit block and this will be the propagate signal from the 16 bit block and this is the propagate signal from the 16 bit block.

(Refer Slide Time: 02:27)

Operation of 16-bit 2-level Carry Lookahead Adder

Signals computed	Formulas	Delay
g_i, p_i $i=0..15$	$g_i = x_i y_i$ $p_i = x_i \oplus y_i$	1 gate delay
$g_{[i..i+3]}, p_{[i..i+3]}$ $i=0, 4, 8, 12$	$g_{[i..i+3]} = g_{i+3} + g_{i+2} p_{i+3} + g_{i+1} p_{i+2} p_{i+3} + g_i p_{i+1} p_{i+2} p_{i+3}$ $p_{[i..i+3]} = p_i p_{i+1} p_{i+2} p_{i+3}$	2 gate delays

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, you can cascade another such block here; so, to get a 32 bit block ok; so, that way we can go ahead. So, we can we will looking to an example; so, these are simple formulas that are we have got this delay calculations and all like this g_i, p_i . So, the formula is $g_i = x_i y_i$; so, that is that will 1 1 gate delay and this p_i computation is $x_i \text{ XOR } y_i$; so, that is also 1 gate delay.

Now, this $g_{[i..i+3]}$ and $p_{[i..i+3]}$ they follow this particular formula and here if I am assuming to 2 level realization of the circuit; then that there is one set of AND gates that will be realizing this $g_{i+2} p_{i+3}$ similarly $g_{i+1} p_{i+2} p_{i+3}$. So, this way it will be realizing the AND terms and after that there will be an OR term. So, there will be 2 gate delay in the generate $g_{[i..i+3]}$ and $p_{[i..i+3]}$.

(Refer Slide Time: 03:21)

Signals computed	Formulas	Delay
c_4, c_8, c_{12} $g_{[0..15]}, p_{[0..15]}$	$c_4 = g_{[0..3]} + c_0 p_{[0..3]}$ $c_8 = g_{[4..7]} + g_{[0..3]} p_{[4..7]} + c_0 p_{[0..3]} p_{[4..7]}$ $c_{12} = g_{[8..11]} + g_{[4..7]} p_{[8..11]} + g_{[0..3]} p_{[4..7]} p_{[8..11]} + c_0 p_{[0..3]} p_{[4..7]} p_{[8..11]}$ $g_{[0..15]} = g_{[12..15]} + g_{[8..11]} p_{[12..15]} + g_{[4..7]} p_{[8..11]} p_{[12..15]}$ $+ g_{[0..3]} p_{[4..7]} p_{[8..11]} p_{[12..15]}$ $p_{[0..15]} = p_{[0..3]} p_{[4..7]} p_{[8..11]} p_{[12..15]}$	2 gate delays

So, for this c_4 , c_8 and c_{12} ; so we will have 2 gate delays for g_0 to $g_{0..15}$ and $p_{0..15}$ that will also have 2 gate delays.

(Refer Slide Time: 03:33)

Signals computed	Formulas	Delay
$c_{i+1}, c_{i+2}, c_{i+3}$ $i = 4, 8, 12$	$c_{i+3} = g_{i+2} + g_{i+1} p_{i+2} + g_i p_{i+1} p_{i+2} + c_i p_i p_{i+1} p_{i+2}$ $c_{i+2} = g_{i+1} + g_i p_{i+1} + c_i p_i p_{i+1}$ $c_{i+1} = g_i + c_i p_i$	2 gate delays

Now for this c_{i+1} , c_{i+2} , c_{i+3} for i equal to 4, 8 and 12; they will see 2 gate delays; so, because of this expression if you looking into this expression; so you will see how many gate delays are necessary.

(Refer Slide Time: 03:47)

The slide has a yellow header bar with the title "16-bit 2-level Carry Lookahead Adder". Below the header, there is a table with three columns: "Signals computed", "Formulas", and "Delay".

Signals computed	Formulas	Delay
$s_{i+1}, s_{i+2}, s_{i+3}$ $i = 4, 8, 12$ <i>i.e.,</i> $s_5, s_6, s_7, s_9, s_{10}, s_{11}, s_{13}, s_{14}, s_{15}$	$s_i = p_i \oplus c_i$	1 gate delay

A horizontal line separates the table from the following text:

Total: 8 gate levels in the CLA adder vs.
32 gate levels in the ripple carry adder

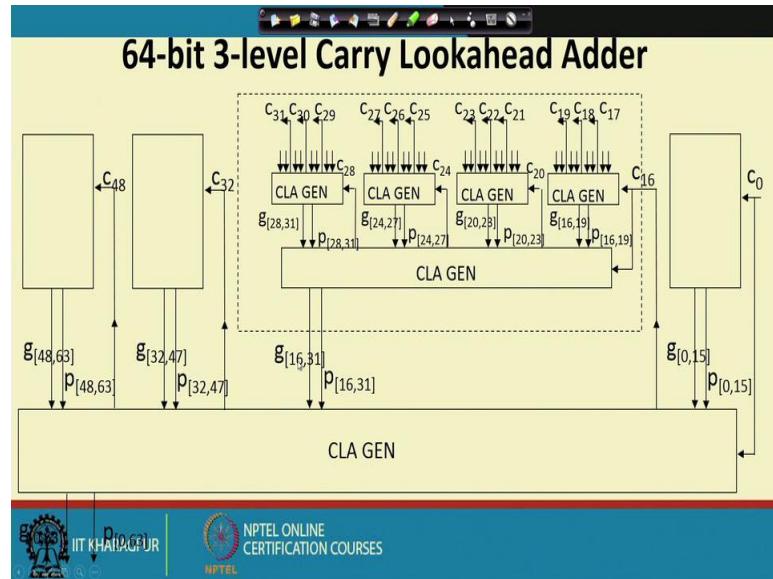
The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player showing a speaker.

So, this way it can total 8 gate levels in this carry look ahead adders versus 32 gate levels in the ripple carry adders. So, if I am if I am had having a 16 bit adder then there will be 32 gate level in the ripple carry adder because this individual stage it will have that sum and carry. So, that will be the carry calculation will have 2 stages ok. So, that way it will have total 32 stage, but this if you are using this carry look ahead mechanism; so, that will have only 8 stage.

That is why this carry look ahead adder; so they are going to be much faster than this ripple carry adder and in many application; so where we need fast addition, subtraction. So, we go for this carry look ahead adder instead of ripple carry.

Of course the penalty that we pay is the number of gates that are needed becomes very high. So, this is the thing that I was talking about.

(Refer Slide Time: 04:45)



So, this is suppose I have called this 16 bit carry look ahead blocks ok. Now from there; so, I can I can have some 32 bit block and from there I can have some 64 bit block. So, here it is 64 bit block that is generated that is shown here in terms of this cascading of this clock generator circuit.

(Refer Slide Time: 05:00)

64-bit 3-level Carry Lookahead Adder		
Level	Signals computed	Delay
PRE	$g_i, p_i \quad i=0..63$	1 gate delay
1	$g_{[i..i+3]}, p_{[i..i+3]} \quad i=0, 4, 8, 12, \dots, 56, 60$	2 gate delays
2	$g_{[i..i+15]}, p_{[i..i+15]} \quad i=0, 16, 32, 48$	2 gate delays
3	$g_{[0..63]}, p_{[0..63]}, c_{16}, c_{32}, c_{48}$	2 gate delays
2	$c_{20}, c_{24}, c_{28}, c_{36}, c_{40}, c_{44}, c_{52}, c_{56}, c_{60}$	2 gate delays
1	$c_{21}, c_{22}, c_{23}, c_{25}, c_{26}, c_{27}, \dots, c_{61}, c_{62}, c_{63}$	2 gate delays
POST	$s_{21}, s_{22}, s_{23}, s_{25}, s_{26}, s_{27}, \dots, s_{61}, s_{62}, s_{63}$	1 gate delay

And these are different gate delays calculation that we have; so it is similar to the previous calculation.

(Refer Slide Time: 05:06)

Delay of a k-bit Carry-Lookahead Adder

$$T_{\text{lookahead-adder}} = 4 \lceil \log_4 k \rceil$$

k	$T_{\text{lookahead-adder}}$	$T_{\text{ripple-carry-adder}}$
4	4	8
16	8	32
32	12	64
64	12	128
128	16	256
256	16	512

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, in that in summary you can say that a look ahead adder; so, now total it has a carry bit k bit carry look ahead then the total look ahead adder delay is $4 \log_4 k$. So, that gives the advantage like k equal to say if I have got the adder size has a 32, then look ahead adder delay will be 12 stages and this ripple carry adder will have 64 stages. So, that ways for 256 bit add addition; so, look ahead adder will have got a delay of 16 stages and carry look ahead ripple carry will have 512 stages; so that is pretty high.

(Refer Slide Time: 05:46)

Decimal Adders

8421 weighted coding scheme or BCD Code

Decimal Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Forbidden codes: 1010, 1011, 1100, 1101, 1110, 1111

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next will be looking into decimal adders; so, decimal adder means that I have got the binary coded decimal digits BCD and then I want to add them. So, in case here; so here I have got the digits important for 0 to 9 only; so rest are not important. So, they are coded like this; so, 9 is coded as 1001 and rest of that 4-bit combinations are not valid for the BCD addition; so, these are the forbidden codes 1010, 1011 and all.

(Refer Slide Time: 06:19)

Decimal Adder

- Inputs: $A_3A_2A_1A_0$, $B_3B_2B_1B_0$, C_{in} from previous decade.
- Output: C_{out} (carry to next decade), $Z_3Z_2Z_1Z_0$.
- Idea: Perform regular binary addition and then apply a corrective procedure.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for decimal adder; so, we have got the inputs A_3 , A_2 , A_1 , A_0 , B_3 , B_2 , B_1 , B_0 and C_{in} from the previous cascade state. And output was C_{out} and Z_3 , Z_2 , Z_1 , Z_0 and so what we do for decimal adder is that we perform regular binary addition and then we do some corrective procedure.

(Refer Slide Time: 06:40)

Decimal Sum	K	P ₃	P ₂	P ₁	P ₀	C _{out} ^{Same}	Z ₃	Z ₂	Z ₁	Z ₀
0-9										
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1

So, for example here we have got this number 0 to 9 for this part it is same ok. So, actually this likely shifted; so, this is for 0 to 9 if the decimal sum is equal to 0 to 9 then it is same; however, if the decimal sum is 10; 1010. So, it will be represented in the BCD forms; so they will be they will be represented has carry bit being 1 and the Z₃, Z₂, Z₁, Z₀ being 0.

Similarly, for 11 if the decimal sum is 11 then the C_{out} will be equal to 1 and the Z₀ will be equal to one and the rest of them will remain as 0, 12 will be like that. So, basically this C_{out} bit will be set to 1 whenever the sum becomes greater than 9 for the BCD addition ok. So, this way I can; so, C_{out} is 0 for this part and for other part; so, it will be equal to 1.

(Refer Slide Time: 07:36)

The slide has a yellow background with a black header bar containing icons. The title 'Decimal Adder' is centered in a large, bold, black font. Below the title is a bulleted list of points:

- No correction needed when the decimal sum is between 0-9.
- Must apply a correction when the sum is between 10-19.
- Case 1:
 - 16-19: K is set to 1. Add binary quantity 0110 to $P_3P_2P_1P_0$.
 - 10-15: $KP_3P_2P_1P_0$ are set to 01010, 01011, . . . , 01111. Need to add 6. Use a K-map to obtain a Boolean expression to detect these six binary combinations.

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

So, no correction is needed when the decimal sum is between 0 and 9 and we must apply a correction, when the sum is between 0 and 19. So, at most we can add two 9 and 9; so that is at most the value can be 18. So, this one a value the between 0 and 10 and 19; so, you have to do some correction.

(Refer Slide Time: 08:00)

The slide has a yellow background with a black header bar containing icons. The title 'Rules of BCD adder' is centered in a large, bold, black font. Below the title is a bulleted list of points:

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8 , we specify further that either Z_4 or Z_2 must have a 1.

Below the list, the equation $C = K + Z_8Z_4 + Z_8Z_2$ is displayed.

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

So, this; so, these are the rules for BCD addition. So, when the binary sum is greater than 1 0 0 1 we obtain a non valid BCD representation. And the binary the addition of binary 6 to the binary sum; it will convert the number to the correct BCD form.

So, whenever it becomes more than 1001; so, you just add this 6 to the binary to the, to whatever sum you have got and that gives the corrective action. So, to distinguish between 1000; that is binary 1001 and 1001; so, which will also have one in position Z_8 . So, we specify that either Z_4 and Z_2 must have a value 1. So, $C = K + Z_8Z_4 + Z_8Z_2$.

(Refer Slide Time: 08:46)

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

Block Diagram of a BCD Adder

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

So, this is the BCD sum part; so, this is 0100; so, this will be added if this Z_8 , so, that if $Z_8 Z_4$.

(Refer Slide Time: 09:00)

Rules of BCD adder

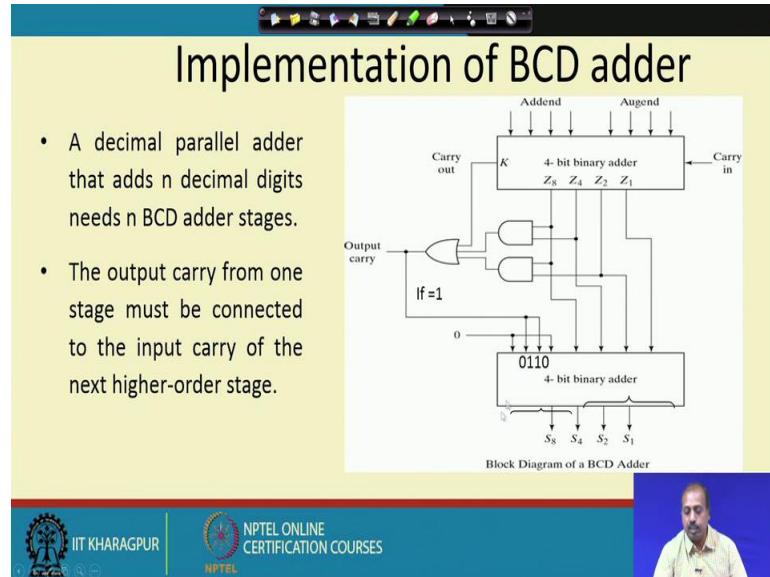
- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8 , we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

So, if you look into this expression is $K + Z_8Z_4 + Z_8Z_2$. So, if this is situation then only the carry is generated.

(Refer Slide Time: 09:07)



So, what we are doing? For with this 4-bit addition result; so, you are trying to add this 6, but that is conditional, conditional subject to the case that there was a carry or this Z_8Z_4 was equal to 1 or Z_8Z_2 was equal to 1.

So, if you draw the corresponding truth table then will find that under these conditions; so, we need to do the correction of adding 6 to the binary addition result to get the BCD addition. So, a decimal parallel adder that adds n decimal digits needs a n bit; n BCD adder stages. And the output carry from one stage must be connected to the input carry of the next higher stage; so, that is obvious.

So, if you need more number of decimal digits to be added. So, then this entire block has to be repeated one after the other to get the BCD addition for higher number of digits.

(Refer Slide Time: 10:00)



(Refer Slide Time: 10:08)

A presentation slide with a yellow background. At the top, there is a toolbar with various icons. In the center, the title 'Two bit Multiplication' is displayed in a large, black, sans-serif font. Below the title, there are two tables. The first table is labeled 'a b | a b' and shows the results of a AND operation for all combinations of a and b (00, 01, 10, 11). The second table is labeled 'a b | a x b' and shows the results of a binary multiplication for all combinations of a and b. At the bottom, there is a blue footer bar. On the left side of the footer, there is the logo of IIT Kharagpur and the text 'IIT KHARAGPUR'. In the center of the footer, there is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man speaking.

a	b	a b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a x b
0	0	0
0	1	0
1	0	0
1	1	1

Next we look into the multiplier; now you see that if I have got 2 bits a and b has input then the multiplication is like this a b a and b it is 0 1 0 0 1. So, if you compare with the AND operation you see that you are getting the same thing. So, this AND is some sort of 2 bit binary multiplication; so, this 2 bit binary multiplication is same has the AND of the 2 bits.

(Refer Slide Time: 10:34)

Three bit Multiplication

- Partial products are: 101×1 , 101×1 , and 101×0
- Note that the partial product summation for n digit, base 2 numbers requires adding up to n digits (with carries) in a column.
- Note also $n \times m$ digit generates up to an $m + n$ digit result (same as decimal).

$$\begin{array}{r} 1 & 0 & 1 \\ \times & 0 & 1 & 1 \\ \hline 1 & 0 & 1 \\ 1 & 0 & 1 \\ \hline 0 & 0 & 0 \end{array}$$

For 3 bit multiplication; so suppose; so we have will generate this partial product terms, so 1 0 1 1 0 1; so, this 1 0 1 multiplied by 1. So, that generates this partial term 1 0 1 then this 1 0 1 multiplied by the second one that generates this partial sum and then multiplied by 0 that generates this partial sum and then we have then we add all this partial sum to get the result.

So, we have the partial products and this partial product summation for n digit base 2 numbers requires adding up the up to n digits in a column. So, that way we are going to add these digits. So, that if I am multiplying and n by one n bit number by an m bit number, then we have to generate it will generate and $m + n$ digit number ok.

(Refer Slide Time: 11:25)

Multiplier Boolean Equations

- We can also make an $n \times m$ "block" multiplier and use that to form partial products.
- Example: 2×2 – The logic equations for each partial-product binary digit are shown below:
- We need to "add" the columns to get the product bits P_0 , P_1 , P_2 , and P_3 .
- Note that some columns may generate carries.

$$\begin{array}{r}
 & b_1 & b_0 \\
 & \times & a_1 & a_0 \\
 & & \hline
 & (a_0 \cdot b_1) & (a_0 \cdot b_0) \\
 + & (a_1 \cdot b_1) & (a_1 \cdot b_0) & \hline
 P_3 & P_2 & P_1 & P_0
 \end{array}$$





So, that is they of they are from multiplication. So, Boolean equation for multiplication; so, it will be like this. So, if I am taking as a 2 bit multiplication for example, then this generates a_0b_0 and a_0b_1 as the this partial sum. And then this a_1b_0 and a_1b_1 as another partial sum then this partial sums are added the P_0 , P_1 , P_2 , P_3 that generates the product terms ok.

(Refer Slide Time: 11:52)

A 2x2 binary multiplier

- The AND gates produce the partial products.
- For a 2-bit by 2-bit multiplier, we can just use two half adders to sum the partial products. In general, though, we'll need full adders.
- Here C_3-C_0 are the product, not carries!

$$\begin{array}{r}
 & b_1 & b_0 \\
 & \times & a_1 & a_0 \\
 & \hline
 & a_0b_1 & a_0b_0 \\
 + & a_1b_1 & a_1b_0 & \hline
 c_3 & c_2 & c_1 & c_0
 \end{array}$$



So, this is the circuitry for that; so, this generates A_0B_0 ; then this generates A_0B_1 , A_1B_0 and A_1B_1 and then this A_0B_0 directly gives; it is not added with anything. So, that that

come directly comes at the multiplication the result output C_0 ; then this A_0B_1 and A_1B_0 are to be added.

So, I can use a half adder here because there is no carry. So, I can just use a half adder to do that and then from this some carry may be generated and that carry has to be added with A_1B_1 . So, this A_1B_1 and that carry comes and that will generate the sum C_2 and the carry C_3 .

So, this way I can use this circuitry to get a 2 by 2 binary multiplier.

(Refer Slide Time: 12:43)

Multiplication: a special case

- In decimal, an easy way to multiply by 10 is to shift all the digits to the left, and tack a 0 to the right end.
 $128 \times 10 = 1280$
- We can do the same thing in binary. Shifting left is equivalent to multiplying by 2:
 $11 \times 10 = 110$ (in decimal, $3 \times 2 = 6$)
- Shifting left twice is equivalent to multiplying by 4:
 $11 \times 100 = 1100$ (in decimal, $3 \times 4 = 12$)
- As an aside, shifting to the *right* is equivalent to *dividing* by 2.
 $110 \div 10 = 11$ (in decimal, $6 \div 2 = 3$)

Now, some special cases; so, in decimal an easy way to multiply by 10 is to shift all digits to the left and put a 0 to the right end. So, 1 to 8 multiplied by 10; so, we know the result is 128 and a 0. So, you shift the numbers digits by to the left and put a 0 at the right end. So, for the same thing we can do with binary one.

So, so we can shift left is equivalent to multiplying by 2. So, a shifting left is multiplying by 2; so, for example, this 1 1 multiplied by 1 0 is in decimal system it is 3×2 which is equal to 6. So, if you converted into binary notation; so, it is 110.

So, what has happened is that this 11; so, it has been left shifted and this less significant bit position we have put a 0 ok. So, that gives the binary shifting or this multiplication by 2 and if you shift it twice; shift left twice then it is a multiplication by 4. So, for example,

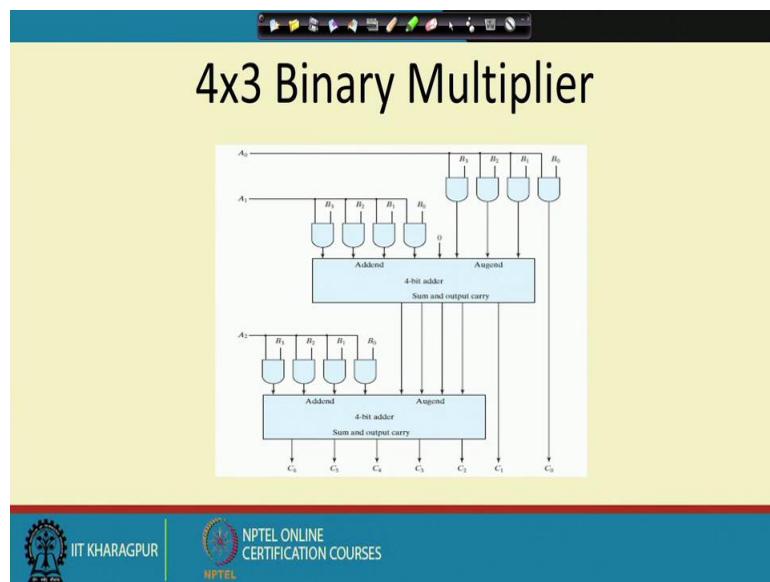
this 11; so, if I left shift twice then I will get 1100 and this number is nothing, but 12. So, this 3×4 is 12; so, this is basically left shifting by 2 bits.

So, shifting to the right is equivalent to dividing by 2. So, if you are; so for example, this 110 that is 6. So, if you are shifting it by right position; so, you are shifting it right means this 0 will go. So, 11 will remain; so, that is 11 the result is 3 this is nothing, but 6 divided by 2 is 3.

So, this multiplication and multiplication by 2; so, this is basically left shifting the number by one position, multiplication by 2^i is left shifting the number by i positions and similarly division by 2 is right shifting the number by one position and division by 2^i is right shifting the number by i positions.

So, this way we can have the special cases for multiplication by multiplication and division by powers of 2.

(Refer Slide Time: 14:50)



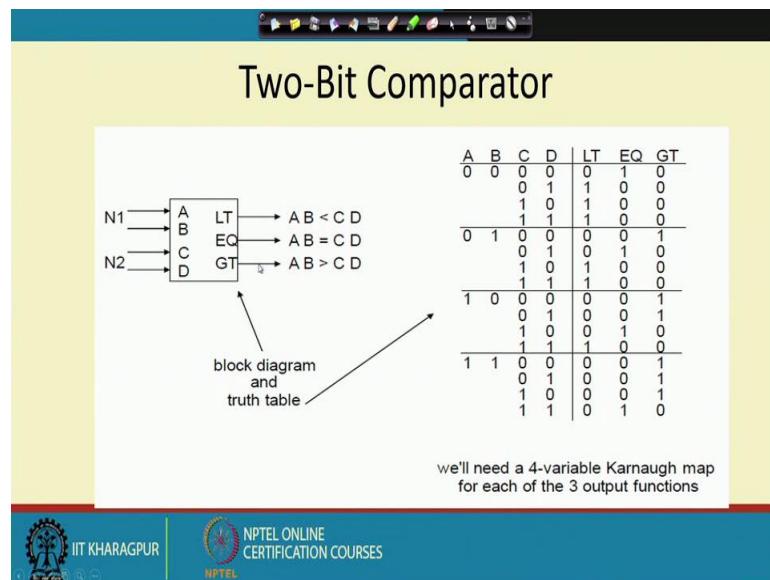
So, for 4-bit multiplication; so 4 by 3 binary multiplier; so, we have got one 4-bit input B_0, B_1, B_2, B_3 and we have got 3 bit input A_0, A_1, A_2 . So, what we can do? So, we can generate these partial sums in this fashion ok. So, then they can be passed through some adders ok; so, two 4-bit adders will be necessary and we can pass them through the adder and ultimately we can get this sum which is consisting of 7 bits. So, $8 C_0$ to C_6 plus there will

be an another carry; so, $4 + 3 = 7$, the result may be 8 bit also. So, that way there will be another carry generated ok.

So, this way we can have this partial sums generated and we can do the multiplication. Of course, this multiplication process that we have talked about is pretty slow and in case of integrated circuit chips particularly in VLSI domain. So, even find many other efficient multiplication algorithm that have come up. So, this is the basic multiplication algorithm that we have discussed and the idea is to see how can we do it using logic gates and all.

Next we will be looking into another very important arithmetic module which is known as magnitude comparator. So, I have got 2 inputs I need to tell whether the one of them is equal to the other greater than the other or less than the other.

(Refer Slide Time: 16:21)



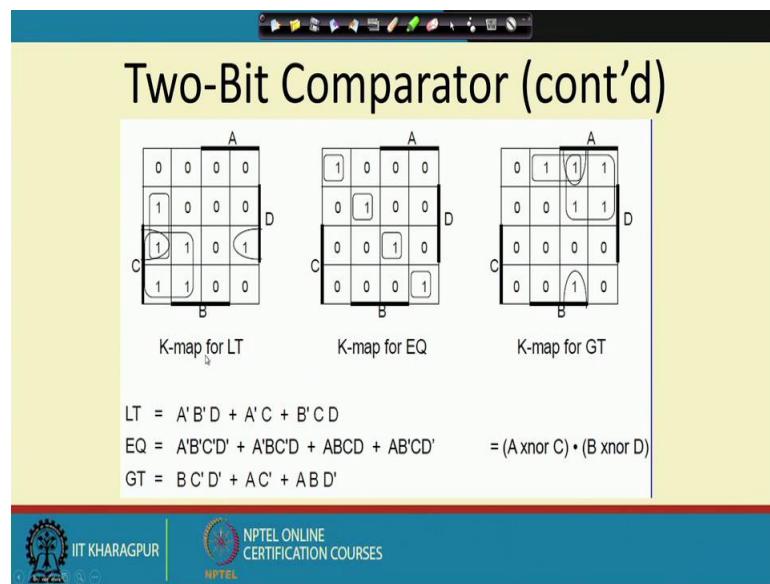
So, if I have got a two-bit value; so, it is like this. So, see suppose AB is the first number and CD is the second number. So, what is what is happening is that this AB is AB is 00; then if CD is 0 then it less than is 0, equal is 1 and greater than is 0 that is these 2 numbers are equal. So, this EQ output is 1 and this less than and greater than these 2 outputs are 0.

Similarly, if AB is 00 and CD is 01; so, this less than output is 1 equal output is 0 and this greater than output is 0. Then this is 1 0 again the same thing; so, this is this AB is 0 0. So, whatever be the CD apart from 0 0 this less than will be 1, others will be 0. So, this way I

can draw a truth table for I can make a truth table for writing the entire behavior of this block of this, this comparator block.

So, for doing the minimization; so, we will need 4 variable Karnaugh map for each of this 3 output functions ok. So, it 4 variable Karnaugh map and accordingly you can do the minimization, get the circuitry and ultimately we can say it is less than equal to and greater than these 3 outputs we can see we can I will draw the corresponding logic circuit.

(Refer Slide Time: 17:44)



So, this is the thing; so K map for less than is $\bar{A}\bar{B}D + \bar{A}C + \bar{B}$, $EQ = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + ABCD + A\bar{B}C\bar{D}$ that is either A and B actually the logic is that A and B both are 0 C and B should also be 0. A is 0, B is 1; so C should be 0 and D should be 1. So, this way I can do it.

So, this equality; so if you look it more carefully; so, we can find that this can also be written as $(A \text{ XNOR } C) \cdot (B \text{ XNOR } D)$ ok; that way it can be there.

(Refer Slide Time: 18:22)

Equality Comparator


 $Z = X \text{ XNOR } Y$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1



IIT KHARAGPUR



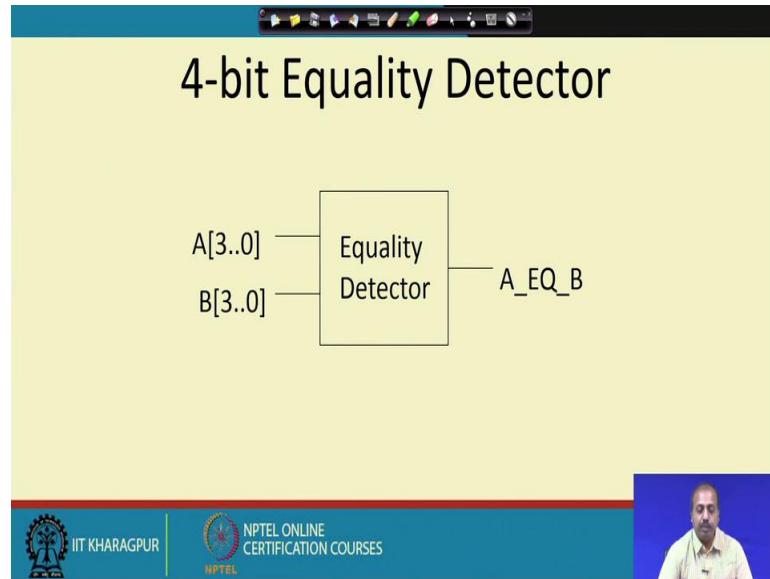
NPTEL ONLINE
CERTIFICATION COURSES



So, this is the; so, So, that way we can draw the corresponding truth table and from there I can get the logic functions realized in terms of basic logic gates. Now let us look into to this equality comparator more carefully; suppose I have got only 2 bits to be compared and I have say whether the bits are equal or not. So, in terms of truth table; so, if it is 0 0 then the output is 1, if it is 0 1; it is 0 1 0 is 0 and 1 1 is also 1, for these 2 cases it is 1.

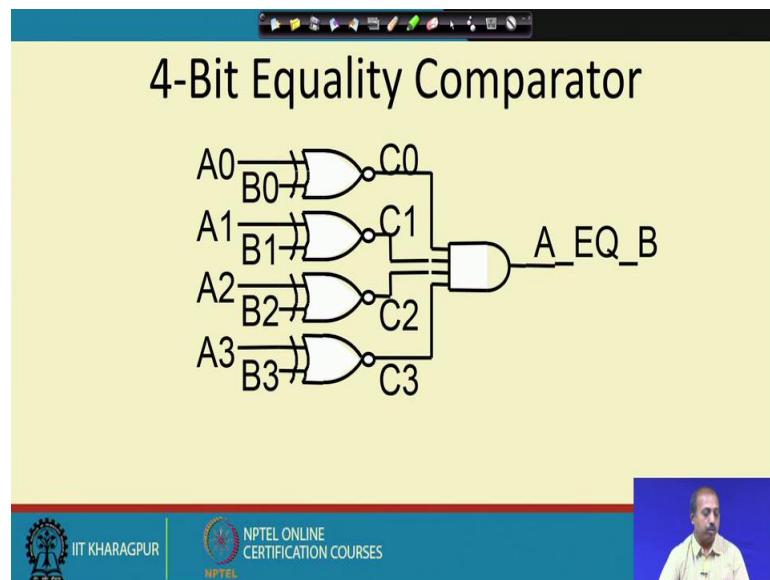
Now, you see the this truth table correspond to simple XNOR logic. So, it is a 2 input XNOR gate; so, this is what is happening here. So, Z is equal to X XNOR Y; so, this XNOR gate is an equality comparator.

(Refer Slide Time: 19:20)



So, if I have if I need a 4-bit equality comparator equality detector; then this may be the block diagram of that. So, I have got this A 4-bit input A_3, A_2, A_1, A_0 and similarly B_3, B_2, B_1, B_0 . So, this is the this is the B 4-bit B input and it answers A equal to B. So, this output is 1 only when this A and B input this A and B input are equal to each other.

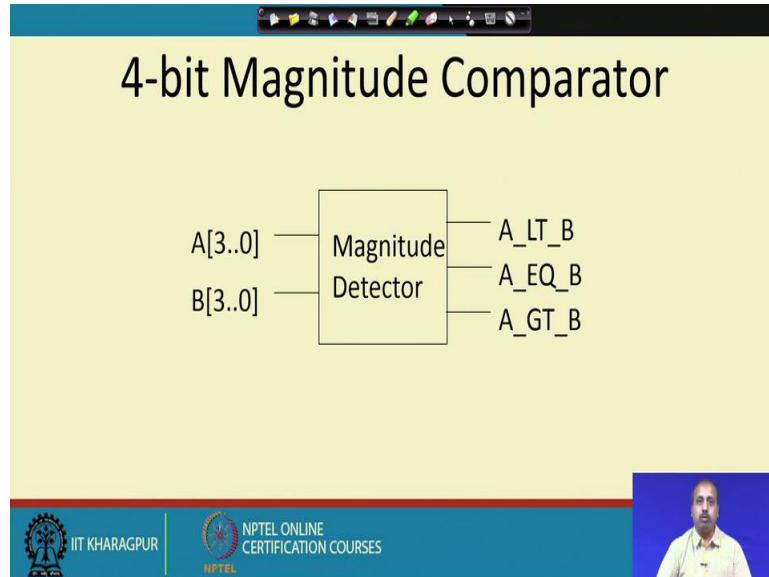
(Refer Slide Time: 19:37)



So, I can take a very simple circuit consisting of 4 XNOR gates this A_0 and B_0 fed to the first XNOR A_1, B_1 fed to the next XNOR like that. And now this C_0, C_1, C_2, C_3 ; so, they

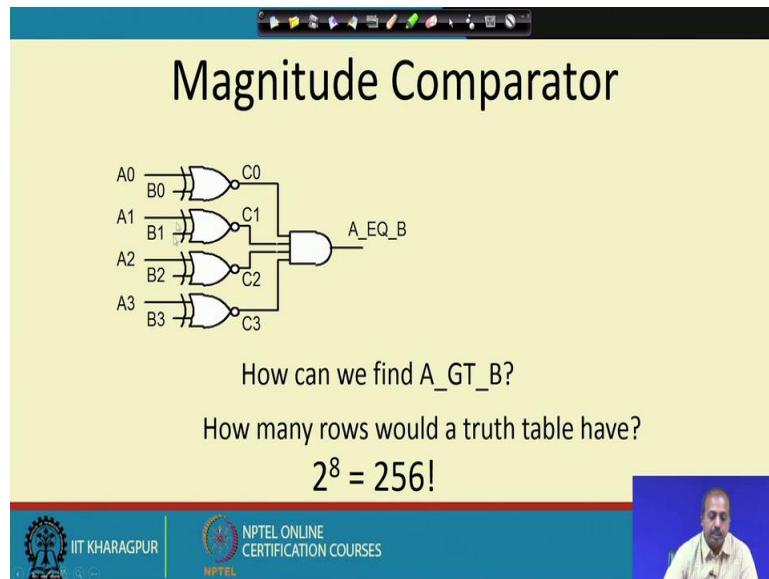
are; they are to be ANDed. So, this AND gate is not; so, they are to be ANDed to get A equal to B ok; so, that can be a 4-bit equality comparator.

(Refer Slide Time: 20:00)



But what about magnitude comparison like; so not only equality, I also want less than and greater than output. So, the A is less than B or A is greater than B.

(Refer Slide Time: 20:15)

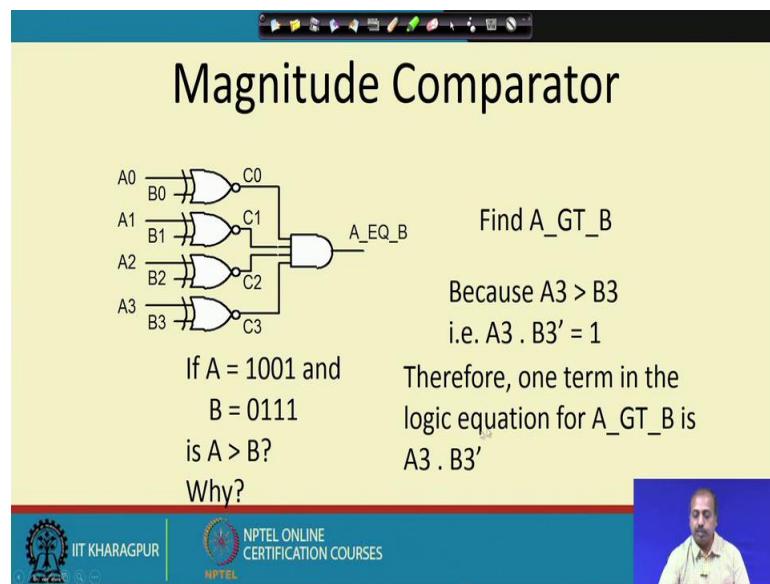


So, equality is definitely fine; so how can we get A greater than B. So, how many rows in the truth table have? So, if you try to answer this question then since there are 8 inputs A

0 to; since we have got 8 inputs A_0 to A_3 and B_0 to B_3 ; total number 8 input say if I am trying to draw a truth table; so, there will be 256 entries.

So, if I am trying to draw a Karnaugh map; then it is an 8 variable Karnaugh map that we have to minimize. So, that is a very clumsy way that is way very; so for in our course we have seen that we can draw K map up to 6 variables not more than that. So, that makes it difficult; so we have to do some something else for getting the minimized form.

(Refer Slide Time: 21:00)



So, if you see that if A equal to 1001 and B equal to 0111 then A is greater than B; why? This is because this most significant bit A_3 is equal to 1 here and B_3 is 0 here; so A is greater than B. Now because A_3 greater than B; so the condition is $A_3B_3' = 1$.

(Refer Slide Time: 21:36)

Magnitude Comparator

A_GT_B = A3 . B3' +
Because A3 = B3 and
A2 > B2
i.e. C3 = 1 and
A2 . B2' = 1
Therefore, the next term in the
logic equation for A_GT_B is C3 . A2 . B2'

If A = 1101 and
B = 1011
is A > B?
Why?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, one term in the logic equation for A greater than B is $A_3 B_3'$. Now what about this one? Say if I have got A equal to 1101 and B as 1011. So, here also A is greater than B because the first 2 bits A_3 and B_3 were same and A_2 was 1 and B_2 was 0. So, I can say that $A_3 = B_3$ and $A_2 > B_2$ that is why I have got $C_3 = 1$. So, $C_3 = 1$ if $A_2 B_2' = 1$ and $A_3 = B_3$.

So, we have got ah; so, so C_3 was A_3 equal to B_3 ok, so, the conditions becomes $C_3 A_2 B_2'$.

(Refer Slide Time: 22:22)

Magnitude Comparator

A_GT_B = A3 . B3' + C3 . A2 . B2'
+
Because A3 = B3 and
A2 = B2 and
A1 > B1
i.e. C3 = 1 and C2 = 1 and
A1 . B1' = 1
Therefore, the next term in the
logic equation for A_GT_B is C3 . C2 . A1 . B1'

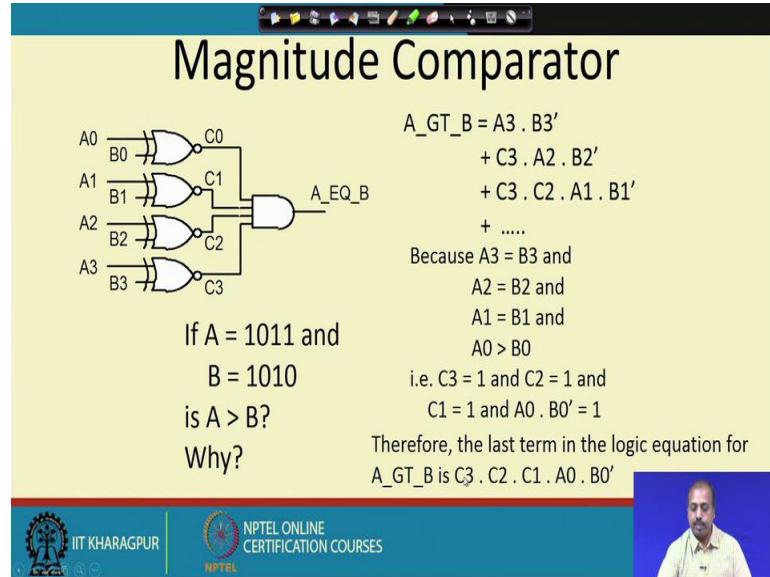
If A = 1010 and
B = 1001
is A > B?
Why?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

That way if we just shift by one more position; so, you will get the condition one more condition there. So, that will be this is the thing $A_3 = B_3$, $A_2 = B_2$ and $A_1 > B_1$. So, we have got C_3 equal to 1, C_2 equal to 1 and $A_1 B_1' = 1$.

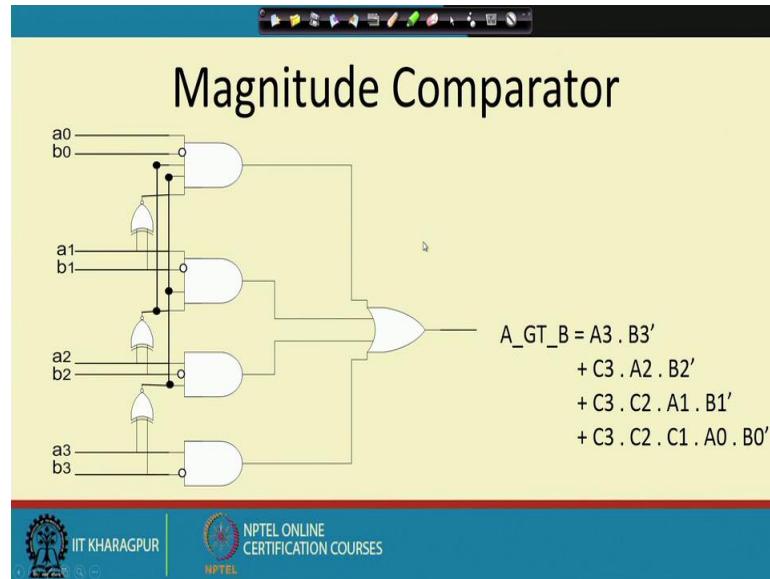
So, we see that the A_3 greater than B terms becomes $C_3 C_2 A_1 B_1'$.

(Refer Slide Time: 22:51)



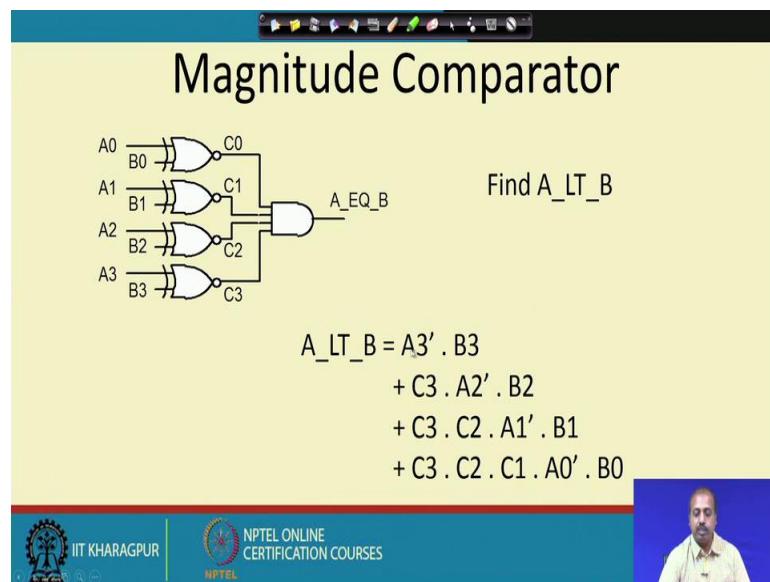
And naturally, I will have one more stage where it will be doing this B_0 also like only the B ; B_0 A_0 and B_0 bits are differing. So, if ah; so, this is the situation the $A_0 B_0'$ is equal to 1, C_3 equal to 1, C_2 equal to 1 and C_1 equal to 1. So, I will get the last term as $C_3 C_2 C_1 A_0 B_0'$.

(Refer Slide Time: 23:15)



So, the final expression becomes like this; so, A greater than B is $A_3B_3' + C_3A_2B_2' + C_3C_2A_1B_1' + C_3C_2C_1A_0B_0'$. And this can be realized by this circuit you see that we have got a very clean circuit here and we do not take help of the Karnaugh map and all. So, sometimes that makes that process makes it very difficult and we have to use some other logic to come to the circuit.

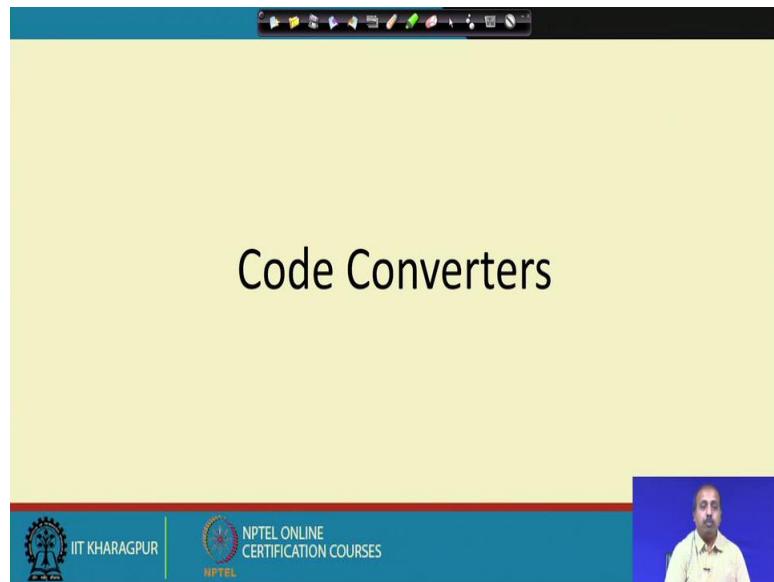
(Refer Slide Time: 23:54)



So, similarly you can say A less than B is given by this formula A'_3B_3 . So, A_3 is 0 and B_3 is 1 or A_3 B_3 are same detected by this condition $C_3A'_2B_2$ is 0 and B_2 is 1 or $C_3 C_2$ that is

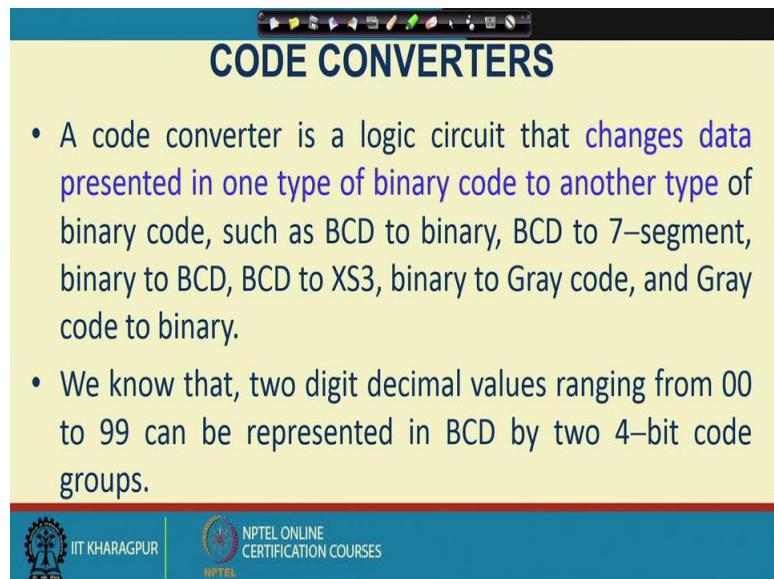
A_3, B_3 are same A_2, B_2 are same and A_1 is 0 and B_1 is 1 and this condition A_3, B_3 same A_2, B_2 same $A_1 B_1$ same and A_0 is 0 and B_0 is 1. So, that way I can write down the condition for A less than B.

(Refer Slide Time: 24:24)



Next, we will be looking into another important topic which is known as code converters. So, many a times we want to convert one, numbers coded in some number system into some other number system. So, that is that is known as code converters; so, they coded in different form.

(Refer Slide Time: 24:45)



So, this is our code converter it is a logic circuit that changes data represented in one type of binary code to another type of binary code. For example, BCD code to binary code, binary code to BCD code BCD to 7 segment binary BCD to 7 segment display code, then binary to BCD; BCD to XS 3 code, so like that binary to gray code gray to binary code.

So, there are different types of codes that are there and many a times we need to interchange between this type this type of code. So, ok; so, how to do this code conversions? That we will see now; so, 2 digit, 2 digit decimal values ranging from 0 0 to 99 can be represented by BCD, BCD by two 4-bit code groups; so, that can be utilized for doing this conversion.

(Refer Slide Time: 25:36)

The screenshot shows a presentation slide with a yellow background. At the top, the title 'BCD-to-Binary Conversion' is displayed in bold. Below the title, a text block states: 'One method of BCD-to-Binary code conversion uses adder circuits :'. A numbered list follows: 1. The value, or weight, of each bit in the BCD number is represented by a binary number 2. All of the binary representations of the weights of bits that are 1s in the BCD number are added 3. The result of this addition is the binary equivalent of the BCD number. At the bottom of the slide, there is a footer bar with three sections: 'IIT KHARAGPUR' (with a logo), 'NPTEL ONLINE CERTIFICATION COURSES' (with a logo), and a video thumbnail of a man speaking.

So; first will be looking into BCD to binary conversion, so one method for BCD to binary conversion is by some adder circuit. So, value or weight of each bit in the BCD number is represented by a binary number and all of the binary representations of the weights of bits that are 1 in the BCD number; they will get added. So, this way we can have some adder circuitry and using that adder we can convert the BCD number to binary number.

(Refer Slide Time: 26:08)

Contd...

For example, 46_{10} is represented as

4	6	Decimal
0100	0110	BCD

- The MSB has a weight of 10, and the LSB has a weight of 1.
- So the most significant 4-bit group represents 40, and the least significant 4-bit group represents 6 as in Table.

IIT Kharagpur | NPTEL Online Certification Courses | NPTEL

So, it is like this. So, 46; so 46 this is the 4 is this is the notation and 6 this is the thing. So, this is the decimal number and this is the BCD number. Now this most significant bit it has got a weight of 10 and the least significant bit has a weight of 1.

So, the most significant 4-bit group represents 40 and the least significant 4-bit represent group is 6 ok. So, because this is this weight is 10 and this weight is 1. So, this is $4 \times 10 + 6$; so that we can do.

(Refer Slide Time: 26:44)

	Bit position							
Decimal weight	$10^1 = 10$				$10^0 = 1$			
Binary weight	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
BCD bit Weight	$2^3 \times 10$	$2^2 \times 10$	$2^1 \times 10$	$2^0 \times 10$	$2^3 \times 1$	$2^2 \times 1$	$2^1 \times 1$	$2^0 \times 1$
	80	40	20	10	8	4	2	1
Bit designation	h	g	f	e	d	c	b	a
46_{10}	0	1	0	0	0	1	1	0
		40				4	2	

IIT Kharagpur | NPTEL Online Certification Courses | NPTEL

So, we can say that this bit position; so, this is decimal weight is bit position 1 and bit position 0. So 10^0 and $10^1 \times 10$ so, if the binary weight of that. So, if you look into that number at weight 10 and look into the corresponding. If you look into the corresponding binary number binary weights, this is $2^0, 2^1, 2^2$ and 2^3 . So, the BCD number; the BCD bit weight is $2^3 \times 10$, this is $2^2 \times 10$, this is $2^1 \times 10$ and $2^0 \times 10$.

Now, so for example, this 46; so 46 it has got 0100. So, this 0 gives a contribution of 0 then this; so, this 4 this 4 is represented by the first 4 bits and the 6 is represented by the next 4 bits. And for the first 4 bits they have got a weight of 10 now this numbers; so, now this 0 multiplied by 10 that gives me 0; 1 1 is. So, this is 2^2 ; so, that is 4 multiplied by 10 that gives me 40, then this is 2 multiplied by 0 2^1 into that value is 0; so that is that is giving me 0.

So, that way; so this part gives me 10 and this part gives me 6.

(Refer Slide Time: 28:11)

BCD bit position	BCD bit weight	Binary representation
a	1	1
b	2	10
c	4	100
d	8	1000
e	10	1010
f	20	10100
g	40	101000
h	80	1010000

So, so binary equivalent of each BCD bit will be is a binary number and it is represented as a BCD weight bit. So, this is BCD position a, b, c, d, e, f, g, h as we have seen here this, this bit designation a, b, c, d, e, f, g, h. So, they have got this BCD weight values like this and corresponding binary representations like this ok.

(Refer Slide Time: 28:39)

The result from the addition of the binary representation for the weights of all the 1s in the BCD number is the binary number that corresponds to the BCD number.

4 6

0 1 0 0	0 1 1 0		0 0 0 0 0 1 0	2
			0 0 0 0 0 1 0 0	4
		+ 0 0 1 0 1 0 0 0	4 0	
		0 0 1 0 1 1 1 0	4 6	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window shows a man speaking.

So, this way we can do this binary to BCD conversion like the result from this addition of binary representation of the weights of all the ones in the BCD number. So, that will give me the corresponding binary numbers.

So, this is; so 0 4 is 0 1 0 0; so that is represented if you take the corresponding weight and all; so, value was 40; so that is giving you. Similarly this was these 2 these bit is 1 and this bit is 1 and for these bit; I know that the corresponding weight is 4 ok. So, this 4 is added and for these bit I know the corresponding weight is 2; so, this 2 is added. So, these gives me this number; so after doing this all this additions. So, it gives me 46 in the binary number.

(Refer Slide Time: 29:20)

Example :
Convert the BCD equivalent of 26 to binary.

Solution

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, similarly suppose I have to convert this 26 into binary. So, again we do we do the same thing that; so, this part is all of them they will contribute to 10 and this this is $2^0 = 1$, this is $2^1 = 2$, $2^2 = 4$ and this is 8.

So, this is this is 2; so, 2×10 ; so that gives me 20. So, I write down the binary representation for 20 then in thus in this part. So, this part is this weight is 10^0 for this all the 4 bits the weight is 10^0 . So, this is this bit is 1, 2; so 2 into 10 power 0 that is 1; so that that gives me 2; so, I write down the binary representation for 2. Similarly, here I write down the binary representation for 4 and I sum them up; so, this gives me the binary 26.

So, in this way we can convert BCD numbers to equivalent binary numbers; of course, we have got other avenues. So, we can have a truth table and write down the corresponding Boolean functions for doing this conversion also; as well.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 22
Arithmetic Circuits (Contd.)

So, we were discussing on this code conversion techniques.

(Refer Slide Time: 00:23)

INPUT (BINARY)				Decred	OUTPUTS (GRAY CODE)			
B3	B2	B1	B0		G3	G2	G1	G0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
0	0	1	0	2	0	0	1	1
0	0	1	1	3	0	0	1	0
0	1	0	0	4	0	1	1	0
0	1	0	1	5	0	1	1	1
0	1	1	0	6	0	1	0	1
0	1	1	1	7	0	0	1	0
1	0	0	0	8	1	0	0	0
1	0	0	1		1	1	0	1
1	0	1	0		1	1	1	1
1	0	1	1		1	1	1	0
1	1	0	0		1	0	1	0
1	1	0	1		1	0	1	1
1	1	1	0		1	0	0	1
1	1	1	1		1	0	0	0

So, today we will first look into another code converter which will convert binary code to gray code. So, gray code is a special type of code where this successive decimal values, so they differ by only 1 bit. So, take an example. Here say, we have we take this example. So, say the value 0 0 0 0; so you in this column, so if you just write the corresponding decimal value. Then for 0 0 0 0, so the value is 0; so is 1, 2, 3, 4. So, it goes like this.

Now, you see that if you look into the binary pattern corresponding to successive decimal values, so a 0 to 1, only 1 bit is changing; the LSB is changing. But when we are going from 1 to 2, so you see the 2 bits are changing; the LSB b_0 and b_1 . So, both are flipping. Similarly when we are going from 2 to 3, again only the LSB is flipping. But when we are going from 3 to 4 here, so here, you see that a 3 bits are flipping

Now, this actually sometimes not very much advisable; particularly when you have got applications in communication and all where we want that the successive values. They

should not flip many times. Also when we have got some integrated circuit chips, VLSI chips and we have a set of signal lines are running, the signal lines in successive time instances, so, it will assume successive decimal values.

So, it is expected that those values, those bits will not flip much from 1 1 value to the next value. This is particularly true when we look into the microprocessors where there is address bus connecting to memory. So, address bus, it will generate the successive addresses and then if I have following binary pattern then there will be a large number of bit flips. So, this leads to a good amount of power consumption, bus power consumption.

So, what we do? We convert them into something called a gray code. So, in a gray code that is shown here in the right side. So, here this, if you look into the value from 0 to 1, only the LSB is flipping; from 1 to 2, only the bit g_1 is flipping. So, bit g_0 continuous to be 1. Similarly from 2 to 3 only bit g_0 is flipping, other bits remain unaltered. So, in this way if you look into any two successive bit patterns, say successive decimal values, their gray codes, they differ only by a 1 position; only 1 bit will be flipping. So, others will remain unaltered.

So, we define something called a hamming distance. So, hamming distance between two patterns say x and y is equal to the number of bit changes from x to y . So, this is defined as the number of bit changes between x and y . So, in case of, in case of binary number system; so you see that it is not a fixed value. So, many sometimes only 1 bit flips sometimes particularly if you look into this 7 to 8, this flipping; you see that all the 4 bits are flipping. So, there a hamming distance becomes high.

On the other hand, if you look into the gray code pattern from 7 to 8 also, you see only this bit is flipping, others are not flipping. So, in case of hamming, in case of gray code; this hamming distance is always equal to 1 between two successive decimal values. So, this has got many application in digital systems and so, this code conversion, is this decimal to binary to gray code conversion is sometimes very useful.

So, the way it is done; so this is the logic. So, it is a MSB is copied directly. So, from this MSB, it is XORed with the next bit to produce the next one, produce the next most significant bit. And again, so this binary code; so these are this is the binary code that we are trying to convert into gray code ok. The 5 bit binary code we are trying to convert into gray code. So, this is binary digits, they are XORed to get the next code.

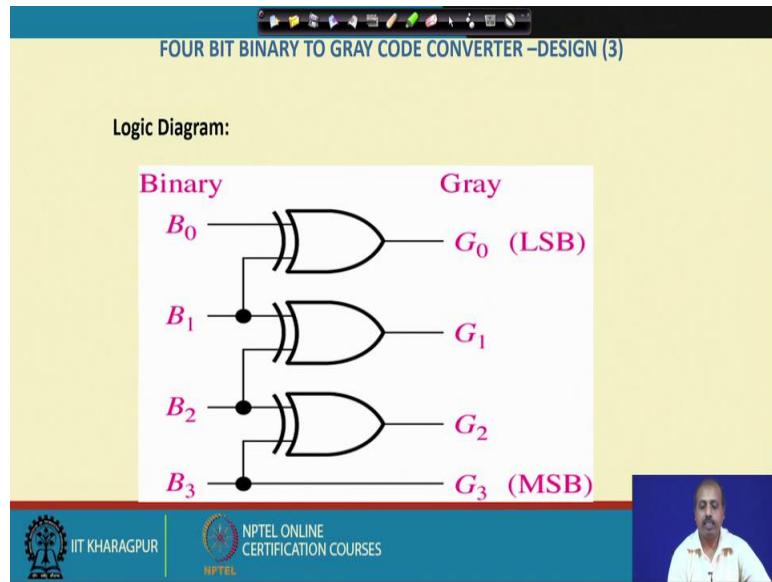
(Refer Slide Time: 05:09)

So, we will come to this definition via this truth table. So, we will start with the truth table here and then we will show you that this actually leads to this functionality. So, let us take, let us take this simplification of this code. So, if we look into the corresponding Karnaugh maps for G_3 , G_2 , G_1 and G_0 ; that is G_3 is this column, G_2 is this column.

So, if you look into the corresponding Karnaugh maps. So, G_3 is 1 from whenever B_3 is equal to 1. So, whenever B_3 is equal to 1 G_3 is equal to one. So, here you see in the karnaugh map, we have done it like this and if you do a grouping and do a minimization then G , the equation that we get is G_3 equal to B_3 . Similarly for G_2 , the function turns out to be $\overline{B_3}B_2 + B_2\overline{B_3}$ which is nothing but $B_3 \oplus B_2$.

Similarly, G_1 gives me $B_2 \oplus B_1$ and G_0 gives $B_1 \oplus B_0$. So, this is as per the definition that we have here. You see that here also the basic definition of gray code is like that and you see that the corresponding Boolean functions. So, that is also leading to the same thing.

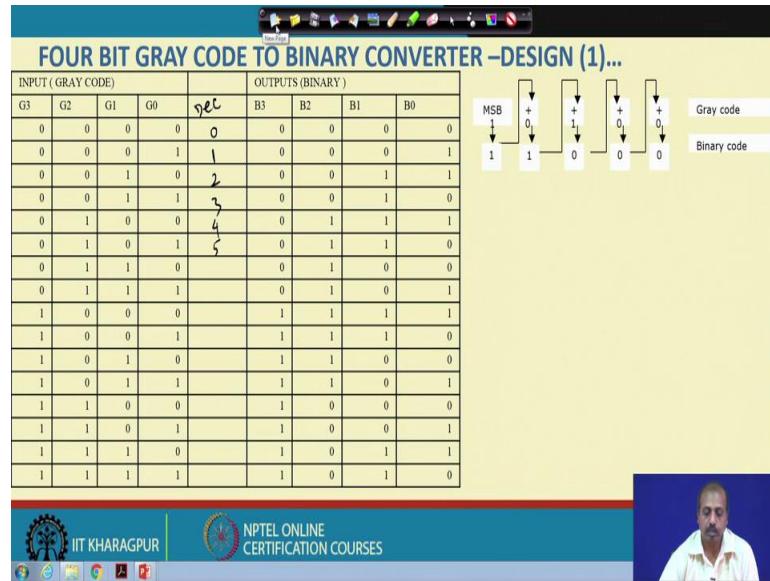
(Refer Slide Time: 06:11)



So, you can very easily draw a logic diagram which can convert a 4 bit binary code into a gray code where this B_3 will be directly copied to G_3 as per this equation G_3 equal to B_3 . G_2 equal to B_3 XOR B_2 . So, G_3 for getting G_2 so, B_3 and B_2 they are XORed. So, get G_1 G_2 . Similarly, G_1 is B_1 XOR B_2 . So, this is XORed here and G_0 is B_0 XOR B_1 . So, that gives the LSB. So, this way you can have a very simple gray code converter using XOR gates.

Next we consider the other way that is starting with the gray code. How can we reach the binary codes? So, this is the standard code conversion that we have. So, if I again, if we say that these are the decimal values; so, we say that here I am writing down the decimal, then this is 0, 1, 2, 3, 4, 5 etcetera.

(Refer Slide Time: 07:07)



Now, here also these values are known. For gray code, the values are like this for binary code; the values are like this. Now the way it is done is this MSB will be directly copied and then so this is the gray code that I have 1 0 1 0 0. Now that is 1 is copied directly to the binary MSB. So, that 1 is copied. Then this bit is XORed with the next gray coded bit 0 to produce the next bit. So, this way it goes.

So, this binary code produced, binary bit produced, so that is XORed with the next gray code bit to produce the next binary bit ok. So, that is the logic. So, we will see that this really happens, if we are looking into the truth table.

(Refer Slide Time: 08:08)

FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (2)...

		B_3			
$G_3 G_2$		00	01	11	10
00	$G_1 G_0$	0	0	0	0
01		0	0	0	0
11		1	1	1	1
10		1	1	1	1

$B_3 = G_3$

		B_2			
$G_3 G_2 G_1 G_0$		00	01	11	10
00		0	0	0	0
01		1	1	1	1
11		0	0	1	0
10		1	1	0	0

$B_2 = \overline{G_3} G_2 + G_3 \overline{G_2}$

$B_2 = G_3 \oplus G_2$

		B_1			
$G_3 G_2 G_1 G_0$		00	01	11	10
00		0	0	1	1
01		1	1	0	0
11		0	0	1	1
10		1	1	0	0

$B_1 = \overline{G_3} \overline{G_2} G_1 + G_3 G_2 G_1 + \overline{G_3} G_2 \overline{G_1} + G_3 \overline{G_2} \overline{G_1}$

$= G_1 (\overline{G_3} \overline{G_2} + G_3 G_2) + \overline{G_1} (\overline{G_3} G_2 + G_3 \overline{G_2})$

$= G_1 (G_3 \oplus G_2) + \overline{G_1} (G_3 \oplus G_2)$

$= G_1 \oplus G_3 \oplus G_2$

$B_1 = G_1 \oplus B_2$



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, let us see that ok, this is the truth table for the function. So, B_3 equal to G_3 . B_2 becomes equal to $\overline{G_3}G_2 + G_2\overline{G_3}$. So, B_2 is G_3 XOR G_2 . Similarly B_1 becomes G_1 XOR B_2 ok. So, this way we can get the individual bits.

(Refer Slide Time: 08:33)

FOUR BIT GRAY CODE TO BINARY CONVERTER –DESIGN (3)...

		B_0			
$G_3 G_2 G_1 G_0$		00	01	11	10
00		0	1	0	1
01		1	0	1	0
11		0	1	0	1
10		1	0	1	0

$$\begin{aligned} B_0 &= \overline{G_3} \overline{G_2} \overline{G_1} G_0 + \overline{G_3} \overline{G_2} G_1 \overline{G_0} + \overline{G_3} G_2 \overline{G_1} \overline{G_0} \\ &\quad + G_3 G_2 \overline{G_0}, G_0 + G_3 \overline{G_2} G_1 \overline{G_0} + G_3 \overline{G_2} \overline{G_1} \overline{G_0}, \overline{G_0} + G_3 \overline{G_2} G_1 G_0 \\ &= \overline{G_3} \overline{G_2} (\overline{G_1} G_0 + G_1 \overline{G_0}) + \overline{G_3} G_2 (\overline{G_1} \overline{G_0} + G_1 G_0) \\ &\quad + G_3 G_2 (\overline{G_1} G_0 + G_1 \overline{G_0}) + G_3 \overline{G_2} (\overline{G_1} \overline{G_0} + G_1 G_0) \\ &= \overline{G_3} \overline{G_2} (G_1 \oplus G_0) + \overline{G_3} G_2 (G_1 \oplus \overline{G_0}) \\ &\quad + G_3 G_2 (G_1 \oplus G_0) + G_3 \overline{G_2} (\overline{G_1} \oplus \overline{G_0}) \\ &= (G_1 \oplus G_0) (\overline{G_3} \overline{G_2} + G_3 G_2) + (\overline{G_1} \oplus \overline{G_0}) (\overline{G_3} G_2 + G_3 \overline{G_2}) \\ &= (G_1 \oplus G_0) (\overline{G_3} \oplus \overline{G_2}) + (\overline{G_1} \oplus \overline{G_0}) (G_3 \oplus G_2) \\ &= G_0 \oplus G_1 \oplus G_2 \oplus G_3 \\ B_0 &= G_0 \oplus B_1 \end{aligned}$$



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, B_0 is big expression like this and then you can simplify and finally it boils down to this B_0 is G_0 XOR B_1 . So, actually this B_1 is given by G_1 XOR B_2 and B_2 is given by G_3 XOR B_2 . So, if I just expand it, so this can be in the previous state at this point. If you see, B_1 turns out to be G_1 XOR G_3 XOR G_2 .

Now in the next one, you see that we have got this $G_1 \text{ XOR } G_2 \text{ XOR } G_3$. So, that part we can simplify to have $G_1 B_1$ directly. So, here this the logic diagram of this gray to binary code converter is something like this. So, we have got this G_0, G_1, G_2 and G_3 on the left side. So, G_3 is directly copied to B_3 and this G_3 and G_2 are XORed to get B_2 .

Now for getting B_1 , I need to XOR G_1, G_2 and G_3 . So that will require 3 input XOR gate. So, to simplify, so $G_2 \text{ XOR } G_3$ has already been computed in B_2 . So, this B_2 is taken here and we do $G_1 \text{ XOR } B_2$ to get B_1 . Similarly for G_0 , I, I need G_0 ; for getting B_0 , I need to XOR all this G_0, G_1, G_2, G_3 . So, what we do, B_1 is already having $G_1 \text{ XOR } G_2 \text{ XOR } G_3$. So, that is XORed with G_0 to get B_0 . So, this is possible because a XOR is a linear function that we have seen and so, it is associative in nature.

So, you can do any XOR at any, you can do the XORs in either order. So, that way it is giving as the flexibility. So, you can do this for this gray code to binary code conversion.

(Refer Slide Time: 10:40)

Exercise

1. Convert the binary number 0101 to Gray code with XOR gates
2. Convert the gray code 1011 to binary with XOR gates

Solution:

(a)
(b)

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

So, we can convert binary number 0 1 0 1 to gray code like this. So, this is 0 1 0 1, this is the binary number. Now for the gray, so this is the circuit for gray code converter. So, 0 will be copied here. Then this 0 and 1 are XORed to get 1; then this 0 and 1 are XORed to get 1. So, this way it goes. So, final solution it becomes so, 1 0 1 0. It is when it is converted to gray code, it is 1 1 0 0 or if we want to convert the gray code number 1 0 1 1 into binary, so this gray code bit is 1 0 1 1. So this 1 is directly copied here. Then the MSB is directly

copied, then this MSB and this bit G2, so they are XORed to get the bit B2. Then B2 XOR G1, so that gives me B1. So, that way it goes. So, this is the pattern that I get is 1 1 0 1.

(Refer Slide Time: 11:44)

Input (Std BCD code)					Output (XS3 Code)			
A	B	C	D		w	x	y	z
0	0	0	0	X0	0	0	1	1
0	0	0	1	1	0	1	0	0
0	0	1	0	2	0	1	0	1
0	0	1	1		0	1	1	0
0	1	0	0		0	1	1	1
0	1	0	1		1	0	0	0
0	1	1	0		1	0	0	1
0	1	1	1		1	0	1	0
1	0	0	0		1	0	1	1
1	0	0	1	✓✓	1	1	0	0
1	0	1	0		X	X	X	X
1	0	1	1		X	X	X	X
1	1	0	1		X	X	X	X
1	1	1	0		X	X	X	X
1	1	1	1		X	X	X	X

Next we look into another code which is known as BCD to XS 3 code. So many times what happens is that we want to add some base to the code. So, we do not allow some of the codes to appear. For example here, so XS 3 code means whatever be the value from there you need to subtract 3 to get the actual value. For example, so if I look into this pattern, you see that here the values that I am getting is 3. If I just take it, the binary representation so, this is 3.

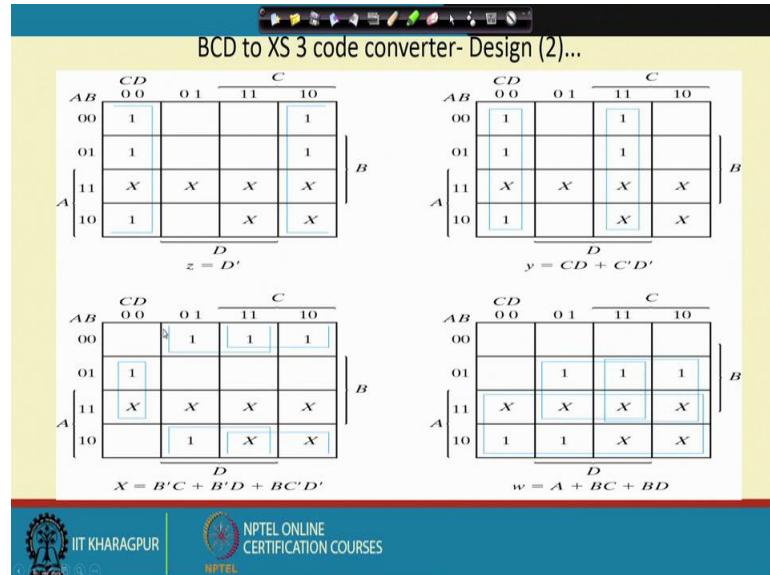
So, XS 3 means from this I need to subtract 3. So, if I $3 - 3$ is 0, so this actually the corresponding decimal value is 0. Similarly, so this is 0 1 0 0 that is 4. So, to get the actual decimal value, I have to do $4 - 3$ that is 1.

Similarly 1 0 1, so this is 5; so $5 - 3$, 2. So, XS 3 code means it has got an excess of 3 from the value. So, if I am talking about BCD to XS 3 conversion then, so BCD can go up to the digit 9. So, it is going from 0 to 9. So, this is my zero and we have 9 here, 1 0 0 1. So, up to this much.

So, nine will be converted to 12 ok. So, that the $9 + 3$ twelve. So, this is the XS 3 representation of 9. For rest of the um bit pattern 4 bit pattern, I do not need to consider because in BCD, we do not have this combinations 1 0 1 0, 1 0 1 1 and all. So, for them I

can take the output to be simply don't care because they will not appear in the conversion a in the conversion process ok. So, this co patterns will never come.

(Refer Slide Time: 13:48)



So, with this, we will be we can go to the circuit for getting the converter BCD to XS 3 converter. So, it is like this that we have this say so, the first the z bit; so x,y,z and w, so this z is the first one. So, z is this LSB. So, corresponding to that if you look into the circuit, it becomes $z = \bar{D}$.

Similarly y, this column; so, this will give us $CD + \bar{C}\bar{D}$. So, this is the XNOR function. Then this part x; so this x, so, this gives us the function $\bar{B}C + \bar{B}D + B\bar{C}\bar{D}$. So, then for w we get $A + BC + BD$. So, I can design a logic circuit for this w x y and z and get this BCD to XS 3 conversion.

(Refer Slide Time: 14:43)

BCD to XS 3 code converter- Design (3)...

- After the manipulation of the Boolean expressions for using common gates for two or more outputs, logic expressions can be given by

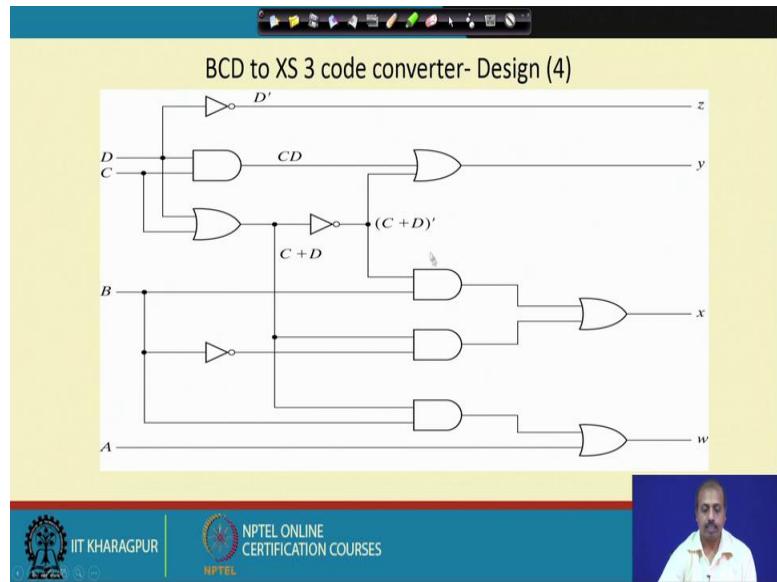
$$z=D'$$
$$y=CD+C'D' = (C+D)'$$
$$x= B'C + B'D + BC'D' = B'(C+D) + BC'D'$$
$$w= A + BC + BD = A + B (C+D)$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, after manipulation of these Boolean expressions using these common gates for two or more outputs, logic expressions are given by this. So, this is this y is $CD + \bar{C}\bar{D}$ that is a 1 NOR gate. So, we can say $CD + \bar{C}\bar{D}$, so XNOR gate. So, this is actually though written as +, but this is actually 1 XNOR function. So, this is the XNOR function.

Similarly here, so $\bar{B}C + \bar{B}D + B\bar{C}\bar{D}$. So, this is XNOR function and then so, this \bar{B} common. So, it is $(C + D)\bar{B}$. So, this $C + D$ is an OR gate and then with that we can take AND with \bar{B} . Then $A + BC + BD$ so, we can take B common and $C + D$ comes as a common function. So, this $C + D$ can be taken as this $C + D$ can be taken as the common function for doing the manipulation and then we can realize the circuit ok.

(Refer Slide Time: 16:00)



So, this is the function for x y z and w and in fact, in this case this XNOR gate and NOR gate so, they are becoming similar. So, we can use this y , so instead of doing XNOR so we can do NOR also. So, that will also give us the similar result because of the presence of don't cares.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

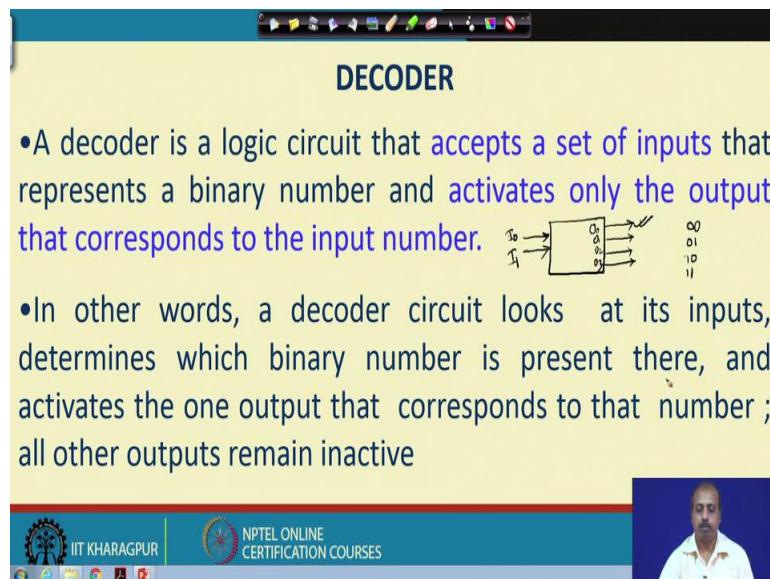
Lecture - 23
Decoders, Multiplexers, PLA

So, we next look into a topic that includes of some more combinational logic elements, Decoders Multiplexers and PLA's. So, these are slightly higher level than the gates logic gates that we have considered so far, they are essentially composed made up of logic gates only; but for so, they will have a good number of logic gates in them.

So, for our understanding, so for many a time we will find that the function that we want to realize is similar to one of these component decoder multiplexer and PLA. So, as a result so, there are many integrated circuit chips that have been developed for you that realizes this functions in them.

So, we can use those chips directly ok. So, we do not have to do in terms of basic logic gates. So, that way the design process becomes simpler ok. So, we will be looking into this one by one by one so, first we look into decoders.

(Refer Slide Time: 01:20)



DECODER

- A decoder is a logic circuit that **accepts a set of inputs** that represents a binary number and **activates only the output** that corresponds to the input number.
- In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the one output that corresponds to that number ; all other outputs remain inactive

So, a decoder it is a logic circuit that accepts a set of inputs that represents a binary number and, activates only one of the outputs that corresponds to the input number. So, what we

mean is that it has got so, if this is the box corresponding to decoder, this is the logical block that realizes the decoder, then it has it accepts some input ok. So, suppose it accepts to input and so, this 2 bit 2 input that I have say I have say I input 0 and input 1.

Now, they correspond to some binary number binary number like say, if these two bits are 0 0; that means, it is in the it is represent in the number 0, if this is 0 1 it is representing the number 1, 1 0 is 2 and 1 1 is 3.

Accordingly this in the output side I will have four lines ok, I will have four lines and out of this four lines. So, if I give say pattern 0 0, then this line will be this line output will be high or and others will be 0 ok. So, if I write in terms of so O_0 , O_1 , O_2 and O_3 so, depending upon the binary pattern that I am feeding. So, one of the four outputs will be active and others will be inactive. So, it active means it may be high were others are low, or it may be the reverse way that active means the output is low and while rest of the outputs are high. So, either way it can be done.

So, a decoder circuitry looks at its inputs determines which binary number is present, there and activates the one output that corresponds to the number, all other outputs they remain in active.

Now, you can very easily design some circuit using some basic logic gates and in fact, it is done that way only, but if your circuit is very function that you want to realizes quite complex, then it may be that a good number of gates. So, we are putting, we are replacing by a decoder to make the design simple to understand ok.

(Refer Slide Time: 03:48).

In its general form, a decoder has N input lines to handle N bits and form one to 2^N output lines to indicate the presence of one or more N -bit combinations.

The basic binary function

- An AND gate can be used as the basic decoding element because it produces a HIGH output only when all inputs are HIGH

Diagram illustrating the basic binary function of an AND gate:

Inputs: $0 \quad 0 \quad 1 \quad 1$ Outputs: $0 \quad 0 \quad 1 \quad 1$

NPTEL ONLINE CERTIFICATION COURSES

So, so, in a general form a decoder has got N input lines to handle N bits and form 1 to 2^N output lines, to indicate the presence of one or more N bit combinations. So, this has this is the basic decoder. So, we will say it is 1 to 2^N decoder. So, that way it is it is taking N input lines and depending upon the value of N one of the 2^N outputs will be made active.

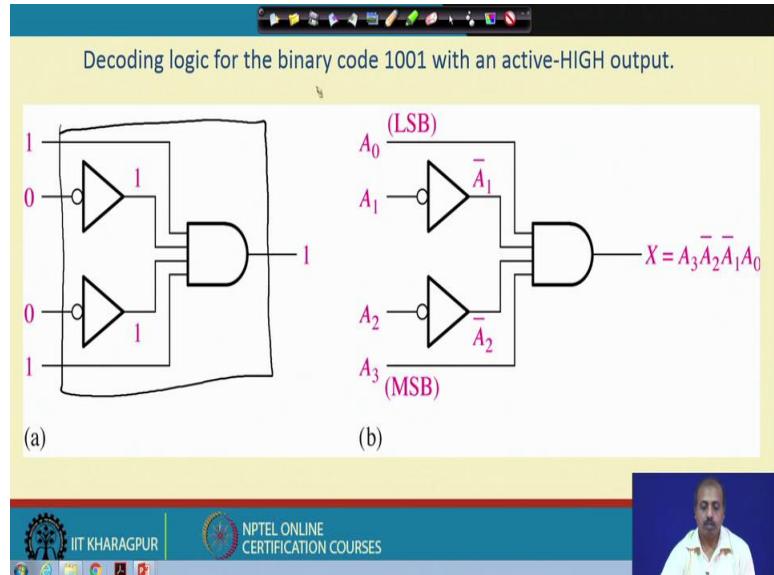
So, you can see that this basic AND gate. So, that acts as a decoding element because it produces high when all its inputs are high. So, if I take one AND gate you see the if you the AND gate. So, this output it has got 1 output and 2 inputs. So, this is also in some sense decoder, because for when the input pattern is 1 1, then only it is giving 1 all the other cases it is giving 0 output is deactive. So, since it has got only 1 output and that 1 output is high, when 1 1 is high.

On the other hand if you say OR gate so, OR gate in that sense it is not a decoder because, it is it produces one at for 3 different combinations so, 0 1, 1 0 and 1 1, for all the 3 combinations so, it is producing 1. So, by if I just look at this output and see that this is 1, I do not know I do not I can I cannot tell what was the corresponding input pattern, but for AND gate it is true that as soon as you look into the output and it is find a 1 there, you will know the input pattern is 1 1. So, AND gate in some sense can be said to be a decoder.

Of course you can tell that for OR gate also, if I get a 0 at the output, if I get a 0 at the output I definitely know that the input patterns are 0 0. So, if I am looking for AND active low decoder circuit ok that is it lowers the output when the corresponding input pattern is

applied, in that situation OR gate will act as a decoder whereas, AND gate will not act as a decoder. So, that is very rudimentary type of decoder that we are talking about.

(Refer Slide Time: 06:08)

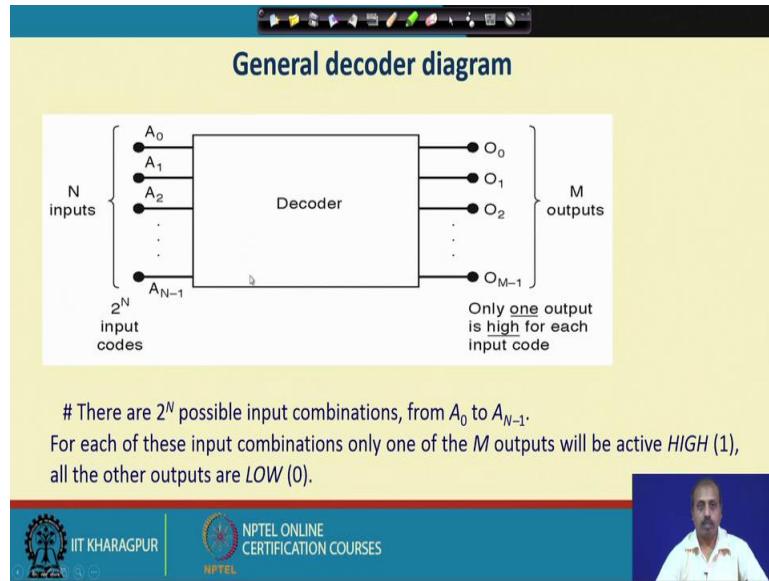


Now, so, this is the decoding logic for the binary code 1 0 0 1, with an active high input. So, this so, if you get if I have to design a black box, or some logic circuit such that for 1 0 0 1 it will give the high output. So, I can do it like this so, the I can put I can design a circuit, where all this things is put inside the box and, this box takes 4 bit input and so, only when this 4 bit input is 1 0 0 1 as per the design of the circuit you see that the output is 1.

Similarly, if you are if you are talking in terms of this value so, A₀, A₁, A₂, A₃. So, here I am getting $\overline{A_1}$ here I am getting $\overline{A_2}$ so, they are ANDed. So, we get $A_3\overline{A_2}\overline{A_1}A_0$. So, when this particular input combination comes at the block input, then only this output will be made active.

So, this way we can realize circuits to get the decoders.

(Refer Slide Time: 07:20)



In a general form so, we have got this A₀ to A_{N-1} so, N inputs are given to a decoder and, there are M outputs O₀ to O_{M-1}.

Now, this N inputs so, they can correspond they can give values, which may be any one of this 2^N input code set ok. So, this is 2^N input codes are coming and, then it is producing output which is one of this M outputs will be made high. So, O₀ to O_{M-1}, one of the outputs will be made high.

So, there are 2^N possible input combinations from A₀ to A_{N-1} and for each of this input combinations only one of the M outputs will be active high all others will be low all others will be low. So, there will be so, if high is 1 and low is 0. So, you can say it is 1 and 0. So, this is the basic decoder definition ok. So, we represent it by square like this.

(Refer Slide Time: 08:26)

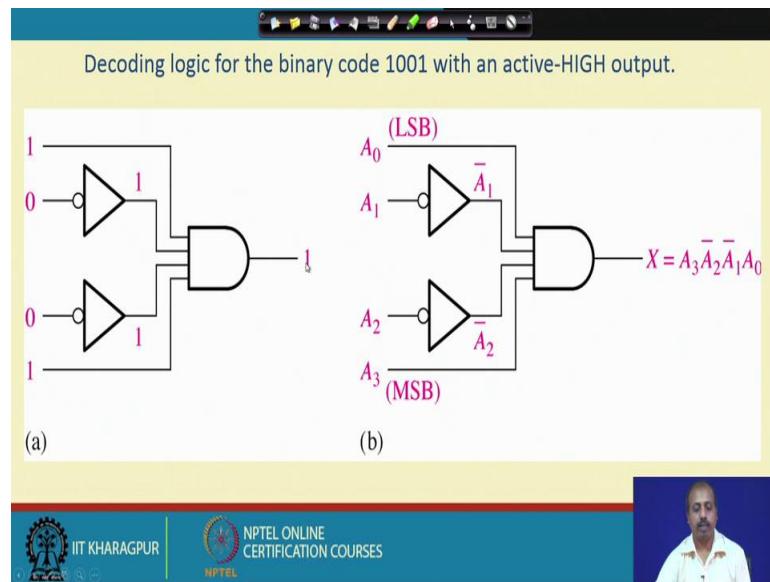
- If an **active-LOW output** (74138, one of the output will low and the rest will be high) is required for each decoded number, the entire decoder can be implemented with
 1. NAND gates
 2. Inverters
- If an **active-HIGH output** (74139, one of the output will high and the rest will be low) is required for each decoded number, the entire decoder can be implemented with
 - AND gates
 - Inverters

As I said that there can be two variants of the decoder one decoder is active low output, where the, whichever output is active. So, that output will be 0 and the rest will be 1. And other category is active high output, where I have got this decoder whose active output will be 1 and the rest will be 0.

And there are IC chips that have been designed for both categories like 74138 is an IC chip, which has got which is a decoder function with active low output and, 74139 is again a decoder, but it has got active high output.

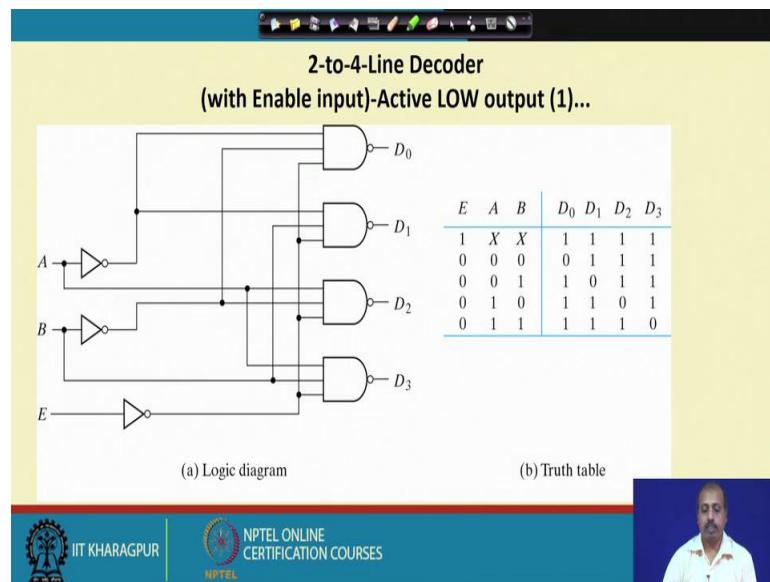
So, this in case of active low output type of decoder, the entire decoder can be implemented with NAND gates and inverters whereas, this active high output the decoder can be realized using AND gates and inverter. So, as we have seen previously like in this diagram.

(Refer Slide Time: 09:27)



So, here I was looking for active high output, so, I could do that using inverters and AND gate. So, if I was looking for active low output, then I have to use inverters and NAND gates.

(Refer Slide Time: 09:43)



So, the first one we will be considering is a 2 to 4 line decoder with enable input. So, what is there so, so here, this is the 2 input A and B that are the, that gives us to the output number that should be made active ok.

So, this is active low. So, corresponding to A and B whichever output is getting selected that output will be low and, the rest of the outputs will be high. So, for example, when A B is 0 0, then D₀ is 0 and rest of the D₁, D₂, D₃, they are 1. Similarly for 0 1 D₁ is 0 and the rest of the outputs are one like that and, there is one additional input called enable or E and, this the outputs will be coming properly only when this E is equal to 0.

If E is equal to 1, then all this D₀, D₁, D₂, D₃ so, they will be giving with the value 1 so, none of the outputs are active because, we have taken 1 as active is 1 is the non active state and 0 is the active state. So, if this E is equal to 1, then all the outputs are 1. So, the all of them are non active node non active mode. So, this way we can augment the decoder to have an enable input as well.

(Refer Slide Time: 11:09)

**2-to-4-Line Decoder
(with Enable input)-Active LOW output (2)**

- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0.
- Only one output can be equal to 0 at any given time, all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs A and B
- The circuit is disabled when E is equal to 1.

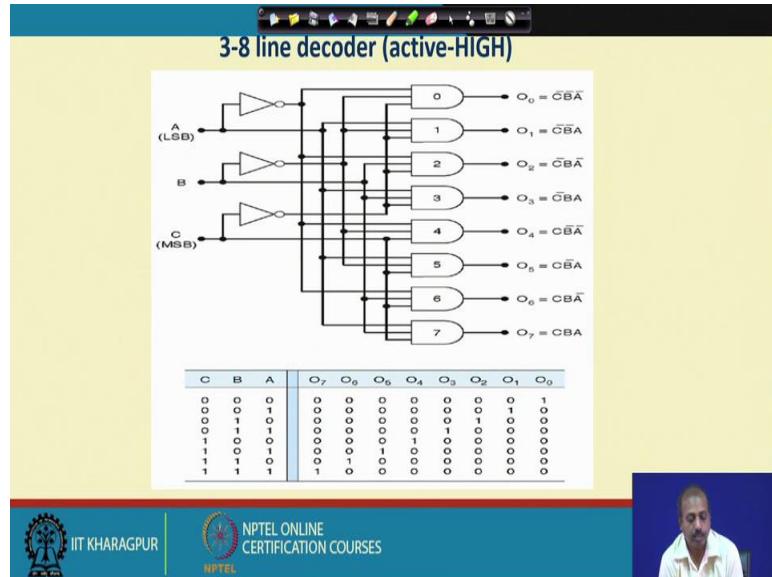
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

So, next we will look into so, this the circuit operates with complemented outputs and, a complement enable input as we have said. So, the output active output is 0 and, decoder gets enabled when E is equal to 0. Only one of the output can be equal to 0 at any given time all other outputs are equal to 1, output whose value is equal to 0 represent some min term selected by inputs A and B.

So, A and B they it can select 4 min terms, 2 input functions. So, it has got 4 min terms and in this so, so this is a D₀, D₁, D₂, D₃ they are actually corresponding to min term 0 1 2 and 3. So, based on this A B value the corresponding min term is getting selected

and corresponding D bit is made equal to 0 and, circuit is disabled when E is equal to 1, for getting the proper operation of the circuit E should be set to 0.

(Refer Slide Time: 12:13)



We look into another example so, it say 3 to 8 line decoder. So, 3 bit input so, if the N equal to 3. So, naturally I can have 2^N that is 8 as the M value or the number of outputs that I can that it can have.

Now, here you see that what do you need is for O₀ output will be will be equal to 1, when all this A B C bits are 0. So, how have we represented it? So, this is an AND gate here and you see the inputs are coming from \bar{A} , then the, this is coming from this is coming from this \bar{B} ok. So, this is this is this is the line \bar{B} . So, inverted B so, this is the \bar{B} and also \bar{C} , when $\bar{A}\bar{B}\bar{C}$. So, these two values are equal to 1, then this output O₀ will be equal to 1 and all other outputs will be equal to 0.

So, this is the truth table that we have so, when this A B C is 0 0 0, O₀ is getting selected O₀ is make becoming equal to 1 and rest of the outputs are 0. So, for any other combination one of these outputs will be made equal to 1. So, this way we can have 3 to 8 line decoder.

(Refer Slide Time: 13:42)

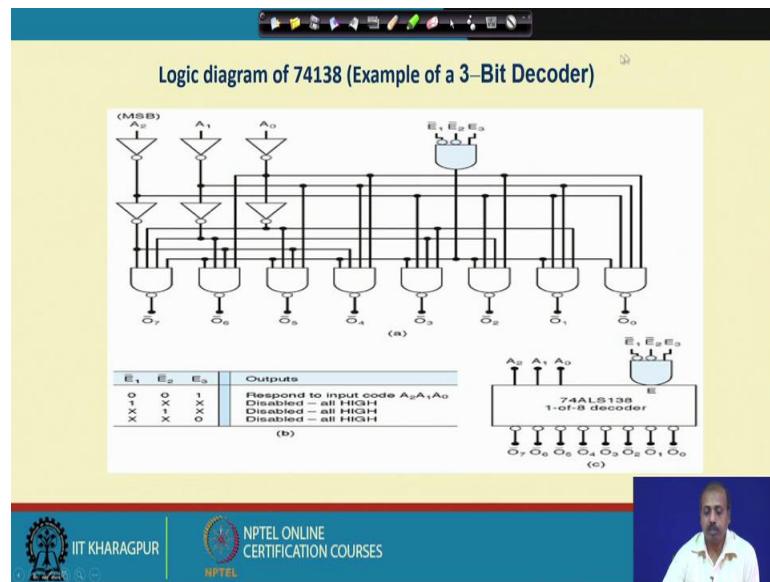
- This decoder can be referred to in several ways. It can be called a **3-line-to- 8-line decoder**, because it has three input lines and eight output lines.
- It could also be called a binary-octal decoder or converters because it takes a three bit binary input code and activates the one of the eight outputs corresponding to that code. It is also referred to as a **1-of-8 decoder**, because only 1 of the 8 outputs is activated at one time.

This decoder can be referred to in several ways; first of all it can be called a 3 line to 8 line decoder. As there are 3 input lines and 8 output lines. So, it has 3 input lines and 8 output lines. Now see this 3 and 8 so, these information they are redundant like as soon as I say the 3 are the input lines. So, 8 are the output lines.

Similarly if I say 8 as the output line, then 3 is the input line. So, it is also it can also be called a binary to octal decoder ok, or converter because it takes a 3 bit binary input code and activates the 1 of 8 outputs corresponding to that code, it is sometimes called ones it is also sometimes called 1 of 8 decoder.

So, these are varies terminologies that we have 3 line to 8 line decoder, 1 of 8 decoder. So, 1 of 8 decoder means that out of 8 only one of them will be made active. So, it is called 1 of 8 decoder. So, naturally since output there are 8 outputs. So, we must have the input as 3 3 bit the 3 input lines should be there. So, this way there are varies names for the circuit.

(Refer Slide Time: 14:54)



So, next we will look into the logic diagram of this integrated circuit chip 74138. So, this 74138 is a 3 bit decoder. So, we have got A₀ A₁ A₂ as the input and, it is when this so, it has got a number of enable line. So, this E₁ E₂ and E₃, there are 3 enable lines out of that the circuit is active when $\overline{E_1}$ is E₁ equal to 0 that is $\overline{E_1}$ equal to 1, E₂ equal to 0 that is $\overline{E_2}$ equal to 1 and E₃ equal to 1.

In that case this AND gate output will be 1 and only when this AND gate output is 1. So, this NAND gates are there and their output can be made equal to the proper value. If this if this AND gate output is 0, then this all the outputs are tied to 1 ok. So, all this 8 outputs are tied to 1. So, for getting the operation from the circuit I should set E₁ equal to 0 E₂ equal to 0 and E₃ equal to 1, then only this will have.

So, apparently it seems why do we need so, many enable line. So, E₀ E₁ E₂ and E₃ so, this helps in the logic design many time. So, may be in a minimize circuit I have got several decoders and, the same input combination is going to a number of decoders and, but I do not want or them to be active simultaneously.

So, this availability of this multiple enable line so, it helps in the process ok. So, that is why the 74138 has got a structure like this.

(Refer Slide Time: 16:31)

Truth table of 74138 (Example of a 3– 8 Bit Decoder) active-LOW									
Inputs					Outputs				
Enables		2^2	2^1	2^0	Active-LOW				
E_3	\bar{E}_1	\bar{E}_2	A_2	A_1	A_0	\bar{O}_7	\bar{O}_6	\bar{O}_5	\bar{O}_4
X	X	H	X	X	X	H	H	H	H
X	H	X	X	X	X	H	H	H	H
L	X	X	X	X	X	H	H	H	H
H	L	L	L	L	L	H	H	H	H
H	L	L	L	L	H	H	H	H	L
H	L	L	L	H	L	H	H	H	L
H	L	L	L	H	H	H	H	L	H
H	L	L	H	L	L	H	H	L	H
H	L	L	H	L	H	L	H	H	H
H	L	L	H	H	H	L	H	H	H

IIT Kharagpur
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

So, this is the truth table for 74138. So, this is when this E_3 is so, when whenever this is a E_2 is high. So, \bar{E}_2 becomes low so, this your decoder is disable. So, all the outputs are high, or if E_1 is high whatever be the value of E_3 and E_2 , then also all the outputs are high. And if E_3 is low whatever be the value of E_1 and E_2 so, again all the values are all the all the outputs are high.

So, there is no priority between E_1 E_2 E_3 so, say for correct operation of the circuit E_3 should be high E_1 should be low and E_2 should be low. Once I have put this values so, rest of the bits A_0 A_1 A_2 , if it is if the all of them are low, then \bar{O}_0 will be low and rest will be high. Similarly if this is if I put this a A_1 and A_2 as 0 and A_0 as 1 so, this will be putting \bar{O}_1 to low and rest of the lines to high. So, this is the 74138, 3 to 8 bit decoder which is an active low 1 of 8 decoder.

(Refer Slide Time: 17:56)

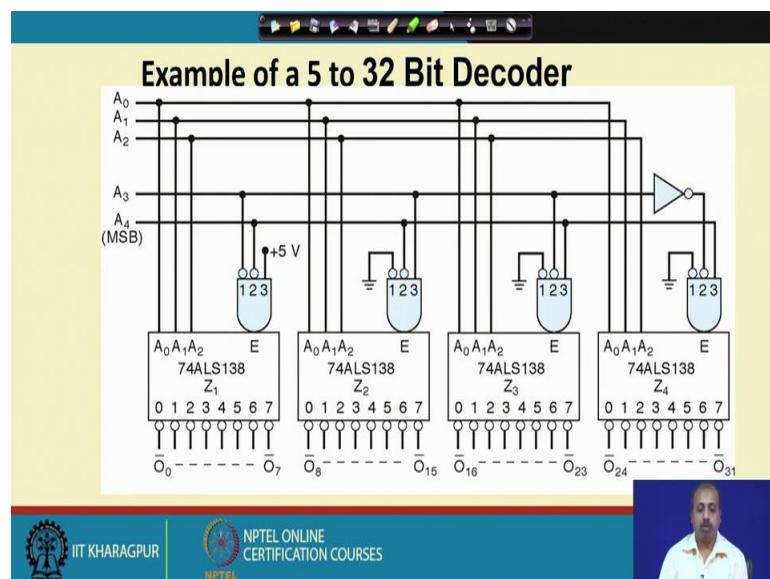
74138 (Example of a 3–8 Bit Decoder)

- There is an enable function on this device, a *LOW* level on each input E'_1 , and E'_2 , and a *HIGH* level on input E_3 , is required in order to make the enable gate output *HIGH*.
- The enable is connected to an input of each NAND gate in the decoder, so it must be *HIGH* for the NAND gate to be enabled.
- If the enable gate is not activated then all eight decoder outputs will be *HIGH* regardless of the states of the three input variables A_0 , A_1 , and A_2 .

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there is an enable function on this device a low level on each of the E'_1 , E'_2 and a high level on input E_3 is required to make the enable gate output high and the enable is connected to an input of each NAND gate in the decoder. So, it must be high for the NAND gate to be enabled, if the enable gate is not activated, then all the 8 decoder outputs will be high, regard less of the states of the 3 input variables A_0 , A_1 , A_2 . So, that is for the enable part.

(Refer Slide Time: 18:30)



Next we will look into how can I make a five to 32 bit decoder using this 3 to 8 decoders ok. So, so, I want to make a hierarchical structure so, in my application I need a 5 to 32 decoder whereas, in my library I do not have this 5 to 32 decoders available as a chip, what I have are only 3 to 8 decoders 7 4 1 3 8.

So, what we can do? So, since there are 32 outputs. So, I will need four such 7 4 1 3 8 chips. So, these are the 4 such 7 4 1 3 8 chips. Now, since there are 5 input lines. So, this A_0 to A_4 so, these are the input lines that I have. Now, this A_0 , A_1 , A_2 so, they are fed to A_0 , A_1 , A_2 of all the 3 all the 4 decoders all the four 7 4 1 3 8's.

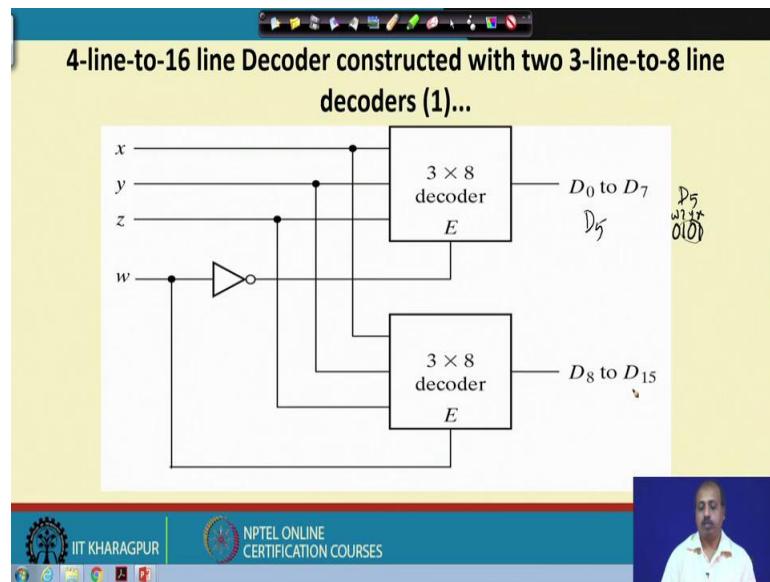
Now, using this A_3 and A_4 we select between the 7 4 1 3 8 that we want for example, the first the first 7 4 1 3 8 chip that I have put. So, that controls the outputs O_0 to O_7 second 7 4 1 3 8 controls the outputs O_8 to O_{15} so, third one O_{16} to O_{23} and a forth one O_{24} to O_{31} .

Now, what is required is if all this A_0 , A_1 , A_2 , A_3 , A_4 are 0, in that case only this $\overline{O_0}$ should be low and all the all these output still $\overline{O_{31}}$ they should be high. Now, how to do that so, this A_0 , A_1 , A_2 is fed here. So, these enable line so, we connect A_3 , A_4 and this the other the E_3 line. So, E_3 line we tie to high and this E_1 and E_2 so, we connect this A_3 , A_4 so, when this so this enable E is active A_3 equal to 0, A_4 equal to 0 and this is already 1. So, this is so this, the decoder will be active, when A_3 equal to 0 and A_4 equal to 0.

For the second decoder what we have done, we have connected the line A_3 to E_3 and then and A_4 to E_2 and the E_1 line we have grounded so this is so, E_1 is always there but if I give A_4 equal to 0 and A_3 equal to 1 ok. If I give A_3 equal to 1 and A_4 equal to 0. Then this decoder will get selected. So, this enable will be active and all the other enables will be deactivate. So, as a result all other outputs will be high, but depending upon the value of A_0 , A_1 , A_2 one of the outputs from this set will be low.

So, you see that I do not need any extra I do not need any extra gate, only thing is that we can just apply this E_3 , E_1 , E_2 properly ok. So, E_3 , E_1 , E_2 properly so, that this decoders will be this 4 to 8 decoders, they can be used to construct 5 to 32 decoder. So, that is why we have got this multiple number of enable lines.

(Refer Slide Time: 21:55)



A simpler example how to construct a 4 line to 16 line decoder using two 3 line to 8 line decoders so, we have got two 3 to 8 decoders and we use them to get a 4 to 16 line decoder. So, we assume that each of this decoders they have got one enable line and, for the decoder to operate this enable line should be equal to 1 ok.

So, this x y z so, these are the 3 inputs coming, now there is another input w so, w is the most significant bit. So, and x is the least significant bit so this x y z line. So, they are fed to both the decoders and this w lines is so line is fed to the enable of the lower decoder and this inverted w is fed to this upper decoder.

So, this upper decoder is so, is enabled when w equal to 0 and, when w equal to one the lower decoder will be enabled. So, this I can say that for D₀ to D₇'s the for 4 bit patterns that we have say for say D₅.

In 4 bit so, it is coded as 0 1 0 1 so, this x y z and w so, this is x this is y this is z and this is w ok. So, this w is equal to 0; that means, the lower decoder is disabled only the upper decoder is enabled and the upper decoder in the line x y z is getting 1 0 1 as a result the D₅ output will be made high and all other outputs will be made low.

And since this lower decoder is disabled by making E equal to 0 ok. So, all the outputs will be low. So, ultimately if you look into this entire set of 16 outputs D₀ to D₁₅, only the D₅ output will be high and all other outputs will be low. So, in this way you can use 3 to 8

decoders to make 4 to 16 decoder, or even higher like we are previously we have said we have seen the 3 to 8 decoder used for getting 5 to 32 decoders. So, that way also we can be done.

(Refer Slide Time: 24:13)

4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders (2)

- When $w=0$, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's , and the top eight outputs generate min-terms 0000 to 0111.
- When $w=1$, the enable conditions are reversed. The bottom decoder outputs generate min-terms 1000 to 1111, while the outputs of the top decoder are all 0's.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will be looking into how can I get 4. So, this is the description of that 4 to 16 line decoder. So, when w equal to 0 top decoder is enabled and the other is disabled and, the bottom decoder outputs are all 0's because, the EB the enable input is 0 and, the top a and top 8 outputs generate min terms 0 0 0 0 to 0 1 1 1.

So, that is corresponding to those min terms the outputs. So, the 8 outputs will there and, when w equal to 1 the enable conditions are reversed that is the upper decoder is now disabled and the bottom decoder is now enabled. So, it will realize the functions 1 min terms 1 0 0 0 to 1 1 1 1 and upper decoder will be outputting all 0's.

(Refer Slide Time: 24:59)

Combinational logic implementation

$S(x, y, z) = \Sigma(1, 2, 4, 7)$

$C(x, y, z) = \Sigma(3, 5, 6, 7)$

The circuit diagram shows a 3-to-8 decoder with inputs x, y, and z. The outputs are labeled 0 through 7. The sum output S is connected to the outputs of decoder outputs 1, 2, 4, and 7 via an OR gate. The carry output C is connected to the outputs of decoder outputs 3, 5, 6, and 7 via another OR gate. To the right is a truth table for the full adder function.

x	y	z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, this decoders they can be used for realizing combinational functions directly for example, suppose we look into that adder function ok. So, in case of adder you know that the sum is given by $\Sigma 1, 2, 4, 7$. So, this $x y z$ it is a full adder.

So, so if I have got full adder. So, this is the truth table so, $x y$ and z and sum and carry. So, this is so 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, when this so, this is come in 0 0, 1 0, 1 0, 0 1, 1 0, 0 1, 1 0 and sorry sorry.

So, this is 0 1 and this is 1 1 now, first sum is 1 for this combination this combination that is 1 2 4 and 7 for this cases some is equal to 1 and the carry is equal to 1 for 3 5 6 and 7 so this. So, this is the 3 then 5 6 and 7 for this cases carry is equal to 1. So, this is the min term based representation. So, if you are trying to realize some circuit using decoder. So, first we have to come to the min term representation of the function, where it lists down the min terms corresponding to the function.

Now, so, now we use a decoder so this is a 3 to 8 decoder. So, this $x y z$ so, they are connected to the input side. Now, output sum will be equal to 1 if the combination 1 2 4 or 7 are chosen such that is if this $x y z$ value is such that this output 1, 2, 4 or 7 so, the so any of them will be equal to 1 in that case sum output will be equal to 1. And we have got this carry output. So, this will be equal to 1 if this 3 5 6 and 7 any of this outputs are 1 so, this 3 5 6 and 7 so, they are outputs are 1.

So, from the min terms, we can figure out the lines that are important for the function and, we OR them to get the corresponding function. So, this way we can realize any combinational function by means of decoders. So, the point is that you have to convert it into min term representation and from the min term representation. So, we have to go it is not from the Karnuagh map it is from the min term representation of the function.

(Refer Slide Time: 27:55)



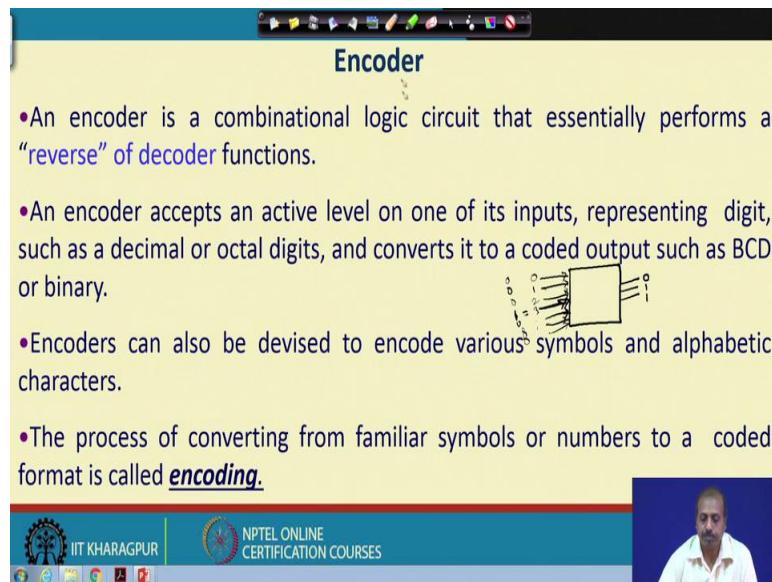
So, next we will be looking into encoders. So, encoders just the reverse of decoders so, in case of decoder from less number of line so, you are going to more number of lines, for the encode will do just the reverse, we will start with the more number of lines and from there we will come to less number of lines.

So, that is also useful in many cases particularly, when we are trying to reduce the amount of information that we need to send from one place to another ok. So, number of bits used for transmission and all. So, we use encoders so, that we can said more amount of information using lesser number of bits. So, we will see that in successive class.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 24
Decoders, Multiplexers, PLA (Contd.)

(Refer Slide Time: 00:20)



Encoder

- An encoder is a combinational logic circuit that essentially performs a "reverse" of decoder functions.
- An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting from familiar symbols or numbers to a coded format is called encoding.

NPTEL ONLINE
CERTIFICATION COURSES



So encoders, they are used for one purpose which is for which you say it is a reverse of this decoder functionality. So, it is a combinational logic circuit that essentially performs the reverse of the decoding function and an encoder, it will accept active level on one of its inputs representing the digit such as a decimal or octal digit and convert it into a coded output such as a BCD or binary.

So, what you mean is something like this. So, it is a box ok. If this is the logic circuit for the encoder, so it will be accepting a number of lines as input. So, these are inputs. And at any point of time only one of the inputs will be will be active. So, other should be in deactive. So, it does not allow by definition more than one input to be high simultaneously. So, it will allow, so it is accepted that whoever is operating this encoder circuitry will make only one of the inputs active at a time and accordingly you have got this output. So, this output will tell the number of the input which has been made active. What I mean is suppose this side, I have got say 8 inputs. So, I have got say 8 inputs and then say input number, so these are the input numbers; so 0, 1, 2, 3 etcetera.

Now, suppose this input number 3 has been made active, so this is equal to 1 and rest are all 0's as per as the value is concerned. So, this is made equal to 1 and rest are all equal to 0. Then on the output side, what we will happen? So, this will output the bit pattern 0 1 1 telling that the third input had been the input 3 has been made high. So, this is actually encoding the number, the input number which is active now ok. So, that is how it is the encoder.

So, instead of telling the number directly, so you can you can send a 3 bit pattern like if you want to tell at this point of time which input is high. So, if you do not have this encoder circuitry, then you have to give this 8 bit pattern and in that 8 bit pattern whichever input is 1, so that we will tell us that that input is active. So, that will require 8 bit of information to be communicated. However, if you use this encoder circuitry, so you can transmit a 3 bit information ok; so, that has got the number directly. So, that is the purpose of the encoder.

So, encoders can also be devised to encode various symbols and alphabetic characters as I was telling. The process of converting familiar symbols or numbers to a coded format is known as encoding. So, encoder performs encoding operation

(Refer Slide Time: 03:28)

- Most decoders accept an input code and produce a HIGH (or a LOW) at one and only one output line.
- In other worlds, a decoder identifies, recognizes, or detects a particular code. The opposite of this decoding process is called encoding and is performed by a logic circuit called an encoder.
- An encoder has a **number of input lines**, **only one** of which **input is activated** at a given time and produces an N-bit output code, depending on which input is activated.

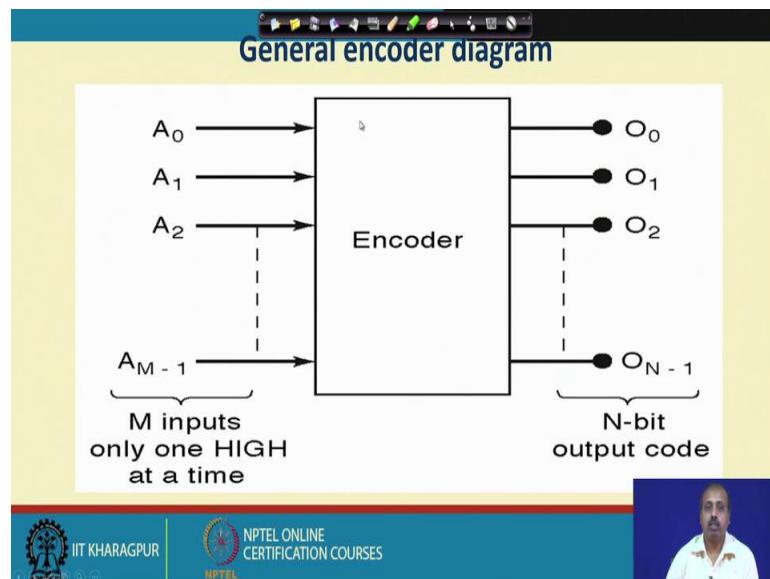
At the bottom of the slide, there is a video player interface showing a person speaking. Below the video player is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and several small icons.

So, most of the decoders accept an input code and produce a high or low at one and only one output line. So, why is it told that encoder is the reverse of decoder? So, this is the thing. So, in a encoding, decoder what happens is that if this is a decoder, so this decoder;

it accepts some input say N inputs and then it has got 2^N outputs. It has got 2^N outputs and depending upon the input combination 1 of the output is made high or made active, others are made low.

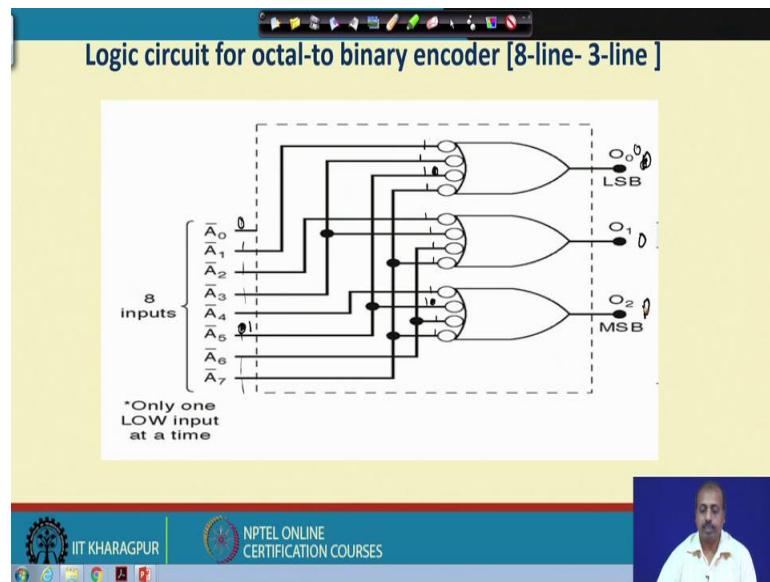
So, decoder we can say it identifies recognizes or detects a particular code. So, it can tell you which pattern you have applied here accordingly. So, from the code, so it can identify the pattern that you want to apply. Then opposite of the decoding process is the encoding and it is performed by the logic circuit that does it is an encoder. Encoder has a number of input lines only one of which will be active at a given time and produces n bit output code depending upon which input has been activated. So, we will see how to design this circuit and all

(Refer Slide Time: 04:50)



So, this is the block diagram. So, we have got this A_0 , A_1 , A_2 up to A_{M-1} as input lines and then we have got so there are M input lines out of which only 1 is high at a time and then at the output side, we have got O_0 to O_{N-1} . So, N output lines. So, that will be identifying which M which of these M inputs is high. So, that is the encoder functionality.

(Refer Slide Time: 05:20)



Now, how do you realize the encoder by means of basic logic gates? So, this is octal to binary encoder. So, it has got 1 octal input and then it is say, if the octal input \bar{A}_0 , A_1 up to A_7 . So, whichever value you want to output. So, 0 or 3 or 4 or 5 or 7 accordingly, the corresponding input line will be made equal to 0.

So, for 5 for example, the A_5 line will be made equal to 0 and all other lines will be made equal to 1. So, this output, output will be coming in terms of it will identify which input has been made active. So, accordingly it will be output in the bit pattern. So, if the 5'th input has been chosen here, so if I as an input; So, if I give an A_5 equal to 0 and rest are all 1, then it is expected that here I will get the bit pattern 1 0 1.

So, how does it happen? So, if all of them are 1 and this A_5 is 0, you see that all these bits are so, wherever this A_5 is going. So, this line is 0. You are getting a 0 here and here it is getting a 0 and on all other places I have got a 1, at all other places I have got 1 in this circuit. So, what is what will happen? So, it is inverted and then OR so, this is basically the NAND gate and NAND gate all inputs being equal to 1 output is 0 and if any of the input is equal to 0 output is equal to 1. So, you will get the patterns 1 0 1 at the output.

So, this A_5 being equal to 0, output is 1 0 1. If A_0 is equal to 0 and rest are all equal to 1, then what happens? If A_0 equal to 0, then and all other so, A_0 line is not connected to anybody. So, all these inputs are that we are getting are 1. So as a result, the NAND function, so this will give all 0 here. This also sorry this will be 0 and this will

also be 0. So, the 0 0 0 will be the output which definitely identifies the A₀ input. So, this way I can use, I can design this encoder circuitry by using NAND gates ok, the simply connecting some NAND gates. So, we can design this circuit.

(Refer Slide Time: 08:08)

Truth table for octal-to binary encoder [8-line- 3-line]

Inputs								Outputs		
\bar{A}_0	\bar{A}_1	\bar{A}_2	\bar{A}_3	\bar{A}_4	\bar{A}_5	\bar{A}_6	\bar{A}_7	O_2	O_1	O_0
X	1	1	1	1	1	1	1	0	0	0
X	0	1	1	1	1	1	1	0	0	1
X	1	0	1	1	1	1	1	0	1	0
X	1	1	0	1	1	1	1	0	1	1
X	1	1	1	0	1	1	1	1	0	0
X	1	1	1	1	0	1	1	1	0	1
X	1	1	1	1	1	0	1	1	1	0
X	1	1	1	1	1	1	0	1	1	1

A low at any single input will produce the output binary code corresponding to that input. For instance , a low at A_3' will produce $O_2=0$, $O_1=1$ and $O_0=1$, which is binary code for 3. A_0' is not connected to the logic gates because the encoder outputs always be normally at 000 when none of the inputs is LOW


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

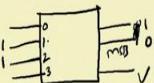
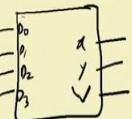

So, a low at any single input will produce the output binary code corresponding to that input. So, if I have a low at \bar{A}_3 here. So, if I have got \bar{A}_3 , so it produces O_2 equal to 0, 1 equal to 1 and O_0 equal to 1 which is the code for 3. So, this way it is done. So, we have already explained how this circuit is operating. So, we can get it like this.

(Refer Slide Time: 08:37)

Design of 4-input Priority Encoder
(4-line-to 2 line priority encoder) (1)...

- A priority encoder is an encoder that includes the **priority** function
- If two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- **Truth Table of a 4-input Priority Encoder:**

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	v
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES


Now, one important constraint that we have put on the operation of the encoder is that at any point of time only 1 of its inputs can be equal to 1. So, many a times, it is a difficult to ensure that in the practical operation. So, it may so happen that the inputs are coming from different sources and as a result, it is simultaneously more than one input is becoming active. But even if they are active, so we may want to give priority to some of them. What I mean is that so, suppose I have suppose I have got a block like this. I have got a block like this and it has got say 4 inputs, so 4 inputs. Now it has got 2 outputs, this is an encoder.

So, these are line 0 1 2 3. So, if the line if the line 1 is active and 0 2 3 are inactive then the output should be 0 1 that will identify, if I take this as the MSB, if I take the lower one as MSB, so this will be 0 1 that will identifying the for the input which is equal to 1.

Now if this 1 and 2 are simultaneously equal to 1, then also we may say that I will give priority to 1 and this 2 the 2 input also be equal to 1 will just be ignored as long as 1 the input 1 is having equal to 1. So, that way I this input 1 has got higher priority than input 2. So, priority encoder is an encoder that includes the priority function. So, if 2 are more inputs are equal to 1 at the same time, the input having the highest priority. We will take precedence. So, it may be like this that see here. So, this if all the inputs are 0 then, so this truth table. So, apart from this x y, so there is another output valid ok. So, there is another output valid. So, you can say that it is a 4 bit input. So, you have got this D_0, D_1, D_2, D_3 . So, this is D_0, D_1, D_2, D_3 plus there is so, there are two outputs x and y and there is another output which is valid v.

So, for the operation, so we have to look into the valid bit. So, if the valid bit is not equal to 1; that means, whatever output is produced as at x y. So, that is not a valid combination. So, when this D_0, D_1, D_2, D_3 so, they are all equal to 0. So, you say that x and y they can assume any value, but this valid bit should be equal to 0; so, as in invalid bit. Then if it is 1 0 0 0 that is only D_0 is equal to 1, so then there is no problem; so only one of the inputs are become equal to 1. So, we can say that output should show that input 0 was 1. So, it is the x and y is 0 0 and this valid bit is equal to 1 telling that it is it is valid ok. The pattern output it is valid.

Now, if D_1 equal to 1, then whatever be the so if D_1 equal to 1, D_2 equal to 0 and D_3 equal to 0, then without looking into the value of D_0 , So, it will output 0 1. So, if this is the condition that $D_2 D_3$ are equal to 0, but D_1 equal to 1. So, it will not consider what is the

value of D_0 will straight way output 0 1. So, that is it will tell that now input 1 is equal to 1.

Similarly, if it is if input if the input D_2 is equal to 1, then irrespective of the values that we have in D_0 and D_1 . If D_3 remains equal to 0, so then, this x y should be 1 0 and the valid bit should be 1 telling that input 2, D_2 is 1 and finally, if D_3 is equal to 1, it does not consider whatever the values we have in D_0 , D_1 , D_2 . It produces output as 1 1 telling that D_3 is high.

So, you see that this line D_3 , it has got the highest priority. So, if D_3 is equal to 1, we do not look into other values of values of D_0 , D_1 and D_2 and we are straight way telling if the output is 3. On the other hand, if D_3 remains 0 and D_2 equal to 1 we are do not considering D_1 and D_0 . D_3 , D_2 both being equal to 0 and D_1 equal to 1, we are not considering D_0 and we are considering D_0 only when this D_1 , D_2 , D_3 all of them are equal to 0.

So, I can say the D_3 has the highest priority followed by D_2 followed by D_1 followed by D_0 for getting identified at the output. So, this is the function of this priority encoder. So, as if we have got 4 inputs and there are priorities. So, priority of D_3 is the maximum followed by D_2 followed by D_1 followed by D_0 .

(Refer Slide Time: 14:00)

The slide has a yellow header bar with the title "Design of 4-input Priority Encoder" and a subtitle "(4-line-to 2 line priority encoder) (2)..." in blue text. Below the header is a bulleted list of four points in black text:

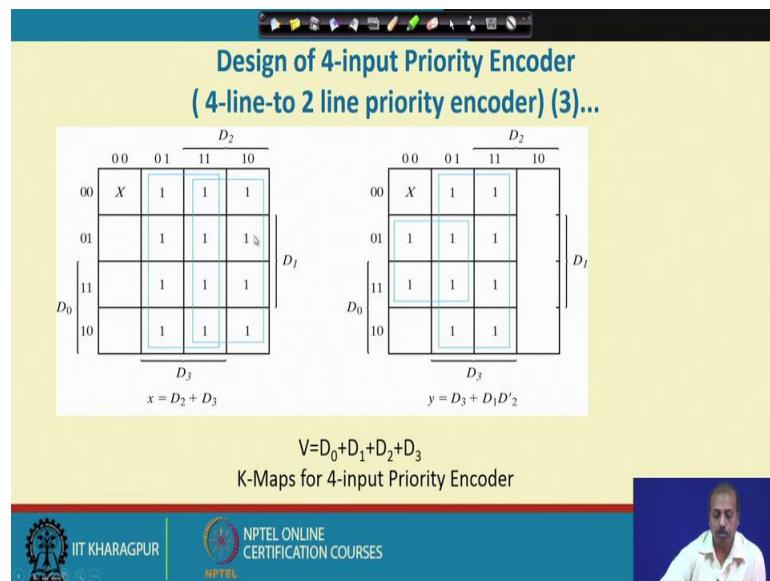
- In addition to two outputs x , and y , the truth table has a third output designated by V , which is a valid bit indicator that is set 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.
- X's in the output column indicate don't care conditions, the X's in the input columns are useful for representing a truth table in condensed form.
- The higher the subscript number, the higher the priority of the input. Input D_3 has the highest priority, so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3)

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". To the right of the footer, there is a small video window showing a man speaking.

So, we can, so in addition to two outputs x and y, the truth table has a third output V which is which is a valid bit valid bit indicator and is set to 1, when 1 or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.

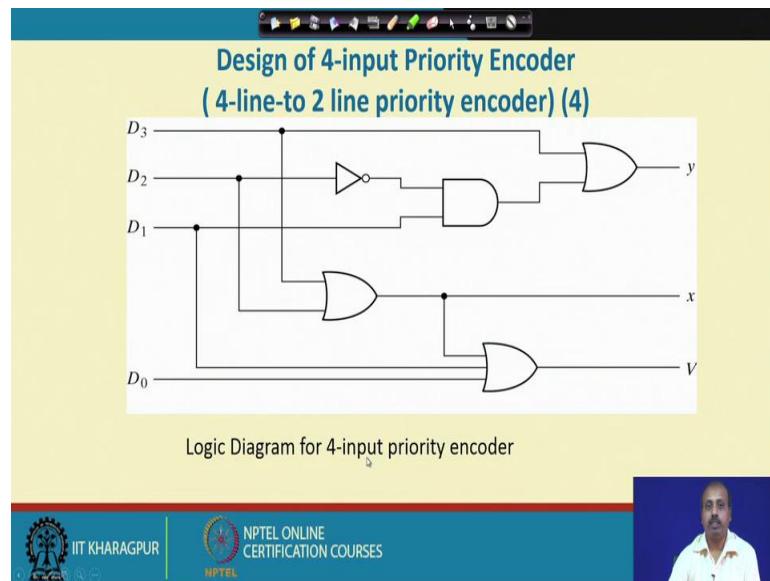
Now, don't cares that we have in the output column indicate that this input columns, similarly we have got don't cares in the input column also. So, that is that actually makes the truth table more condense and the priority that we have is like this. The higher the subscript number, higher the priority of the input. So, input D_3 has the highest priority. So, regardless of the values of the other inputs, when this input is equal to 1, output x y is equal to 1 is equal to 3; 1 1. So, this way we can have this priority function.

(Refer Slide Time: 15:00)



So, if we are looking into this design. So, we have got this x, if we draw the truth table then it will be like this. So, x equal to $D_2 \text{ OR } D_3$, then this $y = D_3 + D_1\overline{D}_2$ ok.

(Refer Slide Time: 15:18)



So, similarly from this you can x and y you have got and this valid bit is equal to 1, if all of them are not equal to 0; at least 1 of them is equal to 1. So, this here there is a slight modification in the circuit. So, instead of ORing D_0, D_1, D_2, D_3 separately, so we have take we have already done an OR of D_2, D_3 here for getting the value of x . So, that has been utilized here for getting the value of v . So, in this way we can design a very simple logic circuit for realizing this 4 input priority encoder.

(Refer Slide Time: 16:00)



Next we look into another fundamental circuit element which is known as multiplexer. So, multiplexers; so they are actually they are also called data selector. So, it is like this that

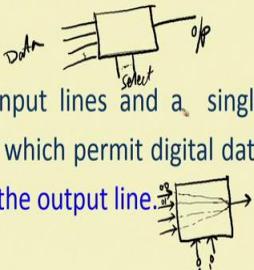
multiplexer, it is a device that allows digital information from several sources to be routed onto a single line for transmission or over that line to a common destination.

(Refer Slide Time: 16:22)

MULTIPLEXERS (Data Selectors)

A multiplexers (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.

The basic multiplexers has several data input lines and a single output line. It also has data- select inputs, which permit digital data on any one of the inputs to be switched to the output line.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, multiplexing is very common like in say when we are say a telephone line, it is carrying signals from several people and it is multiplexing between the that people that are talking so that way so there is a multiplexer.

So, it is actually sometimes, it is selecting one data item; some other time, it is selecting some other data item to be transmitted through the channel. So, the basic multiplexers has several data input lines and a single output line and it also has data select inputs which permit digital data on any of its inputs to be switched to the output line. So, a basic multiplexer is block diagram wise. So, you can say that it will look something like this that we have got this has multiplexer and then we have got some data lines. So, it has got two types of lines, some of the lines are called Data lines and some of the lines are called Select lines. So, these are called select lines.

Now, depending upon the values of this and the output this is the output and output is only 1 bit. So, we have got data lines. So, which can be many we have got select lines which can also be many and then depending upon the values that we are put into the select lines, one of these inputs will be selected and that will go to the output. For example, if I have got say 4 data lines, 4 data lines and 2 select lines like this; 4 data lines and 2 select lines, then depending upon the value, so suppose it can it may say that if the value is 0 0, then

this output will be routed to the, this input will be routed to the output. So, this corresponds to the select lines 0 0.

Similarly, if the value is equal to 0 1, then this input will be routed to the output. So, this is corresponding to the pattern 0 1. This is for 1 0 and this for 1 1; so depending upon select line combination that I have one of this inputs that will be routed to the output. So, this is very useful in many circuit design problem, we will see that this multiplexers; they make it very simple for realizing functions and that will be used in designing many systems.

(Refer Slide Time: 19:08)

MUX-continued...

A modern stereo system may have a switch that selects music from one of four sources: a cassette tape, CD, a radio tuner , or an auxiliary input such as audio from a VCR or DVD. The switch selects one of the electronic signals from one of these four sources and sends it to the power amplifier and speakers.

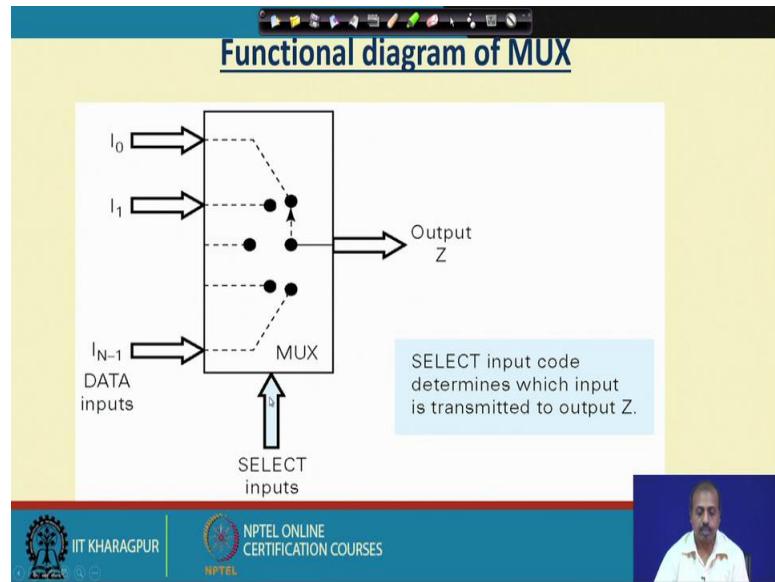
In simple terms, this is what a multiplexer (MUX) does; it selects one of several input signals and passes it on to the output.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



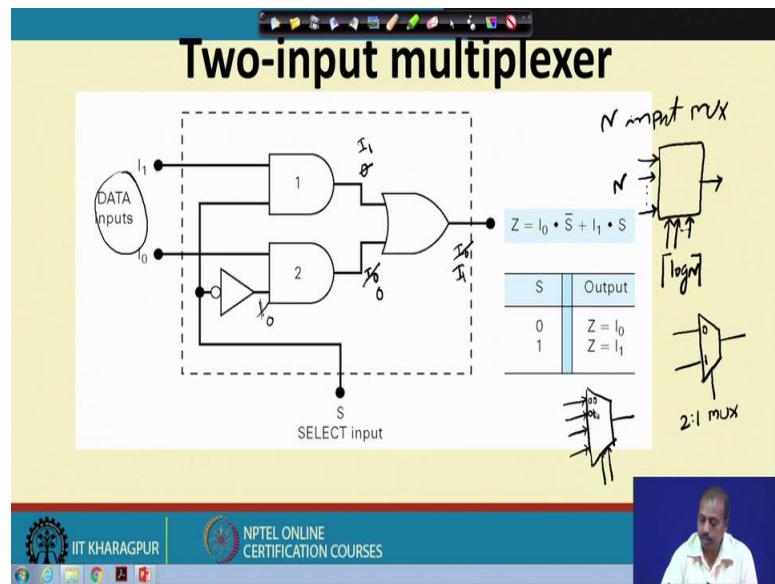
So, a modern stereo system, a typical example; so it may have a switch that selects music from one or one of four sources, when we may be it has got a cassette, tape, CD, radio tuner or an auxiliary input from VCR or DVD or may be some other sources also. Then switch selects one of the electronics signals from one of these from one of these four sources and sends to the power amplifier and speakers. So, that is nothing but some sort of multiplexers.

(Refer Slide Time: 19:49)



So, a multiplexer does, what it does is that it selects one of several input signals and passes it on to the output. So, that is the basic function of multiplexer. So, this is the SELECT line. So, SELECT inputs are applied here. So, depending upon the SELECT input as if logically you can say that this if this switch change the changes its position to from between these points. And accordingly one of the DATA inputs, we will get selected to be sent to the output. So, SELECT input code, the SELECT input code, so this will determine which input is transmitted to the output.

(Refer Slide Time: 20:20)



So, start with a two Two-input multiplexer. So, Two-input multiplexer is, how to realize? So, it Two-input multiplexer means so, when I say two-input, so these two is

corresponding to the DATA inputs ok. So, two-input it immediately tells that number of SELECT lines that I will have is only 1. So, it is like this. So, if there are if it an N input multiplexer N input multiplexer, then I has got N input lines. It has got N input lines. So, number of SELECT lines that I should have here is $\log N$. So, this is N inputs. So, at the here I should have $\log N$ number of SELECT lines that will be an output here. So, because otherwise I cannot select all the N inputs by applying the input to the select lines; so we cannot applying SELECT.

Internally how will it look like is something like this. So, this is when say S is equal to 0; when S is equal to 0, then you see that these AND gate output is 0. So, whatever is the value of I_1 , so you will get a 0 at this point. On the other hand, when S equal to 0; so here this input is 1. So as a result whatever is coming at I_0 , so that will be output here. So, this point a I am getting a 0, this point I am getting I_0 and this is OR of these two. So, here I will get I_0 . So, when S equal to 0, I am getting I_0 and when S equal to 1 then what happens? So here, I will get I_1 , 1 will come here. So, this input is 0. So, as a result I will get a 0 at this point. So, they will be ORed and I will be getting I 1.

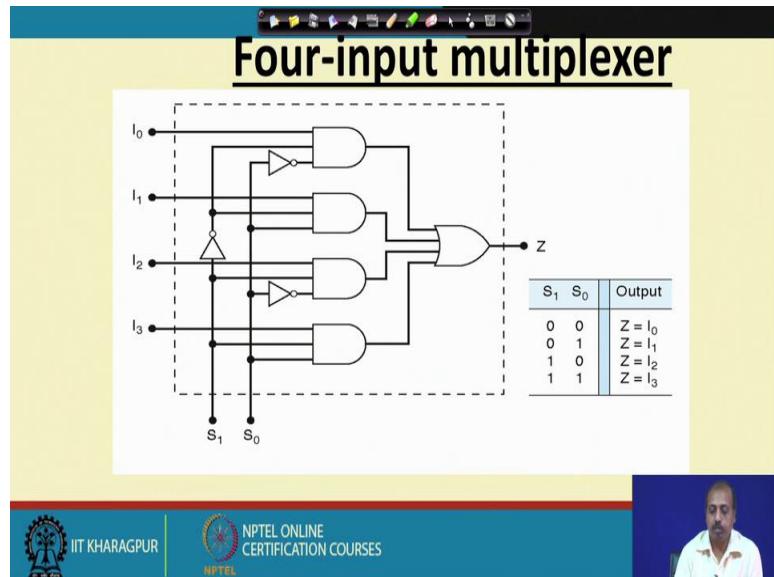
So, when S equal to 0, then this I 0 is transferred from the input at I_0 is transferred to this output Z and when Is equal to 1, then input I_1 is transferred to output Z. So, I can say that the functionality that I am realizing is $Z = I_0\bar{S} + I_1S$ ok. So, this is the basic multiplexer functionality.

So, these multiplexers are often represented by this type of notation. There are several notations for multiplexer. So, this is the most common on. So, we have got this data inputs and this is the SELECT input. So, the it 0 is written here and a 1 is written here meaning that if the input if the control signal is or the select input is 0, then these input will be selected to go to the output and if this select input is 1, then these input will be selected to go to the output. So, this is a 2 to 1 multiplexer or 2 is to1 multiplexer. There are several names or 2 is to 1 multiplexer.

Now, if it is a 4 is to 1 multiplexer, then the corresponding symbol is something like this ok. So, it will have 4 lines, 4 DATA input lines and 2 SELECT input lines and 1 output ok. So, it will have 1 output and again the same thing. So, if it is 0 0, then this input is selected. So, write a 0 0 here. If SELECT lines are 0 1, then this input is selected. If it is 1 0, then these input is selected and if it is 1 1, this input is selected. So, this way we represent

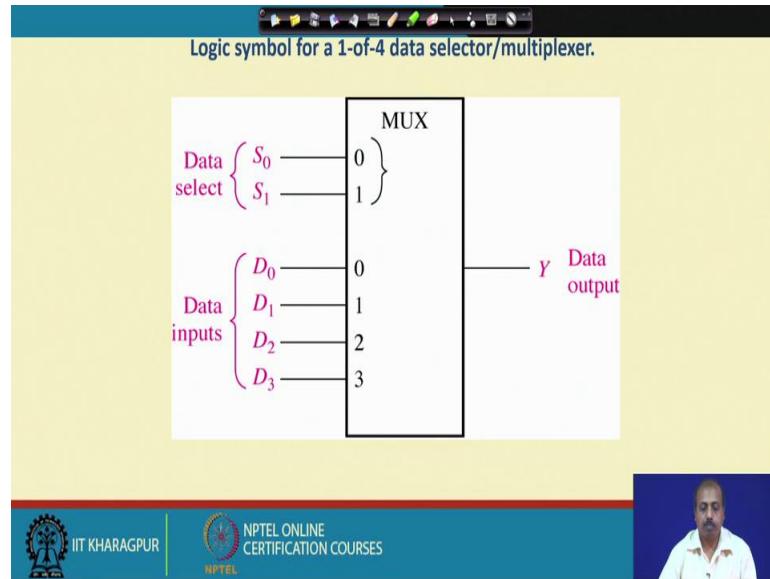
4 input multiplexers in this form ok. So, we will see them in detail. So, this is the 4 input multiplexer.

(Refer Slide Time: 24 is to 19)



So, you see that when S_0 and S_1 , these values are 0 0, then what I want that I_0 should get transferred to Z . So, we can quickly check that it is happening like when $S_0 S_1$ both are 0; then this input is equal so, they when both are equal to 0, then this input is equal to 1 and this input is also equal to 1. So, whatever we are getting at I_0 is transferred to this point, whereas for this second AND gate, so this S this input is 1, but this input is 0. As a result, whatever be the value of I_1 , you are getting a 0 at this point. In this way you can find that for all the cases. So, you were getting a 0 here. So, this OR gate; so, one input is I_0 and rest are all 0. So at Z , I am getting equal to I_0 at Z I am getting equal to I_0 .

(Refer Slide Time: 25:30)



So, other combinations also you can check. So, this is the functionality of a four input multiplexer or Four to one multiplexer whatever you call it. So, this is logic symbol for one of four Data selector or multiplexer. So, it is also represented like this. So, we have showed you another representation where it is a trapezium like structure. So, you can also represent it by means of a rectangular symbol like this ok. So, normally the data inputs are marked as D_0 , D_1 , D_2 , D_3 or we write down the corresponding decimal values that correspond to and in the SELECT input are normally written as a S_0 , S_1 and they are also we tell which one is LSB and which one is MSB where putting 0 and 1 this numbering actually.

(Refer Slide Time: 26:16)

Logic symbol for a 1-of-4 data selector/multiplexer.

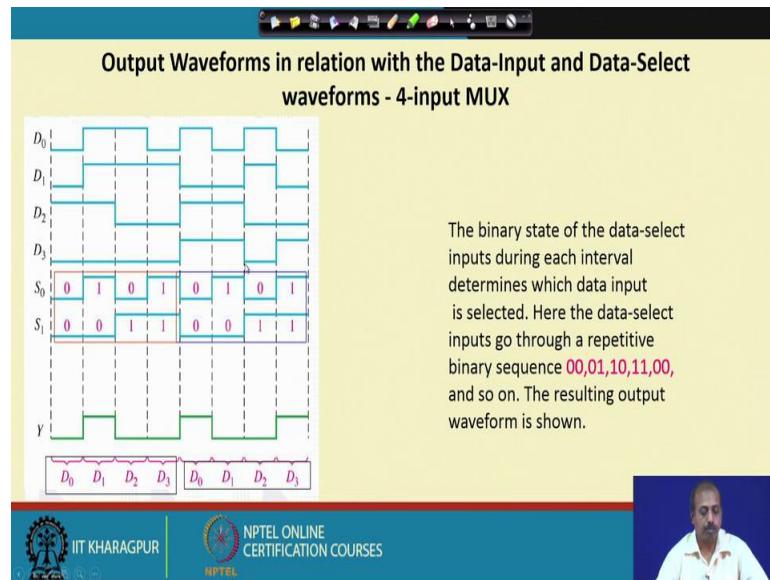
		INPUTS				STROBE \overline{G}	OUTPUT Y
SELECT		DATA					
B	A	C ₀	C ₁	C ₂	C ₃		
S ₁	S ₀	X	X	X	X	H	Z
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

Select inputs A and B are common to both sections.

NPTEL ONLINE CERTIFICATION COURSES

So, this S₀, logic symbol for a 4 is to 1, 1-of-4 data selector or 4 is to 1 multiplexer. So, it can be when the SELECT inputs S₀ S₁, so they are if they are going to be low; if both are low, so this if both are low, then this C₀ will be selected. In fact, I think there is actually this timing diagram explains it better.

(Refer Slide Time: 26:46)

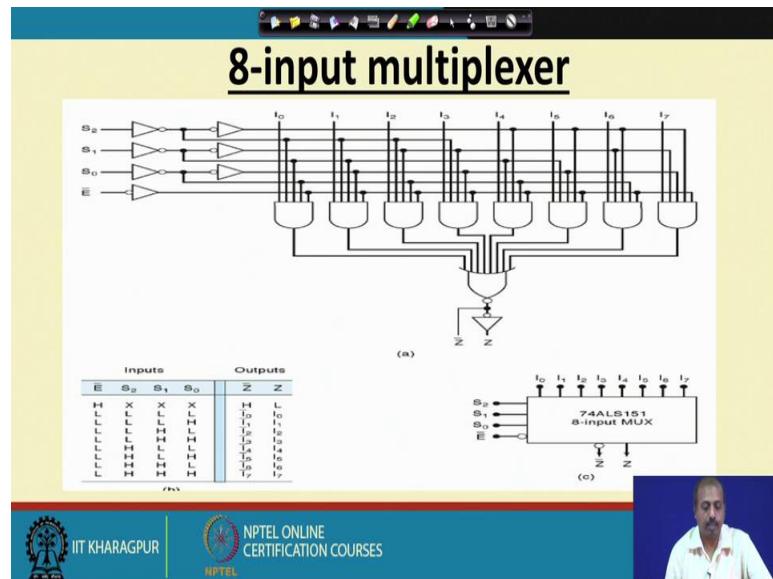


So, this D₀, D₁, D₂, D₃; suppose the values are going like this. Now the SELECT line when it is both of both the inputs are 0, then these D₀ line is selected at the output. When the this S₀, S₁ is having 0 1, then this D₁ line is selected. So, for this much time D₁ is put at the

output. After that S_0, S_1 has become 1 0 and it is 1 0. So, D_2 output will be put at the, D_2 input will be put at the output. So, it is done here. Then it is 1. So, D_3 input will be put. So, it has come here. After that S_0, S_1 again has become 0 0. So, in that case again D_0 will be copy to the output then it is 0 1. So, we will get say this D_1 copied to the output. So, this way it goes ok.

So, by looking so if a varying bit varying signal is applied to this input symbol input D_0, D_1, D_2, D_3 and this S_0, S_1 1 SELECT lines so they also are having some bearing time bearing signal. Then you can draw the corresponding timing diagram like this. So, the binary state of the data select inputs during each interval determine which data input is selected. So, this S_0, S_1 values it will determine which input is selected. Here the data in this particular example, so they are going by a repetitive sequence. So, 0 0, 0 1, 1 0, 1 1 that is it is first selecting D_0 then D_1 then D_2 then D_3 ; again it is selecting D_0, D_1, D_2, D_3 etcetera. So, it is going in the sequence. So, as a result it produces this type of waveform of course, it is specific for this example.

(Refer Slide Time: 28:38)



So, for 8-input multiplexers, so we can have it like this. So, there should be, so we assume that here we have got one enable line as well. So, this is the truth table that we are followings. So, when this enable line is \bar{E} . So, when this \bar{E} is equal to 0, then only this multiplexer is enabled otherwise it is not. So, E bar equal to 1, so this multiplexer is

disabled. So, this \bar{Z} is high and Z is low. So, it is not selecting any of the input. So, it is outputting just a low here.

Now, if this \bar{E} is low that is it is the multiplexer is active. Now it depends on the values of S₀ S₁ and S₂. If all of them are low, then this I₀ line it will be coming to Z as a result \bar{I}_0 will be coming to \bar{Z} . So, this is one particular chip 74ALS 74151. So, it has got 8 input. It has got 8 DATA input, 3 SELECT inputs, 1 Enable line and it has got both outputs and its complemented version. So, Z and Z bar both are available.

So, in this way internally we can think that the circuit is consisting of something like this, which produces the proper output depending upon the values of S₀, S₁, S₂ and the enabled line E.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 25
Decoders, Multiplexers, PLA (Contd.)

We can look into some chips that realizes this multiplexers.

(Refer Slide Time: 00:21)

MSI Quad Two-input Multiplexer

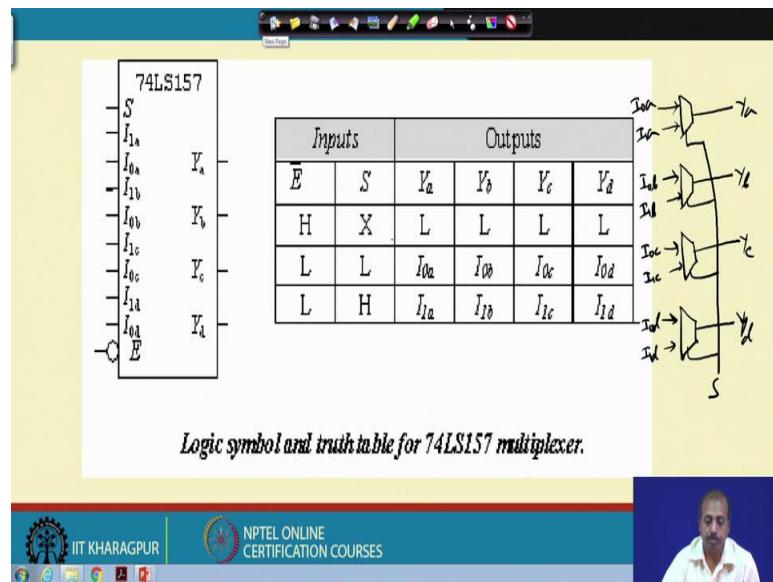
- The 74LS157 contain of quad two–input multiplexers, $I_{0a} I_{0b} I_{0c} I_{0d}$ and $I_{1a} I_{1b} I_{1c} I_{1d}$.
- The logic symbol and truth table is shown in Figure.
- Notice that each of the four multiplexer shares a common data select line and a common *Enable*.
- Each multiplexer has only one data select input because there are only two groups of inputs to be selected.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

For example, this 74157 chip: so it has got Quad Two-input multiplexers; that means, it has got in a in the chip there are 4 multiplexers. So, if this is the full chip, so there are four multiplexers ok. So, they are all Two-input multiplexers that there are 4 such multiplexers. And this multiplexers, so they have input a 0a 0b so, $I_{0a} I_{0b} I_{0c}$ and I_{0d} has the 0 inputs and $I_{1a} I_{1b}$. So, in the first multiplexers; so this is multiplexer numbers 0. So, it has got I_{0a} and I_{1a} as input.

The second multiplexer is multiplexer 1. So, it has got it has got I_{0b} and I_{1b} , so that way then other. So there are 4 multiplexer, one is a b c and d. So, any multiplexer, it has got the so if the multiplexer a, it has got input I_{0a} and I_{1a} multiplexer b, it has got inputs I_{0b} and I_{1b} . So, like that and the select lines are same; so for all of them, so there is one select line only. So, we can just the select line and that enable lines. So, they are common so, for the all multiplexers.

(Refer Slide Time: 02:00)



So, will see how this is working. So, for that we need to look into the corresponding truth table. So, this is the symbol that we have. So, there is a select line, there is an enable line. So, for this multiplexer quad multiplexer to operate, this E line must be equal to 0, then only this multiplexer will operate. Now, this is the actually the \bar{E} . So, if \bar{E} equal to high, then irrespective of the value of S the select line. So all this outputs are equal to low.

However if so, if \bar{E} is low that is then activates enable signal is given to 7 4 1 5 7, then if the select line is low, then in the Y_a output I will get I_{0a} , in the Y_b output I will get I_{0b} , Y_c it get I_{0c} and Y_d I will get I_{0d} . So, essentially you can say that the structure is something like this.

So, I have got I have got 4 multiplexers. So, I have got 4 multiplexers connected like this ok. So, I have got this 4 multiplexers and then the first multiplexer has got I_{0a} and I_{1a} , the second multiplexer has got I_{0b} and I_{1b} , third multiplexer has got I_{0c} and I_{1c} and the fourth multiplexer has got I_{0d} and I_{1d} and all for all this multiplexer the select line is same. So, that is coming from S. So, that is a select line. So, that is same. And this output is called Y_a , this output is called Y_b , this output is called Y_c and this is called Y_d and there is an enable line of course. So, this is the structure, the logically this is the structure that we have inside the multi inside the chip.

(Refer Slide Time: 04:11)

MSI Quad Two-input Multiplexer

- \bar{E} Input is *LOW* – allows the selected input data to pass through to the output.
- \bar{E} Input is *HIGH* – will disable the multiplexers, all of the outputs will be *LOW*.
- When $\bar{E} = 0$ and $S = 1$, the Y outputs will follow the set of I_1 inputs, that is $Y_a = I_{1a}$, $Y_b = I_{1b}$, $Y_c = I_{1c}$, and $Y_d = I_{1d}$.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Now so, \bar{E} input is low. It allow the selected input data to pass through to the output and \bar{E} input is high, it will disable the all the multiplexers and all outputs will be low. When \bar{E} equal to 0 and S equal to 1, the Y outputs will follow the set of I 1 inputs where y_a equal to I_{1a} , y_b equal to I_{1b} etcetera. Similarly when S equal to 1, it will be following the 0 inputs; so in that case Y_a will be I_{0a} , Y_b will be I_{0b} like that. So, that is the standard multiplexer.

(Refer Slide Time: 04:45)

LOGIC FUNCTION GENERATION USING MUX

Exercise 1:

Implement the logic circuit function specified in the table given below by using 74LS151 8-input data selector/multiplexer.

Input			Output	
A2	A1	A0	Y	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	2
0	1	1	1	3
1	0	0	0	4
1	0	1	1	5
1	1	0	1	6
1	1	1	0	7



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

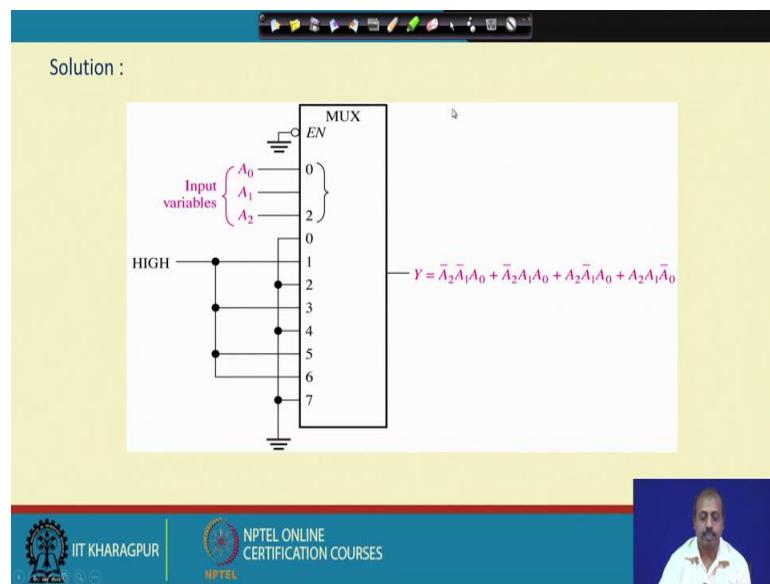


Now we will be looking into some technique by which we can realize some logic circuit function using some multiplexers. So, will take has an example 7 4 1 5 1 which is an 8 input multiplexer. So, it is 8, so 8 8 is to 1 multiplexer we can say. So, we normally we also for the multiplexers, we write it like 8 is to 1 multiplexer that is there are 8 input lines, 3 select lines and 1 output line.

Now, what we do is that so this is suppose this is the function that we want to realize. So, this is the combinational function, 3 input combinational functions. So, for the combination 0 0 1, 0 1 1, 1 0 1 and 1 1 0, the output should be 1 and for the rest of the places the output will be 0.

So, what we do first for doing it for realizing the circuit is that we note down the corresponding decimal value of the different min term ok. So, the first min term is 0, second is 1, third is 2 like that. Now we note down the places where the value is equal to 1 and accordingly we get the circuit. Like here you see that this function can be realized like this. So, what this enable line is made low, so that this multiplexer is enabled.

(Refer Slide Time: 06:07)



Now, this A_0, A_1, A_2 ; so they are connected to the select lines of the multiplexer ok. So, 0 1 and 2 and then if you look into to this place, you see for 1 3 5 and 6. The output is 1 and rest are 0 for 1 3 5 and 6 it is equal to 1. So, for data input 1 3 5 and 6, so they are tied high and this data input 0 0 then 2 and 4 and 7, so 0 2 4 and 7 for them it is low. So, it is for 0 2 4 7, they are tied low.

Now, the way it operates is if you apply some value here A 0 A 1 A 2 some pattern here, depending upon the value one of the inputs lines will one of this data inputs lines will be selected to the output and if it happens that it selects a line from this 1 3 5 and 6. Then you will get a high here or 1 here, but if it selects if the values are such that it is selecting one of this data input 0 to 4 and 7 then you will get a 0 there.

So, this way using this multiplexer, we can realize any combinational logic ok. So, we can just, what we need to do is some of the data inputs are to be tied high, some of the data input are to be tied low. And this combinational function variable, so they should be tied to the select lines of the function or select lines of the multiplexer and definitely the enable line has to be activated ok.

(Refer Slide Time: 07:56)

LOGIC FUNCTION GENERATION USING MUX-Method

- An efficient method for implementing a Boolean function of n variables with a MUX that has $n-1$ selection inputs and 2^{n-1} data inputs is given below:
 - List the Boolean function in a truth table
 - Apply the first $n-1$ variables in the table to the selection inputs of the MUX.
 - For each combination of the selection variables, evaluate the output as a function of the last variable. This function can be 0, 1, the variable, or the complement of the variable. Apply these values to the data inputs in the proper order.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

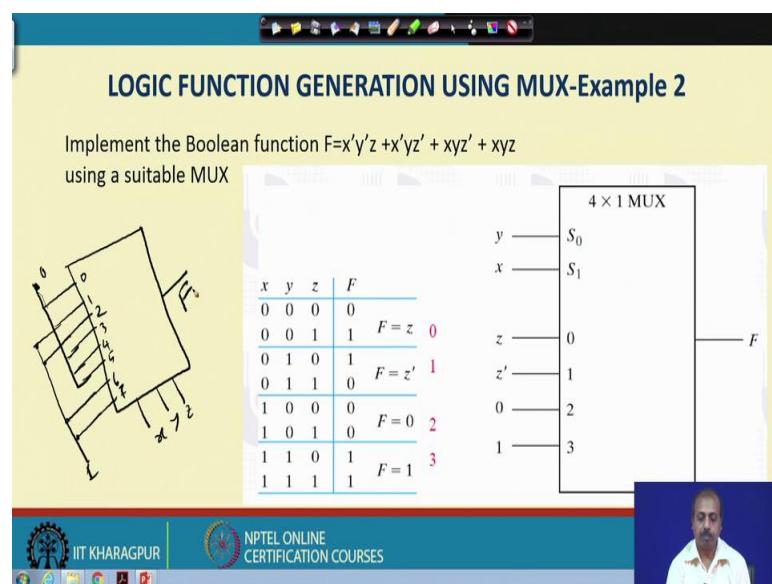
So, how can I do these things? So, this whole operations, which can be summarize like this. So, these an efficient method for implementing Boolean function of n variables with a multiplexer that has $n - 1$ select input and 1 that is and 2^{n-1} data input. So, it is like this that in this previous example in this example, what has happened is that I have got 3 input it is a 3 input function ok. So, this is a 3 input function and for that we needed one 8 is to 1 multiplexer.

So, but if I do that then, what happens is that many times the multiplexer size becomes too high. For example, in my library or I may not have this 8 is to 1 multiplexers available, may be only 4 is to 1 multiplexer are available. That means, what I what is needed is I

need to do this thing. The I have got a I have got an n variable function, but I will I will be realizing using then using multiplexers that has got n -1 select lines that is 2^{n-1} data lines.

So, what we do for this is that we the first make the truth table and the first n - 1 variables of the table, so they are connected to the select inputs of the multiplexer. And for each combination of select variables evaluate the output has a function of the last variable. And accordingly we put the variable has 0 1 or the variable or the variable itself. There is an example.

(Refer Slide Time: 09:34)



So, suppose this is the function that is given to me. So, $F = \bar{x}\bar{y}z + \bar{x}yz + xy\bar{z} + xyz$; so what we do? We have got so this is a 3 variable function. So, you can straight way realize it using 8 is to 1 multiplexers. So, like this. So, I can do it in this fashion, using an 8 is to 1 multiplexer. So, this is the select lines. So, I apply x y and z and this data lines, so these are the data lines 0 1 2 3 4 5 6 and 7 and then I know that for 1 2 6 and 7, the output should be 1. So, for 1 2 6 and 7, I tie them together to logic high. So, this is tied to logic high and 0 3 4 and 5, they should be tied to low.

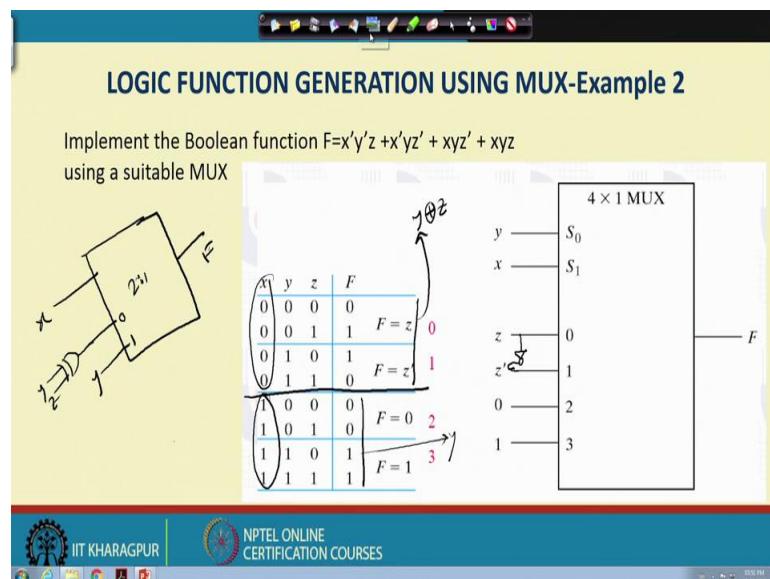
So, what I do? I take these lines and tie them to low. So, this is made 0. So, this way using, so this is my function this is my F. So, by using this 8 is to 1 multiplexer, so I can definitely realize it. But how can I do it using 4 input multiplexer.

So, for doing it with 4 input multiplexer, we have to proceed like this. So, we make the truth table and then we first of this x and y , so these inputs we apply directly to the select lines S_0 and S_1 and for the third input, I figure out like this. First, so for if I look into this first two terms, so 0 0. So, whenever this x y equal to 0 0 that forms 1 group, then 0 1 it forms the second group, 1 0 x y equal to 1 0 form the third group and 1 1 form the fourth group. So, these are the 4 data inputs like this will correspond to the input pattern S_0 x y equal to 0 0, this will correspond to 0 1, this will correspond to 1 0 and this will correspond to 1 1.

Now, if I look into the portion of the truth table where it is 0 0, so that portion of the truth table and if I do a minimization in terms of the third variable, so I will get F equal to z . So, that way in this line I apply z . Next for this is 0 1 for x y equal to 0 1, so I get here as \bar{z} ok. So, if you look into this function here, so that is \bar{z} . So, that is I getting as \bar{z} . Then this part is all 0. So, it is 0 and it is all 1. So, this is again one.

So, I take a 4 is to 1 multiplexer then of course, I need some additional circuitry. So, what is what is actually needed, we were needing is apart from the multiplexer, I need an inverter here for getting this \bar{z} ok. For getting this \bar{z} , I am needing another inverter, but otherwise I can do it like this.

(Refer Slide Time: 13:19)



So, this way you can realize circuits using different number of multiplexers. For example, if I want to do the same thing using 2 is to 1 multiplexer, then what to do? So, it is using 4

is to 1 multiplexer. So, if I have to do it using 2 is to 1 multiplexer, then this 2 is to 1 multiplexer, it will have only 1 select line and 2 data lines. So, on this select line, I apply x and this data lines are 0 and 1. So, I partition this truth table into two partitions. So, in one partition x equal to 0 other partition x equal to 1.

Now if I do a minimization, you see that here the function that I am getting is in this part, if you look into this part, the function that I am getting; so it is 1 when only 1 of y and z are equal to 1. So, this is nothing, but $y \text{ XOR } z$ and in the lower part, in the lower part I am getting the function for this part so, whenever this y is equal to 1, the output is equal to 1. So, for this part I have got the function y . So, what I can do? I can apply y here and I can take 1 XOR gate, I can take an XOR gate and take this y and z inputs to the XOR gate ok. So, I can do it like this. So, this is my F . So, if this is using 2 is to 1 multiplexer.

So in this way, you can have these multiplexers of lesser sizes to realize these combinational functions. So, if we are having same, if we are there is a n variable function. And if you have got a 2^n is to 1 multiplexer, then it is straight forward as it is reducing, then you can you have to do the grouping of this terms and accordingly you have to get the truth table. You have to get the multiplexer realized.

(Refer Slide Time: 15:15)

LOGIC FUNCTION GENERATION USING MUX-Example 2

- The two variables x and y are applied to the selection lines in that order; x is connected to the S_1 input and y to the S_0 input.
- The values for the data input lines are determined from the truth table of the function
 - For ex., when $xy=00$, output F is equal to z because $F=0$ when $z=0$ and $F=1$ when $z=1$. This requires that variable z is applied to the data input 0

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

So, next we will see so, two variables x and y have select line. So, this is the thing that we have discussing; when x y equal to 0, F we can see equal to z . So, we have also seen the 2 into 2 multiplexers like that.

(Refer Slide Time: 15:30)

LOGIC FUNCTION GENERATION USING MUX-Example 3

Implement the Boolean function $F = A'B'C'D + A'B'CD + A'BC'D'$
 $+ AB'CD + ABC'D + ABC'D + ABCD' + ABCD$ using a suitable MUX

A	B	C	D	F
0	0	0	0	0
0	0	0	1	$F = D$ 0
0	0	1	0	$F = D$ 1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	$F = D'$ 2
0	1	1	0	0
0	1	1	1	$F = 0$ 3
1	0	0	0	0
1	0	0	1	$F = 0$ 4
1	0	1	0	0
1	0	1	1	$F = D$ 5
1	1	0	0	1
1	1	0	1	$F = 1$ 6
1	1	1	0	1
1	1	1	1	$F = 1$ 7

8 × 1 MUX

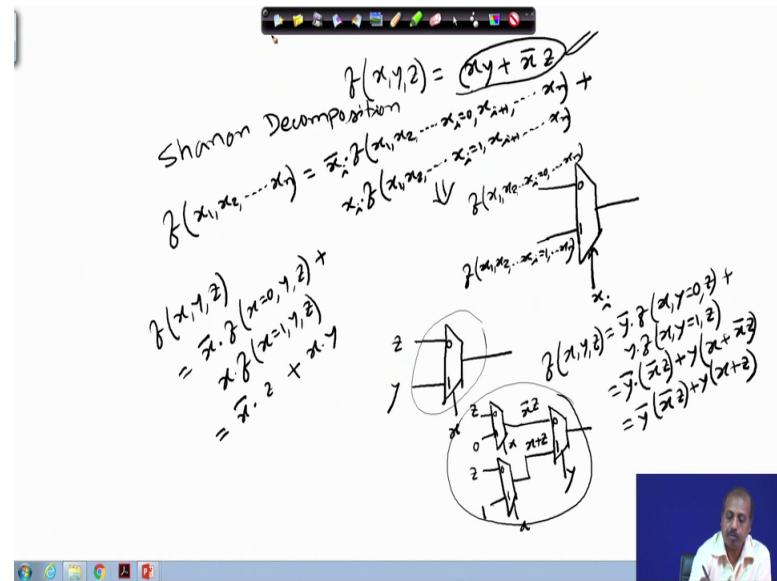
IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this is another example where I have got this 4 variable function and this I want to realize a plus all this. So, this is the 4 variable function, so I will need a 16 is to 1 multiplexer, If I want to do a directly. So, if I want to realizing it using 8 is to 1 multiplexer, then we can do it like this. So, we are applying this A B C to the select lines of the 8 is to 1 multiplexer and then we are dividing the truth table into regions of similar values of A B C.

So, first part ABC values are 0 0 0 and second part is 0 0 1 like that. And if you look into the corresponding function F, then for the first part F equal to D, second part also F equal to D, third part F equal to D', then F equal to 0. So, it goes like this. So, I can take help of a few signals a few lines and inverter to feed the data lines of the multiplexer and accordingly I can get the 4 input function realize using 8 is to 1 multiplexer or 8 by 1 multiplexer.

So, this way we can use multiplexers to realize logic functions. So, another technique by which we can realize this multiplexer based circuit is where we do not go for higher degree multiplexer, but many times we want to realize the circuits using 2 is to 1 multiplexer only. So, if you want to do that say I have got some function F which is a 3 variable function, So, x y and z and this is the function is given by say $xy + \bar{x}z$ like this ok.

(Refer Slide Time: 17:14)



Now, if you want to realize it using 2-to-1 multiplexers, then we can do it like this. So, there is one decomposition which is known as Shanon Decomposition. So, this Shanon Decomposition in the most generic form it tells that if I have got an n-variable function $x_1, x_2 \dots x_n$ then this can be decomposed around the variable x_i , and the decomposition is given by $f(x_1, x_2, \dots, x_n) = \bar{x}_i f(x_1, x_2, \dots, x_i = 0 \dots x_n) + x_i f(x_1, x_2, \dots, x_i = 1 \dots x_n)$. So, if both the cases the function f reduces to a function of

$n-1$ variable because 1 variable has been fixed to some constant. So, that variable is done. So, you have got function of $n - 1$ variable. So, this is known as Shanon Decomposition.

So, if you apply the Shanon Decomposition, so essentially you see what happens is that if I have got this function F ; then if I have got a 2-to-1 multiplexer, then I can do it like this. So, this is the this is the input 0 this is the input 1 and here in the select line I apply the variable x_i and when x_i equal to 0. So here I have to somehow realize the function. The first function $x_1, x_2, \dots, x_i = 0 \dots x_n$ and here I have to realize the function $x_1, x_2, \dots, x_i = 1 \dots x_n$. Then again these functions can again be decomposed using Shanon Decomposition around another variable ok. So, so that way it goes on.

So, let us take an example say this function that we have talking about. So, if we decide that we will decompose it around single variables say x , then what happens is that $f(x, y, z) = \bar{x}f(x = 0, y, z) + xf(x = 1, y, z)$. This is on the definition of that Shanon Decomposition.

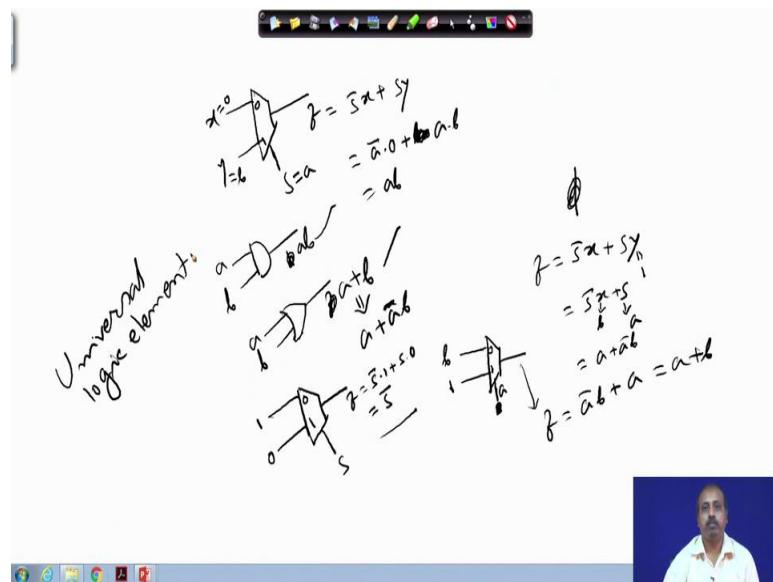
Now in this function, if I put x equal to 0; so what you get is z. So, it says that it is $\bar{x}z$ and it tells that if you for the second case if you put x equal to 1 here; so you are getting y. So, this is xy . So, it tells that you take a multiplexer 2 is to 1 multiplexer and then if you apply x as the select line, then you will be getting the so you apply, so this is 0 and this is 1. So, you apply z here and y here. So, this is the realization of the function using 2 is to 1 multiplexer.

Now, if somebody instead of taking x as the variable around which you decomposed, so takes say y as the variable around which to decompose. Then what happens is so, you gets a realization has $f(x, y, z) = \bar{y}f(x, y = 0, z) + yf(x, y = 1, z)$. So, that is \bar{y} into if I put y equal to 0 here. So, I will get the function $\bar{x}z$ and then if I if I put y equal to 1, then this gives me $x + \bar{x}z$. So, that is nothing, but $x + z$. So, here I am getting $x + \bar{x}z$. So, by Boolean algebra so I can say it is $x + z$. So, it says that at the top level you have got this realization, if you apply y as the control input select input. Then you have got so, here I need to realize function $\bar{x}z$ and here I need to realize the function $\bar{x}z$.

So, do the same thing now. On this $\bar{x}z$, we apply Shanon Decomposition. So, if I apply Shanon Decomposition, then if I select x has the select line; if I put x has the select line like this, if I put x has the select line then when x equal to 0, so this is z and when x equal to 1, so this is 0. So, I can do it like this and for realizing this $x + z$ again, I can do it using Shanon Decomposition around x. So, when x equal to 0, so this is z and when x equal to 1 this is 1. So, this multi 2 is to 1 multiplexer based realization. So, it realizes the same function as this one, but you see depending upon the variable around which we decomposed in Shanon Decomposition in case you need single multiplexer, in another case you are requiring 3 multiplexer. So, that can happen.

So, that way any n input function it can be realized using only 2 input functions and this whenever I am requiring say some complimented input also so, sometimes so, you can also you can also do it using this input said inverter also like whenever we are having a 2 is to 1 multiplexer; so, 2 is to 1 multiplexer the function that we have is say x y and this is the select line s this is the output f. So, f is given by $\bar{s}x + sy$.

(Refer Slide Time: 24:25)



Now, you can use this structure to realize AND gate, how? So, for AND gate, I want that if I have got two inputs a and b, it should produce a 1 here. So, you see that if you it should so, it should give me the function $a \cdot b$. So, you see that if I apply a at S, if I apply a at s and b at y and if I make this x equal to 0, then this function transform to something like this. So, $\bar{a} \cdot 0 + ab$. So, $\bar{a} \cdot 0$ is 0. So, you are getting ab .

So, this way I can realize 1 AND gate using this multiplexer 2 is to 1 multiplexer. Can I get an OR gate? So, for getting OR gate, OR function what I need is $a \cdot b$ and this f should be $a \text{ OR } b$. This, $a \text{ OR } b$ is can also be written as $\bar{a}b + a$. So, you have seen that these two expressions are equivalent. So, what we do in this case? So, if I take this s as a, so if I take a say y equal to 1; so, I have I have got this expression f equal to $\bar{s}x + sy$. Now if try to draw if I put this y to be equal to 1, then this expression turns out to be $\bar{s}x + s$.

Now, if you set this s to be equal to a and this x equal to b, then this is nothing, but $\bar{a}b + a$. So, what I essentially get is that if I have a two input multiplexer, then on the line x, I put b. So, this is 0 and 1 sorry 0 and 1 and this is my s. So, on the 0 line you put b and this y and the y input. So, if this is 1 this is equal to s, this is equal to a, s equal to a.

Now, the function that you get here is f equal to $\bar{a}b + a$ that is nothing, but $a + b$. So, I can get this OR gate also realized. So, can I get an inverter? So, if I have got this function so now this s; so, this is 0 and this is 1. Now you see that if I when s is 0, the output should be 1. So, I make it 1 here and I can take the multiplexer and then the 0 input I connect 1

and 1 input I connect 0. So, accordingly I will get here, output function f as $\bar{s} \cdot 1 + s \cdot 0$ so that is equal to \bar{s} . So, I am getting the inverter also. So, I am getting AND gate, OR gate and the inverter.

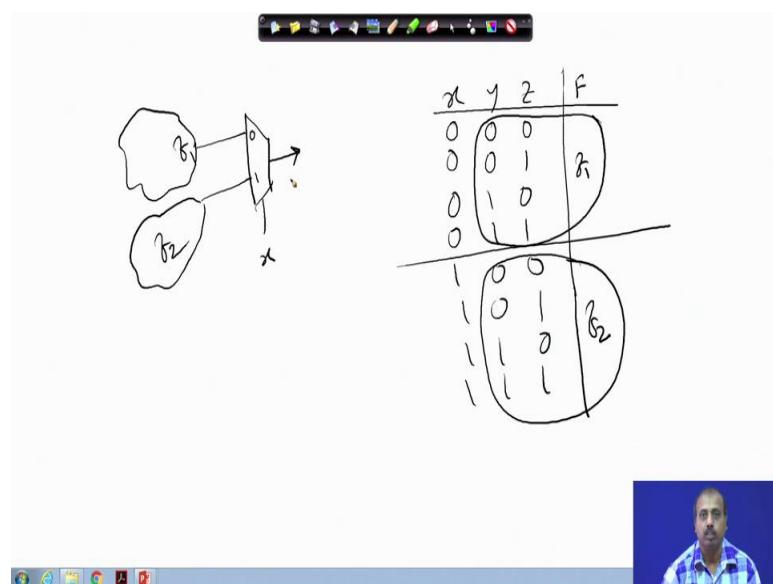
So, this 2 is to 1 multiplexer. So this is also an universal logic element. So, it can realize all the logic circuit that we have we can think about ok. So, this is another universal logic element. Apart from the NAND gate, NOR gate that we have seen this 2 is to 1 multiplexer is also an universal logic element.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 26
Decoders, Multiplexers, PLA
(Contd.)

In our last class we have seen how to realize multiplexer based circuits and we have seen that multiplexers they act as universal gate. So you can realize any of these logic gates using multiplexers. However, the technique that we have seen for multiplexer based circuit realization is that, we start with the truth table of the function and then we partition the truth table so in terms of the select variables.

(Refer Slide Time: 00:42)



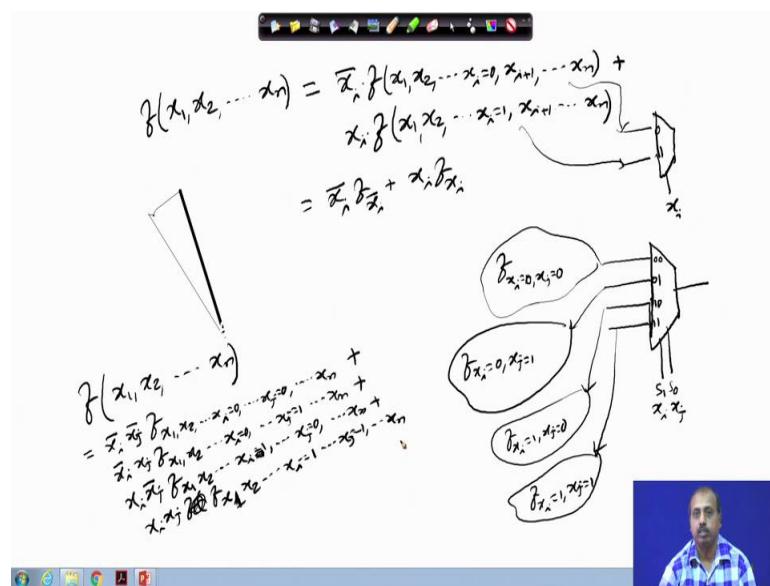
So, what I mean is that if I have got a 3 variable function say a with the variables x y and z and I need to realize it in terms of in terms of a 2 to 1, 2 is to 1 multiplexer then I am doing a partitioning, where this say I am doing a partitioning where this these are all the combination that I have. This is the truth table combination and what we did is we if I have a 2 to 1 multiplexer here then this if x is fed as the control input, then we divide this truth table into 2 parts ok.

So, x equal to 0 and x equal to 1, for x equal to 0 we realize the function given by this and for x equal to 1 we realize the functions. So if this function is say f_1 and this function is

say f 2 ok. So we realize the 2 functions in f 1 and f 2 and this is the sum we put some combinational gates to realize this f 1 and f 2 and that gives the output.

Now, the problem with this type of technique is that, if the truth table is very large and that is the number of variables are a bit high say more than say 5 or 6 then it is very cumbersome to write down the full truth table and we are partitioning the in this way. A better approach for doing this is via the Shannon decomposition that we have seen and in our last class we have seen that Shannon decomposition around a single variable.

(Refer Slide Time: 02:20)



So, if I have got a say and you have got an n variable function $f(x_1, x_2, \dots, x_n)$ so this can be decomposed around variable x_i as $\bar{x}_i f(x_1, x_2, \dots, x_i = 0 \dots x_n) + x_i f(x_1, x_2, \dots, x_i = 1 \dots x_n)$. So, this is Shannon decomposition around a single variable.

Now, this is suitable if we are thinking of realizing the circuit in terms of 2 to 1 multiplexers, because this, the input x_i will be fed as control input to this stage and the function so they will be getting for 0 and 1. For 0 parts so this will be realized by this function and 1 will be realized by this function, so in short so this is written as

$$\bar{x}_i f_{\bar{x}_i} + x_i f_{x_i}.$$

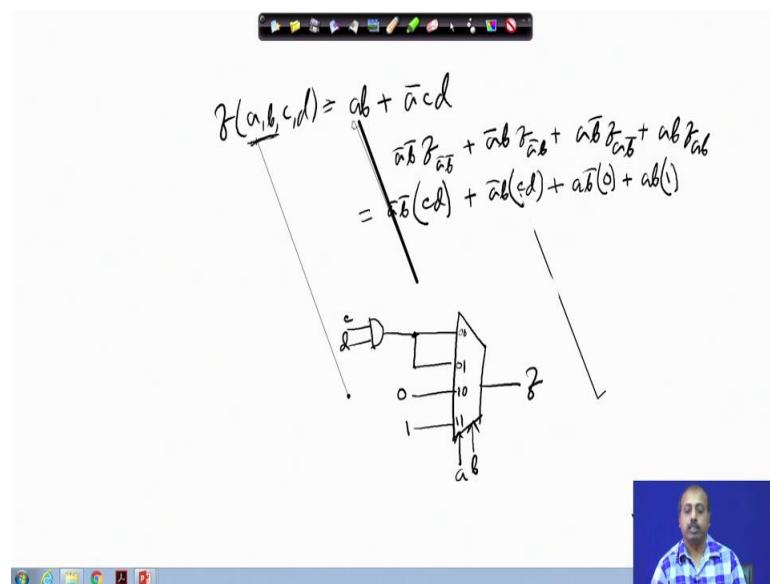
So, this is the notation so $f_{\bar{x}_i}$ means, the function obtained from f by putting x_i equal to 0 and f_{x_i} means the function obtained from f by putting x_i equal to 1 and these we have

functions $f(x_i \bar{x}_i)$ and $f(x_i)$, so they are of n minus 1 variables. Now, how can we generalize the method so if we are if we have got a 4 is to 1 multiplexer instead of 2 is to 1 multiplexer if I have a 4 is to 1 multiplexer, that is there are 2 select lines and we have got 4 input lines. So this is for 0 0, this is for 0 1, this is for 1 0 and this is for 1 1 and these are my select lines say S_1 and S_0 .

So, if I apply say some variable so x_i here and x_j there x_i to S_1 and x_j to S_0 then for at this part I will look for the function obtained by putting $f_{x_i=0, x_j=0}$. Similarly, here I will look for the function obtained by putting $f_{x_i=0, x_j=1}$, here I look for the function $f_{x_i=1, x_j=0}$ and here I will put the function $f_{x_i=1, x_j=1}$. That is I am looking for decomposition around two variables, so if I write in terms of the similar expression that I have here so we can say do it like this. So we can do it like this we can say f_{x_1, x_2, \dots, x_n} .

So, if I am decomposing around the variables x_i and x_j in this can be written as $\bar{x}_i \bar{x}_j f_{x_1, x_2, \dots, x_i=0, \dots, x_j=0, \dots, x_n} + \bar{x}_i x_j f_{x_1, x_2, \dots, x_i=0, \dots, x_j=1, \dots, x_n} + x_i \bar{x}_j f_{x_1, x_2, \dots, x_i=1, \dots, x_j=0, \dots, x_n} + x_i x_j f_{x_1, x_2, \dots, x_i=1, \dots, x_j=1, \dots, x_n}$. So, this way we can decompose function around any number of variables and accordingly we can get the functions to be sub functions to be realized in different inputs of the multiplexer, so we can take an example.

(Refer Slide Time: 07:33)



Suppose I have got a 4 variable function $f(a, b, c, d) = ab + acd$ fine. So if we want to realize it is using say 4 is to 1 multiplexer then we have to first select the choose the select

variables, so let us say that we will be using a b as the select variable. So, if we do that a b as the select variable then I have to decompose around a b, so I have to write it as $\bar{a}\bar{b}f_{\bar{a}\bar{b}} + \bar{a}bf_{\bar{a}b} + a\bar{b}f_{a\bar{b}} + abf_{ab}$.

So if I just expand it so this becomes $\bar{a}\bar{b}$ into so if I in the original function if I put a equal to 0 b equal to 0 so you get cd fine. Get the functions c d then $\bar{a}bf_{\bar{a}b}$ if I put a equal to 0 and b equal to 1, so the first term vanishes in the second term it becomes c d sorry the second term becomes c d again, because I have if I put a equal to 0 then this term vanishes and this is this becomes 1 and c d is there so this will also become c d plus $a\bar{b}f_{a\bar{b}}$.

So, so first term again vanishes and the second term since a equal to 1, so this term also vanishes. So this is 1. So my final function becomes $\bar{a}\bar{b}$, so if I say that I have got a 4 is to 1 multiplexer then this is my f and a and b are fed as input to the select lines then the data line input so this is for 0 0, this is for 0 1, this is 1 0 and this is 1 1 fine. So, here I have to put an AND gate with the input c and d the input c and d then 0 1, so it also has got cd as input so you can directly feed it here also then $\bar{a}\bar{b}$ is 0 so you can put it as 0 here and ab.1, so you can put a 1 here. So this gives me the realization in terms of 4 is to 1 multiplexer so without you going into truth table.

So, you can use this generalized Shannon expansion and go to the truth table realize, go to the multiplexer based realization of the circuit. So, as the number of select inputs increased, so you have to the decomposition becomes a bit complex and lengthy. So, the decomposition has to be for all the input permutation all the input combinations of the select variables ok. So, with that we will be going to a new topic which is demultiplexer.

(Refer Slide Time: 11:36)

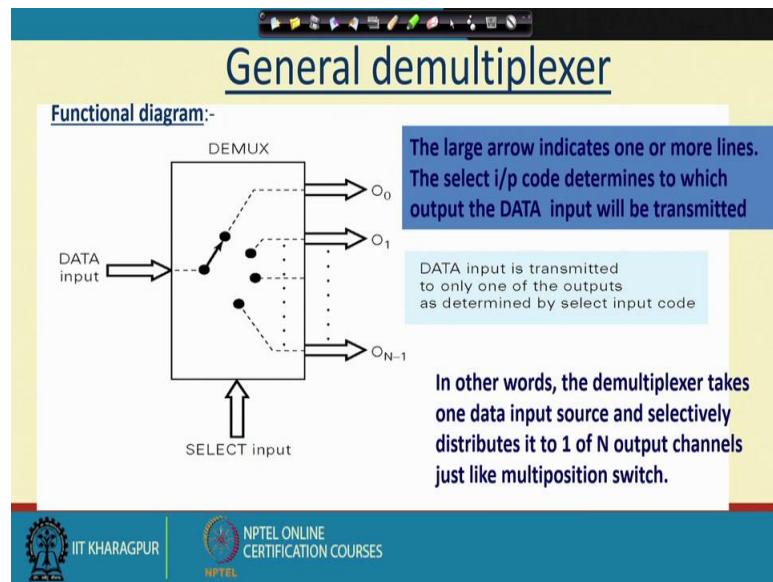
The screenshot shows a presentation slide with a yellow background. At the top center, it says **DEMULITPLEXERS**. Below that, there is a block of text: "A DEMULITPLEXER (DEMUX) basically reverses the multiplexing function. It takes data from one line and distributes them to a given number of output lines. For this reason, the demultiplexers is also known as a data distributor." To the right of this text is a small diagram showing a single input line branching into two output lines. Below this text, another block of text says: "A multiplexer takes several inputs and transmits one of them to the output." To the right of this text is a small diagram showing multiple input lines merging into a single output line. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

So, this demultiplexer is just the opposite of multiplexer, so in a demultiplexer so we have got some it reverses the multiplexing function, takes data from one line and distributes them to a given number of output lines. So, to which output line the input will go so that is determined by the select input. So, demultiplexer is also known as data distributor as if the data is coming to the block and it is getting distributed to one of its outputs. A multiplexer on the other hand it take several inputs and transmits one of them to the output, so demultiplexer is just the reverse of multiplexer and symbolically also you remember that a multiplexer we are showing it like this.

So, we are a multiplexer we are showing it like this, these are the data input so this is the output and so this is the select line so this is the way we are showing the multiplexer. Similarly for a demultiplexer we will show it like this the data input there is a single data input and there are a number of data outputs. So if I have got a single select line then there will be 2 data outputs.

So if the select line is 0 then the output will be going there, if the select line is 1 then the input will be going to this output. If it is 0 it will go to the upper the input will go to the upper output if it is equal to 1 then it will go to the lower output. Now, a demultiplexer performs the reverse operation it takes a single input and distributes it over several outputs.

(Refer Slide Time: 13:20)

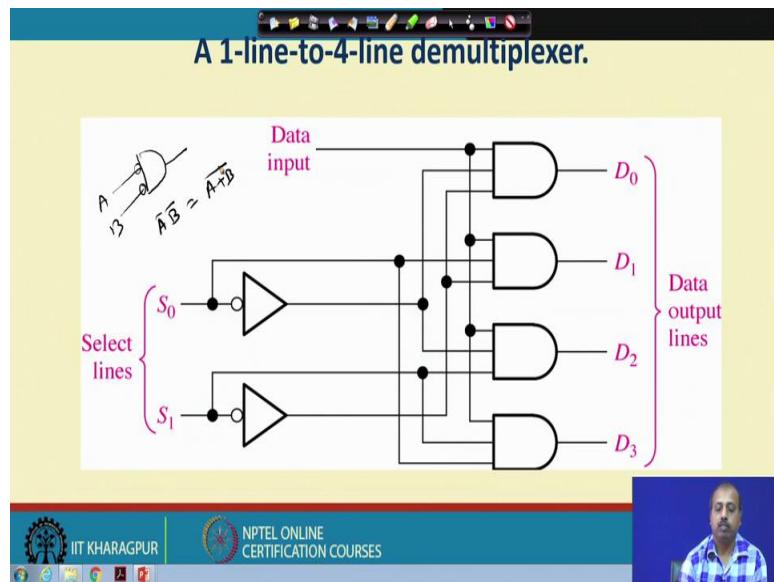


So, then also you can look at it as if there is a switch that goes from that can get attached to one of the input positions so this to transfer the data input to that line. So here this data input is coming and depending upon the select input so this switch will be set to one of these positions accordingly the input will go to one of the output, so this large arrow indicates one or more lines.

So, it may be the case that instead of having a single line so we have got a multiple line. The select input output code determines to which output the data input should be transmitted. So, it may so happen that instead of getting a single bit input so from it is coming from some block that has got say 16 bit output and the 16 bit output is connected to this demultiplexer as to the input of this demultiplexer to be transferred to a one of many such 16 bit outputs so that way it so this arrow.

So, that may not represent a single line there may be multiple lines also, so data input is transmitted to only one of the outputs as determined by the select input code. So, demultiplexer takes one input data source and selectively distributes it to one of n output channels just like the multi positions switch, there is a multi position switch so it is doing it like that.

(Refer Slide Time: 14:46)



So, as far as the logic level diagram is concerned so this diagram shows a 1 line to 4 line demultiplexer or 1 is to 4 demultiplexer. So multiplexer we are writing like 4 line to 1 line or 4 is to 1 multiplexer, for demultiplexer we will write as 1 line to 4 line or 1 is to 4 demultiplexer.

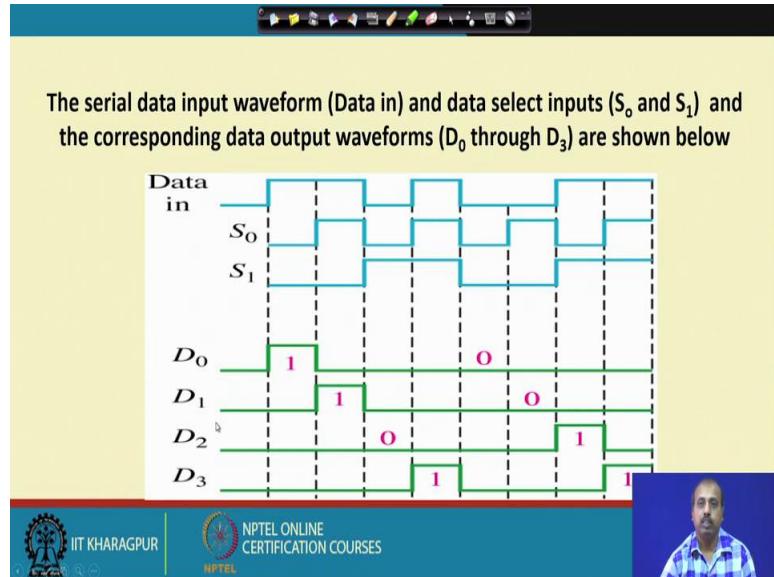
So, here we have got the select lines S_0 and S_1 both are 0 0 then what will happen see the this is 0 and this is also 0. So as a result all these 3 and gate output so they will be 0 fine, only so this d_0 so it gets the input from this inverter and it also gets the other input also it gets from the from the output of this inverter.

So, that way it they are coming as 1 and these data input is coming, so whatever be the data input that will pass on to d_0 where as $d_1 d_2 d_3$ they will get 0s. Now, this some so here you get a new notation, so this bubble of the inverter so it is shown before the inverter; So that is also quite common so it does not matter, so it as if the input is inverted here and then buffer or if the inverter is at the output; that means, the input is buffered and then inverted. So that way it is same previously also we have seen that these bubbles can be put before the gate or after the gate.

So, you have to go by the logic and say what is the function realized for example, if I have got a gate like this and here I have got bubbles then I know that the function realized is $\bar{A}\bar{B} = \overline{A+B}$, so that is a nor gate ok. So, by applying Demorgan's theorem we can get it

so this way we have to look into the schematic that we have the logic symbols that we have and interpolate what is the corresponding signal.

(Refer Slide Time: 16:56)

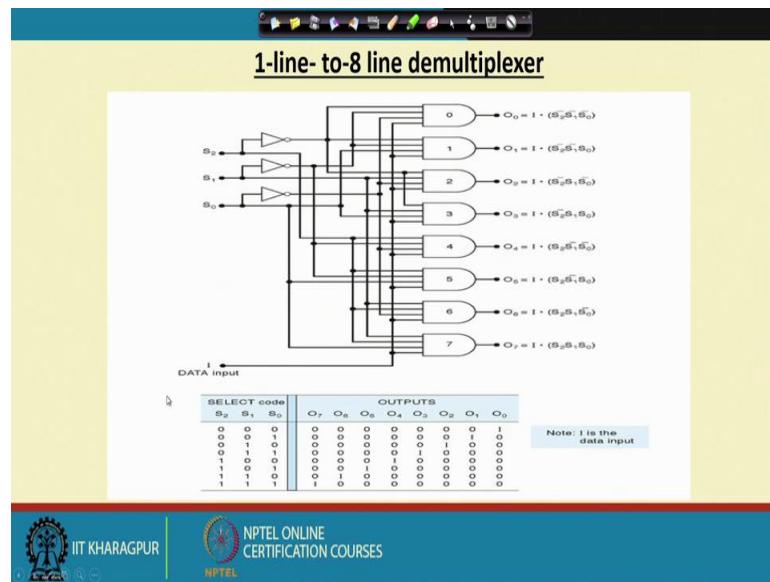


Typical, timing diagram may be like this so this is the serial so data input suppose I have got a data input like this and this S_0 , S_1 so they are trans doing some transmit they are going like in this fashion. So, S_0 is toggling every in every such boundaries at these dotted boundaries and S_1 is toggling every alternate such dotted boundary. So, initially both are 0 so D_0 will be getting the value of data in and others will all be 0 irrespective of the value of data in after sometime.

So, this S_0 becomes 0 and S_0S_2 becomes 1 and S_1 remains 0, so as a result D_1 get selected so D_1 get this data input at its output and others become all others becomes 0. So in this way you see that at any point of time only one of the outputs will get the value of the data in with it so others will be always others are all 0.

Now, in the of course, you can say that it does not distinguish between 0 which is a coming from the data input and a 0 which is occurring due to non selection of the particular output. So, we will see later that we can have some concept of tri state buffer by which we can get rid of this situation and we can come to a new logic level which is neither 0 nor 1, so that will see later.

(Refer Slide Time: 18:25)



So 1 1-line-to-8 line demultiplexer so that previously what we have seen is a 2 to 4 demultiplexer 2 1 line to 4 line de multiplexer. So, this is 1-line-to-8 line demultiplexer, so naturally I need 3 select lines S_0, S_1, S_2 and then we have got a number of and gates and this symbols are ANDed like say this is this is this output is 1. So the here I do not have any data input to be transferred.

So you only say that if it is which one is so this sorry the data input is there and data input is transferred to O_0 if this S_0, S_1, S_2 all are 0s, in that case the data input goes there. If it is equal to 0 is 1 and S_1, S_2 are 0s then this O_1 will get the value of data inputs. So this way this logic circuitry has been made accordingly in the truth table you can see that this I can the input I can pass can come to only 1 of the outputs O_0 to O_7 at any point of time depending upon the select lines S_0, S_1, S_2 .

(Refer Slide Time: 19:38)

The slide is titled "Buffers". It contains the following text:

- Buffer:
 - Doesn't change the input.
 - Only amplifies.

Below the list is a hand-drawn symbol of a buffer, which is a triangle with a curved arrow pointing from the input to the output. To the right of the list is a schematic diagram of a buffer. It shows a triangle symbol with an "in" terminal on the left, an "EN" (Enable) terminal at the top, and an "out" terminal on the right. A small checkmark is present near the "in" terminal. At the bottom of the slide is a navigation bar with icons for back, forward, search, and other controls, along with the text "IIT KHARAGPUR" and "NPTEL ONLINE CERTIFICATION COURSES". On the right side, there is a video player window showing a man speaking.

Next we will be looking into another fundamental concept which is known as buffer, so what happens is that may when whenever we are transmitting signals over long distance so due to this noise and this RC drop of the line. So the voltage level of the signal drops so maybe you have transmitted a logic high, but due to the length of the wire at the other end when you measure, so you do not get a strong logic high so you get some intermediary value or in the worst case it may even become a logic low level. So, what is required is that we should do some amplification of the signal so such signal voltage level has to be restored to the proper values. So, if I have a line like this so if I have a line like this.

So, what I means so at this point I am sending some value signal S at this point when I measuring S , I find that it is not that strong so what is done in between we put some buffer like this buffer as it is shown here. So, this buffer will amplify the signal so it does not so it is basically a true value transfer, so whatever is coming as input will go to the output provided this enable line is equal to 1, if this enable is equal to 1 then in output equal to input. So it does not do any other logic change to the signal in.

(Refer Slide Time: 21:13)

Three-State Buffers

- Buffer output has 3 states: 0, 1, Z
- Z stands for High-Impedance \cong Open circuit

EN in out

EN	in	out
0	X	Z
1	0	0
1	1	1

EN = 0 \rightarrow out = Z (open circuit)
EN = 1 \rightarrow out = in (regular buffer)



IIT KHARAGPUR



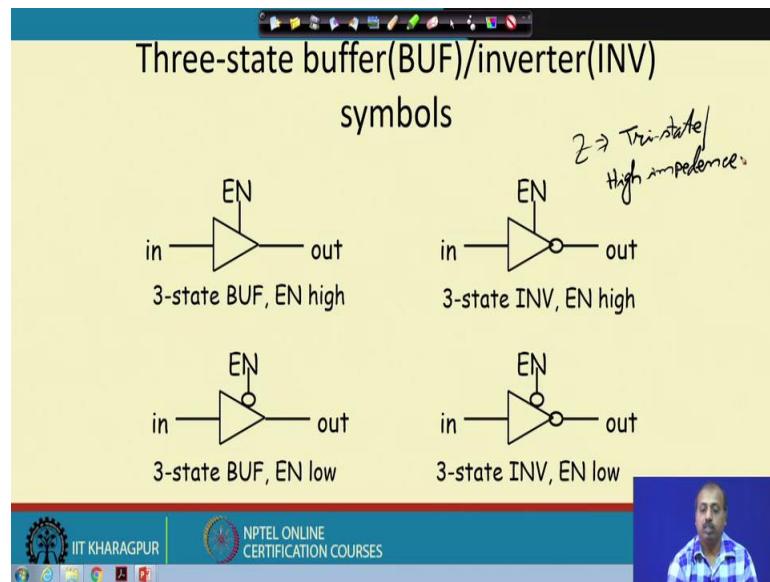
NPTEL ONLINE
CERTIFICATION COURSES



So, so, these are also known as 3-state buffers because, when this what happens when enable equal to 0 when enable equal to 1 then I have said that output gets input, so what if output sorry enable line is equal to 0. So, this type of buffers where we have got an enable line so they have got 3 output states 0, 1 and Z.

So, Z is Z is known as the high impedance state or open circuit state, so as a as if the circuit is open it is not connected to any voltage source. So, enable equal to 0 output equal to Z which is equivalent to the open circuit and enable equal to 1 output is output is equal to in that is the regular buffer operation that will occur there. So, enable equal to if we if we do not want the output to get the value of input.

(Refer Slide Time: 22:14)



So, we can put enable equal to 0 and that will stop the output getting the value of input. Now, there are many such 3-state buffer configurations so is so I can have these buffer with enable high, enable high means when this enable equal to 1 then only this output will get the value input. Sometimes we have got a so this is a buffer though this is a 3-state inverter so this is a basic inverter logic, but before that we have got so we there is an enable line also so if the enable is high then only this circuit will act as an inverter otherwise in this output will be tri stated.

So, output is the output will get the value Z so this Z is also known as that tri state. Z that I am talking about so that is also called the tri state. There are several names one name we have already seen the high impedance state, the high impedance state or the tri state state like that we have got several names for this.

Now, this 3 so this so we have got this thing we have got this enable and if enable is 1 so this is a basically an inverter with some enable line. This enable can be the enable itself may be an inverted enable that is this buffer will get the value of output; output will get the value of in provided this enable line is equal to 0. If enable line is low then output gets the value of input and if these enable line it is similarly I can have an inverter where the enable line itself is low. So, this is we have got this enable equal to 0 then only this circuit acts as an inverter, so that is possible. So, we can have various combinations of this buffer.

(Refer Slide Time: 24:05)

Open Collector/Drain Gates

- Outputs of two (or more) gates must not be wired together:

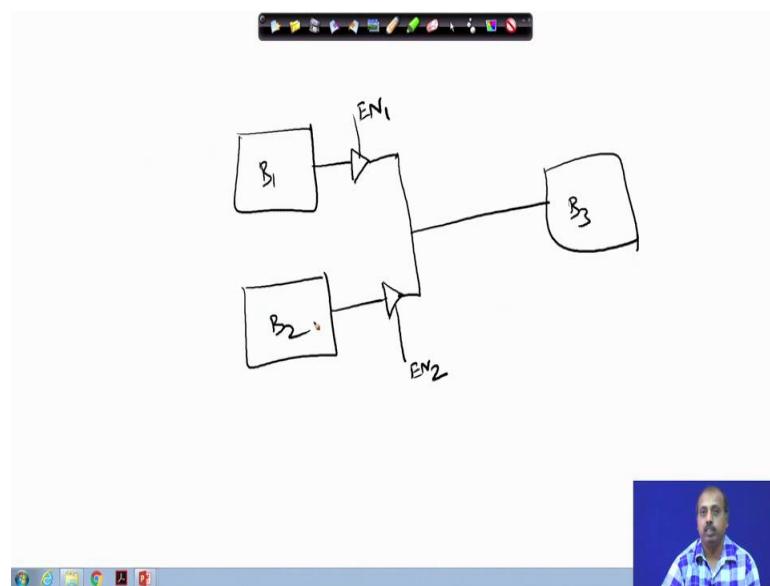
If $A = B$
→ $\text{out} = A = B$

If $A \neq B$
→ a large enough current
can be created, that causes excessive
heating and could damage the circuit

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will look into let us see like why this is necessary? So this is necessary because many times what happens is that we have got different logic blocks they are giving me some sources.

(Refer Slide Time: 24:26)



So, this is see this is say block 1 whatever it is, so this is block 1 this is block 2 so they are all logic signal they are all logic blocks computing some logic operation and then we need to give this output of both the blocks to block B₃. So block B₃ so this output should be this input should get it should be coming from the output of B₁ or B₂, but at any point of time

only 1 of them has to be connected and sometimes this input may not be connected also what I mean is that sometimes this B_3 gets the value from B_1 or B_2 or it may be that it is not connected at all.

Now, we will see later that if I do it like this if I connect it in this fashion, so what I really want is something like this that these are the outputs from B_1 and B_2 and this is going there to B_3 . Now, this type of connection is difficult because what may happen is that there may be conflict in the outputs of B_1 and B_2 and in that case so what is the voltage level at this point so this becomes a question fine.

So to avoid that situation so we can use the tri state gates, the high impedance gate so what we do is we put buffer here sorry. So let us take a new page. So, we take we take we have got this B_1 and output of it is put to a buffer and we have got a block B_2 and this output of B_2 it passes through another buffer and this two are now connected and that goes to the block B_3 fine. Now, this enable line so they will be so this is say enable 1 and this is enable 2.

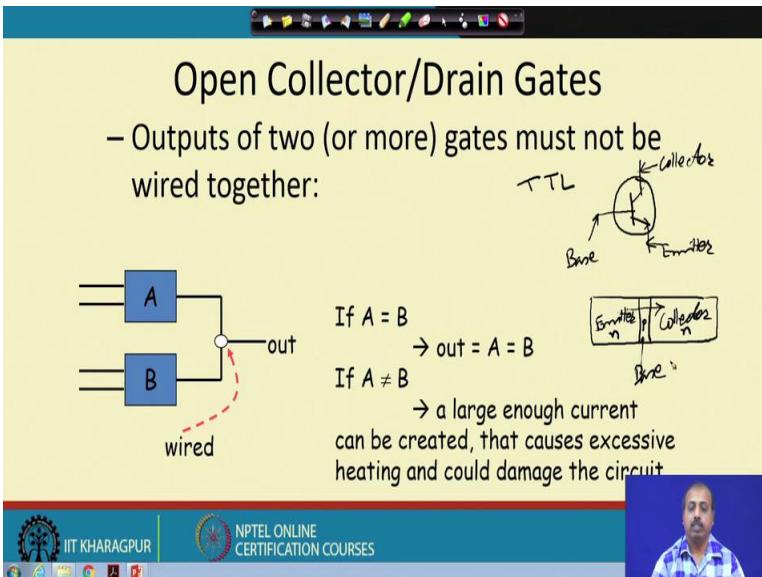
So, if enable 1 so this B_1 output will be coming to the input of B_3 if enable 1 is high and likewise a B_2 will B_2 output will come if enable 2 is high. Now, you see that if enable 1 and enable 2 they are not activated simultaneously then I do not have any problem. So, this B_1 output is never getting connected to shorted to B_2 output ok, so that will not happen. So, this way we can use this tri state logic for realizing for realizing this type of connections where multiple outputs from different blocks they may be connected to a common input of another block.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture - 27
Decoders, Multiplexers, PLA
(Contd.)

Next we look into another problem that occurs when we are connecting outputs of 2 or more logic gates together or say logic blocks together.

(Refer Slide Time: 00:26)



Open Collector/Drain Gates

- Outputs of two (or more) gates must not be wired together:

If $A = B \rightarrow \text{out} = A = B$
If $A \neq B \rightarrow$ a large enough current can be created, that causes excessive heating and could damage the circuit.

TTL
Collector
Base
Emitter

Emitter of Collector

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this comes under the broad heading called open collector or open drain gain. Now this, so, this if you remember this term drain we have seen when we talked about this CMOS transistors and this collector. So, collector term we have not seen, so this is basically, this is basically coming from the fact that previously if you look into the in the logic families, so there was a family called TTL Transistor Transistor Logic and in the transistor transistor logic, so the circuits were realized by means of transistors. So, a transistor has got 3 terminals, one is known as the base terminal. So this is the symbol of a transistor, this is the symbol of a transistor. So, this terminal is called the base terminal, this is called the collector terminal and this is called the emitter terminal ok.

So, architecture wise it is like this so we have got this, if you say that we have got a silicon wafer then some part of it is the emitter then after that we have got a thin base, so this

region is called base and then this is called the collector. Now this particular transistor that I have drawn here this emitter is of n type, base is of p type and this collector is of n types, so this is an n p n transistor it is called ok.

So, electrons they go from emitter cross this by base and reach the collector region so that way the current flow is done. Anyway so, that is a different issue so this logic gates can be realized using this type of transistors also because, that also acts as switch, now the current that can be drawn like say if I have got some circuitry here.

(Refer Slide Time: 02:36)

Open Collector/Drain Gates

– Outputs of two (or more) gates must not be wired together:

If $A = B$
→ $\text{out} = A = B$

If $A \neq B$
→ a large enough current
can be created, that causes excessive
heating and could damage the circuit

IIT Kharagpur | **NPTEL ONLINE
CERTIFICATION COURSES**

So, if I have got some circuitry here which is say TTL circuitry and then the so it is connected to the supply voltage which is in general known as **VCC** So, this is the output of the, this is the some logic is realized here and this is output of that logic and it is feeding some load here, load is generally shown as a capacitive load because the charging current is necessary for taking the particular signal point to logic high. So, the amount of current that can be drawn here is limited ok.

So, it is limited by the TTL family TTL logic family so it is not very high. Now so, if you have to connect large number of input large number of inputs from this output so it creates problem. So, another difficulty that we have with this TTL family of logic is that suppose here we have got 2 blocks A and B, we have got 2 blocks A and B and then. So, this outputs of A and B so they are shorted together now if A equal to B then output is so there is no conflict so A and B are same so output also is equal to that value.

However, if A is not equal to B so, what can happen is that say A equal to, A output is 1 and this output is 0, so that means, A is trying to pump in current like this and B is trying to sink current like this. So, as a result there is a there may be a large amount of current flow through this circuit ok.

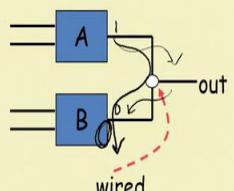
So, a large enough current that can be can get created and this current may be so high that this transistors that are there in the lower part of this B the logic, so that may not be with may not be able to which stand that current. So, it may damage the circuit, so a large enough current can be created that causes excessive heating and could damage the circuit, so what is the way out ok.

So, this open collector is a solution to this problem and so this is also there are there are on the CMOS family if you look into so they are called open drain connection. So, open drain so you know that in CMOS it is in CMOS there are 3 terminals like this one is called drain another is called source and another is called gate ok.

(Refer Slide Time: 05:11)

Open Collector/Drain Gates

- Outputs of two (or more) gates must not be wired together:



If $A = B$
 $\rightarrow \text{out} = A = B$

If $A \neq B$
 \rightarrow a large enough current
can be created, that causes excessive
heating and could damage the circuit

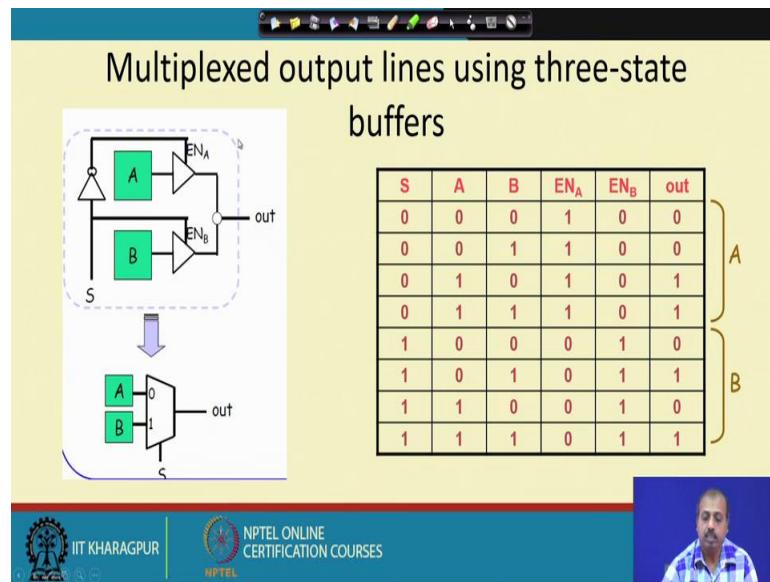
D
G
S

 IIT KHARAGPUR  NPTEL ONLINE
CERTIFICATION COURSES



So, here also with the if this drain is connected normally it is connected to V_{DD} side and this is ultimately connected to the ground and this G is getting the input. So, in the process, so it may so happen that I mean some of the logic family so this drain is not connected to the supply voltage, because of the same reason that if you have to drive a large load it may create some problem.

(Refer Slide Time: 05:41)



So, we will see how this can be solved like say here so 1 possibility is that we can use some sort of multiplexer, so multiplexed output lines using 3 state buffers. So, this solution we have seen some time back say like this A and B. So, these are the 2 block output and we put some tri state buffer here this is controlled by EN_A and this is controlled by EN_B and this S this actually for multiplexer it is select since in the multiplexer, so it will select one of these 2 outputs and that will go to the outline. So, if S equal to 0 then this input is equal to 1 and this enable B is equal to 0.

So, this B equal to if S equal to 0 so this is equal to 1. So, as a result this A output is passed through this buffer to the output. Similarly if S equal to 1 then enable B is equal to 1 and enable A equal to 0; as a result this output of B it passes through this buffer and it comes to the output. So, if you look into the truth table you see that for this combination when enable A equal to 1 and naturally in that case enable B has to be 0.

So, whatever be the value of A that comes to the output and similarly if my enable A is equal to 0 and enable B has to be 1 in that case then whatever be the value of B that comes to the output. So, this way we can realize this, we can we can use this 3 state buffers or tri state buffers for buffer this type of multiplex realization. But you see that this is, this does not always solve the problem because, many times we do not want this type of multiplexing we just want some logic by which that will be followed when this type of shorting of 2 lines are there.

(Refer Slide Time: 07:36)

Open Collector/Drain Gates

- Outputs of **some** gates can be wired:
 - The result: $O_1 \text{ AND } O_2$
 - Open Collector Gates in TTL Technology
 - Open Drain Gates in CMOS Technology

The diagram illustrates a logic circuit where two outputs, O_1 and O_2 , are connected to the inputs of a tri-state AND gate. The output of this gate is labeled $\text{Out} = O_1 \cdot O_2$. The inputs to the tri-state AND gate are represented by small squares at the junctions of the wires from O_1 and O_2 .

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, they come under this under the broad heading called open collector or open drain gates. So, outputs or some gates maybe wired like this so if it is A and B, so they are connected like this.

Now, this so this what is the logic that you get at this point. So, that is the functionality that you get is dependent on the logic family that you are using; for example, if you are using TTL technology then this connection so this will result into AND operation, so O_1 AND O_2 so you will get the AND operation. Similarly for open drain gate since CMOS technology.

So, that will you will get a get an AND operation ok, so you will get a output as the AND of these two. Now can I connect this O_1 AND O_2 like this so logic level is fine so logic level logic value I get as AND of this and something or some things like that. So, if I do not use open collector if I use normal TTL gate or normal CMOS gate, then also I will get the output as O_1 O_2 , but the problem that will come is the current flow through the lower device, the current flow through the through the devices that can occur if one output is high and another output is low.

So, for that purpose we need some protection, so what we do we put a tri state gate here. So, this is the, this so this is known as the wired AND so this AND gate is physically not there. So, it is just the 2 wires that are connected that are shorted at this point, but it

essentially gives the AND operation ok. So, this gate is the wired AND gate we say and we but physically there is no such AND gate.

(Refer Slide Time: 09:21)

Open Collector/Drain Gates

- Open Collector (Open Drain) NAND Truth Table:

A	B	F = (A · B)'
0	0	Z
0	1	Z
1	0	Z
1	1	0

Z (Hi-Z) (Hi-Impedance):

- As if it is unconnected
- The gate cannot pull up the output
- needs a resistor to pull it up if an input is '0'

NPTEL ONLINE CERTIFICATION COURSES

So, open collector or open drain NAND truth table is like this, so if you this is open drain so if this A B is 1 1 then only output is 0, so since the problem with his open collector or open drain is that you see if this is your open drain gate, this is the open drain gate so this is my drain, so this is gate, this is source and whatever be the connection so and this lower part of the logic. Suppose this gate is equal to 1 and that we are taking the output from this point sorry not from this point we are taking the output from this point.

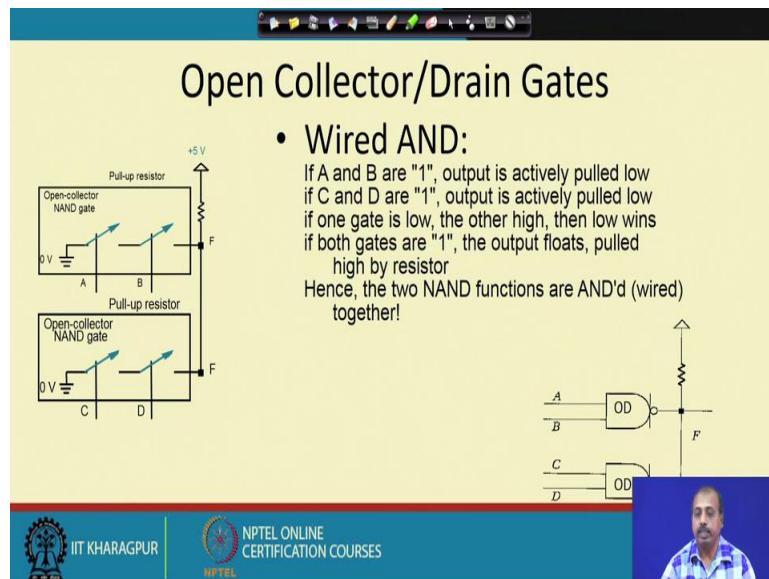
Now, if this transistor is say oh if this transistor is say on then this side is eventually connected to ground so this will be going down to 0. So, if this NAND gate both the inputs are 1, then this then there is a current flow path; like in the NAND operation this lower n network is shown like is like this, so this is A and B. So, we have got so this is grounded and suppose this is open drain.

So, I am connecting it, so this side is open it is not connected to V_{DD} , now what will happen is that if I put a 1 1 here. So, these 2 transistors will be on, so this is also on and this is also on as a result this value voltage will be come this point voltage value will become equal to 0. So, for 1 1 I am getting is 0, but if I if any of them is equal to 0 sorry if any of them is equal to 0, then I do not know what is the value here because that that side so this lower

side is cutoff and upper side is also not connected to any voltage source so this is open fine.

So, this is as it is so it is the it is high impedance or high Z as if it is unconnected. So, this if the switches so this switches are connected both of switches are equal to 1, then this switch will drop down as if I will get a connection to 0 so this way it operates. So, what to do if I do not have this out if I do not have any connection to the supply voltage? So, what it says is that you need a resistor to pull it up when one of the input is equal to 0, let us see how this is done.

(Refer Slide Time: 12:02)



So, this is the situation so we have got this point. So, we have got this if this A and B both of them are 1 then definitely this F point is getting connected to 0 and it is getting 0 here. So, but if it is if both of them are equal to 0 in that case what I will do, I will be pulling this to this point to high via this resistor ok.

Similarly, so if ABCD all of them are equal to all of them are equal to 0 ok, if all of them are equal to 0. Then so all these switches are open, so in that case at this addition of this extra resistor here so that will ensure that at this point you get a 5 volt value. So, this so this is outside the chip, so this resistor and the supply so this is outside the chip. So, from inside the chip I am getting just these switches as open. So, this point this line is open similarly this line is open and by means of this external resistor and this pull up this voltage value. So, I am getting a 5 volt value nearly 5 volt value at this point.

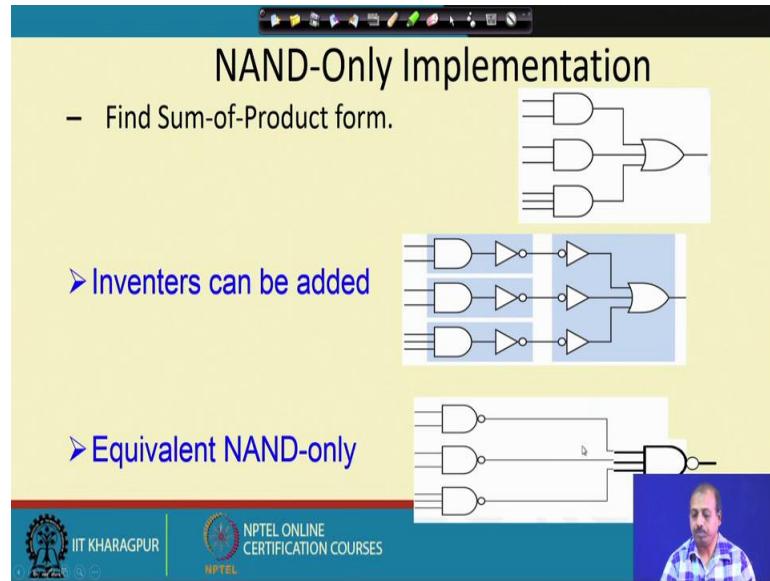
So, this is the advantage of this wired and connection so you see that pictorially you can see that the individual gates that I have, so they are actually open drain NAND gates. So, this is an open drain NAND gate and this is another open drain NAND gate. So, they are connected together and then when they are shorting so we will be connecting via this open, we will be connecting via this external resistor and pulling it up to high. So, that way it is this is going to F ok. So, this is the wired AND connection and the advantage that you get is that see depending upon the load current that you need, so you can control the value of this resistor.

So, as a result you can make this can you can make this resistance value low if you want a large current to drive the next state device next state devices, which is not possible with the normal gates because they are the current is fixed and it is fixed by the technology. So, if A and B are 1 output is pulled low so similarly if C and D are 1 then also output is actively pulled low, if one gate is low and the other gate is high then low wins.

So, that is the wired AND feature that we are getting here. If both gates are low this will be so if both gates are low.

So, this output of this so if at this point, so if both A B are 0 in fact so whatever is there so what we mean is that if we get the both 2 both of these 2 gates as open, then we can then this has to be pulled to high via this resistor ok. So, if the 2 NAND gates they function as they functions as ANDed as their functions are ANDed together so this is wired together to get the output.

(Refer Slide Time: 15:51)



So, this is the wired AND connection so we will take an example and see how this NAND implementation can be done. So, we start with this sum of product form suppose this is the sum of product realization that I have, then we add invertors between so after for every line. So, we say at the output of the, at the in the beginning of the line and at the end of the line we have got inverter. So, this inverters are added and then so this is the, this is equivalent to NAND because this AND invert inverter, so that is equal to NAND. So, this is my NAND operation and similarly this inverter, see if I take it here, so this becomes

$\bar{A} + \bar{B} + \bar{C}$. So, that is \overline{ABC} , so here also you get the NAND operation.

(Refer Slide Time: 16:44)

NAND-Only Implementation

➤ NAND = AND + NOT

➤ Use Open Collector/Drain NAND Gates:

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, that way we can get the NAND operation and NAND is nothing but AND + NOT. So, then this and part we can replace by wired AND so this AND we can forget and then we can just join this together by means or to make some wired AND connection. So, then this open collector or open drain gates NAND gates can be used here to get the wired as a final circuit realization.

(Refer Slide Time: 17:11)

Another Method

➤ Instead of finding the circuit for F, find the circuit for F' in the first stage

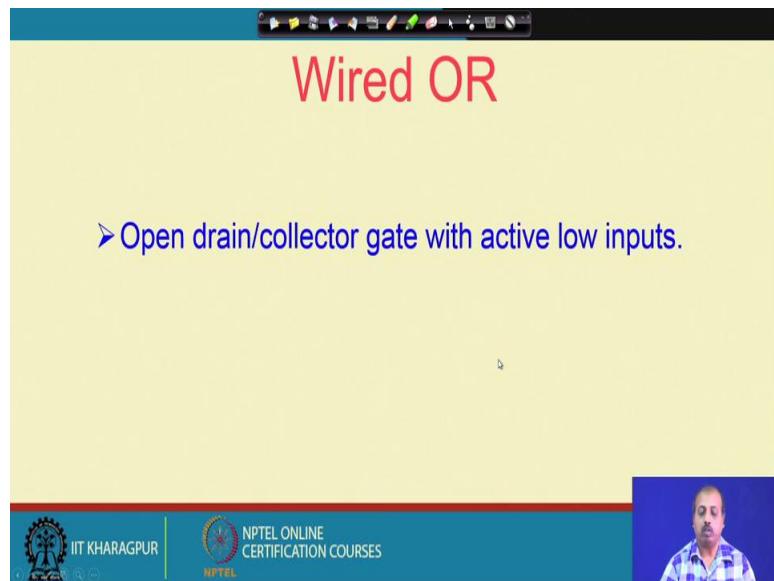
➤ Then there will be no Inverter at the output

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Another method may be like this instead of finding the circuit of F the circuit for F' found in the first stage and then there will be no inverter at the output. So, we can do it for F dash

instead of F and then we proceed like this and in that case I do not need the inverter at the output.

(Refer Slide Time: 17:30)



Similarly, we can have wired OR connection when open drain or collector gate with active low input, so the some logic families so they will tell you so if you connect wires together, so, you will get or type of logic and that or type of logic so there so that will give us the wired OR connection so that depends on the technology that we are using. So, if it support so if it says that if you connect 2 gate output it will be acting as OR gate then it will be an wired OR connection rest of the discussion remains same, so you have to pull it low or pull it high ok. So, this that way we have to put some additional resistance and taking it to high or low.

(Refer Slide Time: 18:15)

PLA Logic Implementation

Key to Success: Shared Product Terms

Equations

Example:

$$\begin{aligned} F_0 &= A + \bar{B}\bar{C} \\ F_1 &= A\bar{C} + AB \\ F_2 &= \bar{B}\bar{C} + AB \\ F_3 &= \bar{B}C + A \end{aligned}$$

Personality Matrix

Product term	Inputs			Outputs		
	A	B	C	F ₀	F ₁	F ₂
AB	1	1	-	0	1	1
$\bar{B}\bar{C}$	-	0	1	0	0	1
$A\bar{C}$	1	-	0	0	1	0
$\bar{B}C$	-	0	0	1	0	1
A	1	-	-	1	0	0

Reuse of terms

Input Side:

- 1 = asserted in term
- 0 = negated in term
- = does not participate

Output Side:

- 1 = term connected to output
- 0 = no connection to output

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

So, next we will be looking into another logic device which is known as programmable logic array or PLA. So, PLA is like this that many times what happens is that we need to realize multi output functional block. So, what I mean that if I have if we have got some function, we have got a logic block where these are the inputs. So, this A B and C they are they come as input and there are 4 outputs F₀, F₁, F₂ and F₃. Now this so this function if we want to realize suppose this is the, so these are the functions, so these functions we have got after drawing the truth table after making the Karnaugh map and so after doing the minimization so, we have got it like this.

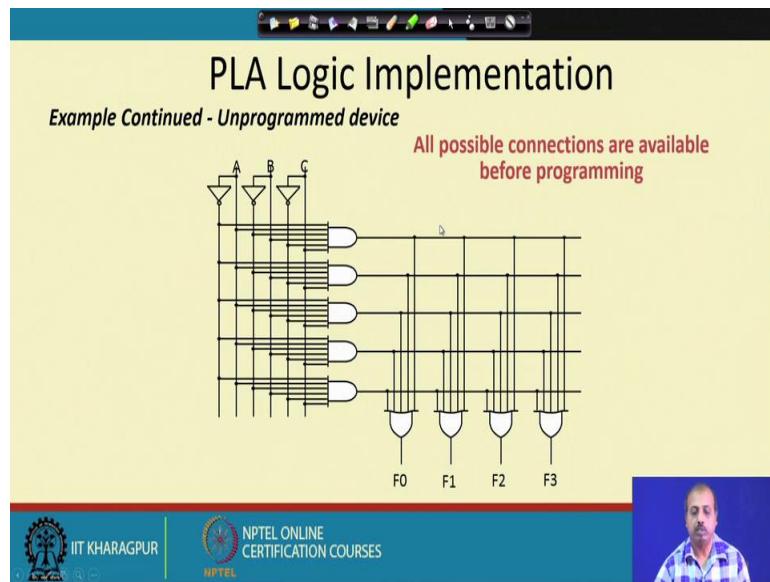
Now if you realize them separately so you will need 1 gate for doing this operation then 1 gate for doing this OR operation like that; however, if you look into the terms you see that there are many terms which are common between them; for example, say this F₀ say this $\bar{B}\bar{C}$. So, $\bar{B}\bar{C}$ is used here as well as here similarly your term AB is used in function F₁ as well as in function F₂ this term AB is used similarly.

So, many cases we have got reuse of terms ok. So, on the input side so 1 equal to a asserted term and 0 is negated term so and dash means it does not participate. So, that way it is just drawing the, it is just noting down the terms like $A\bar{B}$ etcetera. So, if AB the term A B term can be represented as a equal A and B these are 1 and C is not there in the term, so C is put as a dash.

Similarly, B bar C so A is not there A is put as a dash and B is put as 0 and C is put as a 1 ok. So, this way I am drawing writing the input side part corresponding to the product terms that we have in this expansion and then in the output side if the corresponding product term is being used, then I will be putting a 1 otherwise I will be putting a 0. For example, the term A B is used by F 1 and F 2 it is not used by F 0 and F 3.

So, this F 0 and F 3 I have got a 0 for the first row and I have got ones for the F1 and F2. So, that way so this is known as the personality matrix of the PLA, so or the programmable logic array so, we will see what is a programmable logic array, but from this function itself we can make this matrix, the this personality matrix. That is you identify the product terms you write down the product terms in terms of inputs and in terms of the variables, and then whether it is present or not and if it is in the output side you write down the functions and a put 0 as a 0 and 1 corresponding to the situation if the product term is not present in the function or present in the function.

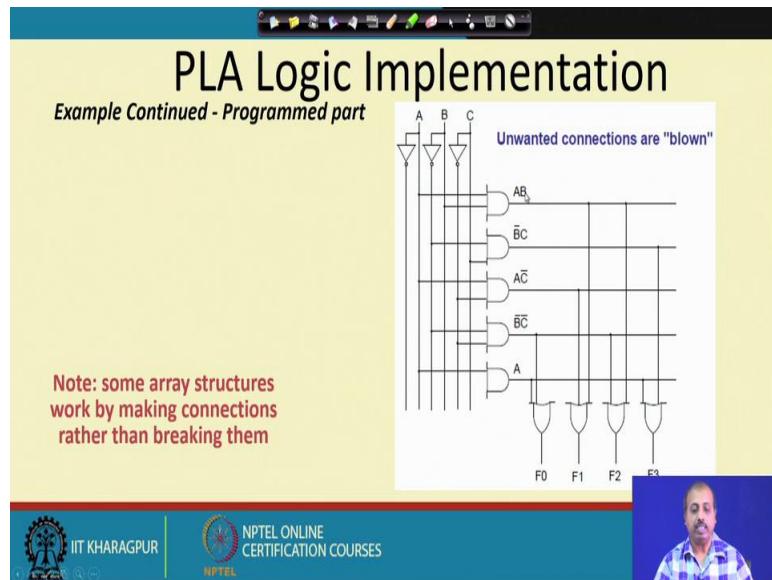
(Refer Slide Time: 21:42)



So, next we will see how this can be realized like see. So, we can see that as if I have got this ABC as the 3 inputs, so we have drawn this inverted lines $\bar{A}\bar{B}\bar{C}$ and these are all possible connection. So, as if I have got this AND gate has got connection from all the lines. So, second AND gate has been got connection from all the lines, so like that similarly this OR gate I have got connection from all the 5 rows that are running so it connected here.

So, this is an un-programmed device, suppose this device is given to you where you have got this type of connections available and I tell you that you choose the connections that you need to retain for getting the functionality. So, you can very easily choose the line connection that should be there to realize the function.

(Refer Slide Time: 22:32)



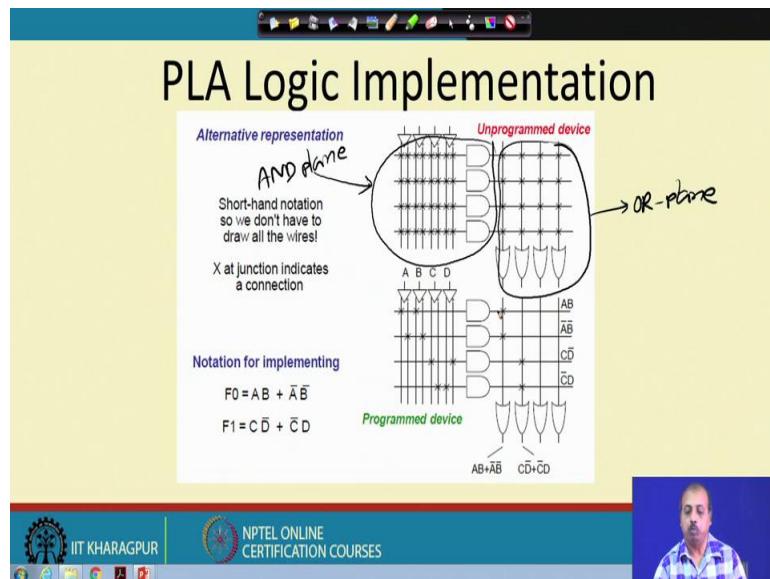
So, all possible connections are available before programming and after that we do some programming and then we take out some of the connections, so unwanted connections are blown. So, there are some switches and those switches can be blown out by applying appropriate voltages to those points and accordingly I can get a customization of the, of this structure.

So, we have got for the first term I need AB , so rest of the connections are blown out for this and term similarly $\bar{B}C$ I blow a blow out the next of the rest of the connections. So, AC bar that is also a rest are blown out. So, this way finally this A so this is only this AND gate is not necessary. In fact, so but this is this AND gate is physically present because that was part of the un-programmed device. So, this A is coming directly here and that is there is only 1 inputs, so it is 1 input AND gate and then the OR gate we blow out the remaining connection and retain the minimum one.

So, some array structures work by making connections rather than breaking them, so this is also another type of PLA or programmable logic array that we have where we make the

connections rather than breaking the connections. So, both are available both the facile both the families are available.

(Refer Slide Time: 23:53)



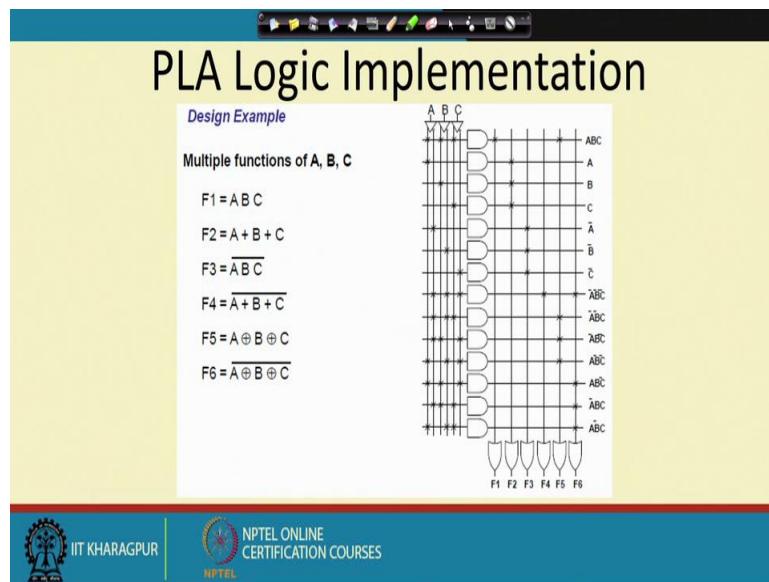
So, this is another representation of this AND and OR so this PLA so in this PLA you see that the first part of the circuit so where I have got say this region. So, this region is representing the product of variables and in this part I am realizing the OR of the product terms. The first in the first in the left side whatever we have that is the generating the product terms which are actually AND of this variables or it is complements and on the right side we have got the portion where it is doing the OR of the product terms. So, this part is known as AND plane of the PLA this is called and plane of the PLA and this part this right side, so this is known as or plane of the PLA.

So, the AND plane it realizes the product terms and in the OR plane we have we realize the functions the final output function in terms of the product terms. Now, this connections so this in the un-programmed device. So, that is represented by the presence of all this crosses so it indicates that there is a connection and there are fuses ok, so each connection has got a fuse and you can blow it if you do not want the connection.

For example, if you are trying to implement this function F_0 and F_1 , F_0 is $AB + \bar{A}\bar{B}$ and F_1 is $C\bar{D} + \bar{C}D$ ok. So, then I have to do it like this so there is there no sharing of term here. So, this AB so these 2 crosses are retained so this 2 the rest of the fuses they are blown out, so you get at the end this line on this line so, you get the term AB .

Similarly, on this line you have retain the connections $\bar{A}\bar{B}$. So, they are ANDed here and you get the line $\bar{A}\bar{B}$ similarly you get here this $C\bar{D}$ and here you get $\bar{C}D$. So, those connections are there and now on the OR plane we make we retain these 2 connections and blow out these 2 as a result I get this $AB + \bar{A}\bar{B}$ and on the second OR gate I realize these 2 connections are retained others are blown out, so I get $C\bar{D} + \bar{C}D$. So, this way this F_0 and F_1 are used are realized rest of the lines and rest of the, so product if there are more product lines available in the device. So, they are not used, similarly if there are more number of output lines so they are also not used.

(Refer Slide Time: 26:38)



So, it is a more complex example so we have got a 3 variable function ABC and we have got 6 functions F_1 to F_6 multi output function with 6 outputs. Now here also the AND plane consists of this ABC lines and their inverted lines and we are so you identify first the product terms.

Now, you see this ABC and this $A + B + C$, so they are all separate, separate terms ok. So, this $\bar{A}\bar{B}\bar{C}$ are coming here. So, they are ORed here to get F_3 and for getting say F_1 , for getting say this $\overline{A + B + C}$ F_4 ; so how are you realizing F_4 so this is by means of this product line. So, this is $\bar{A}\bar{B}\bar{C}$ ok, so this is by mean so $\bar{A}\bar{B}\bar{C}$ are obtained and they that is coming to this OR gate to produce F_4 .

So, this way we can realize any function so this F_5 F_6 so they are pretty complex ok. So, if you want to realize in terms of discrete gate, so that will require a large number of gates,

but you see that once you have generated all the product lines and so you can just OR them together. For example, this $A \oplus B \oplus C$ the function F_5 , so F_5 you see so it is consisting of all this terms $\bar{A}\bar{B}C, \bar{A}B\bar{C}, A\bar{B}\bar{C}$ ok.

So, this all the 3 odd lines odd terms are OR together to get a F_5 , so that is the $A \oplus B \oplus C$ function. So, this way we can very easily realize functions using PLA where the functions are, the terms are shared between the PLA between the functions. So, for multi output function realization this PLA gives us a very important strategy for getting them.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 28
Sequential Circuits

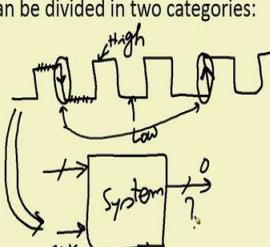
So, far we have seen combinational circuits in this course on Digital Circuits. Next we will be looking into sequential circuits. So, in sequential circuits what happens is that there is a sequence of events that can take place. Maybe you do not change the primary input, but still with the passage of time the situation in the circuit changes. So, typically if there is a moving display, then we get some characters on the display for some time, then after sometime some other character gets displayed.

So, that type of systems there is a sequence of events that are taking place. So, this type of devices or this type of systems they are known as sequential digital systems. So, for a circuit design behind that so, they will be coming under this broad heading of sequential circuits.

(Refer Slide Time: 01:04)

Sequential Digital Circuits

- Sequential circuits are digital circuits in which the outputs depend not only on the current inputs, but also on the previous state of the output.
- The basic sequential circuit elements can be divided in two categories:
- Level-sensitive (Latches)
 - High-level sensitive ✓
 - Low-level sensitive ✓
- Edge-triggered (Flip-flops)
 - Rising (positive) edge triggered ✓
 - Falling (negative) edge triggered ✓
 - Dual-edge triggered



?

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, sequential circuits are digital circuits in which the output outputs will depend not only on the current input, but also on the previous state of the output. So, it is not only in case of combinational circuit like if we take one AND gate. So, the AND gate output is

determined by the input combination that you feed at this point of time. So, it is not dependent on whether the AND gate was previously 0 or 1.

But if you give a 0 0 then the AND gate will always produce a 0. Or if you give 1 1 irrespective of the previous output of the AND gate so, it will produce a one value; however, in case of sequential circuits. So, that may not happen because the output will also depend on the previous state of the output. So, this basic sequential circuits can be divided into 2 categories, one category is known as level sensitive, another category is known as edge triggered. So, level sensitive means like when we are looking at the operation of the circuit so, when are we expecting the circuit to change its state.

So, it may be that we want during some interval of time, or you may want it at some precise instant of time. So, if I, for example, so, if I draw a signal so, which is in sequential circuits so, that is commonly known as clock signal. So, if I have got a signal like this ok, so, it has got a high period and a low period. So, you see this signal it has got some portion of the signal where the value is high.

And there are some portion where the value is low, and there are some other portions other points of interest may be, this is a point of interest when the signal is going from high to low. Similarly, say this is another point of interest, so, where the signal is going from low to high. Now if we say that we are interested about the system behavior when this particular if this is the system if this is the system that we have designed.

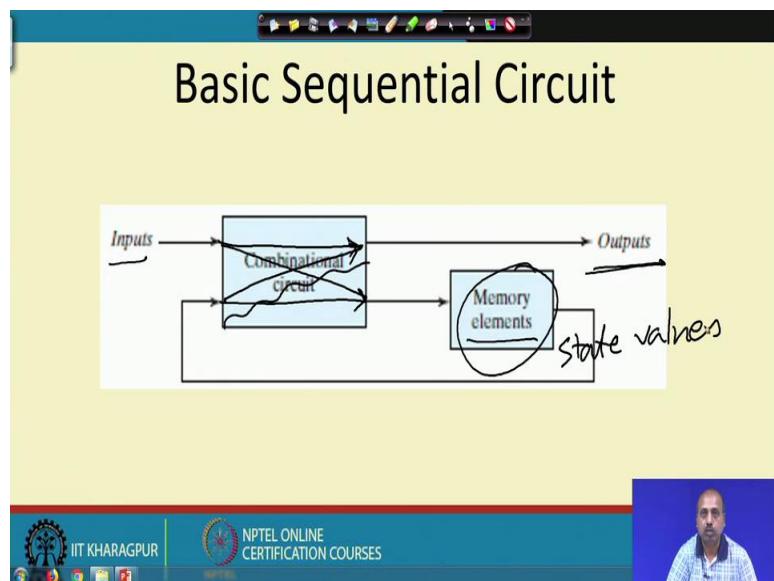
So, this is the, and it has got a number of inputs and number of outputs, and there is a special signal which is commonly known as the clock signal; where we feed an input pattern like this. Now we may be interested in the behavior of the system when this clock signal value is high, or when the clock signal value is low.

So, when this thing happens so, when we have interested about the behavior of the system when the clock signal is high, we say the circuit is high level sensitive. Similarly, if we say that we are interested about the system behavior when the clock signal value is low so, we say it is a low level sensitive. And as I have said the other type of option is when this edges occur. So, it maybe that we are bothered we are interested about the system behavior when the clock signal makes a low to high transition.

So, this is called a rising or positive edge triggering, and in case of this signal if you are interested about when the signal is falling down, see if you are interested in that type of situation, then will say that the system is a system is considering falling edge triggered. So, call the system is a falling on negative edge triggered system. Now depending upon our requirement so, we can we can design a system to be a level sensitive one or an edge triggered one. And between them also we can think about a high level sensitive, low level sensitive or rising edge triggered, falling edge triggered.

So, depending upon the requirement so, we can design the circuit. Now another possibility is that we are interested in both of these events; that is, this following edge as well as this rising edge. So, we are we are interested about the system behavior when the clock signal is rising or the clock signal is falling. So, this type of systems so, they will be known as dual edge triggered system. So, in a in this part of lecture so, will slowly look into all this different categories of a sequential circuits, and how to design those circuits.

(Refer Slide Time: 05:19)



Now, coming to the basic sequential circuit so, this is a sequential circuit. So, it, since it remembers the previous state of the signal of the system. So, there must be some memory element in it. So, you see that there are some memory elements in the system. So, if you forget about this part is memory element, then it has got some input and it has got some output so, that that part is a pure combinational circuit.

But as if this combinational circuit it has got some further input coming from the memory element. And this combinational circuit it can also be divided into for a, into 2 parts for example. One part is determining this thing from input and this previous state from memory to memory element values. So, it determines the output, another part is from the input and this memory element value it determines the next value of the memory elements.

So, as a user of the system so, we are this memory elements are not of interest for us. So, we are bothered about this outputs only but since this memory element inputs are also coming to this combinational circuits, so, what will happen is that, this combinational circuit output will depend not only on this present input, but previously what was the content of the memory elements so, that will determine the output. So, that is the sequential behavior so, it remembers what was the previous state of the system.

So, this memory elements so, they are also known as the state values. They also known as the state values so, this state values of the signal of the system is remembered, and then when this next input comes this state values are also taken into account to determine the output. So, we'll, so, that way we can think about this sequential circuits to be consisting of the basic combinational part, and a set of memory elements.

(Refer Slide Time: 07:10)

The slide has a yellow header bar with a toolbar icon at the top. The main title 'Synchronous vs. Asynchronous' is centered in a large, bold, black font. Below the title, there is a statement: 'There are two types of sequential circuits:' followed by a bulleted list. The list contains two items: 1. 'Synchronous' sequential circuit: circuit output changes only at some discrete instants of time. This type of circuits achieves synchronization by using a timing signal called the *clock*. 2. 'Asynchronous' sequential circuit: circuit output can change at **any** time (clockless). At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES Sequential Circuits'. On the right side of the footer bar, there is a small video window showing a man speaking.

Another type of categorization that we can have in case of sequential circuit is synchronous sequential circuit versus asynchronous sequential circuit. So, in case of synchronous sequential circuit, circuit output changes only at some discrete instance of time, and this

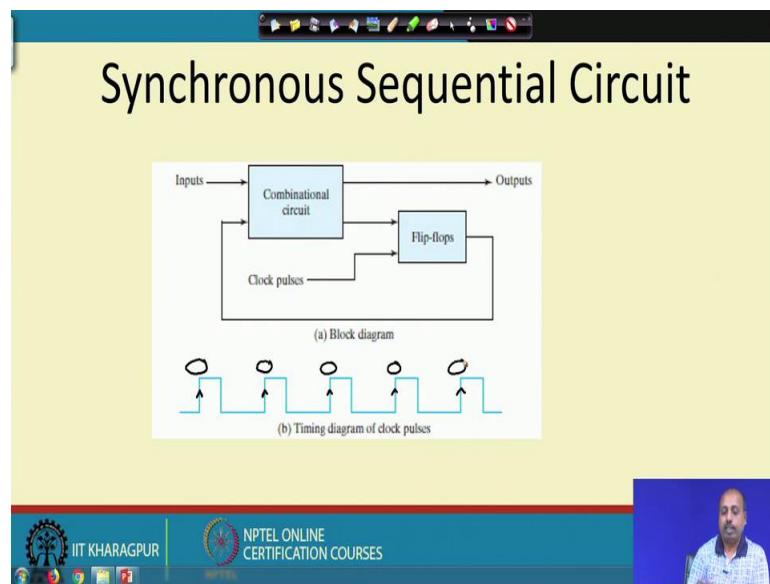
type of circuit they achieve synchronization by using a timing signal called the clock. So, just sometime back I have already introduced the signal clock.

So, naturally the question comes can we have a sequential circuit that does not have any such timing signal like clock, ok, the answer is yes there can be. So, for the type of sequential circuits where we have such a distinct clock signal, so, they that is known they are known as synchronous sequential circuits. On the other hand, there is asynchronous sequential circuit where there is no concept of a clock. So, if there is no clock and the output can change at any point of time.

In case of synchronous sequential circuits so, we have the output can change only at the clock boundaries, maybe for level triggering it may be at over all clock is high or clock is low, depending upon whether it is high level to high level triggered or low level, high level sensitive or low level sensitive. On the other hand, if it is edge triggered, so, if it is whether it is rising edge triggered or falling edge triggered.

So, based on that only at those points the circuit output can change. Whereas, for asynchronous circuit the output can change at any point of time. So, we will see some example.

(Refer Slide Time: 08:41)



So, this is a typical structure of a synchronous sequential circuit. So, we have got this combinational input, the combinational circuit that gets the input + there are some memory elements, so, which are commonly known as flip flops.

So, will see this flip flops after sometime. So, this flip flops are there so, this clock pulses are actually feeding this flip flops. So, this flip flops are activated when the clock signal is active. So, if we say it is the high level sensitive then when the clock signal is high that is at this point, then only the values that are coming from the combinational circuit to the flip flops will be stored in the flip flop.

Otherwise when the clock signal is low for example, during this period, whatever happens to this combinational circuit so, these lines, these lines will be changing, but that will not affect the flip flops. So, as a result the state of the state of the circuit will not change, state of the circuit will remain unaltered. Only if, when the clock signal is active then the input changes, then only this flip flops will change.

Similarly, if we are thinking about say edge triggering, then if we say that it is rising edge triggered so, the circuit which state will change only if the input changes around this. So, the input changes around this time, then only the circuit the state out whether the, this flip flop contents will change, otherwise it will not. So, this gives us advantage because others are design becomes simpler as will see.

However, in some system so, we may not have this type of clock signal, and they are known as asynchronous circuits.

(Refer Slide Time: 10:25)

The slide title is "Asynchronous Sequential Circuit". The circuit diagram shows a sequence of three flip-flops (00, 11, 01) with their outputs (q00, q11, q01) connected to the inputs of an OR gate, which produces the final output "out".

So, this asynchronous sequential circuits so, this is an example so, here you do not have any clock signal in the system; however, if you are if I ask you suppose I give a equal to say 0, and sorry if I give a equal to 0, and say b equal to 0, a equal to 0 and b equal to 0.

So, can you tell me, what is the value what is the output? So, it is difficult because a equal to 0, b equal to 0, means these 2 bits are getting 1 1. So it is dependent what was the previous value of this out, so, it is dependent on that. So, until and unless I tell you what is the previous vale of this out you cannot determine the next value of out. So naturally so, this type of circuits so, this is also a sequential circuit, because it depends on the previous value or previous output of the circuit.

However, it is there is no explicit clock signal in this system. So, it is not asynchronous circuits so, this is an asynchronous circuit. So, this asynchronous circuits are also used sometimes. So, we will see the relative advantages and disadvantages of these 2 types of systems.

(Refer Slide Time: 11:38)

Applications - Synchronous

- Predominates asynchronous circuits
- Typically used to perform activities that need to happen at precise times

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, applications of synchronous so, this synchronous sequential circuit they predominate asynchronous sequential circuits. So, normally whatever sequential circuits we design so, they have a clock signal in them and then the so, most of the circuits are synchronous in nature. So, you will not find much asynchronous sequential circuit, because of several reasons that will see.

And this typically used to perform activities that need to happen at precise times. So, at some regular time interval something has to happen, that example that I took at the beginning like if there is a display then some character is displayed so, there is a fixed amount of time for which one character will be displayed after that the next character will be displayed.

So, this passage of timing so, when the character will change so, that is the space that is fixed. So, you can say that at fixed intervals of time, we are expecting something to happen, something to happen in the system so, they are synchronous systems. So, they are typically used to perform activities that need to happen at some precise times, or if there is a conveyor built on which items are moving and at regular intervals of time, so, we can we there may be a robotic hand which is picking up the items from the from the belt.

So, that also happens at some regular intervals of time. So, all this examples so, they will be leading to synchronous sequential circuits.

(Refer Slide Time: 13:01)

Applications – Asynchronous

- Signal processing
- Fast arithmetic unit
- Simple microprocessors
- Memory(static, RAM, FIFOs)
- ILLIAC

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

On the other hand, asynchronous sequential circuits so, they have got applications in signal processing because you do not know, when the exact signal will what is the clock of the clock period of the arrival of this signal so, that is not known.

So, we have to we have to do it in an asynchronous fashion, then one typical problem with the synchronous circuits is that so, you are your the speed of the circuit is bounded by the clock signal that you are feeding. So, this clock is a clock signal is not very high so, you cannot do you cannot do operations fast.

So, I may have different components in my in my system, so, they are operating at they are all of them are synchronous sequential circuit, then the delay of the slowest modules so, that will determine the maximum clock frequency that I can have for the system. Where as if this all the systems are asynchronous in nature, then what can happen is that all of all the systems they can operate at their own speed, and we may get a faster system.

So, maybe you can design an arithmetic unit which is much faster than this register and all in the system. So, that way we get a fast arithmetic unit, then simple microprocessors. So, they are made the asynchronous because of this speed achievement. So, we can do it like memory, the static memory then this ram FIFO first in first out buffer so, they are all made this they are made of asynchronous system.

(Refer Slide Time: 14:37)

Advantages –Synchronous

- Simplicity
- Widely taught and understood
- Available components
- Simple way to deal with noise and hazard

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

ILLIAC 4 machines so, this is one of the computer early computers so, they are made in asynchronous mode. The why do we go for synchronous? Asynchronous has got so many advantages; the major advantage is the simplicity. So, asynchronous system so, since we have to consider what is the previous value of the output at, whenever we are trying to determine like, what is the current value of the output, we are we are faced with the question the what is the previous value of the output.

Now, this previous value means before how many how much time? So, that question has to be answered. And in case of synchronous sequential circuits so, you can just say it is at the previous clock period what was the situation, or previous to previous clock period, what was the situation. So, we have got a fixed interval or fixed instant of time at which of which we are talking about.

But in case of asynchronous circuits we do not have this instants of time. So, that they are the time becomes a continuous value there. And then this previous time definition of this previous time becomes difficult. So, synchronous circuits are generally much simpler to design, and they are widely taught and understood ok, for digital circuits classes so, we normally, we normally go to the synchronous design only because that is that is much easily understood and can be easily taught.

And in our course also for most of the designs so, will be going to synchronous designs also available components. So, there are many chips which are synchronous in nature. So,

they are realizing some sequential circuits which are synchronous in nature. So, that is why we go for that, then simple way to deal with noise and hazard.

So, if some noise signal comes so, that can change the state of the system. Now you see for synchronous sequential circuit we have the advantage that if the noise does not come when the clock is active, then that noise is will not be able to affect the system. So, we can be we can concentrate only for noise protection during the clock active time. Similarly, hazard that we discussed in our previous classes. So, that also is that affect will come only when we have got this clock signal active.

So, otherwise particularly for edge triggered system, so for very small amount of time we have to have the system noise and hazard free. Otherwise this noise and hazard so, they are unable to affect the system state. What is the disadvantage?

(Refer Slide Time: 17:02)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'Disadvantages –Synchronous' is displayed in a large, bold, black font. To the right of the title, there is a small diagram of a circuit board with a central component and several wires. Below the title, there is a bulleted list of disadvantages:

- Sensitive to variations in physical parameters
- Not modular
- Power consumption
- Clock distribution is difficult due to clock skew
- The maximum possible clock rate is determined by the slowest logic path in the circuit, otherwise known as the critical path

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is a logo for IIT Kharagpur and a link to NPTEL Online Certification Courses. On the right side of the footer, there is a small video window showing a man speaking.

So, disadvantages of synchronous circuits are like this. It is sensitive to variations in physical parameters. Because physical parameters like say if I have got a big circuit, then this the various parts of if they are operating at a different speeds.

And then with the change of this temperature and particularly in this power consumption and temperature those values are also changing, the delay values are also changing. So, that makes it difficult. So, that becomes sensitive to variation in physical parameter. It is

not modular so, you cannot just add another module to this because that will determine the clock speed also.

Power consumption so, whenever the circuit is active so, it will consume power. Now in case of sequential, synchronous sequential circuit what is happening is that the at least the clock signal is changing at regular intervals of time. And we have seen that for CMOS type of technology, the power consumption occurs only when the input signals change or output to the input signals are changing the values.

As a result, the output is also changing, but in case of sequential circuit you know that even if you have got the inputs unchanged, input remain same. So, that output is also same, but the clock signal will always toggle. So, as a result the power consumption due to this clock signal will come into picture. So, the synchronous sequential circuit power consumption will be high.

Clock distribution is difficult due to clock skews. So, clock skew means so, if I have got a board, on which I have got some systems made, now the clock signal is entering here. So, I have got a chip here, I have got a chip here, I have got a chip here, and all of them are operating in a synchronous mode.

Now, this clock signal is distributed like this, ok. If it is distributed like this, then you see that this delay of this path so, they are proportional to the length. In fact, it is square proportional to the length. So, as the length increases the delay increases severely. So, this is intended that when the clock signal reaches this module. At the same point of time the clock signal reaches here and also reaches there.

But the way I have laid out this clock line so, it will not be possible to ensure that. So, there will be a skew, the time at which the clock reaches here. This third module will get the clock after sometime delay so, this is known as clock skew. And it is. So, this clock skew if it is there, when naturally when this circuit is producing some outputs. So, it is to be used by this module, say functional output produced by the module one is to be used by module 3.

And if it is so then this module one it has produced the output when the clock signal was active, but at that time this module 3 was not getting the clocks it could not use the value.

By that time maybe some other thing has changed. So, that way there that is why this clock skew is a severe problem.

And for synchronous sequential design so, it is a challenge like how to distribute the clock signal through the, IC through the integrated circuit chip, ok. So, that is the VLSI layout problem. So, will not go further into that so, this clock distribution becomes difficult due to this clock skew, the maximum possible clock rate is determined by the slowest logic path in the circuit.

(Refer Slide Time: 20:35)

The screenshot shows a presentation slide with the title "Disadvantages –Synchronous" in bold black font at the top. Below the title is a bulleted list of disadvantages:

- Sensitive to variations in physical parameters
- Not modular
- Power consumption
- Clock distribution is difficult due to clock skew
- The maximum possible clock rate is determined by the slowest logic path in the circuit, otherwise known as the critical path

To the right of the list, there is a small diagram showing a central node connected to several wavy lines representing clock signals, with one line labeled "clk". At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player window showing a man speaking.

So, if I have got some if I have got some combinational circuits, suppose this is a combinational circuit, and this is feeding some flip flop, ok. So, this is feeding some flip flop, this output is stored like the way we have seen in the sequential circuit. Now what is the; what type of plot I can apply to this to this to this flip flop, ok. So, if this flip flop operates at a clock so, this combinational circuit it get some input from outside it does some computation, and accordingly it produces the value here.

So, the if this for clock is faster, then the delay of this combinational block then what will happen? This flip flop will get some wrong values. So, the values when I change this input values, the before the proper output has been computed the flip flop will get a, get the values.

So, as a result the value will be incorrect so, what is required is that this clock should be slower than the delay of this particular block. And there are several paths through this for through this network. So, one path may be like this going through several gates another path may be like this. Another path may be like this now among these 3 paths.

So, whichever path is the slowest so, that is the critical path. So, this critical path delay so, this will determine the clock frequency, the maximum clock frequency at which I can operate the system, ok. So, the slowest logic part in the circuit so, that will determine that is the critical path so, it will determine the clock period. So, that is another problem with the synchronous sequential circuits.

(Refer Slide Time: 22:04)

Advantages –Asynchronous

- High performance
- Low power dissipation
- Low noise and EM emission
- Good match with heterogeneous system timing

GALS
Globally Asynchronous
Locally Synchronous

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Asynchronous circuit is the major advantage that we get is the high performance because there is no clock signal. So, I am not limited by the clock. So, I can operate whichever module gets the data ready. So, it can start operating on the data and they reproduce the result fast.

Low power dissipation so, if the inputs do not change, then the output will also not change so, as a result there is nothing like at every clock interval. So, that the clock toggling will occur so, that that sort of thing is not there. So, this power dissipation is often much less compared to the synchronous sequential circuit. Low noise and electromagnetic emission so, this is also another issue, like this generally it is seen that it does not produce this high noise and electromagnetic radiations are also less, because there is no such toggling.

And good match with heterogeneous system timing, like even a system I have got a different types of components, and they are operating at different frequency, then also we can have this thing. So, in fact, for this a heterogeneous timing so, this leads to one particular type of structure which is known as GALS structure, which is globally asynchronous locally synchronous type of structure.

So, that takes the advantage of both this synchronous and asynchronous design. What the idea is that so, suppose I have got a system like this, now in this system I have got several component so, this is one component so, this is another component, this is another component. So, they are interacting between them so, there are some signal lines running between them so, they are interacting.

Now, they have got they are running at their individual clock. So, this is running on clocks c_1 . So, this is running at clock c_2 , this is running at clock c_3 , but the clocks are not same so, c_1 , c_2 , c_3 are not same. So, individually when you look at the systems so, they are synchronous in nature, but if you look into the overall system. So, that is asynchronous system, because there is no common clock synchronizing all the 3 subsystems.

So, this type of designs have become very popular, because it combines the advantages of synchronous design, that is the ease of design and all, and the advantages of this asynchronous design where you do not have to have this global clock signal, then this power consumption, high power consumption like that so, they are not there, so that way this GALS style of design have become quite common.

(Refer Slide Time: 24:51)

Disadvantages – Asynchronous

- Substantial circuit level overhead
- Lack of CAD tools
- Delay

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next at disadvantage of asynchronous system is the substantial circuit level overhead because of this lot of feedback lines will be running since the clock signal is not there, and this sequential elements are often realized by with other combinational elements.

So, as a result there will be a large number of lines running backwards so, that is the circuit level overhead is high. And the set of cad tools so, that you for asynchronous design so, you do not have this very good cad tools available so far so, that is the other problem. And delay so, delay in the sense that you have to you have to match the delays between 2 stages, if both the stages that are asynchronous in nature so, you need to match the delay between the 2 stages.

And that also become a concern, so, delay becomes a concern.

(Refer Slide Time: 25:40)

The Set/Reset (SR) Latch

The Set/Reset latch is the most basic unit of sequential digital circuits. It has two inputs (S and R) and two outputs outputs Q and \bar{Q} . The two outputs must always be complementary, i.e if Q is 0 then \bar{Q} must be 1, and vice-versa. The S input sets the Q output to a logic 1. The R input resets the Q output to a logic 0.

Circuit Diagram

Truth Table

S	R	Q_+	\bar{Q}_+	Function
0	0	Q	\bar{Q}	Latch
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Illegal

Logic Symbol

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, will start with some latch circuits, the first latch that will look into is known as the SR latch, the first sequential circuit that will look into is known as SR latch or set reset latch. So, this is the most basic unit of sequential circuit. It has got 2 inputs S and R, and 2 outputs Q and \bar{Q} , the 2 output must always be complemented; that is if Q is 0, \bar{Q} is must be one and vice versa.

So, this is by the specification of the S R latch. So, it says that it is it has to be a sequential circuit, that will have 2 input set and reset. And there will be 2 outputs Q and \bar{Q} , and by specification, \bar{Q} should be complement of Q. It is further stated that the S input will set the Q output to logic one, and the R input resets the Q output to logic 0, that is why S is called the set input R is called the reset input.

So, the set input will set Q to 1, and reset input will set Q to 0. So, a basic circuit diagram for this can be like this. So, it can we can take 2 cross coupled NOR gates, ok, now you can understand that if I set this, in this nor gates. So, if I set this S to be equal to 1, if this S is equal to 1, then this \bar{Q} will be getting 0. And this so, so, it if it is 0 here then what ever be the \bar{Q} so, it will be coming here so, this Q will be coming back to this.

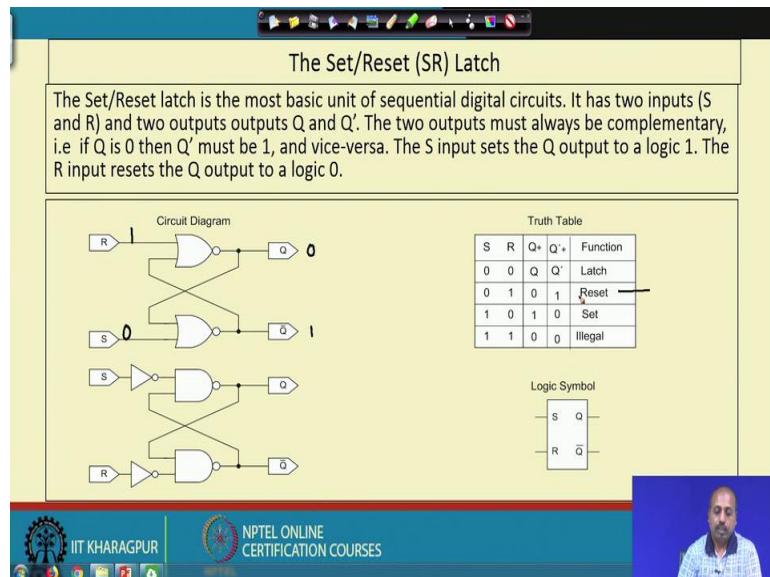
So, this set and reset inputs so, let us look into this truth table and try to understand. Suppose S and R both are 0, 0, so, if both are 0, if say if both of them are 0, then what happens is, so this is 0, and this is 0 so, this is this Q is coming and this is a NOR gate. So, what will happen is that this Q whatever be the Q value so, it will be coming at say \bar{Q} . So,

this Q value will be coming and this is just a NOR gate. So, it will be \bar{Q} , and this is \bar{Q} so, when it is when it is getting this so, this is equal to; this will remain equal to Q.

So, if you look into this truth table when this S and R so, Q_+ and \bar{Q}_+ . So, Q + means what I mean is that, at the next time you need what will be the value of Q. So, that is Q_+ and whatever and Q, \bar{Q}_+ means what is the value of \bar{Q} at the next time unit. Now you see that what happens is that if you give S and R as 0 0 then this Q remains at Q_+ remains at Q and Q \bar{Q}_+ remains at \bar{Q} .

That is whatever the value it was having previously so, that value remains there. Now let us take the other combination, let us take say this R S equal to 0 and R equal to 1.

(Refer Slide Time: 28:52)



Then since this is a NOR gate irrespective of the value of \bar{Q} , this Q will become equal to 0, and once this Q and this and this Q was coming here. So, this will be this 0 comes here so, this 0 will be making it 1. So, you see that when this S is equal to 0 and R equal to 1, we get the value Q_+ equal to 0 and \bar{Q}_+ equal to 1. So, that is the reset so, as if the flip flop has been reset or the latch has been reset.

(Refer Slide Time: 29:31)

The Set/Reset (SR) Latch

The Set/Reset latch is the most basic unit of sequential digital circuits. It has two inputs (S and R) and two outputs outputs Q and Q'. The two outputs must always be complementary, i.e if Q is 0 then Q' must be 1, and vice-versa. The S input sets the Q output to a logic 1. The R input resets the Q output to a logic 0.

Circuit Diagram

Truth Table

S	R	Q ₊	Q' ₊	Function
0	0	Q	Q'	Latch
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Illegal

Logic Symbol

The other combination like say 1 0 so, if I give S equal to 1 and R equal to 0, then what will happen? This will be setting this by the similar logic. So, this \bar{Q} output will be equal to 0, because this S input is 1, this NOR gate and this 0 0 going there so, this will make this thing to be equal to 1. So, what is happening is that Q_+ is becoming one and \bar{Q} is becoming 0 so, that is the value is set.

And if it is 1 1, then it is we say that if I give it 1 1, then both the outputs will become 0 0.

(Refer Slide Time: 30:03)

The Set/Reset (SR) Latch

The Set/Reset latch is the most basic unit of sequential digital circuits. It has two inputs (S and R) and two outputs outputs Q and Q'. The two outputs must always be complementary, i.e if Q is 0 then Q' must be 1, and vice-versa. The S input sets the Q output to a logic 1. The R input resets the Q output to a logic 0.

Circuit Diagram

Truth Table

S	R	Q ₊	Q' ₊	Function
0	0	Q	Q'	Latch
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Illegal

Logic Symbol

And by the specification so, this is called illegal so, this is illegal so, we should not apply this 1 1. Similarly, if we take the NAND gate based realization also. So, you will get you will see that if we do it take 2 cross coupled NAND gates and take 2 inverters before after S and R you get the similar type of truth table. So, this is the logical symbol for this S R latch, ok. So, we just we give it this is a shown like S and R being different values, if we put then Q and \bar{Q} will be following this truth table value.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 29
Sequential Circuits
(Contd.)

(Refer Slide Time: 00:20)

The Set/Reset (SR) Latch

The Set/Reset latch is the most basic unit of sequential digital circuits. It has two inputs (S and R) and two outputs outputs Q and Q'. The two outputs must always be complementary, i.e if Q is 0 then Q' must be 1, and vice-versa. The S input sets the Q output to a logic 1. The R input resets the Q output to a logic 0.

Circuit Diagram

Truth Table

S	R	Q+	Q'	Function
0	0	Q	Q'	Latch
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Illegal

Logic Symbol

So, in the S R latch we have seen that the combination 1 1, we have said that this is illegal. So, why is it illegal? So, suppose we give this particular value say sorry, if we give this particular value say 1 1 then you see by; since these are simple nor gates, so you will get straightway 0 0 there fine now suppose after giving this after giving this 1 1.

So, we change this 1 1 to the values 0 0 we change both of them to 0 ok. So, we number these gates as gate number 1 AND gate number 2 ok. Now what happens is that so suppose these gate number so what is the. So, this is I have applied 1 1 so this is 0 0 and then I am applying here 0 0 and this is gate number 1 this is gate number 2.

(Refer Slide Time: 01:20)

The Set/Reset latch is the most basic unit of sequential digital circuits. It has two inputs (S and R) and two outputs outputs Q and Q'. The two outputs must always be complementary, i.e if Q is 0 then Q' must be 1, and vice-versa. The S input sets the Q output to a logic 1. The R input resets the Q output to a logic 0.

Circuit Diagram:

```
graph LR; R((R)) --> OR1(( )); S((S)) --> AND1(( )); OR1 --- AND1; AND1 --> Q1((Q)); Q1 --> OR2(( )); S --> AND2(( )); AND2 --- OR2; OR2 --> Q2((Q'));
```

Truth Table:

S	R	Q ₊	Q' ₊	Function
0	0	Q	Q'	Latch
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Illegal

Logic Symbol:

```
symbol_rect S((S)) --- symbol_rect Q((Q)); symbol_rect R((R)) --- symbol_rect Q_bar((Q'));
```

Race condition

Now suppose, so, suppose I am looking into gate number 1, then what happens is that it both the inputs are 0 as a result this output becomes 1 and then this lower gate, so this case 1 input is 0 another input is 1 so NOR gate, so these value remains at 0. So, it says that after the pattern 1 1 if so 1 1 gives me the 0 0 after that if I apply the input pattern 0 0, I am getting the pattern 1 0, but is my answer correct ok. So, to see that; so let us now consider the gate number 2 first; so this was so this was at a value 0 0 both the outputs were at 0 and then we consider gate number 2 first, then getting 0 0 so this will make this 1 as 1 and getting this 1 and 0.

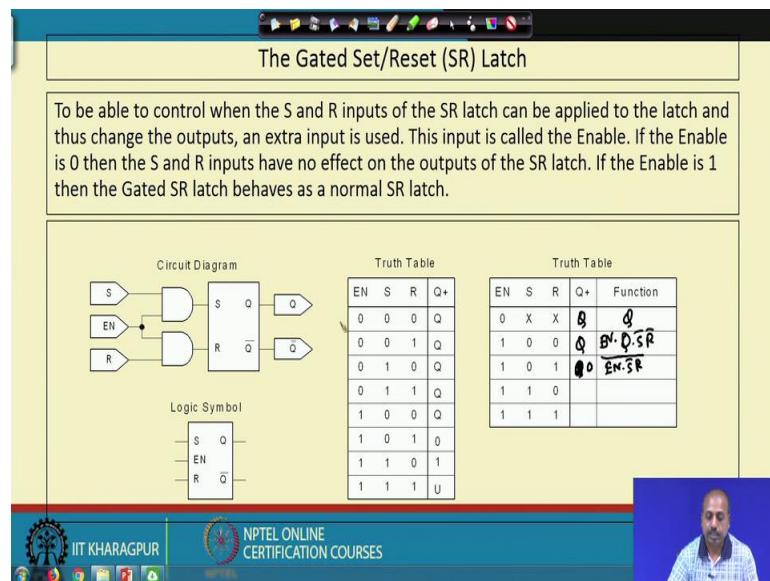
So, this will make this 1 as 0 so that means, another possibility is from 0 0 it can also give rise to 0 1 and that is the problem, that is the problem; you see if you apply 1 1 once ok, then after that if you whatever pattern you will apply. So, this gate number 1 and gate number 2, so they will race with each other whichever gate makes the transition fast and accordingly the output will be determined.

So, you see that in case of this gate number 1 and gate number 2, so physically they are made on the silicon. So, they are not exactly of same speed ok. So, what will happen is that 1 of them will be faster than the other, but we do not know which 1 is faster that it is not possible to determine. So, we cannot say which transition will take place first gate number 1 transition or gate number 2 transition.

So, as a result we cannot say whether the circuit will go to the output 1 0 or 0 1. So, this is the problem and this particular problem is known as the race condition. This particular problem is known as the race condition, as if the two gates 1 and 2 they are racing between each other to make the transition first. So, whichever gate means so sets the output to 1 ok.

So, that is the problem now with this so the that is why whenever we using S R latch, so we should we should be careful that we do not apply this pattern 1 1 to it see if we apply 1 1 both the outputs will become 0 so that part is fine if both of them are strong zeros, but after that the circuit behavior will become un predictable. So, as the gates will start racing between each other to make the transition, so we should avoid giving 1 1 to S R latch.

(Refer Slide Time: 04:11)



So, there is we can think about some enable signal. So, this they are known as gated S R latch, so to be able to control when the S and R inputs are S R latch can be applied to the latch. And thus, change the outputs and extra input is used so this is the enabled signal. So, this if this you can understand that if this enable is equal to 0, this if the enable is equal to 0 then both these S and R so this AND gates are getting 0. So, in case of in case of AND gate in case of S R latch so if you give 0 0. So, you see the circuit output does not change so they Q and \bar{Q} they hold their previous values.

So, if I have this enable line so if this enable line is 0 so whatever values we give to S and R, so these 2 inputs will get 0 0 as a result Q and \bar{Q} will maintain their previous values. Whereas, if we if we take this enable signal to be equal to 1 we take this enable signal to

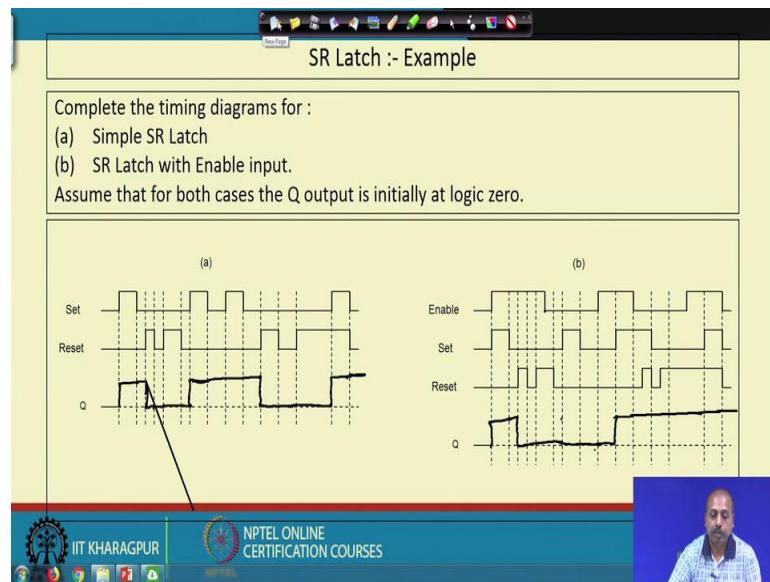
be equal to 1, then this S and R whatever values are coming so they will be coming to this S and R input and accordingly the circuit behavior will be determined.

So, we can say that so the a detailed truth table can be like this so in enable is 0 S and R being 0 0, 0 1, 1 0 or 1 1 so they will maintain the previous value Q and whenever this output is equal to whenever this enable line is equal to 1, then the behavior will be determined by the then the behavior will be determined by this S and R inputs ok.

So, for example in this particular table so you can see that this when whenever this enable is equal to 0, so this is this will be holding the Q value. So, function is equal to Q and then whenever it is 1 0 0, then you know that the S R latch it does not changes it's state. So, that is equal to Q and this 0 1. So this is function is basically $EN \cdot Q \cdot \bar{S} \bar{R}$ and then this function is again this resetting the Q output so this is equal to 0.

So, we can say so this is output is the combination is $\overline{EN \bar{S} \bar{R}}$ because output is 0 in this case. So, this way we can finish you can write down this truth table or you can just make a carno map and try to figure out what is the function realized for this particular case.

(Refer Slide Time: 07:16)



Next we will look into another example say this S R latch how will it look like, so I will urge you to finish this diagram. So, as it is said that we can say this is initially this Q output is low and here this S signal is going high. So, when the S signal is high the Q output will be high and then it will the after that S signal goes low this S and R both are 0 0. So, 0 0

means it will continue holding the previous value and then the R signal is going high, so as a result so this will become low ok.

So, this remains low and then both of them are low so it will remain low then this R signal is again going high, so it continues to be low till this S signal becomes high. So, at this point S signal is high so this is going high and again. So, this is this is this will continue to be high till this R signal is becoming high so at this point R is high. So, this will be reset and then this will continue to be reset and then again at this point S signal is high so it will be going high.

So, it will continue being high because S and R both have become 0. So, this way you can you can trace through this diagram. So, in similarly in this case everything remain same only thing is that the enable line also has to be considered. So, at this point enable is high and set is high as a result the signal will become high. And after sometime so enable is high up to this point, so till this point whatever changes are occurring in S and R so they will be considered.

So, it will come up to this point then this it will go down then again the S signal is high at this point, so it continues to be going like this and then at, but at this point it cannot change because enable is low. So, next enable is high at this point so up to this point the signal value Q value will be low and then when the enable is high the set line is going high at this point. So, this will become high and then again it will be then after sometime enable has become low.

So, it will continue like this and at this point enable become has become high active again, but here the S signal is again high so it continues like this. And then after this point enable signal is becoming low so it will continue like this. So, this way you can try to draw the diagram for this timing diagram for this S R latch operation ok. So, you just need to follow the truth table.

(Refer Slide Time: 09:58)

The Data (D) Latch

A problem with the SR latch is that the S and R inputs can not be at logic 1 at the same time. To ensure that this can not happen, the S and R inputs can be connected through an inverter. In this case the Q output is always the same as the input, and the latch is called the Data or D latch. The D latch is used in Registers and memory devices.

Circuit Diagram

Logic Symbol

Truth Table

EN	D	Q	Q ₊
0	0	0	Q
0	0	1	Q
0	1	0	Q
0	1	1	Q
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Truth Table

EN	D	Q ₊	Function
0	0		
0	1		
1	0		
1	1		

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

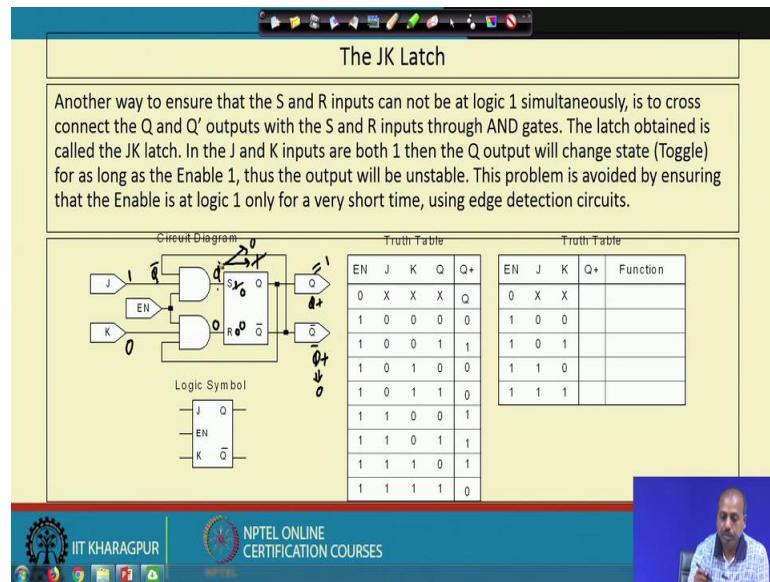
Next we will be considering another latch which is known as the D latch. So, D latch so essentially it this logic symbol is like this so it has got to 2 inputs 1 D input and 1 enable input; whenever D whenever this enable line is high whatever is coming in the D input will be copied on to the Q and \bar{Q} is definitely the compliment of that. So, you see whenever enable is low enable is 0 this Q_+ remains the holds the value of Q so it does not change.

But when this enable signal is 1 then D whatever irrespective of the value of Q the value is copied on to Q_+ , so it does not depend on the on the previous Q value so it copies this D value to this, otherwise it will be remembering the if the enable is low it is remembering the previous value of Q, but if the enable is high. So, it is copying the current D value to the Q value so it is going like this. So, this particular latch this D latch can be realized by just connecting an inverter between S and R inputs of this of this SR latch.

So, you see that in case of in this case what is happening is that if I apply a so enable I am taking as permanently 1, now if I give D equal to 1 then what will happen so this S will come as 1 and this R will come as 0. So, as a result this Q output will be set to 1 and \bar{Q} will be set to 0 that is from the behavior of SR latch. Now if I do if I put D equal to 0 and this enable equal to 1 then this is 0 and this is 1 and by the behavior of a SR latch we know that it will reset this Q output to 0 and \bar{Q} to 1. So, you see if we make an SR latch and we just connect and inverter between this S and R inputs. So, we get the D latch and here of course there is no race around condition because, in into if the in our SR latch so it never

gets this. So, both S and R inputs as 1 so as a result the S R around condition is not there say you again finish off, you can complete the truth table for the S R for the D flip flop. Not doing it now.

(Refer Slide Time: 12:30)



Another interesting type of latch that we have is known as the J K latch, so in case of J K latch what is done is that it is a 2 input it is a 2 input circuit. So, J and K these are the 2 inputs and it has got a third enable input. So, when this enable line is 0 by our definition we know that whatever be the values of this J and K. So, it will continue to hold the previous value of Q, but when this enable line is 1 then the behavior of the system will get affected how.

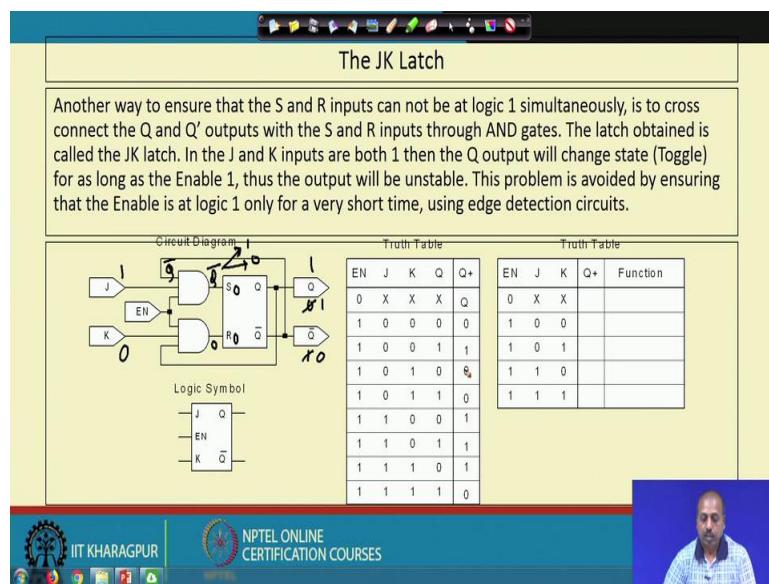
So, this is 1 way to ensure that S and R inputs cannot be at logic 1 simultaneously is to cross connect Q and \bar{Q} outputs with the S and R inputs through AND gates what is done this \bar{Q} line is taken back and it is fed as an input to this AND gate. So, this AND gate it has got 3 inputs. Now J \bar{Q} and enable and then this K it has got inputs like this Q this Q line is coming here. So, K Q and enable so these 3 are put into this AND gate and they are put into this S R latch input.

So, so the type of latch that we get is known as the J K latch now otherwise, so when there is a behavior is almost similar to the S R latch like if I have this say J equal to 1 and K equal to 0. So, what will happen is that this R input will be equal to 0 and then this and then this \bar{Q} line is coming here Q bar line is coming here so this gate is the Q line.

So, Q may be equal to 0 or may be equal to so, so this is a previous Q ok. So, this is the previous Q so this Q if I say if I write it as Q_+ and \bar{Q}_+ that way. So, what have what happens is that so this previous $Q_+ \bar{Q}$ was coming here so that is ANDed with J equal to 1. So, it get a previous Q value coming here now this Q there are 2 possibilities this Q may be equal to 0 or this Q may be equal to 1. Now if this Q is equal to 1 in that case this case input is 1 and R input is 0 as a result this S R latch will be set.

So, this Q_+ will become equal to 1 and \bar{Q}_+ will S become equal to 0, on the other hand so if the Q value is equal to 0 if the Q value is equal to 0 then what you are getting is you are getting a 0 here and 0 there, as a result this Q it will be maintaining it is previous value. So, you see that when these J input is 1 so this when this J input is 1 then if this Q value is 0 if the Q value is 0 then it is going to sorry here it is here it is \bar{Q} is coming I am just explain it once more.

(Refer Slide Time: 15:48)

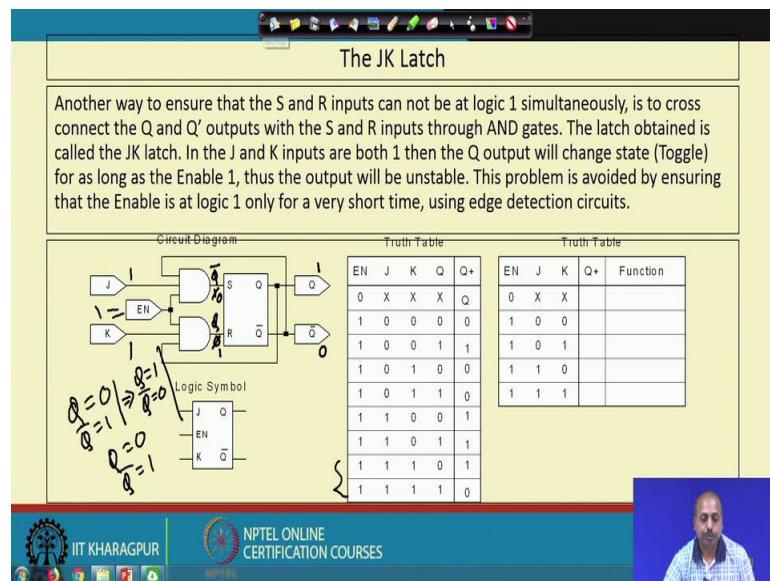


So, I said J equal to 1 and Q equal to K equal to 0 and this so this \bar{Q} line is coming here, so this \bar{Q} line is coming here. So, this \bar{Q} can be equal to 1 or can be equal to 0 this line is 0 there is no problem. Now if this \bar{Q} line is equal to 1 in that case this Q line was previously it was 0, Q bar line was 1 now, because this is this is S R latch and this Q input this S input is equal to 1, so this will set it to 1 and this will go to 0. On the other hand if this \bar{Q} line was 0 ok, that means this Q line was equal to 1, so it will continue to hold the value 1. Since I am getting a 0 0, here so it will continue to hold the value 1.

So, in case of J K flip flop what J K latch what happens is that if you give this J and K J inputs as 1 and K input as 0, then irrespective of the previous value of Q the new value of Q becomes equal to 1 and you can also trace the other way that is if the K value is K is made equal to 1 and J is made equal to 0 then irrespective of the previous value of Q, so it will become equal to 0; so $Q +$ will become equal to 0.

So, that way it is behaving more like a S R latch, but only thing is that that racing condition cannot occur because a illegal combination cannot be applied. Of course, there is another issue like if I make this both this J and K equal to 1, so I make this J and K both of them equal to 1.

(Refer Slide Time: 17:28)



So, what happens is that this Q line comes here. So Q line the previous Q value will be coming here and the previous Q bar value will be coming here. Now suppose Q was equal to 0 Q was equal to 0, so this is equal to 0 and this is equal to 1 as a result this line will become 1 and this line will becomes 0 ok. Now you see that after that what will happen after that this 0, so I had Q, I had Q equal to 0 \bar{Q} equal to 1 from there it has gone to a situation where Q equal to 1 and \bar{Q} equal to 0. Now then what will happen this 0 will be coming back here so this will make it 0 and this and this 1 will be coming back here will make it here equal to 1 and by the S R latch property next time what it will do it will make Q equal to 0 and \bar{Q} equal to 1, it will reset the latch and so it will make Q equal to 0 and \bar{Q} equal to 1.

So, what is happening is that this will go on happening continually. So, if this as long as this enable line is equal to 1 if J and K both are made high then this latch so it will go on toggling the states. So, once this once it will be equal to 0 and then it will become equal to 1 and that will happen continually. So, this is explained in these two rows like, if this enable is 1 then if J and K both are 1 then if the previous value of Q is 0 now the value will become 1 and since the enable is 1. So, now what will happen is that this 1 will come here as a result it will again change to 0. So, the Q line will continually flip and this **Q bar** line will also continually flip, so this is the problem with the J K latch that as long as this enable is equal to 1 output will be unstable.

So, this J K, J and K inputs are both 1 then Q output will change state for as long as the enable equal to 1 and the does the output will be unstable. The problem can be avoided by ensuring that enable is at logic 1 only for very short period of time by using some edge detection circuit.

So, this type of if the problem with this circuit is that is as long as enable signal is high, so it is thus whatever values in J are coming in J and K so they are taken into consideration. So, if we can do something so that this enable is not high for a significant amount of time only for a very small amount of time it is active. So, we get we go from this level triggered circuit to edge triggered circuit, then for the edge triggered circuit this type of problem will not occur anyway we will see that.

(Refer Slide Time: 20:25)

Latches and Flip-Flops

- Latches are also called transparent or level triggered flip flops, because the change on the outputs will follow the changes of the inputs as long as the Enable input is set.
- Edge triggered flip flops are the flip flops that change their outputs only at the transition of the Enable input. The enable is called the Clock input.

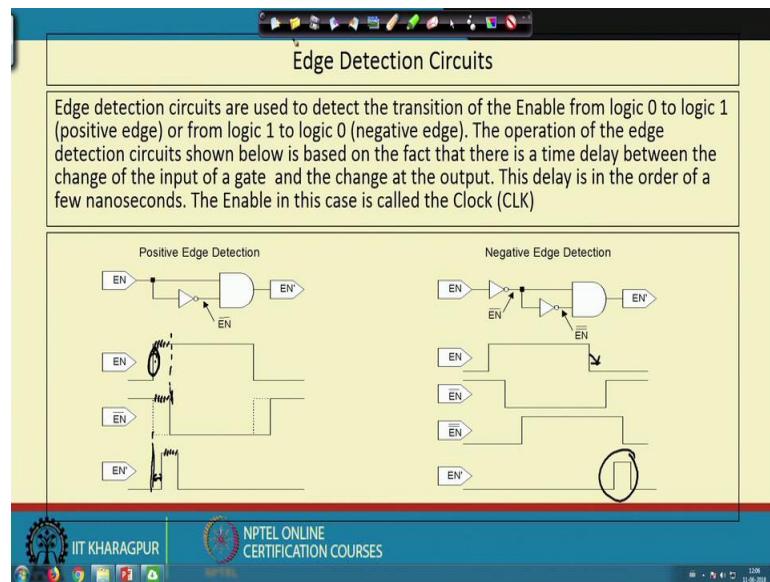
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the latches are also called transparent or level triggered flip flops because, the change in the outputs will follow the changes of the inputs as long as the enable input is high. So, so far what we have seen is that we have got if this is whatever S R latch, S R, D, JK whatever it is so we have got an enable line and we said that as long as this enable signal is high. So as long as this enable signal is high whatever inputs are that is going to affect the output. So, these type of sequential circuits so they are known as latches or transparent or level triggered flip flop. So, these three terms are synonymous latch transparent flip flop and level triggered flip flops, so these three terms are synonymous.

So, latch, transparent flip flop and level triggered flip flops, so these 3 terms are synonymous. On the other hand, see, if we have some edge triggering, so, edge triggering means this clock this enable line or clock line whatever you call it. So we put a hat at the, triangular shaped structure at the beginning and then whatever will be the input so that is coming. So, this is input output now this is so this means that we will be considering the input changes only at the edges of this signal ok.

So, whatever you call it. So, normally we call it a clock signal so this whenever this clock edges come. So, they will be considered so you can make symbols for this low level triggered and falling edge triggering. So, low level triggered so this is represented by a diagram like this enable line we put a bubble at the end so that is the low level triggered; similarly for this falling edge triggered flip flop, so this clock signal is shown like this the clock signal is shown like this so that is a falling edge triggered. Whereas, this one is a rising edge triggered so this type of notations are used for due to distinguish between different types of flip flops we have.

(Refer Slide Time: 22:48)



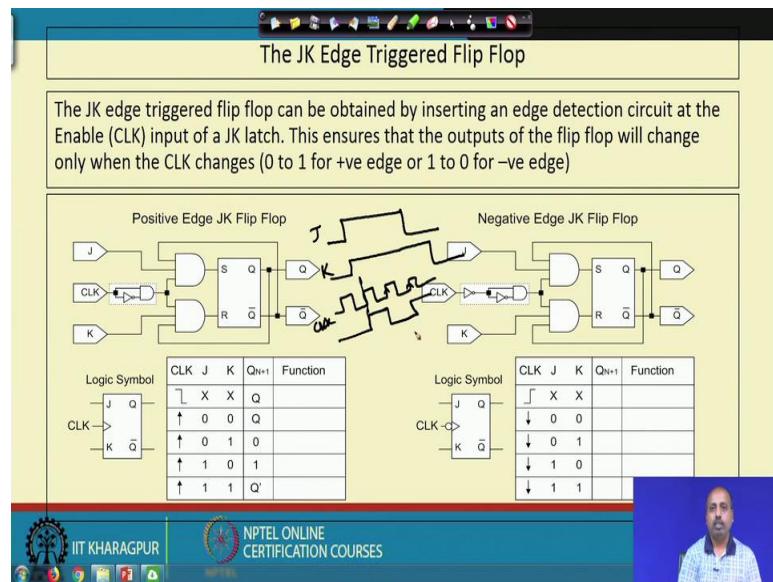
Next we will look into how to detect the edge so we have said that I want to detect the edge, but how to detect it. So, edge detection circuits they are used to detect the transition of the enable line from logic 0 to logic 1 or from logic 1 to logic 0. So, it can be done like this see suppose I have got this enable signal and we put an inverter and end these 2 signals, then what will happen if this is the enable signal it was low for this much time now then it is became high and then again it is low.

So, you want to detect this situation that when the transition occurred from low to high. For detecting that situation we put an inverter so due to the delay of the inverter this \overline{EN} signal, so this will be delayed by some amount of time ok; so this is delayed by some this much of time. Now if you take this AND gate then what will happen this so this output of AND gate will be high only for this small amount of time. So, this is not so if you considered the delay of and get also.

So, you see that so for this much for this period of time both the signals are high ok. So, that will be coming to that that comes to the output of the AND gate so that corresponds to this much time and it is slightly time shifted and that delay from here to here. So, this is due to the delay of the AND gate, but anyway so by using this circuit. So, I can detect the precise time at which this transition has occurred from low to high, similarly if you are going for to detect high to low transition then we can go we can make the circuit like this, so this enable line is inverted and then another inversion.

So, that comes like this and then if you go on doing this ending you will see that ultimately it will land into a pulse on this line when the which actually corresponds to this transition this high to low transition, so this corresponds to this transition. So, this way we can use these digital gates to realize this edge detection circuits, and then use that this EN' signal as the edge triggering clock signal for the flip flops.

(Refer Slide Time: 25:16)



We will next look into edge triggered J K flip flop. So, this edge triggered J K flip flop, so as I said that edge triggering can be positive edge triggered or negative edge triggered. Now you see that this circuit is otherwise similar to whatever we have seen this previously we this name of this line has changed from enable to clock otherwise the lines are same.

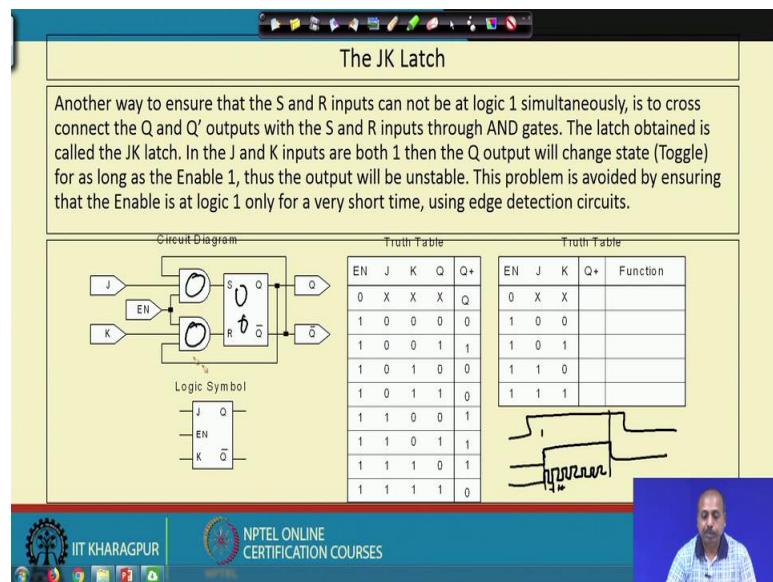
So, previously so this line was drawn directly and connected here, now we have put a small edge detection circuitry between before this. So, that way we are we have done it like this now what happens is that this only during the time for which this line is high and that precisely happens when the clock is making a transition from high to low. So, only when this making a transition from high to low sorry, when this transition is low to high like this then only this output will be equal to 1 and then only this J and K values will be allowed to come to this is S and R. So, whenever this clock is high or clock is low or clock is making a transition from high to low. So, this output at this point will be 0 as we have seen in the previous slide, so you see that when only when this was making a transition from

low to high. So, this pulse came otherwise this EN' line was always 0, so this following edge was not detected here.

So, that is why so if it is a falling edge or high level or low level of the clock signal with respective of the values of J and K this Q will maintain its previous value; whereas, if there is a rising edge like this then this depending upon the value of J and K this Q the Q value will change. So, it will be becoming equal to 0 or 1 or either it will maintain its previous value or it will be changing.

Now because of this edge triggering so we cannot have the we cannot have these toggle because, this by the time this toggle if the toggle has to occur then this clock has to appear again this clock transition has to appear again and for that we do not bother. Because, in the previous case what was happening is that you will be depending upon the delay of this J K flip flop, depending upon the delay of this gates so this is S and R settings were being done.

(Refer Slide Time: 27:53)



So, what was happening is that: so if this J is high for this much of time and K is high for say this much of time, so during this time this Q output will be continually changing like this, Q output will be continually changing like this. So, why this thing happens so this particular delay is equal to the delay of this gates this, the AND gates and all and this the NAND gates or NOR gates that we have inside. So, this is determined purely by the

combinational elements that you have in the circuit; whereas, for the edge triggered circuit so we are not we will be getting this type of behavior, but only at the clock boundaries.

So, that is why we are we are it is much safer, like here also if the J and K both the signals are high, if J and K both the signal are high like say J is high for this much time and K is high for this much time. Now in between if you have the clock signal going like this so this is the clock then the transitions you will see the Q output is, so at this point the Q value will change the Q was Q will change here. Again at this point at this point the Q will change again at this point Q will change, so that is determined by the clock boundary.

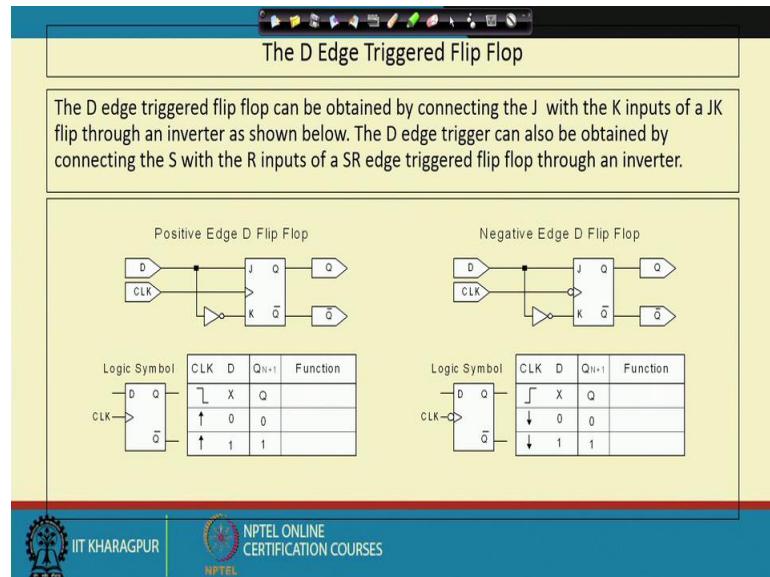
So, it is controlled it can be controlled externally by controlling the clock signal you can control the toggling rate whereas, for the latch JK latch we do not have that control, so that is purely determined by the delays of the constituents gates in the latch. So, this J K edge triggered flip flop it solves that toggling problem and that way it is helpful.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 30
Sequential Circuits
(Contd.)

Next, we will look into D edge triggered flip flop. So, it is similar to the JK edge triggered flip flop.

(Refer Slide Time: 00:21)



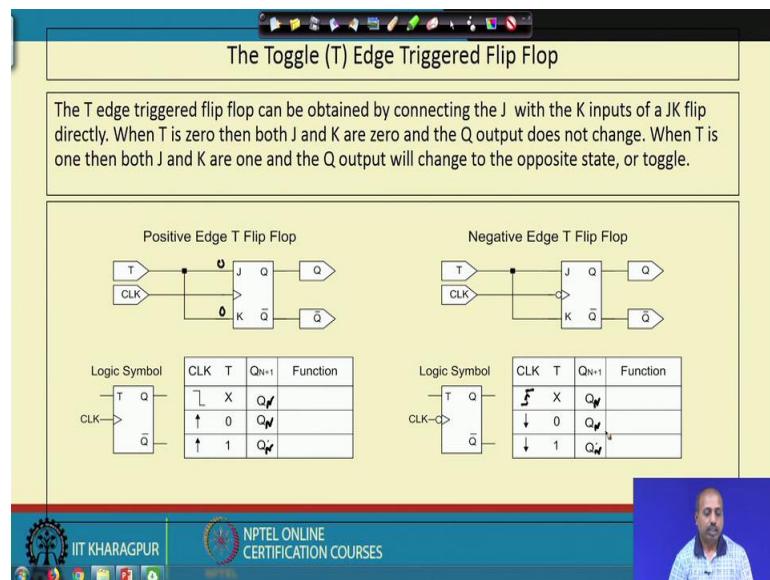
But, this D edge triggered flip flop can be obtained by connecting the J and K inputs or JK flip flop using an inverter. So, just like that instead of JK, it can be SR also.

But it does not matter, but this if we connect an inverter between J and K, then it will behave as a D flip flop. So, it can also be obtained by connecting inverter S and R inputs through an inverter and the behavior is already behavior is whatever we have said so far that as long as the clock signal is a high or low, all it makes a high to low transition. The Q value does not change and so we write it as Q_{N+1} . Sometimes, we write it as Q_{N+1} . So, to mean that at the $N + 1$ -th time instant, what will happen? So, if we are currently at the N -th time instance.

So, that is Q_N and then at the $N + 1$ -th time instant what will happen? So, to be precise; so, you can write it as Q_N ok, so to be more precise instead of writing Q ; so, you can write Q_N . So, otherwise when the clock signal is making a transition from low to high. So, depending upon the value of D , it will either become 0 or it will become 1.

So, this way we can have this D type edge triggered flip flop.

(Refer Slide Time: 01:45)



Another flip flop that we get from this JK flip flop is known as the toggle flip flop or T flip flop. So, this T flip flop is simply the JK flip flop when this J and K inputs are tied together. So, they behave like this since this is JK flip flop internally. So, you see, whenever I make this T value equal to 1, then this J and K, both are they both of them are getting 1 as a result whatever we was the previous value of Q. So, that will be complemented.

So, you see here that when this T value is equal to 1, then at the rising edge of the clock this Q_{N+1} will be equal to Q'_N to be more precise. So, you can write it as Q'_N whereas, if the T value is 0; that means, this J and K both are both of them are getting 0. So, as a result, the value will not change. So, it will remain as Q_N and if the clock is not active that is it is high low or making a high to low transition, then also it remains at the previous value.

So, this way, we can have this T flip flop made, it is a positive edge triggered T flip flop because the triggering is occurring at the positive edges of the clock, we can for all the flip

flops that we have discussed. So, there can be negative edge triggered version also where at the falling edge so have seen previously, how to detect the falling edge of the clock signal.

So, when the falling edge is coming, then the signal the value of the Q will change. So, here when the clock signal is low or high or making a low to high transition in those cases Q_{N+1} remains equal to Q_N . So, the value does not change where as if it is making a transition from high to low, then if the T value is 0, then this Q_{N+1} will may maintain the previous value of Q that is Q_N and if it is equal to 1, then it will toggle.

So, it will change. It will become equal to \bar{Q}_N . So, this T flip flop is also a special case of JK flip flop. So, you see that often this JK or SR it is actually the main flip flop that we design and from there, we can derive this remaining type of flip flops. So, from SR flip flop. So, you can come to D or JK flip flop or you can from the JK you can come to D or T flip flop. So, like that you can get one flip flop from the other.

(Refer Slide Time: 04:28)

Flip Flops with asynchronous inputs (Preset and Clear)

Two extra inputs are often found on flip flops, that either clear or preset the output. These inputs are effective at any time, thus are called asynchronous. If the Clear is at logic 0 then the output is forced to 0, irrespective of the other normal inputs. If the Preset is at logic 0 then the output is forced to 1, irrespective of the other normal inputs. The preset and the clear inputs can not be 0 simultaneously. If the Preset and Clear are both 1 then the flip flop behaves according to its normal truth table.

Positive Edge JK Flip Flop with Preset and Clear

CLK	PR	CLR	J	K	Q_{N+1}	Function
0	0	X	X	X		
↑	0	1	X	X	1	
↑	1	0	X	X	0	
↑	1	1	0	0	Q	
↑	1	1	0	1	0	
↑	1	1	1	0	1	
↑	1	1	1	1	\bar{Q}	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, some flip flops they come up with some asynchronous inputs which are known as preset or clear. Now one question like whenever I am suppose I give you 1 flip flop.

Suppose, I give you 1 flip flop and it is some type, say D type, ok. Now suppose this is say D and this is say Q and \bar{Q} . And if I say; if I give D equal to 0, if I give equal to 0, there is

a clock, then what is the value of Q? Ok the answer that we have got so far; it says that the value of Q will be same as its previous value, but what is the previous value of Q.

So, that is again dependent on the previous the time before that. So, that way how long we go back. So, that is we cannot go back infinitely. So, what is done is that there must be a starting point ok. So, there must be like if this circuit, if I make a circuit like this and switch on the circuit then what will happen is that this Q and \bar{Q} . So, they will pick up some arbitrary values and from that point onwards depending on the value. So, it will go on operating.

But the system behavior will become unpredictable for SR latch, we have seen that we should not apply the pattern 1 1, but we do not have any control over that ok. So, we somehow want to the Q and \bar{Q} should always be complementary that is also said, now we do not have any control, if we do not have any; if we just switch on the circuit and this Q and \bar{Q} can pick up any value.

Now, how can we control it ok? So, for that purpose, most of this flip flops that we have studied, they come up with two extra controls which are asynchronous in nature one is known as preset another is known as clear. So, preset and clear as long as this, they are be 0 0. So, if it is in this particular case in this particular case. So, this is a rising edge triggered JK flip flop. So, if it is for these cases, if it is falling as long as they are falling edge falling edges or the high or low, this preset and clear both are 0 0, then nothing happens.

So, this remains it; it continues to the previous value, on the other hand, if this if there is a rising edge like this, if there is a rising edge here from low to high, then this preset and clear, preset is equal to 0 and clear is equal to 1, then it is this Q_{N+1} will be equal to 1. So, that is we can we reset this flip flop value to 1; so, irrespective of the value of J and K. So, if this preset value is preset, control is made equal to 0, then this Q output will become equal to 1.

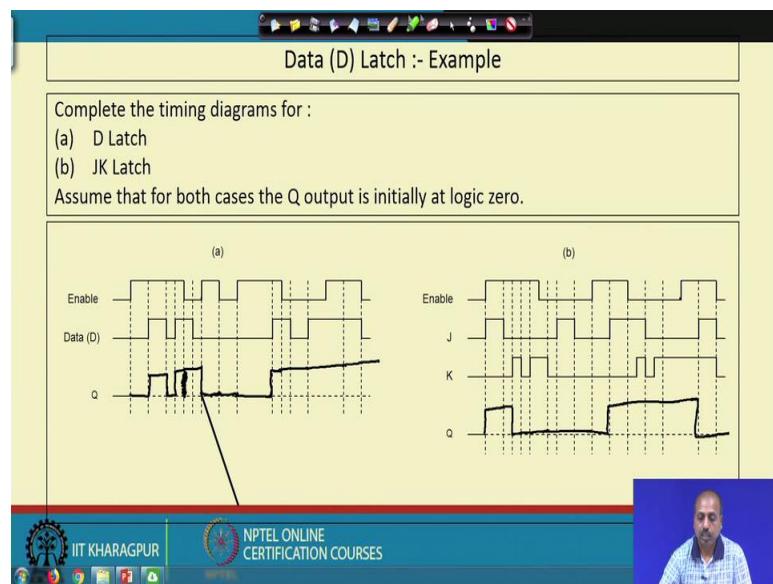
Similarly, if this depending upon this preset and clear, if this clear is made 0; so, irrespective of the value of this J and K, this Q_{N+1} will become equal to 0; the flip flop content will be cleared. So, that is why the name preset and clear. So, irrespective of the values of J and K, if this clock signal comes the rising edge of the clock signal comes, then this preset and clear will be presetting the value on the other hand, for the proper operation of the flip flop, this preset and clear both of them should be made equal to 1. So, they are

always made equal to 1. So, then you can just when they are made equal to 1. So, then this flip flop operates as a normal JK flip flop ok.

So, if we read this particular text, it says that 2 extra inputs are often found on flip flops that either clear or preset the output. These inputs are effective at any time and that is why they are called asynchronous. If the clear is at logic 0, then the output is forced to 0 irrespective of other normal inputs if the preset is at logic 0, then the output is force to 1 irrespective of normal input. The preset at, and the clear inputs cannot be 0 simultaneously. And so, it is a design constant that is they cannot be made 0 simultaneously.

So, that is why it is the designer has to, the user has to ensure that I am not making this preset and clear 0 simultaneously and if both of them are 1, then the flip flop will behave as a normal flip flop ok, otherwise, it will be behaving as per this preset and clear. So, this if the preset signal is 0, then this then this circuit will be preset and if the preset signal is if the clear signal is 0, then the circuit the flip flop will be cleared. So, that is why it behaves like this. So, we have got another exercise.

(Refer Slide Time: 09:30)



So, we have got say this D latch, how does it work? So, it is, it says that complete the timing diagram for D latch assume that both cases; both cases the Q output is initially 0. So, you see that this is D input is high at this point. So, till this much, it continues like this because enable is high, but D is also 0, then it is high, then it will become low, then it is it

will follow that D as long as enable line. Sorry, this is not correct as long as the enable line is active. So, it will follow that D line, then enable becomes deactive at this point.

So, it will continue like this, then enable becomes active at this point again. So, this D value is 0. So, it comes down, then it then it goes like this enable is low enable is high. So, it continues like this D value is also low, then enable is low. So, it continuous like this enable becomes 1, but the D value is 0. So, it goes like this enable is 1 and D is 1. So, it becomes high, it goes like this, then enable becomes low, but D is D was high.

So, it continuous to hold the value like this next change can occur only at this point ok. So, D is already 1. So, nothing happens and then at this point enable is going low and D is also going low. So, it will continue like this. So, you can draw the timing diagram for the JK accordingly. So, here this Q is 0. Now this enable is high and J is 1. So, as a result, it will become high, it will continue like this, then at this point K is high. So, it will come low and then it will be K is again becoming high J is continuing to be 0. So, after this enable has become 0. So, enable becomes 1 at this point, but at this point J and K both are 0. So, as a results; it will continue with the previous value of Q and then at this point J is equal to 1.

So, it will rise like this and it will be then enable becomes low. So, as a result, nothing happens in this part of time, then enables becomes high at this point and then we have got J and K both are high at this point ok. So, it will be making a transition like this and it will continue like this.

So, this way, we can make the timing diagrams for this D latch and JK latch, next we will look into some circuit changes.

(Refer Slide Time: 12:25)

JK Edge Triggered Flip Flop :- Example

Complete the timing diagrams for :

- (a) Positive Edge Triggered JK Flip Flop
- (b) Negative Edge Triggered JK Flip Flop

Assume that for both cases the Q output is initially at logic zero.

(a)

CLK

J

K

Q

(b)

CLK

J

K

Q



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, this diagram also, you can continue, I am not doing it. So, you can just draw the timing diagram fill up this \bar{Q} and Q output in a similar fashion.

(Refer Slide Time: 12:40)

D and T Edge Triggered Flip Flops :- Example

Complete the timing diagrams for :

- (a) Positive Edge Triggered D Flip Flop
- (b) Positive Edge Triggered T Flip Flop
- (c) Negative Edge Triggered T Flip Flop
- (d) Negative Edge Triggered D Flip Flop

(a)

CLK

D

Q

(b)

CLK

D

Q

(c)

CLK

T

Q

(d)

CLK

T

Q



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

This is another example. So, you can I am just putting this slide so that you can just make a few practice ok.

(Refer Slide Time: 12:45)

JK Flip Flop With Preset and Clear:- Example

Complete the timing diagrams for :

- (a) Positive Edge Triggered JK Flip Flop
- (b) Negative Edge Triggered JK Flip Flop.

Assume that for both cases the Q output is initially at logic zero.

(a)

CLK
J
K
CLR
PR
Q

(b)

CLK
J
K
CLR
PR
Q

(Refer Slide Time: 12:48)

Level Triggered Master Slave JK Flip Flop

A Master Slave flip flop is obtained by connecting two SR latches as shown below. This flip flop reads the inputs when the clock is 1 and changes the output when the clock is at logic zero.

Logic Symbol

CLK	J	K	Q	Function
1	0	0	1	Set
1	0	1	0	Reset
1	1	0	0	Toggle
1	1	1	1	Hold

(a) Positive Master Slave JK Flip Flop

CLK
J
K
Q

(b) Negative Master Slave JK Flip Flop

CLK
J
K
Q

Now if we try to there is another type of JK flip flop which is known as master slave JK flip flop. So, the problem with edge triggering was that. So, we were requiring some extra edge detection circuit for making it to operate properly. So, if we instead of that what we can do? We can use a, use 2 such flip flops, 2 such JK flip flops and the ones 1 is called the masters flip flop.

So, this is the master and this is the master flip flop and this is the slave flip flop. So, master flip flop operates at the point when the clock signal is high and the slave operates when

the clock signal is low because there is an inverter connected like this. So, you see that when the clock signal is high then this J and K. So, depending upon this value of J and K this Q and \bar{Q} will come. Now during this, then the then the clock signal will become low when the clock signal becomes low, then what ever values of the whatever the values of this J and K. So, they cannot change this SR flip flop.

So, the previous value will be remaining latched here. Now this slave flip flop will be following the JK flip the master flip flop. So, whatever master have done previously slave will do now. So, this way we can. So, this 2 SR latches. So, the flip flop the reads the value of inputs and the clock is 1 and changes the output when the clock is at logic 0. So, here for example, this J and so, when the clock signal is high this Q output does not change ok. So, this Q output will change when the clock signal is low. So, when the clock signal was high this J and J was 1 and K was 1. So, it evaluates to 1 and when the clock signal becomes low at that time the signal makes a transition, it goes, it becomes high like this.

Now, again at the next time instant; so, by that time when the clock signal was now when the clock signal becomes high depending upon this J and K. So, J value is low here K value is also low here after sometime K value becomes active K value becomes, so, so it becomes to 0. So, it this master flip flop becomes equal to 0 at the end of this clock.

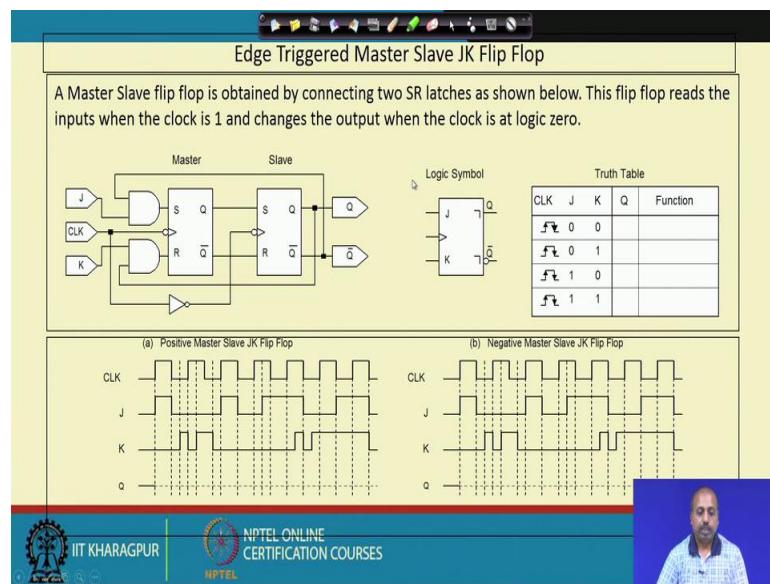
So, at the end of at this clock boundary; at this clock boundary when the clock signal becomes low; so, it will be making a transition to low and it will continue like this. So, this way depending upon this clock signal boundaries, so, whatever the master flip flop has made a transition to the slave flip flop will make the transition in the when the clock signal becomes low. So, this is the master slave JK flip flop that. So, you see that we need a both rising and though both level high and level low situation of the clock for the output to get affected ok.

So, this way so, the advantage is that here also, we do not have that continuous toggling problem, because for this the slave flip flop will change only when once only during when the clock signal is low, and then only those values will be available to the J and K, this the first 2 AND gates. So, these 2 AND gates are values will be available only in the next clock pulse. So, in between previously, what was happening is that the feedback values were coming feedbacks are coming from this points ok, but now the feedbacks are coming from the slave. So, so slave will not change immediately when the master changes. So, as

a result the master will see the change only in the next clock. So, that continuous toggling problem that we had in the JK flip flop or JK latch that will not happen in the master slave organization.

So, this also solves that toggling problem of in JK in JK latch.

(Refer Slide Time: 16:59)



So, we can have this master slave JK flip flop also, the same thing that is you have got this 2 flip flops, and then a 1 flip flop operator on positive edge triggered 1 edge of the clock and the other. So, this works at the negative edge of the clock and this works at the positive edge of the clock, then slave will operate at the positive edge of the clock and accordingly, the master will read the values coming on the J and K.

So, they will be read by the master when the clock signal is, in fact, when the clock signal is 0, then the value will be coming here and when the clock signal is one then the value will be outputted by slave. So, here it is written as when the clock is 1. So, in that case this bubble should not be there and in that it is written that it is evaluated on the clock is 0, in that case, this bubble should not be there.

So, there is a problem with this diagram and this write up anyway, it conveys the basic idea.

(Refer Slide Time: 18:02)

Conversion between Flipflops

SR to JK

JK Inputs		Outputs		S-R Inputs	
J	K	Q _{P+1}	Q _P	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram

K-Map for S = \overline{Q}_P

Q _P	00	01	11	10
0	0	1	X	0
1	1	0	X	1

K-Map for R = $Q_P \overline{Q}_{P+1}$

Q _P	00	01	11	10
0	X	0	1	X
1	0	1	0	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will be looking into some conversion like how can you convert 1 flip flop to another type of flip flop that is very common like; it may be while doing the operation. So, we are looking for some type of flip flop, but what we are getting in our laboratory is of a different type. So, we need to control do some conversion between the flip flop the first conversion will look into is SR to JK flip flop. So, basically we have got an SR flip flop and we want to make a JK flip flop out of that. So, for doing this thing, this type of exercises first we have to make a conversion table. So, in the in the conversion table, we write down the values of this JK and the current value of Q and what is the expected; next value of Q.

So, these if I take these three as input JK and QP; so, this is 3 as input and it see; what is the value expected value of output. So, by from the JK flip flop, we know if it is 0 0 0

Q_{P+1} is also 0 if it is 0 0 1. So, it maintains the previous value. So, this Q_{P+1} equal to 1 that is equal to Q P it is 0 1 0, then irrespective of the value of Q P the output is 0.

So, Q_{P+1} is 0; so 1 0 0. So, irrespective of if the J input is 1 irrespective of the previous and K input is 0 irrespective of the previous value of Q P, the values are one and for if J and K both are equal to 1, then if Q P was 0. Now it will become one if Q P was 1. Now it will become 0. So, if we now we try to mimic this behavior by means of an SR flip flop that is in SR flip flop, we will try to put the values in S and R such that we get this type of behavior that is in a flip flop I have got the present state as 0.

Next state we also want as 0. So, what are the values that we should feed to a S and R and you know that only if I in SR flip flop if I set this S input to 1, then only output will become equal to 1. So, if I maintain this S input as 0 and irrespective of the R input, it will maintain the value 0 0. So, if R input is 0, then also it will remain 0 if R input is 1 that also will be said the flip flop and take it to 0. Similarly, if you have if the current value is 1 and the next value expected is also 1, then what is to be done. So, you can ignore the value of S. So, you can feed it to; we can feed 1 or 0 to S, but you have to feed a 0 to R; so, as a result if you if you set R to 1, then this will change to 0.

So, what is restricted is that this R should, must be equal to 0. So, should get 1 1 here, similarly, we should if we are, if the current state is 0 and the next here also it is same thing; so here also $0 \times 1 0$. So, if we want that the current value is one and the next value, we want as 0, then the R input should be made equal to 1 and S input should be made equal to 0.

Similarly, 0 1 S input should be made equal to 1 and R input should be made equal to 0. So, in this way we make we write down the values to be applied on S and R inputs of the flip flop now we make Karnaugh map that has got the inputs JK and Q_P and the outputs as S and R. So, far the S, we make the Karnaugh map here and for R, we make the Karnaugh map here and after simplification it turns out that we should make it as J, there is there is a it is $J\overline{Q_P}$.

So, S equal to $J\overline{Q_P}$ and R equal to KQ_P . So, if you simplify. So, you will get it like this. So, you see that we are what we have done. So, you have connected two AND gates and this $\overline{Q_P}$ line is taken here and connected here and Q_P line is taken here and connected here. So, as a result, now this structure it behaves as a JK flip flop. So, in this way if we have got one type of flip flop; so, you can very easily convert to another type by making by putting some additional logic into it.

(Refer Slide Time: 22:24)

Conversion Table

S-R Inputs	Outputs	J-K Inputs
S R	Q _P Q _{P+1}	J K
0 0	0 0	0 X
0 0	1 1	X 0
0 1	0 0	0 X
0 1	1 0	X 1
1 0	0 1	1 X
1 0	1 1	X 0
1 1	Invalid	Dont care
1 1	Invalid	Dont care

Logic Diagram

```

    graph LR
      S((S)) --> J((J))
      R((R)) --> K((K))
      J --> QP((QP))
      K --> QP
      QP --> QPplus1((QP+1))
      QPplus1 --> C((C))
      C --> QPplus1
      C --> QP
      QP --> QPplus1
      QPplus1 --> QP
      QPplus1 --> QP
  
```

K-maps

S Q _P	00	01	11	10
0	0	X	1	0
1	1	X	X	X

$J=S$

S Q _P	00	01	11	10
0	X	0	1	X
1	X	0	1	X

$K=R$

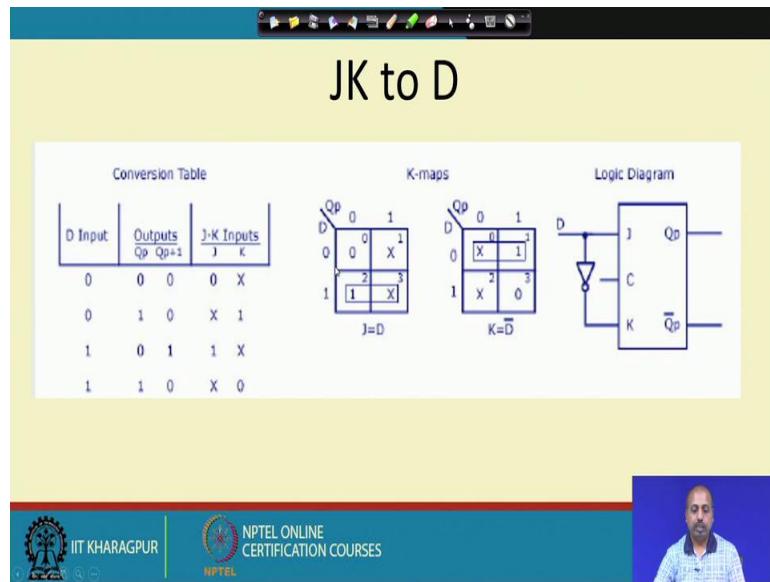
How to do the other side that is from JK to SR for JK to SR?

So, ultimate behavior expected of the S R flip flop is like this that it is 0 0, if it is 0 0, then the if the if the S and R are 0 and Q P is also 0, then Q_{P+1} should be equal to 0 if it is 0 0 and 1, then Q_{P+1} should be 1 that is it is holding the previous value if it is 0 1, then this Q_P whatever be the value Q_{P+1} will be equal to 0. Similarly, 1 0 whatever be the value of Q_P output be equal to one and for SR flip flop or S R latch, we do not apply this 1 1. So, these 2 are not put; so for J K, so, we can simply consider them as don't cares. Now if we want to get this behavior that is what value should be applied to J and K such that we get this behavior that is Q_P is 0 Q_{P+1} is also 0. So, here also this is if you put J equal to 0 if J equal to 1, then this Q_{P+1} will be equal become equal to 1. So, we have to make J equal to 0 K is do not care similarly 1 1. So, K should be equal to 0 and J is a J is do not care, similarly 0 0.

Again the same thing J is 0 and K is don't care. So, in this way, you can fill up this J and K part and then you can do a Karnaugh map and try to simplify this Karnaugh map and then what you get is J equal to S and K equal to 1. So, you do not need any additional gate to convert 1 JK flip flop to SR flip flop and that is quite expected because that is the we have seen previously also the behavior of JK and SR, and we have seen that they work exactly like that JK and SR only thing is that that in SR, this 1 1 input pattern will come that is not there in the JK.

So, we are not bothered about the 1 1 input pattern for JK.

(Refer Slide Time: 24:28)



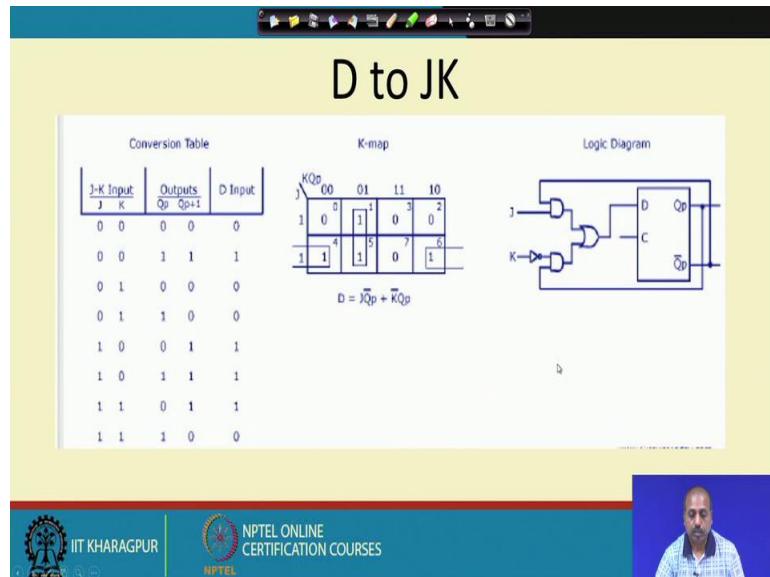
Next, we will be looking into the technique how to convert a JK flip flop to a D flip flop take the strategy is similar. So, for the first, we write down the table for the D flip flop in the D flip flop D input is 0 and Q_P is 0. So, we want that Q_{P+1} should be equal to 0 if it is D is 0 and Q_P is 1, then also it is 0 D is 1 and Q_P is 0. So, this Q_{P+1} should be equal to 0. So, this is the D flip flop behavior. Now if we want to realize this behavior by means of an SR JK flip flop, then what we have to do is. So, I have to apply a 0 at J because to get this K input can be don't care, then it can be since the Q_P is 0 it will always come to 0.

Similarly, so, if we are looking for if the previous value of Q_P was equal to 1 and we are trying to get a 0 here, then this K value must be set equal to 1, then it will be resetting it to 0. Similarly, if the previous value of Q_P was 0 and the next value is 1. So, we should we have to set this J value to be equal to 1. So, that it makes a transition from 0 to 1 and from 1 to 0 again, this x and this K value.

Since this is the D input is 1. So, this is a 1 to this sorry this is D input is 0 previous value of Q_P is also is 1. So, I have to make this, so, then if we do a simplification of this corresponding K map; so, it will turn out that between J and K. So, J is equal to D and K equal to \bar{D} . So, previously you I have said that while making the D flip flop that you have to connect an inverter between J and K inputs. So, these particular derivation shows that

you can do that really that is really the case that we have to connect an inverter between J and K.

(Refer Slide Time: 26:35)



So, can we convert a D flip flop to JK flip flop ok? So, how can we do this now you see that this is a bit cumbersome because this JK. So, we have got this truth table like JK 0 0 0, then the expected. So, this is the expected behavior. Now we try to note down what should be the D value like from 0, I want to go to 0. So, D input should be 0; 1 to 1, so, that should be 1; 0 to 0 0; 1 to 0 0; 0 to 1. So, I need to put a value 1 so that way I write down this D input column.

Now, we make a truth table out of JK and Q P for the D and it if you do a simplification, then you can find that it turns out to be a function D equal to $J\overline{Q}_P + \overline{K}Q_P$. So, if you are given this add this from this J and K we have to derive this two this function for D ok. So, $J\overline{Q}_P$. So, \overline{Q}_P . is taken it is ANDed and $\overline{K}Q_P$ the Q_P is taken K is inverted and ANDed and they are ORed here and then put to D; so this logic. So, this will be converting 1 D flip flop into a JK flip flop. So, internally it remains a JK flip flop; but sorry internally it remains a D flip flop, but you we can realize a JK flip flop out of that.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 31
Sequential Circuits (Contd.)

(Refer Slide Time: 00:21)

The slide is titled "SR to D". It includes the following components:

- Conversion Table:**

D Input	Outputs		S-R Inputs	
	Q_p	Q_{p+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0
- K-maps:**
 - For S = D:

D	Q _p	0	1
0	0	0	0
1	1	X	X
 - For R = \bar{D} :

D	Q _p	0	1
0	0	1	0
1	1	0	1
- Logic Diagram:**

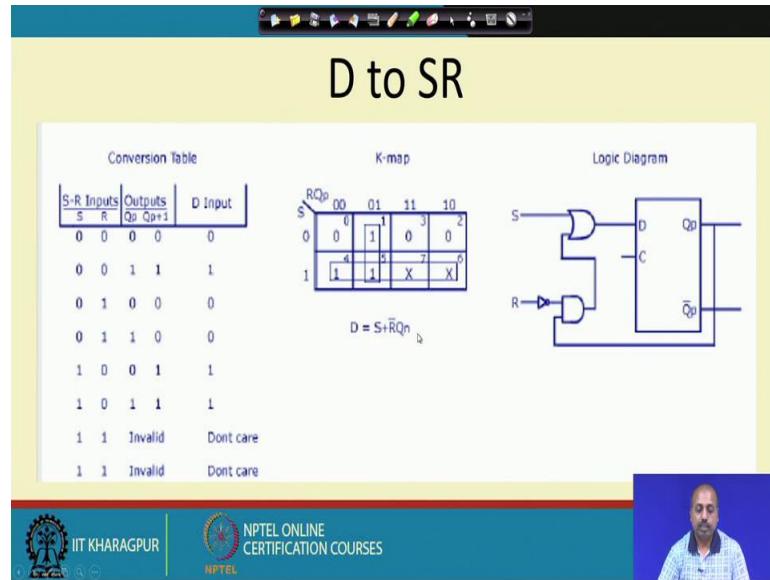
```
graph LR; D((D)) --> OR(( )); OR(( )) --> S((S)); OR(( )) --> C((C)); C(( )) --> NOT(( )); NOT(( )) --> R((R)); R(( )) --> QP((Q_p)); QP((Q_p)) --> AND(( )); AND(( )) --> QP1((Q_{p+1})); AND(( )) --> NOT1(( )); NOT1(( )) --> SR(( )); SR(( )) --> QP1((Q_{p+1}));
```

Next we will look into conversion from SR flip flop to D flip flop. So, here also for the since finally, we want D flip flop. So, we write down the operation for D flip flop that is if the D input is 0 and previous output is 0, then it is output Q_{p+1} should be 0. So, like that now for S and R, we write down the corresponding values that should come like for 0 to 0 this S value should be equal to 0, R value can be don't care. Now, 1 to 0 S value should be 0 and R value should be 1 and from 0 to 1, S value should be 1 and R value should be equal to 0.

So, we have to be careful that we do not put 1 1 to SR flip flop. That is why though I could have written an X here, but you do not do that because that will be putting on that, that leads the chance that will be putting 1 1 on to SR. Just to avoid that so, it is hard coded to 0. Similarly, from 0 to 1 so, this is hard coded to 1 0 and this 0 is not, don't care, and this 1 1 so, this is I can put this R value as 0 and S value as don't care.

So, if you do a simplification and then, we find that between S and R, we need to put an inverter, so S is equal to D and R equal to \bar{D} . So by putting this inverter so we can get this SR flip flop converted to D flip flop.

(Refer Slide Time: 01:51)



The vice versa, the other way D to SR conversion; so, this is also similar, like what we do. We first write down the table for this SR, and then, for 0 0 can Q_p equal to 0, then Q_{p+1} is also 0 0 0, then Q_p is 1, then the Q_{p+1} is 1. So, that way we make this table, but this 1 1, so this is invalid and then, because for SR we are not going to put 1 1 into it. So, they are kept as invalid.

Now, we consider what is the, what should be the value of D like from 0 to 0. So, D is 0 from 1 to 1, D is 1 from 0 to 0, D is 0. It goes like this, then for these invalid cases, since we are not going to apply this pattern, so they are taken as don't care. So, make it Karnaugh map and try to see what is the expression for D and the expression for D becomes $S + \bar{R}Q_n$. Basically this is Q_p somehow not written properly that is that $S + \bar{R}Q_p$. So, R is inverted Q_p is taken, they are ANDed here and they are ORed with S and that is fed to the D flip flop.

(Refer Slide Time: 03:16)

Conversion Table

T Input	Outputs	J-K Inputs
	Q_p Q_{p+1}	J K
0	0 0	0 X
0	1 1	X 0
1	0 1	1 X
1	1 0	X 1

K-maps

Logic Diagram

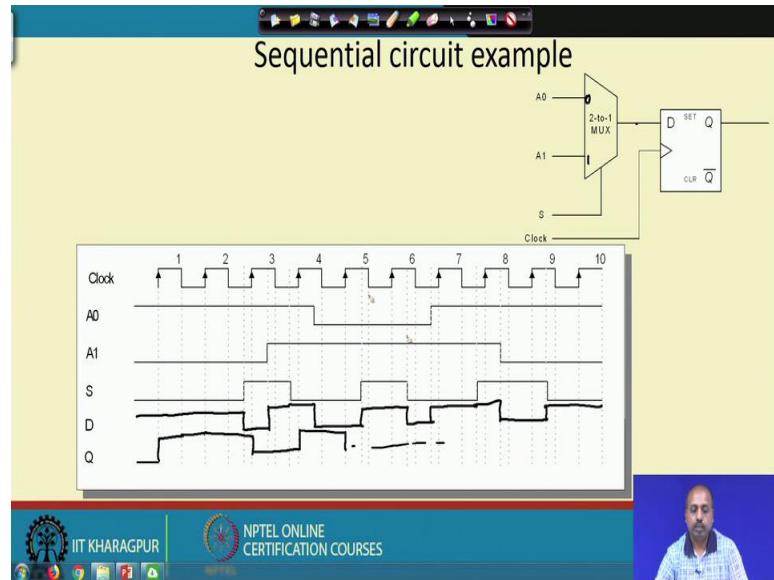
So, this way we can make conversion from D flip flop to SR flip flop. JK to T conversion, so, we already know that you for JK if you tie this, J and K inputs together, so it will give me T flip flop. So, why does it happen like this? For T the behavior expected is if the T value is 0, output will not change. If the T value is 1 output will toggle, ok. So, for 0 if the previous value is 0, the next value is also 0. Similarly if the previous value is 1, the next value is 1. Whereas for, if the T input is 1 so, this Q value is if it was 0 now, it will become 1 and if it is 1 now, it will become 0. The T is 1 if Q_p is 1, then it will become 0.

Now, for this J K what happens is that we have got this. So, for the JK so if you want to get this type of behavior, then from 0 0, this J input must be equal to 0, K input can be don't care for 1 1, K input must be equal to 0 because this J input being 1 will set it to 1, J input being 0, it will maintain the previous value.

So, both the cases, the value will be equal Q_{p+1} will be equal to 1. Similarly here if we are trying to get from 0 to 1, so J input must be made equal to 1, K input is don't care. So, if K input is 0 by virtue J equal to 1, so it will set the flip flop to 1. If K equal to 1 by virtue of toggling property of JK flip flop, it will toggle from 0 to 1. So, in either case, so this value will become equal to 1 and from 1 0 so, this J input can be don't care, where K input is equal to 1. So, it will be resetting the flip flop to 0. Now, if you draw the Karnaugh map and then, we see that minimize it, then it turns out that J equal to T and K equal to T.

So, both the cases so this T input should be connected to both J and K, so, that is the tying of this J and K inputs together and calling it as T input. So, this way we can convert 1 JK flip flop to T flip flop.

(Refer Slide Time: 05:29)



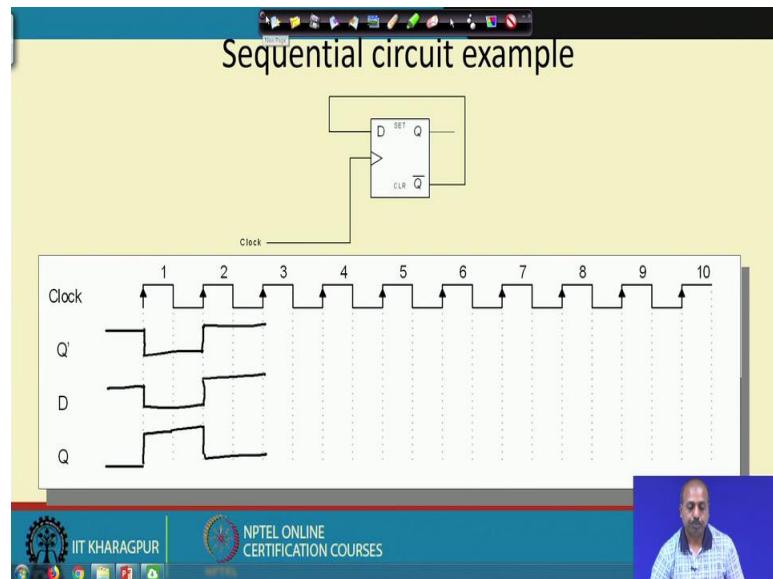
So, here is again another exercise that says that how can you, how this circuit will behave when you will be, when you will set this behavior of this circuit. Now for that matter so you can do it like this. First of all, you can try to get what is the value of D here. So D, so your S signal is low here. So, as long as S is, let us say this is 0 and this is 1. So, when S is low, A0 is selected. When S is 1, A1 is selected.

So, for this much time this D input is high and at this point, it is becoming low and then, it is becoming, it will remain low for this time. Then, A1 is becoming high, ok. So, it continues like this then S is becoming low. So, A0 will get selected, but A0 is high here. So, it will continue like this and then, at this time it will be coming down and then, S is low for this much time. It will be like this then this A1 is high here. So, A1 is high here, A1 get selected and then, S is becoming low here and at this time you see that A0 is also low. So, A0 is low. So, it will come down and then, it will continue like this, then at this point it will be A0 is becoming high. So, it will become high. Now, again at this point S is becoming high. So, A1 will get selected, A1 is high. So, at this point it will come down and then, it will continue like this and then, S is becoming low and A0 is high. So, it will go like this.

So, this is the behavior of D line. Now, from this behavior, you can very easily draw Q line behavior for the D flip flop. So, because at this point your, this Q is high. So, this will be high and then, at this clock edge is basically you have to see. So, this is high then again at this clock edge it is low. So, it will be coming down here and then, again at this clock edge, it is high. So, it will be going high next clock edge, the signal is low, the signal D is low. So, this way you can draw the rest of the part.

So, for any sequential circuit tracing, first you determine the value at the input of the flip flops and then, based on the clock you just try to draw it. So, that is the technique for doing it.

(Refer Slide Time: 08:28)



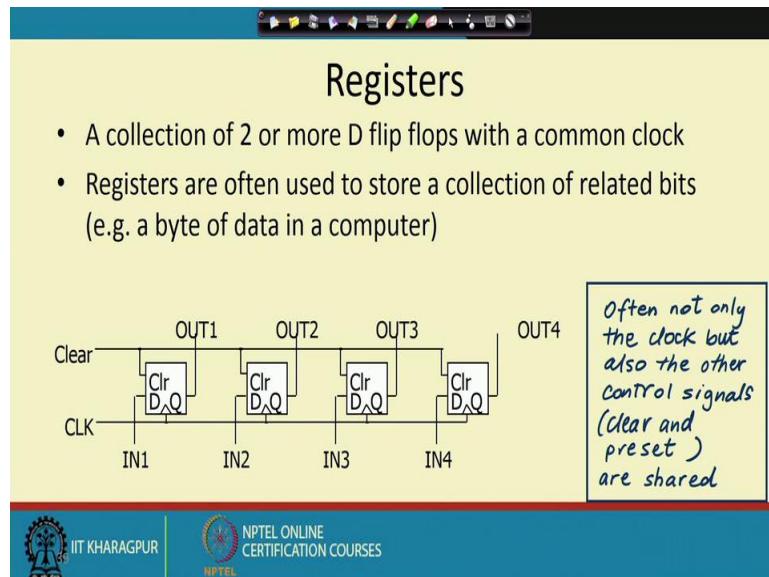
So, this is another example where this \bar{Q} line has been fed back to D. Now, see this. So, this clock is initially, so \bar{Q} line is initially high. So, when Q line is initially low so, you can say that D line is initially high, ok. Now, since this is a D flip flop then this \bar{Q} line is fed back. So, this Q value, so this is getting high at D. So, at the clock edge, this value will be going up then and as soon as this goes up, these goes down, ok. Then, it continues like this till the next clock.

Now, at the next clock what is the value of D? So, that is equal to the value of \bar{Q} , ok. So, this whenever this \bar{Q} has become low, this D value has also become low. So, at this point this \bar{Q} value is low. So, D value is low and this Q will be coming down because it is coming as low here. So, it will be coming down and that it will continue for this much of time and

this then this \bar{Q} line so, that is the compliment of this thing Q will continue till this and D line since it is equal to \bar{Q} , it will go like this.

So, this way you can draw the circuit, you can complete, you can draw the timing diagram and complete the portion. So, I am leaving it for you as a practice.

(Refer Slide Time: 10:08)



Next will be looking into another sequential element which is known as register, so far we have seen flip flop and in a flip flop, you can see that we can store only 1 bit of information. If the clock signal is not given, then if I have already D type of flip flop, then it will remember what was the last value of D given to it.

So, that is why it may be considered as 1 bit register. Now, if I want to remember multiple bits, instead of single bit if I want to remember multiple bits, then what is needed to be done is that we should have more than 1 D flip flop. We should have a collection of two or more D flip flops with a common clock and this type of structures, they are known as registers.

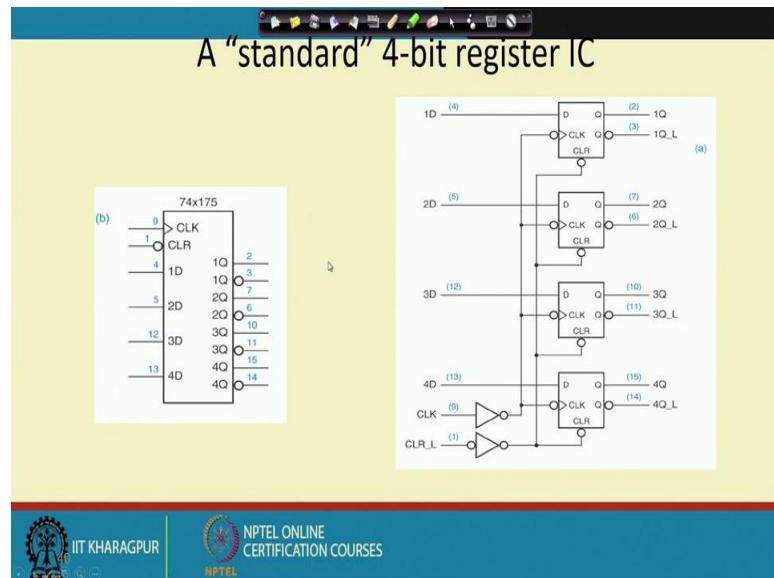
So, registers are often used to store a collection of related bits for example, a byte of data in a computer. So, like here this particular structure you see it, we have got 4 D flip flops and the clock signal is common to all of them. So, at the positive edge of the clock whatever value is coming on to IN1 will be stored into the first flip flop. Similarly, whatever coming

IN2 will be coming in the second flip flop, IN3 will be coming the third flip flop and IN4 will come to the fourth flip flop.

So, I can say that on the positive edge of the clock the 4 bit pattern that I am giving here will be stored in these flip flops and they will be available on this outlines and also, there is a common clear line connected to all of them. All the flip flops they are connected using this clear line. If the clear is equal to clear is set or is active, then all the flip flops content will be cleared. There may be some preset control lines also that is not shown in this diagram. So, that may preset all the flip flops to 1. So, this type of structures are very common inside computers and they are known as the registers.

So, next we will be looking into this construction of the registers using these flip flops.

(Refer Slide Time: 12:20)



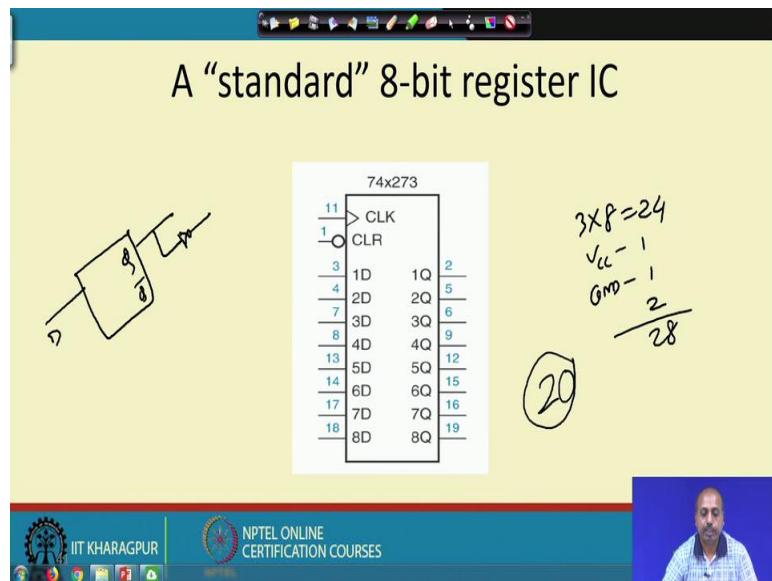
So, this is a standard 4 bit register IC 74 175. So, there are 4 bit means, there should be four such flip flops inside and each flip flop has got 1D input and 1Q output $1\bar{Q}$ output, fine. So, it is written as 1D 1Q 1Q and there is a bubble shown here. That means, the pin number 2 is Q line and pin number 3 is the \bar{Q} line for flip flop 1.

Similarly, these 2D means this is the D line for the second flip flop pin number 5 and its outputs are 2Q and $2\bar{Q}$. So, it is second flip flops Q line is 7 and \bar{Q} line is 6. So, this way this chip is there and this clear and clock. So, these are common for all the flip flops. If the clear signal is made 0, so all the Q values will be made equal to 0 and \bar{Q} line will be made

equal to 1 and then, this the clock signal is required and this symbol shows that it is a positive edge triggered flip flop.

So, internal diagram is like this. So, you have got this 4 flip flops, and this individual flip flops so, they are having this D line clock line and the clear line and there are outputs like Q and \bar{Q} . So, this way so this standard 4 bit register IC there. So, you can use it for design.

(Refer Slide Time: 13:56)



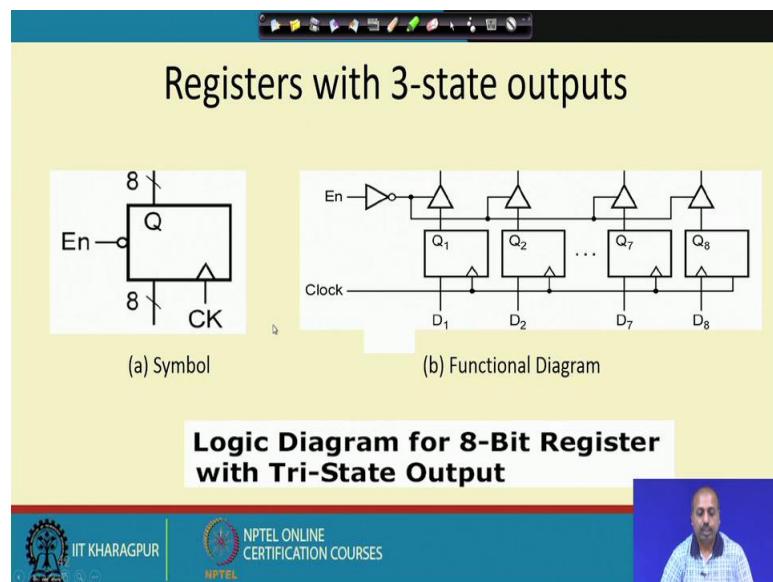
You can have 74 273, this is another IC, another integrated circuit chip where we have got this 8 such flip flops, so, 1D 2D 3D 4D and then this 1Q 2Q. So, interestingly you see that \bar{Q} lines are not taken out ok.

So, \bar{Q} lines are not taken out because for D flip flop. So, if you put both Q and \bar{Q} lines for 8D flip flops, so we will be needing how many pins? So, each flip flop requires 3 pins; 1D 1Q and $1\bar{Q}$. So, 3×8 , so 24 pins will be needed there itself + there is a supply voltage V_{cc} . So, that is one pin, then the ground pin, ground lines that will be 1 pin and this clock and clear. So, there are 2 more pins. So, $24 + 4$, 28 pin, but this 74 273, it has got lesser than 24-28 pins. So, this you can see here up to a pin number 19. So, 1 2 3 4 5 6 7 8 9 and 10 is not shown here, 10 is actually by the supply line, the V_{cc} line and there is another ground line.

So, total it will have 20 pins in it. So, to fit that design into 20 pins, what has been done, the \bar{Q} lines have been cut. So, \bar{Q} lines are not provided. So, if externally you really need \bar{Q}

line in your design, you can put an inverter and do it. So, what I mean is that in, so you have got this D line and this Q lines. So, \bar{Q} pins are not taken out. So, if you need externally \bar{Q} , so put an inverter here and take the \bar{Q} from there ok, but in most of the cases, we do not need both Q and \bar{Q} lines. So, that is why it is many chips will not have this explicit \bar{Q} lines in them.

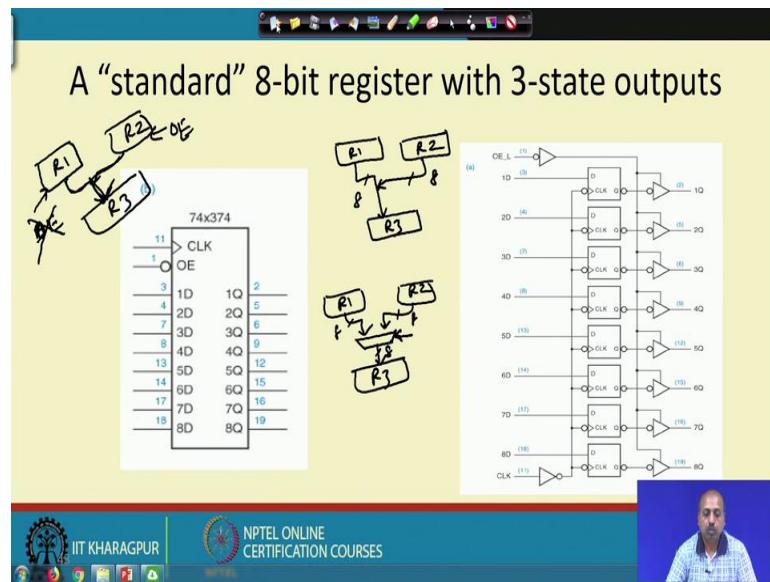
(Refer Slide Time: 16:00)



Some of the registers, they have got 3-state output. So, 3-state output means that the output can be 0. It can be 1 or it can be in a high impedance state that is neither 0 nor 1. So, it is like this. So, there is an enable line. So, if this enable line is active that is enable line is 0 then only this whatever be the state in the register so they will be available on this 8-bit line. So, there is 8-bit input and 8-bit outputs. So, whatever be the 8-bit value that you are giving here that is stored into this flip flop, but that is not immediately visible at the Q output for getting the output here so, you have to make enable equal to 0, then only they will be visible.

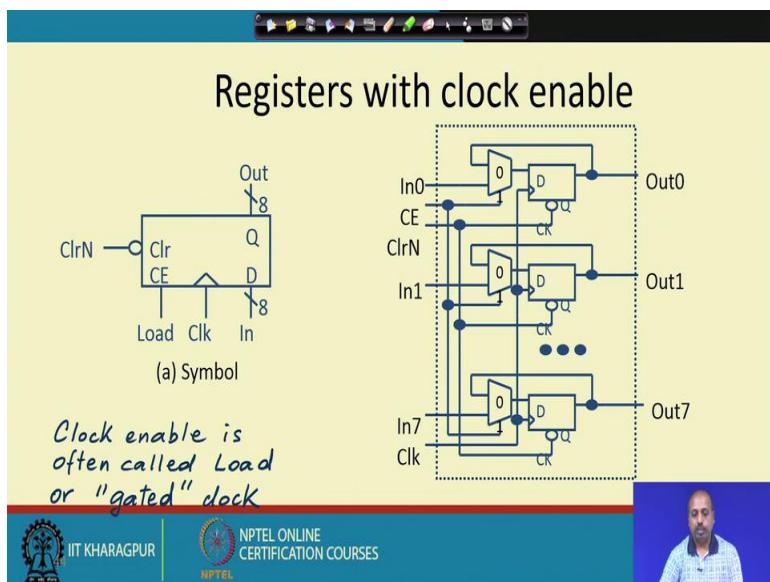
So, this type of gates, they are known as tri-state gates. So, this tri-state gates are enabled by means of this enable pin. So, if this enable line is 0, then this tri-state gates are enabled. As a result this Q outputs are visible at this at the output of the tri-state gate. So, this helps us in connecting a number of register outputs together ok.

(Refer Slide Time: 17:14)



So, like this 74 374, so this is a chip where we have got two controls. One is clock that is the standard clock and there is another pin which is known as output enable or \overline{OE} line.

(Refer Slide Time: 18:20)



So, if you look into this logic diagram, it shows that the \overline{OE} line. So, this is connected to this tri-state buffers. So, if this \overline{OE} line is 0, then this will become equal to 1, as a result all these tri-state buffers will be enabled and the value that is available at Q will be coming to a pin number 2 which is corresponding to the 1 Q line. Similarly, this Q output will be available at pin number 5 which corresponds to the 2Q line.

So, this way we can have these 8-bit register with 3-state outputs, so once apart from this logic high and logic low output so, it has got high impedance state which is neither logic 0 or logic 1 and it is useful while you are connecting number of such devices together, like what I mean is that suppose I have got two registers like this and these two registers outputs, they are fed to another register. So, this output is one 8-bit output. So, this output is also another 8-bit output and both of them are feeding to this register. So, this is R1 and R2 and R3.

Now, if I do not have this type of tri stating facility, then one way to do it is that I have got this R1 R2 and then, I put some multiplexer type of structure, the 8-bit multiplexer where this two comes and then, they finally go to R3. All these lines are 8-bit that is these multiplexer is consisting of 8, such 2 to 1 multiplexers, fine.

Now, I have to have a select line by which I have to select which multiplexer output I want to go to R3, but if I have this type of tri-state facility, then what I can do, I have this R1 and R2 and their outputs are tri-stated and then I can if I enable this R1 register, then this if I enable, if I give enable signal to this OE signal output enable signal to this R1 register and it do not give OE signal to R2 register, then R1's content will be coming here.

Similarly, if we do not give OE signal to this and give OE signal to R2 ok, then these value will coming through R3 like this. So, I do not need to put separate multiplexers if my registers are tri-state buffers or having tri state buffers in them. So, that is the outputs are tri-stated. So, that is why in computer design, so while we have got a number of registers and their outputs are made to be common, so we take help of this tri state registers and then, there is a controller that we have in the computer system. So, that will be controlling this output enable lines and it will dedicate, it will determine like which registers output will be going to which register.

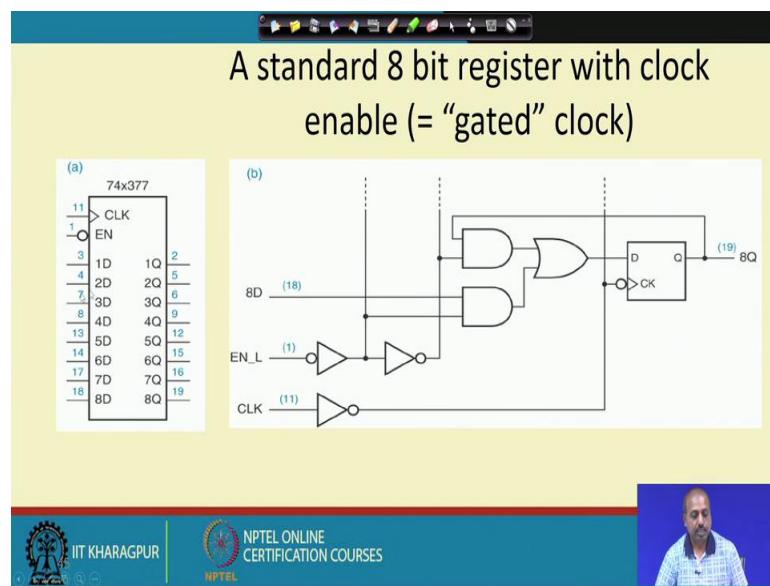
So, we do not need separate multiplexers for them that reduces the overhead. We can have registers with clock enable like so, we have got here this clock and clock enable is often called load or the gated clock. So, this load signal is given. So, here the clock signal is given, but until and unless this load signal is also given, the value that is coming on these D will not be stored into this register, ok. So, this design is shown here. So, you see if this clock enable is equal to 0, if the clock enable line is equal to 0, then whatever is output Out 0 so, that is circulated back into D flip flop.

So, for loading something on to this D flip flop, I must make this CE line equal to 1 or the load line equal to 1, then what will happen if this line is equal to 1 then, this In0 will be coming here. So, this In0 will be, input 0 will be stored into this D flip flop. Similarly, this In1 will be coming on to this line and this will be stored into this D flip flop. I am sorry there is a shift in this 0's and 1's. So, this 0 is with this input and this 1 is with this input. They are actually identifying which input will be selected if C is equal to 0. So, C equal to 0, Out 0 is selected if C equal to 1 In0 is selected.

Similarly, if C equal to 0, Out 1 is selected in this case and In1 is selected if C equal to 1. So, this way apart from this clock and everything we can have another control which is the load control. So, this load control will tell when the value should be loaded into the register when the next clock comes, like if the load is low even if the clock signal is there, the value is not loaded and there is a clear of course that if clear line is 0, then the register will be cleared so, that is there and in this particular case we do not have that tri-state buffering shown here. There can be \overline{OE} control also, output enable control also in the general case.

So, normally in a register we will have the clock input, the clear input and we will have this load input and \overline{OE} , the output enable line. So, all those lines can be there.

(Refer Slide Time: 23:04)

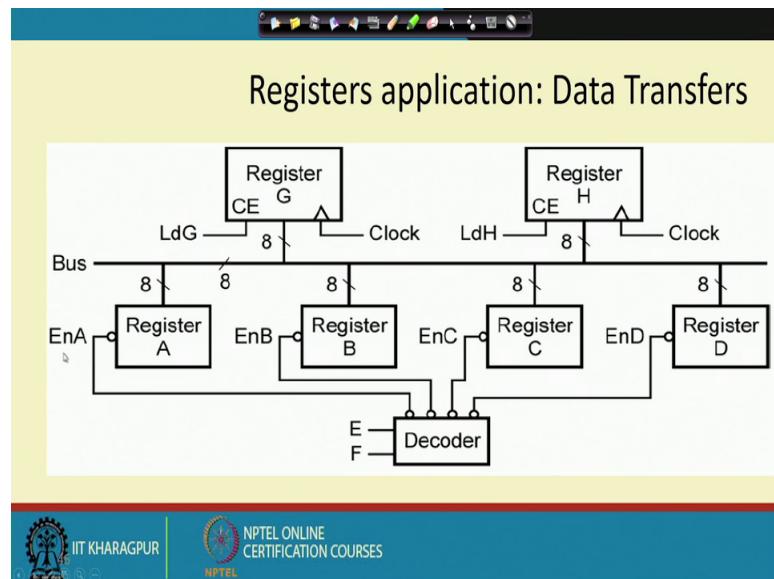


So, this is one chip 74377. So, that has got that has an 8-bit register with clock enable. So, clock enable it is some sort of you can say it is a gated clock type of structure as if this clock signal is given, but if this enable line is low, if the enable line is high in that case, so

this line, this point is 0. As a result this point is 1 and then this Q line will be circulated here and it will be put into this D flip flop into D input. Otherwise, if this enable line is low, so this is 1 so, this is 0 and as a result this output is 0, but this is, this was equal to 1. So, whatever be this D input will be coming through this AND gate and through this OR gate till finally reach this D input and that will be stored into this D flip flop.

So, this is, this shows the structure of these clock enable circuitry. So, you can have this enable and this clock. So, this extra part that we have put here that is like a multiplexer type of structure that is useful for selecting whether I am going to circulate the previous value because enable line was not active or I am loading a new value because the enable line is active.

(Refer Slide Time: 24:27)



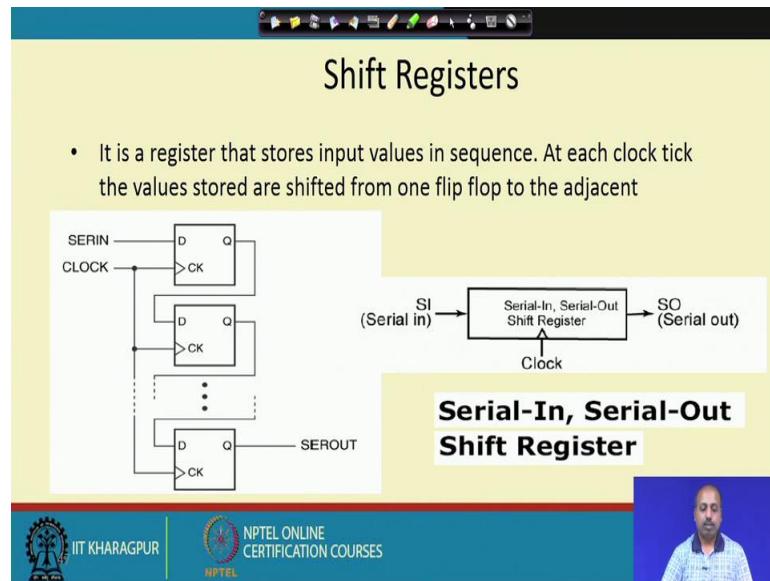
So, this shows a typical diagram like normally in application that include involve data transfer which is very common in case of computer systems or processors. So, we have got a bus and from this bus a number of registers are connected, ok, so this A B C D G and H. So, these are a few registers and each register, they are connected to the same bus, and what we want is that they should be able to communicate between themselves like one register content I should be able to transfer to some other register. Now, suppose I want to transfer the control of register A to register G, now how can I do this? So, for that purpose I have to enable these register output, and I have to enable the load signal for this G register.

So, this enable A and load G, so these two inputs are to be given and accordingly it will be doing the transfer from A to G.

Similarly, if I want to make a transfer from A to H, then this load enable A should be given and this load H signal should be given. So, in a very simplistic situation, it may be the case that I have got these four registers A B C D from where we want to make transfer to these registers G and H. So, I can have some set of decoder where there where this E and F, they come as control and they will say from which register the transfer will take place and only one of this is a decoder only one of this outputs will be made equal to 0. So, as a result only one of these registers enable line will be 1 enable line will be active and these 8-bit will be coming from that register. Other registers those out all the outputs are tri-stated. So, they will not affect the, they will not affect the bus.

Once the content is available in the bus, then it is assumed that there is some other controller it will provide this either this load G signal or load H signal to load the value on to the proper register. So, this way we can have these registers implemented register data transfer implemented by means of individual registers that has got enable and load lines in it.

(Refer Slide Time: 26:50)



Another class of registers, so they are known as shift registers. So, in shift register, it is a register that stores input values in sequence. So at each clock tick, the value stored at shifted from one flip flop to the adjacent flip flop. So, there is a serial input, so problem

with this previous structure is that if you are trying to store some value on to these registers, so I have to give 8-bit parallel lines, but often getting so many parallel lines from somewhere is difficult. So, you want to transfer them, you want to transfer the data serially and that way we have got only.

So, these may be some higher number of, higher size register may be 8-bit register or 16-bit register or whatever, but I am not feeding all those register simultaneously with external data inputs, they are coming through this serial input line and this the serial input line so, it effects only the first D flip flop and this first D flip flop Q output is connected to D input of the next flip flop and it continues like that. This Q output is connected to the next flip flop's D input. So, it continues like this. So, this way with the when the clock pulse comes, then this serial input data gets loaded into D flip flop and the previous value of this D flip flop are transferred to this next flip flop.

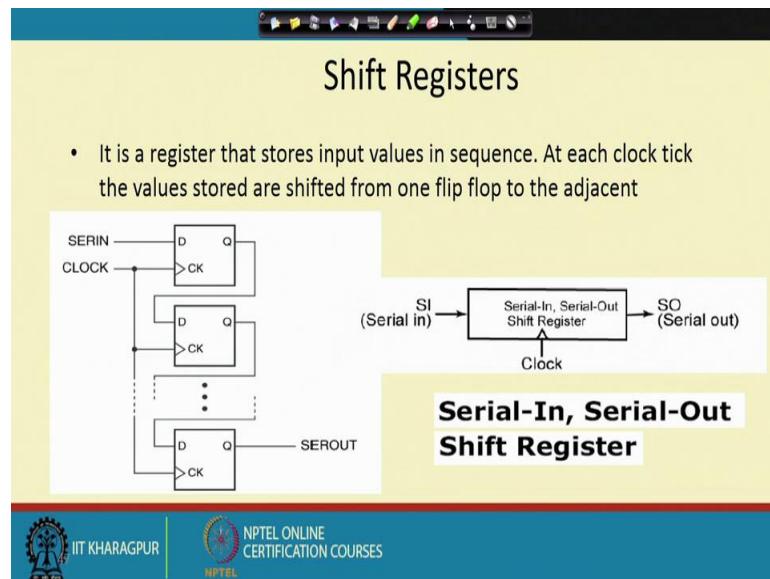
So, this way this acts as a serial change. So, this type of registers, they are known as serial input, serial output register, shift register or SISO register serial input, serial output register and with the individual clock pulse, one data 1-bit data is shifted from this SI input into the first register and 1-bit of data from last register is shifted out. So, this is the serial input serial output type of shift register structure.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 32
Sequential Circuits
(Contd.)

So shift registers: so these are special type of registers, where instead of loading the value parallel into the register we will load it in a serial fashion.

(Refer Slide Time: 00:26)



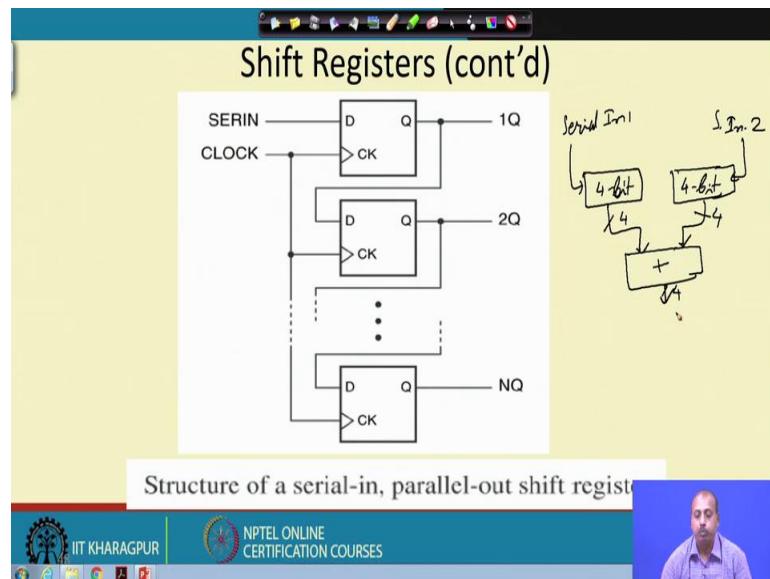
So, here a number of flip flops have been connected in series as you can see here. So, these Q output of this first flip flop is fed to the D input of the next flip flop, similarly Q output of the next the second flip flop goes to the Q input of the third flip flop. And that way we can connect n such flip flops.

So, in a parallel load register this n values the values into these n register n flip flops will be loaded parallelly, so you need n parallel pins to be there in the chip by which you can load feed that n bit pattern. But in many cases what happens is having so many number of pins is difficult in a chip. So, in those situations this serial register they come into use like we have got a single input pin which is serial input. So, here whatever bit we give in the first clock pulse, so that bit gets latched on to the first flip flop and whatever was there in the first flip flop gets transferred to the second flip flop in the chain and so on.

So, as a result if there are n such flip flops connected in a chain like this, then to load the a particular bit pattern, so we have to shift the patterns n times. The bits are to be fed serially n times an every clock 1 bit will be put into the register. So, the bit for the last flip flop should be loaded first, so that in a in the total shifting process that bit finally reaches the final flip flop.

So, this way we have got serial IN serial OUT type of shift register as it is shown here, so we have got a serial OUT pin. So, output of the last flip flop is coming out as the serial output from the register.

(Refer Slide Time: 02:06)

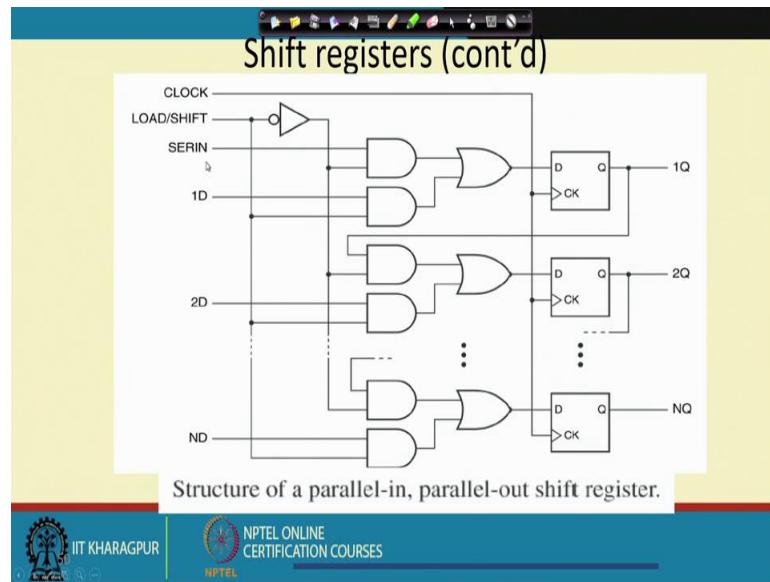


So, we also can have serial input parallel output type of shift register, where the input is coming serially but output is available in a parallel fashion. So, this may be useful in some cases where we need the parallel output; for example, if we have loaded 2 bit patterns into 2 shift registers and just want to sum them up.

So, we have got 2 shift registers like this, so this is 1 shift register this is another shift register and then the individually there may be 4 bit, 4 bit. So, they are shift registers so they have got one serial input line. So, this is serial input 1 and this is serial input 2; serial input 2. So, they are coming so after the 4 bits have been loaded, so they may be given to an adder 4 bit adder and that at that time this 4 bits are feed parallelly.

So, these are 4 bit lines coming from the 2 flip flop 2 registers and the result is also 4 bit. So, this type of cases so we need serial input parallel output type of configuration so that is also used in some situation.

(Refer Slide Time: 03:21)



Then we have got a parallel input serial output type of configuration. So, here the values are loaded parallelly, so these load control when this load is equal to 1 then this load signal comes here as a result this AND gate, at the output here also you will get the value whatever value is given at pin 1D so that comes here. And since this is coming through this load shift and inverted, so load shift line equal to 1, so this will get a 0 here. As a result this OR gate outputs finally this 1 D value and that gets loaded into the D flip flop.

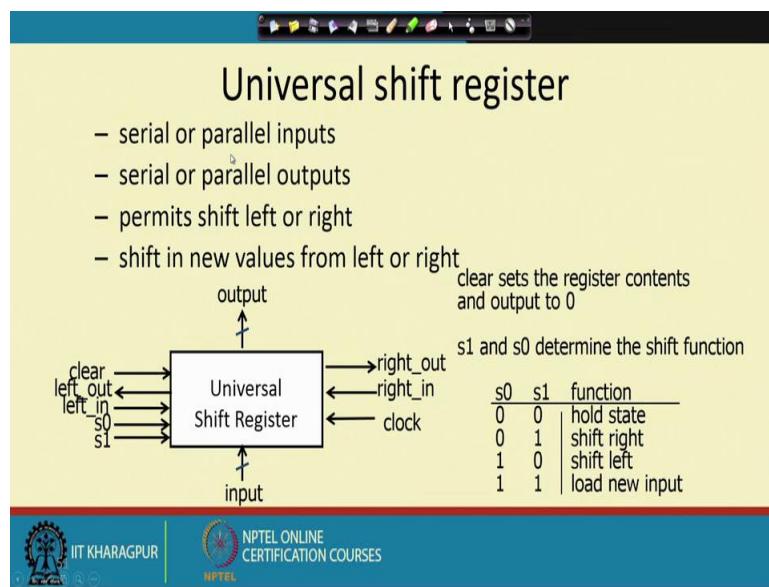
On the other hand the second bit pattern that you have loaded parallel given it parallelly that also gets loaded into the second flip flop, so that way we have got the parallel in type of configuration. Now it is a serial OUT like, if you want to shift out the value, so you have to give the shift you have to make the load shift line equal to 0, as a result this output of this inverter will be equal to 1. So, whatever is coming as serial input so that will be coming to the first D flip flop, so this serial input bit comes to the first D flip flop, output of the first D flip flop is coming as the input of the second D flip flop. So, now it behaves as a serial shift register serial input serial output shift register.

So, we have got this parallel input serial output type of shift register configuration. So, in this you can have different types of shift register configuration, we can also have parallel

in parallel out type of shift register. So, here you can have this parallel input like 1D 2D up to ND. So, n possible data bits that you want to store into the register and then we have got this load shift lines. So, if load shift line is equal to 1 then the value from this 1D 2D to ND, so that will get loaded into this flip flop.

On the other hand if this load shift line is equal to 0, then whatever is coming in the serial input so they will be loaded into the D flip flop. Also I have got this parallel output available so this for getting the parallel output I do not need to do anything because, I just have to have this extra n number of pins in the chip where this Q outputs are available on those pins, so you have got this those n bits. So, this is basically a parallel input it also has got a serial input and then it is a parallel output shift register. So, it is a parallel input stroke serial input and parallel output shift register. So, this way we can think about different types of shift register configurations and they are used in different applications.

(Refer Slide Time: 06:08)



So, the most generic concern structure of this shift register we can think about is the universal shift register. In universal shift register so it should be able to serial or I should be able to choose between serial or parallel mode of inputs serial or parallel mode of outputs, I should permit shift left or shift right. So, in this diagram so if I say that this flip flops are drawn in a horizontal fashion, so this is the left most flip flop this is the right most flip flop.

So, the shifting that I am doing here is from left to right, but it may be the other way also we can connect these Q output to the previous D flip flops input, similarly finally so this Q output to the to this D input and this serial input line can be connected to the last D flip flop.

So, as a result I can get a left shift type of configuration, so that actually gives us the nothing like we can permit shift left or right and we can also shift in new values from left or right, though that is you can you should be able to put a new value into the register either from the left side or from the right side. So, an universal shift register should have so many controls.

So, that's why parallel input output facility, so the this these are these are the parallel lines parallel input and this is the parallel output. I should be, I should be able to have this left serial IN and right serial IN then this the left serial OUT here left out and right serial OUT right out and the clock signal is definitely needed. There is a clear signal, so, that if it is given so it will clear the, all the flip flops, so it will be register content will become 0 and I need to select between these 4 modes ok. So, there here you see that the register can operate in any of these 4 modes. So, I need to select between those 4 modes and for that in need 2 mode bits $S_0 S_1$.

So, it may be like this that if $S_0 S_1$ both are 0 0 nothing happens the flip flop the flip flop contents remain unchanged, so this is the whole state for the register. If it is 0 1 then we are doing a shift right. If it is 1 0 then that control may translate into the shift light shift left command for the shift register and 1 1 it may be for loading new input. So, this way we can think about different types of control different modes of operation for this shift register and we have to set this $S_0 S_1$ pins accordingly to get the job done.

(Refer Slide Time: 08:49)



So, you can extend the designs that you were doing here ok. So, that it can take care of this is $S_0 S_1$ controls you can introduce into this design and get the whole thing done. So, I am not showing that design explicitly because the design will become a bit complex, but this is nothing very difficult, so you just have to draw the corresponding combinational logic that should feed the individual flip flops and that will be giving us the full functionality. Rather we will be looking to the applications of this shift register like in which situation we can use the shift registers.

(Refer Slide Time: 09:28)

A screenshot of a presentation slide titled "Applications" of shift registers. It lists applications like Ring counters, Johnson counters, Pseudo-random binary sequences and encryption, and ready-made shift registers. It also discusses conversion between serial and parallel data. The slide includes a diagram of a shift register and a video player at the bottom.

So, there are several uses some of the sometime we can use it for counter design, so this ring counters and Johnson counters so will see them later. There are some application in pseudo random binary sequences and encryption. So, pseudo random binary sequence basically generate some random number in many applications so we need random number generator.

So, when you are writing a program you in software, so you know that there are normally the operating system provides you with the routines like random number generator, but how those random number generators are working. So, it may be that you have got some random number generation algorithm that runs, but while you are doing in digital circuits how can you have a random sequence?

So, this shift registers they can be used for generating some random sequence and then once you have got this random sequence generators, so they can be used for data encryption type of application. So, there are also readymade shift register that are available in integrated circuit such as 74165. So, they are shift registers are available as chips so you can use it directly.

Another application is conversion of data from serial to parallel and vice versa. So, large scale devices such as universal asynchronous receiver transmitter UART, so they are based on shift register. So, this universal asynchronous receiver transmitter or UART, so they come as a separate chip or they come as part of some processors also; like what happens is that if you want to transmit some data over from one system to another system. So, if you are using a parallel type of transfer then there are large number of lines which has to run parallelly over may be a long distance. But instead of that if we have got less space and we can use a single line because serial if I transmit the data serially then the only one line is needed for doing that data bit transfer.

So, that many times we need this serial transmission and that is asynchronous in nature because, the 2 system at the 2 ends that we have of that transmitting line so they are operating at their own clocks, so naturally that is an asynchronous system. So, this type of model there are this type of systems we need this module which is known as UART ok, so that essentially use as a this shift register.

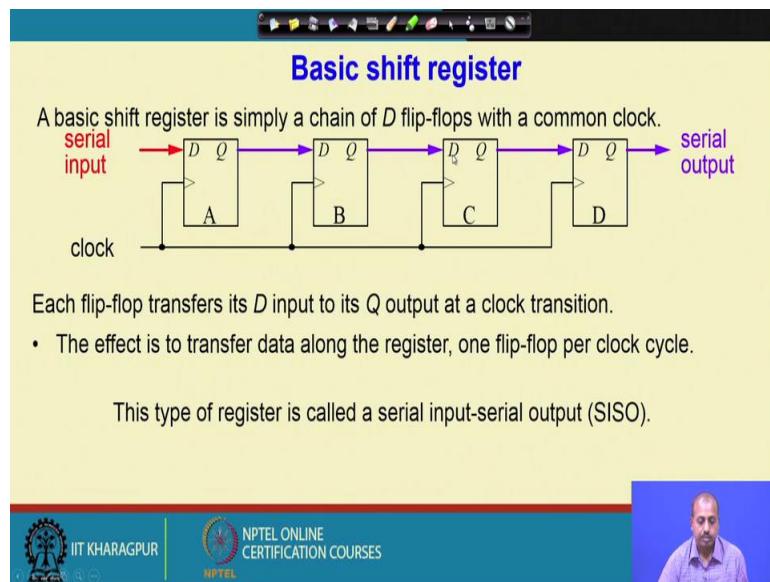
So, from the source if this is the processor from where I am this is the source processor from where I am sending it, so there is a special register here ok. So, the data the processor

loads this register parallelly the value is loaded here, but from here the data is transmitted serially.

So, you need a parallel input serial output type of register, similarly at the receiving end so you have got a similar type of register. So, where this serial line is connected and it outputs the parallel line. So, this is a parallel input serial output type of combination that we need here and at the receiving end we need a serial input parallel output type of combination, so both the cases so you need some shift register for doing the thing. Some functions that available in microcontrollers or microprocessors where the shift on rotate type of instructions.

So, there also you need to shift the bits of a register by number of places and there also we will see that we will have to use this type of shift registers. So, this will be more clear when you go to our discussion on microprocessors particularly 8085 microprocessor. So, then it will be clear further.

(Refer Slide Time: 13:18)

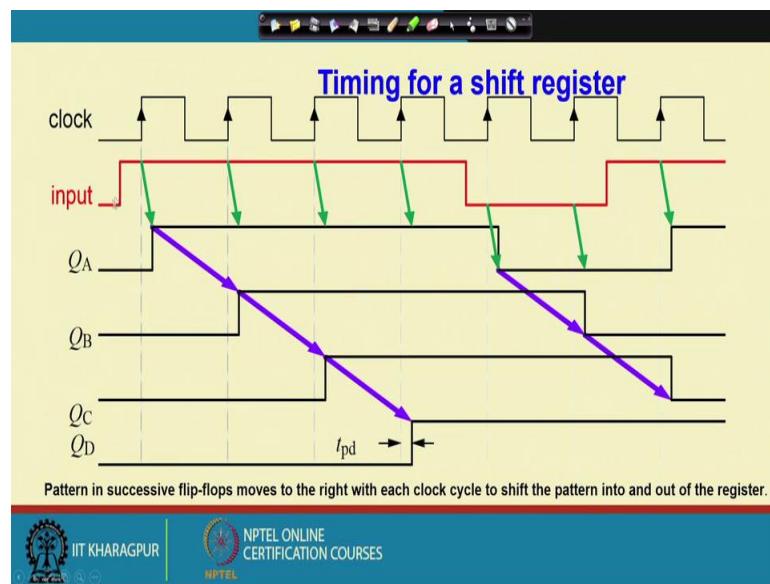


However, so basic shift register from our understanding you see that it is nothing but a chain of D flip flop with a common clock. So, clock is going everywhere and this serial input line is connected to the D, Q output of the first flip flop is connected to the D input of the next flip flop and it goes like this. So, each flip flop it transfers it's D input to it's Q output and at a clock transition; when clock there is a when the when the clock signal

makes a transition. So, this from serial input it is transferred here similarly from this Q it transferred to the next D, so that way it goes.

The effect is to transfer data along the register 1 flip flop per clock cycle. So, it is going 1 flip flop per clock and this type of register is known as serial input serial output register or SISO this is the one terminology which is quite often used, which is SISO register serial input serial output register.

(Refer Slide Time: 14:13)



So, timing behavior is like this so if this is the clock signal that we have got and suppose we have got an input pattern like this ok. So, it was initially low then it is high for some time then it is again becomes low for some time then again high for some time.

Then this Q_A that is the first flip flop, so it is fed directly from the serial input, so at the clock transition so it looks into the value like here there is an edge, so this value is sampled and the Q_A makes the transition accordingly. So, there is a delay associated with the flip flop so this is the delay that we have. So, after that Q_A becomes high then at the further at further clock edge further rising clock edge Q this input is still 1. So, it remains high and it goes on like that so it goes on like that.

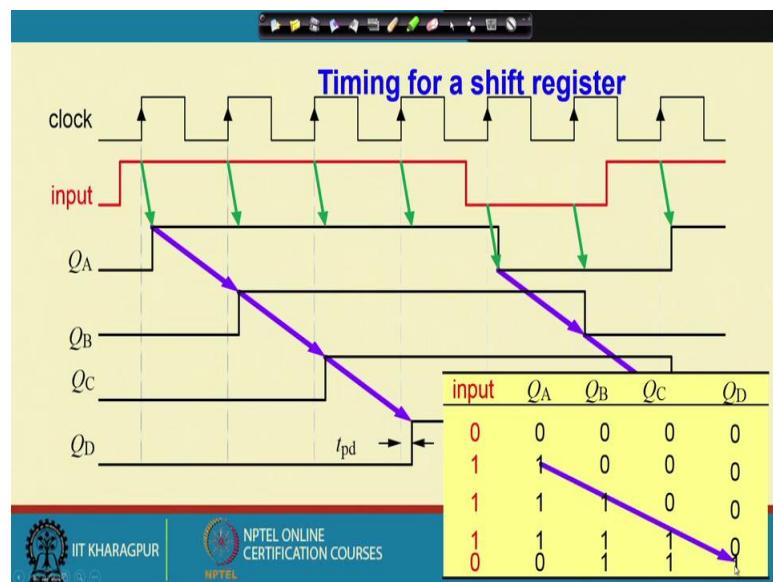
So, similarly at the third clock so it goes high and it when the third clock step it goes high. So, it is also having this input as high, so this value is sampled here so this way it goes on and finally, when it at this particular clock edge it finds that the input is low. So, the Q_A

output becomes low and it now becomes low and continues with the low value till the next clock edge where it is again low, so this is sampled at that time.

If you look into the second flip flop the B flip flop then at the first clock edge it finds that Q_A is 0. So, it remains 0 at the second clock edge it finds that Q_A is 1, so it becomes high and it goes on like this. Similarly Q_C it at the previous clock edge is 0 so it finds at Q_B is low. So, till this clock that is no change but at this clock find the Q_B is high. So, it will be making a transition like this and it will continue like that.

Now, this way so we have got this propagation delay TPD, so this is from the clock edge after how much time the transition will occur. So, that is the TPD or the transition time ok. So, pattern in successive flip flop moves to the right with each clock cycle to shift the pattern in to and out of the register, so that happens in this shift register.

(Refer Slide Time: 16:37)



So, you see how this 1 is getting transmitted so it is shown here. So, this 1 is actually transmitted to this Q_D after 4 clock cycles and that has to be taken into consideration that after 4 clock cycle the value will come there.

(Refer Slide Time: 16:56)

Applications of a basic shift register

1. **Delay line** — N stages delay the signal by N clock cycles
2. **Multiplication and division by powers of 2**, because this just requires a shift of the binary number (like multiplication or division by 10 in decimal)
Example: decimal $3 \times 4 = 12$ becomes $11 \times 100 = 1100$ in binary. The arithmetic logic unit (ALU) of a computer processor uses a shift register for this purpose.

Warning: the 'sense' of a shift — left or right — is usually based on its effect on binary numbers written in the usual way. For example,
 $11 \rightarrow 1100$ is called a **left shift**. This is clearer if both numbers are written with 8-bits as
 $00000011 \rightarrow 00001100$. Similarly, dividing by 2 such as $00010110 \rightarrow 00001011$ is a **right shift**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, another some applications of this basic shift register are like this delay line, like if you want to put an N stage delay that is when a particular clock edge comes, so, you want that the circuit or the actual circuit should get that signal after some time after some N clock pulse this ok. So, you put an N stage delay and then the signal will be coming here. For example what I mean is suppose this is my actual system and the signal comes from some other systems as the signal comes here, but what you need is that when this signal is generated after some delay the signal should be available here.

So, what you do in between you put some shift register of sufficient number of stages ok. So, you put a shift register of sufficient number of stages, so that this signal gets transmitted through this shift register, and then finally, it comes to this register. So, we can have this in this way this shift register behaves as a delay line in our in this particular case.

Now, another application is multiplication and division by powers of 2. So, you see suppose we are doing a multiplication by 3×4 that is 12. So, it becomes 1 3 is 1 1 and 4 is 1 0 0 so it is 1 1 0 0 in the binary arithmetic. Now you see this when we are multiplying 3×4 so it is nothing but we are doing a shifting of this 1 1 towards the left by 2 bits, so this is the left shift operation so what I mean is if I take if 4 bit number and 4 bit number representation then this 3 is represented as 0 0 1 1.

(Refer Slide Time: 18:40)

Applications of a basic shift register

1. **Delay line** — N stages delay the signal by N clock cycles
2. **Multiplication and division by powers of 2**, because this just requires a shift of the binary number (like multiplication or division by 10 in decimal)
Example: decimal $3 \times 4 = 12$ becomes $11 \times 100 = 1100$ in binary. The arithmetic logic unit (ALU) of a computer processor uses a shift register for this purpose.
Warning: the 'sense' of a shift — left or right — is usually based on its effect on binary numbers written in the usual way. For example,
 $11 \rightarrow 1100$ is called a **left shift**. This is clearer if both numbers are written with 8-bits as
 $00000011 \rightarrow 00001100$. Similarly, dividing by 2 such as $00010110 \rightarrow 00001011$ is a **right shift**.

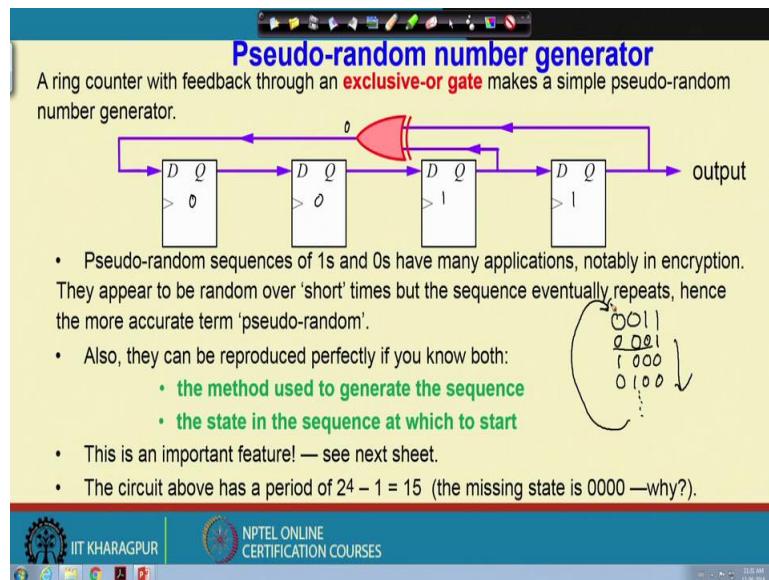
0011 $\xrightarrow{\text{left shift}}$ 1100
by 2 bits

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, when I am multiplying by 4 what I am doing is that I am shifting left by 2 bit position. So, left shift left shift by 2 bits. So, if we do that so it will become 1 1 0 0 so that is essentially multiplication by 2. So, this multiplication by 2 for powers of 2 is a very common operation in computer applications. So, there we can do that by shifting left by left by N bits on the other hand if you are doing a division. So, division by powers of 2 then it is nothing but right shifting the content by appropriate number of bits.

So, this sense of shift left or right so it is based on that it's effect on binary numbers written in the use value; for example, this 1 1 to 1 1 0 0 so this is called a left shift and so this is numbers written in 8 bit format is like this and then you transmit to a shifting towards left. So, this so 2 ones get left shifted by 2 position so it becomes 0 0 0 0 1 1 0 0 and similarly when you are dividing by 2, so this is this factor becomes like this so this is a right shift operation. So, this is a left shift and right shift operation that we have.

(Refer Slide Time: 20:07)



Now another application that we have of this shift register is known as pseudo random number generator. So, this is the shift register but only thing is that so these Q output of the last flip flop, so it is connected back to the first flip flop via some XOR gate and this XOR gate it has got some of the bits tapped from this shift register stages and connected to it. For example, in this case the last 2 flip flop output, so they have been XORed and then fed to the first flip flop.

So, this if you now if you start this D flip flops, if you initialize this D flip flops with any value other than all 0, then what will happen so it will be it will be based on the these values in this these 2 flip flops, these shift value will be determined accordingly it will it will generate a sequence. For example, we can if we say that we started with the pattern say 0 0 1 1 that is these 2 flip flops are 0 and these 2 flip flops are 1, then what will happen so these 2 ones, so they will be XOR here so you will get a 0 at this point. So, at the next clock so this is a shift register otherwise. So, by the shift register operation, so these 3 bits will be 0 0 1 and this bit will become 0.

Similarly, in the next clock so this one will be going out. So, you will be getting 0 0 0 so this 3 zeros will be right shifted by 1 position and this 0 and 1 they will be XORed and coming as input to the first flip flop, so it will become 1 0 0 0. So, at the next clock so you will get this 1 0 0 0 here and then this 0 0 will be XORed and you will get a 0 at the first

flip flop. So, this way it will go on generating a random sequence first it is 3 then 1 then 8 then 4 after generating a random sequence.

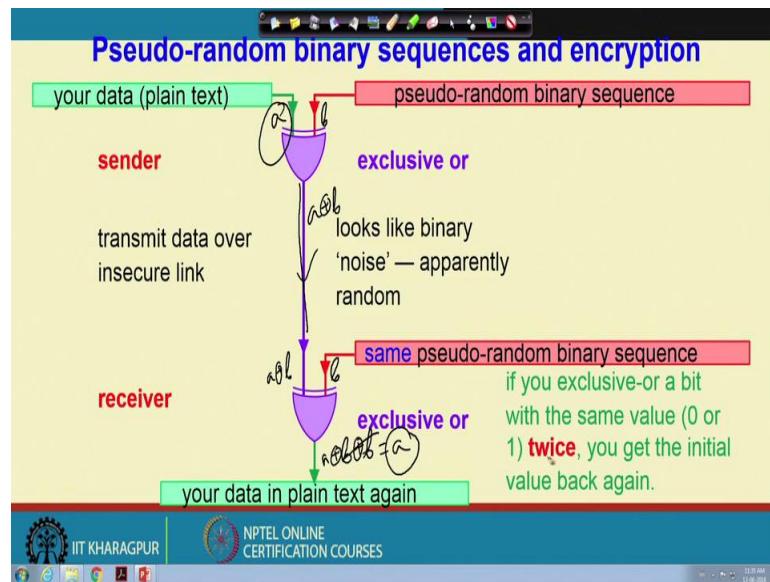
So, this pseudo random sequences of ones and zeros have many applications particularly in the field of encryption we will see, they appear to be random over short time period and because the sequence will eventually repeat. What will happen is that in this way if you trace out this sequence then after generating all the 15 nonzero pattern, so it will again come back to this point. So, it will start generating from the beginning.

So, that pseudo random that is why it is called a pseudo random sequence not a truly random sequence because after sometime the sequence repeats itself, so it is a pseudo random sequence and they are they can be reproduced perfectly if you know the following thing the method used for generating the sequence, that is what is the structure of this what is the structure of this shift register. That is, but in particular what is the where from which points from which flip flops you have taken inputs to be XORed to be fed back to the first flip flop

So, if you know that information and you know what is the initial pattern that you have fed; so given these 2 information you can reproduce this sequence at any point of time ok. So, that is why we must the method used to generate the sequence which is basically this XOR tapping points and the state in the sequence at which it starts. So, that is the first value of the of the shift register and this in this particular case after fifteen cycles it will be returned it will be repeating and the as I said that the state 0 will not come because, if you load 0 then it will always remain in the 0 value.

So, it will not be having any other pattern here, so it will always be at 0. So, it cannot generate the sequence 0 apart from that it will generate all other patterns.

(Refer Slide Time: 24:08)



So, what do you do so this pseudo random sequence generate they have got application in encryption. So, in encryption what happens is that you have got some plain text which is combined with some key value and accordingly it will generate some encrypted result ok. Like in this particular case suppose this is your plain text is coming, I am considering the plain text also as a bit sequence although it is coming here and you have got pseudo random binary sequence, so that is generating the pseudo random pattern.

So, the successive bits that are generating so you are you do XORing of them and then you see; what is the output. Since it is XORed. So, naturally you the nature of this plain text will be lost and you will see something like binary noise ok, so which which will appear to be apparently normal. So, at the sender end, we do it like this we XOR this plain text with this pseudo random binary sequence generated by the shift register and then we just after doing the XOR we start transmitting over this channel.

So, at the receiving end what you do you use the same pseudo random binary sequence. So, you are using shift register where this feedback points are same to the first XOR gate. So this, the same structure is used here and also you start at the same initial pattern. So, once you start at the same initial pattern then the same sequence of random numbers will be generated here and that you can XOR with your plain text. So, there that so since this if it is your plain text is say A and this if your plain text is A and this is B. So, at this point I am getting A XOR B.

Now, what is happening is that here I am using the same pseudo random sequence generator starting with the same initial pattern. So, this is basically B so at this point so this is A XOR B and at this point I get A XOR B XOR B. So, these B XOR B is equal to 0 so I am getting back A.

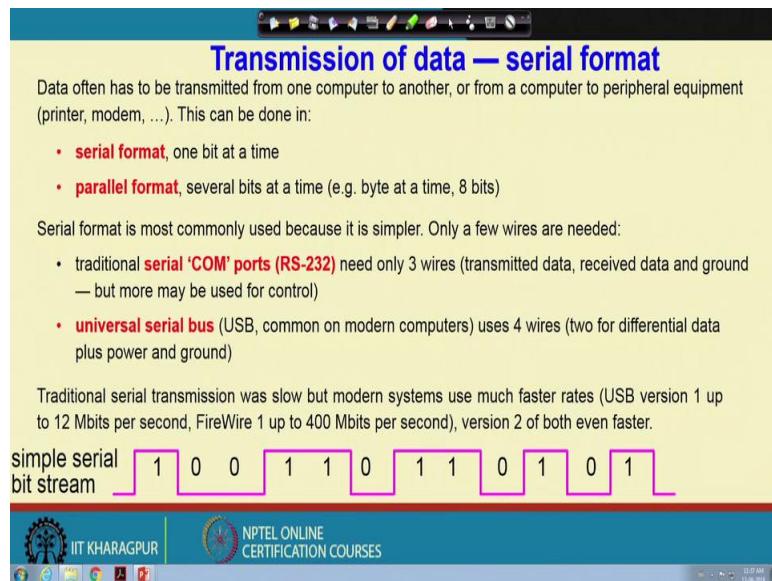
So, whatever A you are transmitting you are getting here, but in between in this while it is getting transmitted through this channel. So, this is encrypted as A XOR B so this is the property that if you exclusive or bit with the same value 0 or 1 twice you will get the initial value back, so that is the property of XOR. So, that is used for this encryption algorithm. So, this pseudo random sequence generator it has got application in this encryption algorithms.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 33
Sequential Circuits (Contd.)

So, next application that we look into is the transmission of data in serial format.

(Refer Slide Time: 00:18)



Now, as we know that the data it has to be transmitted from one computer to another computer or one system to another system and normally what you have may, may be one computer to your printer or modem so like that.

Now while we are doing this transmission, so, it can be done in two formats either in serial format or in parallel format. As I already said that the advantage of serial transmission is that, the number of lines that you need to draw, so that is less ok. So, you have to take only if you have single line for carrying 1 bit value 1 bit at a time is transmitted, but when you are doing in parallel format, then several bits will be transmitted parallelly.

So, if you are transmitting say 8 bits parallelly, then you have to have 8 parallel lines running between the source and destination. So, this is that is why the serial transmission is quite popular in many applications. So, this is simpler and only few wires are needed. So, traditionally if you look into the computer these the serial com ports RS 232 ports. So,

they need only 3 wires transmitted data, received data and ground; but there, there some more controls are needed which is not stated here for doing the set ups and all, so that is there.

Another serial communication is universal serial bus or USB standard that uses 4 wires two for differential data, plus power and ground. So, that way this both of them are serial. So, in case of COM port we need one transmit data, one receive data and ground and in case of USB we need only 4 wires of course, for serial for this COM port RS 232 communication.

So, more number of a lines are needed, but for USB it is 4 only. So, that way USB has become a better standard, but anyway we are looking into a serial transmission. So, traditional serial transmission was slow, but modern system are faster like USB version 1 is up to 12 megabits per second, FireWire up to 1 up to FireWire 1 is up to 400 megabits per second version 2 is even faster. So, we now we have got even USB version 3.0 which is still faster.

So, that way, but essentially what is happening is that for serial transmission. So, we transmit in transmit them as a bit pattern. So, this so 1 is transmitted as logic high, 0 is transmitted as logic low so, that way this bit pattern is transmitted serially.

(Refer Slide Time: 02:54)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'Parallel data' is centered in blue text. The main content area contains the following text:
Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- inside the processor itself, e.g. our microcontroller handles bytes
- inside a computer system on the **bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE)— but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a man speaking.

Whereas when higher speed data transfer is required, we need to transfer several bits in parallel and usually a small number of bytes each of say 8 bits that may be moved at a time, more complicated connections are needed more wires and so, some applications they will need it one difficulty with the serial transmission is that since you are transmitting serially. So, time needed for the total transmission is higher compared to a parallel transmission.

So, it is the parallel transmission takes place inside the processor itself for example, microcontrollers, microprocessors they can handle in terms of bytes. Inside a computer system on the bus for example, you are connecting this PCI you have you are having PCI bus that connects the processor to other disk drives and all or SCSI, IDE disk drives. And they are all having this parallel communication, but because they are, they were initially parallel, because of this thing that this transmission will be faster, but now the serial communication is also being used.

Now, interfacing that you have, so from the processor it comes out as a parallel data, but the transmission is serial. So, we need to have a parallel to serial converter. So, this interfaces are changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at high speed of modern systems. So, you are you are expecting that an 8 bit data, but that 8 bit data if it is coming through 8 parallel lines, then it is there is a possibility that this 8 parallel bits will not arrive at the same point of time. So, there may be a synchronization problem between this 8 bits.

So, it is better that we go for serial communication where this transmission will be done serially. And also there are other effects like cross talk between the parallel lines which effects the logic value that you are sending over the bits.

(Refer Slide Time: 04:55)

The screenshot shows a presentation slide with a blue header bar containing icons. The main title is "Parallel data". Below the title, there is a paragraph of text and a bulleted list. At the bottom of the slide, there is a footer bar with logos for IIT Kharagpur and NPTEL.

Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- **inside the processor itself**, e.g. our microcontroller handles bytes
- inside a computer system on the **bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE)—but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.

How do you interface a serial device to a computer?

How do we interface an external device that transmits serially with the bus of a computer that transfers one byte (8 bits) at a time?

- **Use a shift register.**

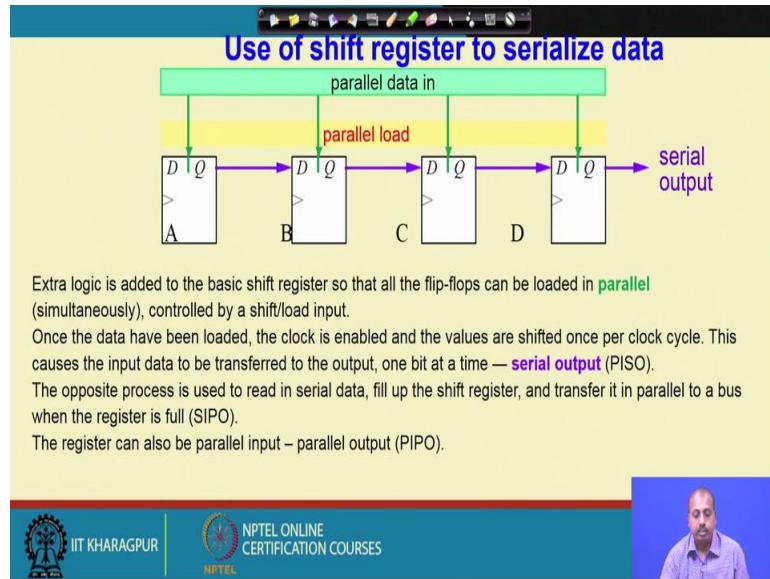
In practice this would almost certainly be buried inside a larger circuit called a **UART** (universal asynchronous receiver transmitter) or something similar.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Where higher speed is necessary, so we have several bits are transmitted that we moved at once, and more complicated connections are needed more wires like inside the processor itself or inside the computer. So, how do you change this serial device to a computer? So, how do we interface an external device that transmits serially with the bus of a computer that transfer only 1 bit.

So, this is this transmission has to be via some shift register, because then shift register only it you can do a parallel to serial conversion and we have got, we have already seen that they can use some sort of UART for doing this communication.

(Refer Slide Time: 05:33)



So, this diagram actually explains how we can have this parallel to serial conversion. So, parallel data is loaded parallelly on to these D flip flops and then this D flip flops are otherwise connected in a series fashion. So, this logic we have seen previously parallel in serial out type of register configuration or this is known as PISO structure, parallel input serial output type of structure.

So, you can also at the receiving end, so we can have this SIPO structure serial input parallel output type of structure. Where the serially the bits will be coming and then they will be converted into parallel data and given to the processor or we can also have parallel input parallel output type of connection.

Next will be looking into another very important digital circuit component which is known as counter so, counters are normally used to count some event ok. So, suppose we are counting like how much time, time is lapsed from the beginning of certain event or when the items are moving on a conveyor belts, so, you want to keep a count on how many items have passed through the conveyor belt. So, like that we can have different application where we need to really count the number of occurrences of the events.

So, they are so how do you do it like occurrences of an event may be coming as a trigger to your circuit, but the circuit has to memorize like how many such things it has seen ok.

(Refer Slide Time: 07:03)

Counters

- * Counters are important digital electronic circuits.
- * They are Sequential logic circuits because timing is obviously important and they need a memory characteristic.
- * Digital counters have the following important characteristics,

1. Maximum number of count
2. Up-Down Count
3. Asynchronous or Synchronous Operation
4. Free-Running or Self-Stopping

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, this will be done by means of counters, so counters are important digital electronic circuits they are sequential logic circuits because timing is obviously, important. And they need a memory to need a memory characteristic because after every, it has to after every event it has to remember, how many events it has seen? So, that is why it is sequential one in nature and digital counters have the following characteristics like there, there will be a maximum number of count because ultimately the values are stored in some flip flops. So, depending upon the number of flip flops that we employ to hold the count value there will be a limit.

Like for example, if I keep say 4 flip flops to keep the count value, then after the values becomes all one then at the next at the next time instant when I try to increment this 1 1 1 1 by 1, so it becomes all 0. So, there is a maximum count value which is 15, in this case the counter may be an up counter or a down counter. So, we can start from some initial value and the counter may be instructed to increase its value by one, after each event occurrence or it may be that I start with some initial value and then with the occurrence of an event we start down counting.

So, this up down counting may be there, the counters may be designed in both asynchronous and synchronous fashion. So, will see that and we may have some free running counter or self-stopping counter. So, may be as soon as we power on the system the counter. So, it may be it may start running continually without any, any intervention

from outside or may be self-stopping after the counter has reached a particular final value it stops at that point. So, depending upon the type of application that we have we have to choose a particular type of counter, so in our course. So, will be looking into how to design this different types of counters.

(Refer Slide Time: 09:02)

The screenshot shows a presentation slide with the following content:

Asynchronous/Ripple Counter

- Asynchronous counters are commonly referred to as ripple counter because the effect of the input clock pulse is first “felt” by first flip-flop (FF0).
- Cannot get to the second flip-flop (FF1) immediately because of the propagation delay through FF0.
- So the effect of an input clock pulse “ripples” through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

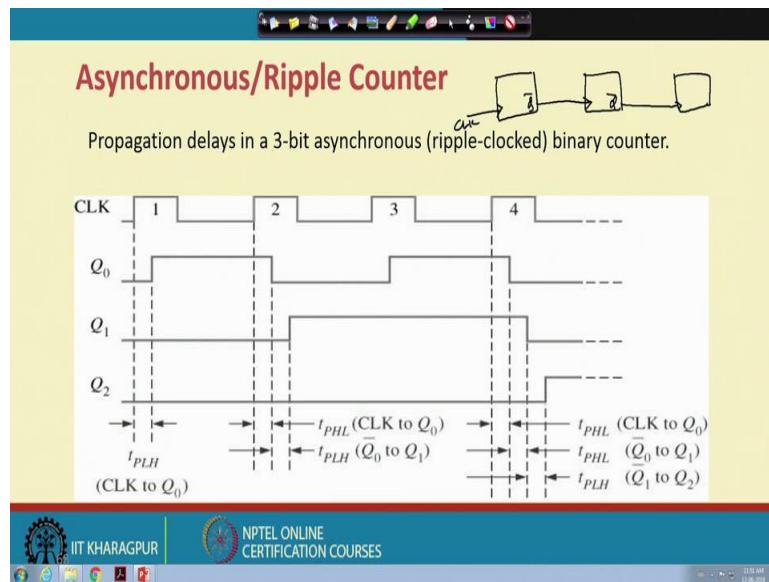
Only the first FF receive clock pulse from the source (clock generator), others FFs receive clock pulse from either Q or Q' of prior FF

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window in the bottom right corner shows a man speaking.

The first counter that will look into is known as asynchronous ripple counter. So, asynchronous counters are commonly referred to as ripple counter, because the effect of the input clock pulse is first felt by the first flip flop. And then it cannot go get to the second flip flop immediately because of the propagation delay of the first flip flop.

(Refer Slide Time: 09:24)



So, it is like this so if I have got say if I takes a 2 flip flops. So, this is 1 flip flop and this is another flip flop then. So, if so this, the clock signal that is coming to the first flip flop then. So, the value when this when this clock signal comes when the first clock suppose I have initially made both of them equal to 0. When both of them are equal to 0 now the first when the clock pulse comes, so this flip flop makes a transition to one, but as an input this flip flop sees still the output of the previous flip flop ok. At that way so it will not see this one immediately when the clock pulse will be arriving.

So, it will still see it as 0 and this flip flop has got a propagation delay that is why the second flip flop will not see the, it will not get the it will not get the one value directly there, so it will not there. So, as a result the effect of the input clock pulse ripples through the counter and so will see how does it operate only the first flip flop.

So, in the structure that we have follow only the first flip flop receive clock pulse from the source other flip flops receive clock pulses from either Q or Q̄ of the previous flip flop. So, what do we mean is that, if I have got say three such flip flops forming the counter then one possible structure may be like this.

(Refer Slide Time: 10:53)

Asynchronous/Ripple Counter

- Asynchronous counters are commonly referred to as ripple counter because the effect of the input clock pulse is first “felt” by first flip-flop (FF0).
- Cannot get to the second flip-flop (FF1) immediately because of the propagation delay through FF0.
- So the effect of an input clock pulse “ripples” through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

Only the first FF receive clock pulse from the source (clock generator), others FFs receive clock pulse from either Q or \bar{Q} of prior FF

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The clock pulse is connected to the first flip flop, and the second flip flop do the clock pulse is connected from the Q output of the second flip flop. Similarly, for the third flip flop the clock pulse may be connected from the Q output of the second flip flop. As a result the clock pulse that, so this clock is coming from the outside world. So, when the clock pulse comes it only the first flip flop sees the clock rest of the flip flops will not see. When this Q makes it transition from say 0 to 1 or 1 to 0 then only this is being felt by the second flip flop.

Similarly, when this flip flop makes a transition from 1 to 0 or 0 to 1 then only this clock will be felt by the third flip flop. So, that is why the last statement that we have here that this only the first flip flop receive clock pulse from the source or the clock generator and other flip flops they receive clock pulse from either Q or \bar{Q} of the previous flip flop. So, if you have got these type of connection then you see suppose, suppose this is the free running clock that we have ok. So, clock is high for this much amount of time then it is low for this much time again it is like this, so it goes like this.

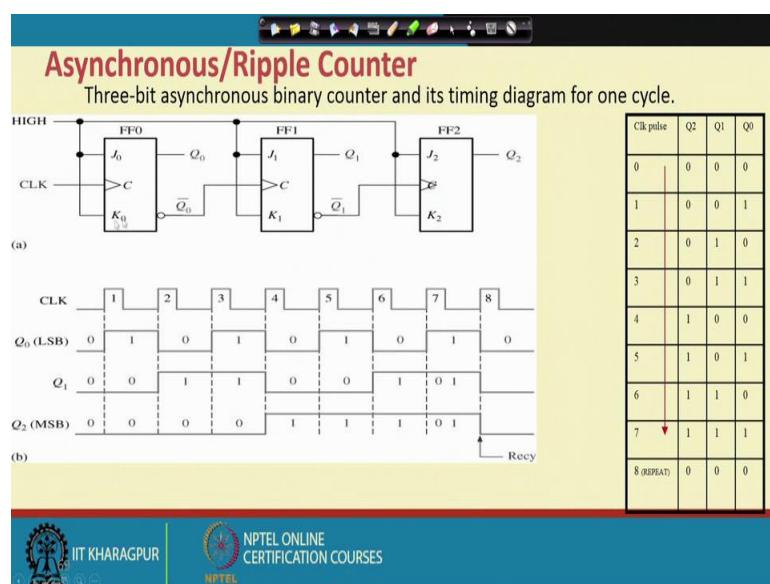
So, per the behavior that we have got so, if we have if we connect this clock, clock to the first flip flop, then due to the delay of the first flip flop, the first flip flop will initially the value was 0. So, it will it will become one after sometime and that time is given by this t_{PLH} , this is the propagation time from clock to Q_0 of the first flip flop, so it makes a transition here.

For the second flip flop, so it does not see the clock pulse immediately, but what happens is that it will see the clock pulse when this thing when this clock pulse to be high when I have got this Q_0 output to be equal to high, so it will see this here. So, then when this is when this particular Q_0 is going low. So, it will see that there is a transition in the Q_0 line. So, then this fellow will be making a transition getting Q_1 to be high and after sometime those this Q_2 will become high. So, at this point, so this is the Q_1 is making at transition from high to low. So, now, it will be making a transition like this.

So, here I have connected, so in this particular case it is connected that it is assume that clock is connected to Q_0 . So, the counter structure that is assumed here is like this. So, this is the first flip flop this is the second flip flop and this is the third flip flop. So, this is the clock is connected to the first flip flop like this and then this the second line second flip flop. So, we have got this \bar{Q} output connected to the clock and here also the \bar{Q} output is connected to the clock. So, you see that at this time this Q_0 is making a transition from high to low that means, that this \bar{Q}_0 is making a transition from low to high, so that is actually sensed here.

Similarly, at this point Q_1 makes a transition from high to low. So, this \bar{Q}_1 is making a transition from low to high and that is sensed by the third flip flop at this point. So, that way we can have this connection of this 3 bit counter to get this rippling effect.

(Refer Slide Time: 14:37)



So, this is the structure, so what we have done. So, all the 3 flip flops we have we have taken J K type of flip flop and this J, J and K, so they are tied high. So, you remember that in a J K flip flop if J and K both are tied high then whenever the clock comes so it will make a transition. Now you see that if this is the free running clock signal that we have so this Q_0 ok.

So, this particular flip flop, so this will transit at every rising edge of the clock. So, initially if the flip flops are content all 0 then at the first clock edge. So, this Q_0 makes a transition it becomes 1 then at the next clock edge, so Q_0 will become 0. So, it will go like this third clock edge again it will become high so it will go, so Q_0 timing diagram is like this

Now, once we have got the Q_0 time timing diagram $\overline{Q_0}$ timing diagram is just the reverse of this. So, whenever Q_0 makes a transition from 1 to 0 $\overline{Q_0}$ makes a transition from 0 to 1 and that is seen as the clock signal for flip flop 1. So, this Q_1 makes a transition when Q_0 goes from 1 to 0 or equivalently $\overline{Q_0}$ goes from 0 to 1, so it makes a transition here.

So, next transition of Q_1 occurs at this point ok. So, at this point again Q_1 makes a transition. So, you see now if you look into this count value if you say that this Q_2 is the most significant bit of my count, and Q_0 is the least significant bit of my count. So, initially if all this flip flops are assumed to be their content is assumed to be 0. So, initially it is 0 0 0 after the first clock pulse high, so it becomes 0 0 1, after the second clock pulse it becomes 0 1 0. Then it is 0 1 1 then 1 0 0, 1 0 1, 1 1 0, 1 1 1 and then it becomes 0 0 0.

Now, from this point onwards this is basically recycling now it again generates the same sequence. So, we can say that if we try to look into the behavior of this counter. So, it starts with 0 0 0 then at successive clock pulses at clock pulse 1 the count value will be 0 0 1 at clock pulse 2, 0 1 0 clock pulse 3, 0 1 1. So, it will go like this till it comes to this step 8 where it is 0 0 0, and now it goes on like that.

So, it is a 3 bit asynchronous binary counter we have got the timing diagram for one clock cycle that is for one cycle that is for 0 to 8. So, that is since it is a 3 bit counter. So, after 8 it will be over flowing and it will be coming back to 0 after 7 basically after 7 it will come back to 0 and now this sequence will be repeating.

Now, you see that this clock signal if you take it as an event for example, it may be on a conveyor belt items are passing, and there is a sensor light sensor which senses

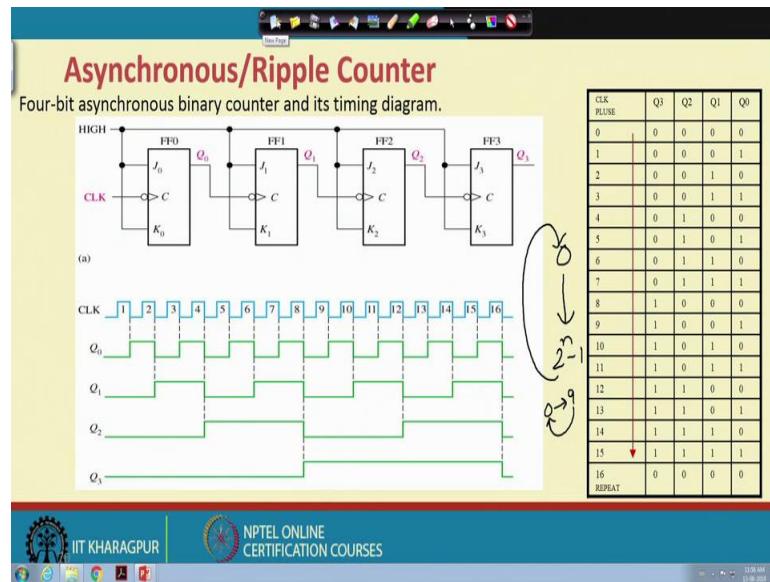
that whether some item is passing or not and whenever some item blocks that light. So, we have we suppose we have got a 1 ok, so that way.

So, this clock signal is generated from that items moving on to the conveyor belt and accordingly this counter will now be counting like how many times it is getting blocked or if we in intro if we have this circuit in the installed at the door of a room. And we have got appropriate sensors for sensing whether somebody is entering into the room or not.

So, every, every time a person enters or some obstruction occurs then the light may be there is a led and a sensor. And they will be sensing that the light is getting obstructed accordingly that can be treated as an event and every time that event occurs it will generate a pulse like this and accordingly the count value will get incremented at the end of the day, we can have a count about how many persons entered the room on that particular day.

So, this way we can have this ripple counter implemented and used in digital systems.

(Refer Slide Time: 18:51)



We can also have asynchronous ripple counter, so this is in this case 4 bit this is a 4 bit asynchronous ripple counter. So, previously it was a 3 bit counter now we have got a 4 bit counter. Where this Q₀ is connected to the clock of the $\overline{Q_0}$ basically this Q₀ and this is inversion. So that means, the falling edge of the clock the transitions will take place.

So, you can reason out in a similar fashion that if you start at time 0 then initially all these contents are 0. Then if you give some clock pulse at the following edge of the clock this Q_0 will become 1. Similarly, this flip flop will transit at the falling edges of Q_0 . So, at the falling edges of Q_0 this Q 1 makes a transition at falling edges of Q_1 Q_2 makes transitions, so like this and at falling edge of Q_2 Q_3 makes a transition.

So, here also if you look into the pattern, so initiate 0 0 0 0 then at this time it is 0 0 0 1 then it is 0 0 1 0, so it generates this particular sequence. So, 0 0 0 0 to 1 1 1 1 and then it repeats the sequence from the state 16. So, this way we have this 4 bit asynchronous binary counter.

(Refer Slide Time: 20:12)

The slide has a yellow background and a blue header bar with various icons. The title 'Asynchronous Decade Counter' is in red at the top left. Below the title is a bulleted list of six points. At the bottom, there are two logos: IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES'.

- The Modulus of a counter is the number of unique states that the counter will sequence through.
- Counter can also be designed to have a number of states in their sequence that is less than the maximum of 2^n .
- Counters with the states in their sequence are called **decade counters**.
- To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states.
- One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear (CLR) inputs of the flip-flops. The inputs the NAND gate are from the Q output from FF1 and FF3 (from 1010 -- **FF3FF2FF1FF0**)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

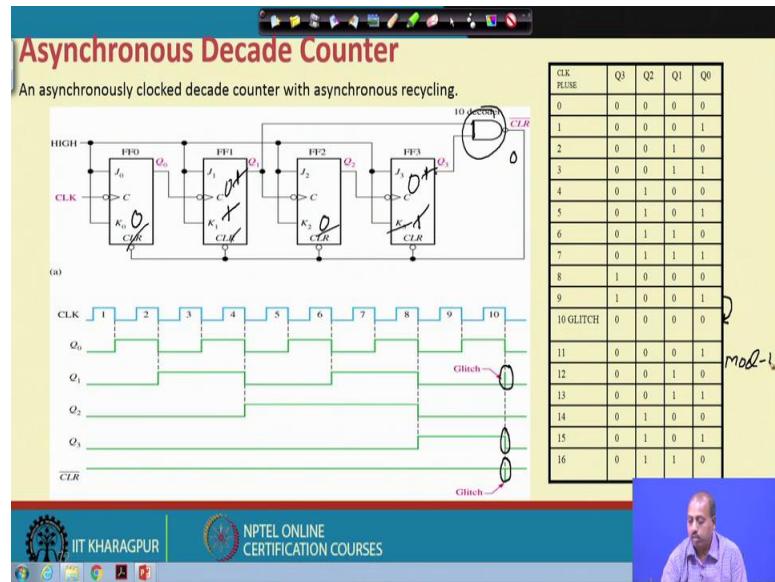
Sometimes, we don't want to count up to say the maximum capacity. So, if you look into this diagram, so if I put n such flip flops then you can understand that the count value, if I start at 0 it will go up to 2^{N-1} and then it will repeat, so this is the standard behavior.

Now, many times we don't want to go up to 2^N for example, in a 4 bit counter we may be interested to count up to from 0 to 9 only, and after 9 we want to come back to 0. So, that is if we are following some decimal system may be interested in some counting up to 0 to 9.

So, this type of counters they are known as decade counter that is it is modulo 10 counter ok. So, for getting that type of counter, so we can do some modification to the circuit that we have drawn here. So, modulus of a counter is the number of unique states that the counter will sequence through. Like previously that was counter that we have so that was mod 16 counter because it was going through 0 to 15, 16 distinct states, distinct, unique states.

The counter can also be designed to have a number of states in their sequence that is less than the maximum value of 2^N . So, decade counter is an example of that category, so where we have got up to 10 ok. So, how can we do that so one possible way of designing a decade counter is that after the counter is, we want to recycle the counter after 9. So, for that as soon as the count value becomes 1 0 1 0, maybe we can put some NAND gate and then make connect it to the clear line.

(Refer Slide Time: 21:58)



So, this diagram actually shows it. So, when the count value becomes 9, after that in the next clock pulse the count value becomes 10 that is 1 0 1 0. So, when it is 1 0 1 0, so the combination is sorry. So, this is the most significant bit so this is 1 0 1 0. So, when it comes like this now what we are doing we are taking out, so this one here and this one here.

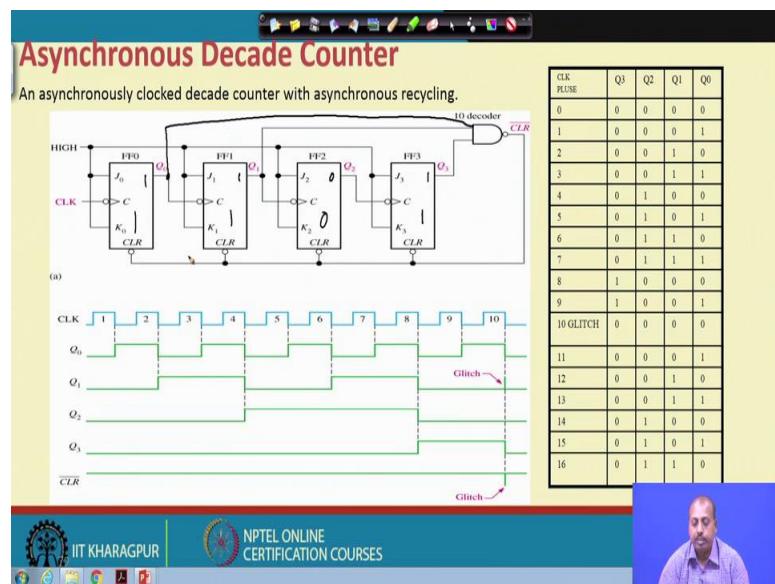
So, these two are taken, so they are given to this NAND gate and so NAND gate output becomes 0. And they are connected to the clear input of this flip flops and since this is

clear bar connection. So, as soon as this clear line becomes 0. So, all this flip flops will become 0, so their content will become 0 they will become 0.

So, essential what happens is that after this counting 9 when it goes to 10, it immediately comes back to 0. Of course, for that is small amount of time it will show the value 10 that is determined by the delay of this NAND gate and the delay of this clear in circuitry in the individual flip flops, but after that the value will become equal to 0. And if that value is much less compared to the clock duration then that value may be negligible, but you will get a glitch like this. So, here you will get a glitch momentarily the value will become 1 0 1 0 and then, value will be momentarily become 1 0 1 0 and then it will be going to all 0 ok.

So, this way you can design you can put some extra gate. So, whatever count value whatever mod value is given suppose, I am asked to design a mod 11 counter, mod 11 counter means 11 in that case, so if it is mod 11. So, it is 1 0 1 1. So, in that case what I have to do is that I have to take a NAND gate where this connections will be there. So, this is 1 0 1 1, so these two ones are already there, so I have to take this one also and connect it as another input to this NAND gate.

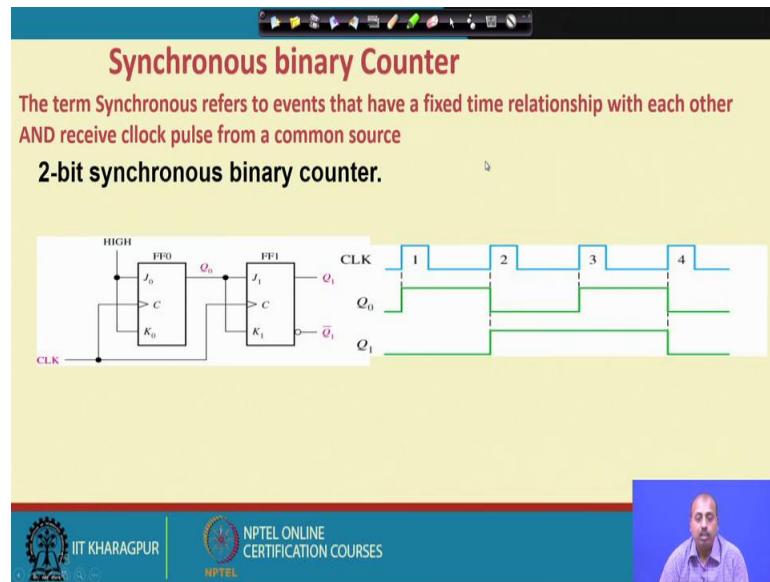
(Refer Slide Time: 24:21)



So, then what will happen is that when the count value becomes 1 1 0 1 or 1 0 1 1 then this clear value will come and it will reset the counter. So, that way you can very easily decide this resetting value. So, depending upon the mod value that you have so, you can decide

whenever that particular count is reached. So, you can say that I will stop at that point and you can get this the decade counter or whatever be the mod value. So, accordingly you can design the counter.

(Refer Slide Time: 24:59)



So, you can also design counters in a synchronous fashion. So, previously we have seen the counters which are asynchronous in nature because the individual flip flop they were not operating at the same clock. So, all the flip flops they had clocks, but the clock was not common throughout the design.

But, so, another possible type of design is when the same clock signal is fed to all the flip flop. So, that will give rise to synchronous design and this synchronous counter design is also simple, like in this particular case what we do? It is a 2 bit synchronous counter. So, there is a, this clock signal is connected directly to both the flip flops, but this is the J K line. So, this J and K lines are tied high for the first flip flop 0, and for the second line J and K, so they are tied high together and connected to the Q₀ lines.

So, whenever this Q₀ equal to 1 then only this J₁ and K₁ they get the value 1. So, as result this flip flop will toggle. So, if you look into the timing diagram, you see that the first flip flop Q₀. So, it toggles after at the rising edge of every clock ok, because this J and K are tied high. So, they, they are actually doing the transition at the rising edge of the clock. On the other end this second flip flop the flip flop 1, it makes its transition whenever Q₀ is equal to 1.

So, at this point Q_0 is equal to 1 now when the clock signal comes at this point Q_1 does not make a transition because Q_0 was equal to 0 at that time. Now so it could not make any transition, but at this point Q_0 was equal to 1. So, Q_1 makes a transition, so it becomes 1. So, after then again at this clock edge Q_0 is equal to 0. So, Q_1 does not transit, but at this point it will again transit.

So, if you look into this value that it is counting. So, it is 0 0 here then it is 0 1 at this point then it is 1 0 at this point, and then it is 1 1 at this point then it is again going back to 0 0. So, that is a, that is a 2 bit synchronous binary counter. So, in this we can design synchronous counters also. It is not mandatory that the counters should be asynchronous in nature.

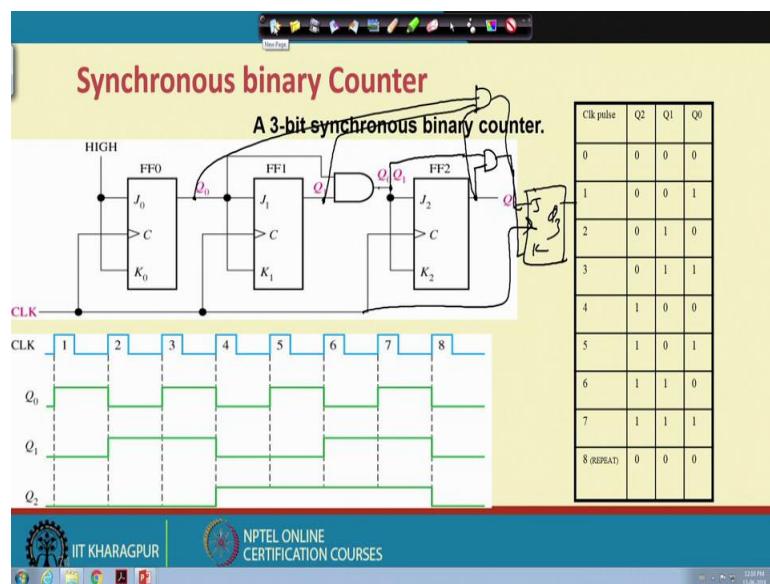
So, you can design synchronous counters also the advantage with the synchronous counter is that the problem of glitch will not be there. So, asynchronous circuits we have those glitch, but in synchronous situations since we are bothered about changes only during the clock boundary. So, only when the clock edge comes the circuit will transit, the flip flops will transit. So, the glitch problem will not be there, so that is the advantage and many of the counters are designed in a synchronous fashion.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
Sequential Circuits (Contd.)

Problem with synchronous counters is that many times, you need more amount of logic.

(Refer Slide Time: 00:20)



For example, say this one here we have got a 3 bit counter, 3 bit synchronous counter, but this 3 bit synchronous counter. So, this flip flop 0, flip flop 1 and flip flop 2. So, this flip flop 2 is the most significant bit flip flop 1 is the next bit and flip flop 0 is the least significant bit.

Now, you see that in the connection pattern. So, what we need is that, every clock pulse this flip flop 0 should toggle ok. So, this should toggle flip flop one should toggle if flip flop 0 was equal to 1 like if we look into this counting sequence. So, this whenever this flip flop 0 it toggles at every clock fine flip flop 1 Q₁ it changes the state on only when Q₀ was equal to 1 at the previous time.

You see whenever this is equal to 1. So, it makes a transition similarly this is equal to 1, it makes a transition, but if Q₀ is equal to 0, in that case, it is not making any transition like here also, it is not making any transition. So, it is coming from one to one only and as far

as Q_2 is concerned. So, Q_2 should make a transition when Q_0, Q_1 both are equal to 1. So, here Q_0, Q_1 both are equal to 1. So, at the next time, it makes a transition to one and they become 0 that is the different thing.

So, while drawing the corresponding logic circuit. So, what is required is that for this J and for the flip flop 0. So, they will transit at every clock pulse. So, this is tied high directly. Now for this flip flop 1 this J and K are shorted and it is connected to this Q_0 line. So, whenever Q_0 equal to 1 at the next clock, this flip flop 1 will transit then this flip flop 2, it is connected again J and K are shorted, but it is connected via AND gate where this Q_0, Q_1 , they are ANDed and then this is connected to flip flop 2.

So, this flip flop 2 will transit when Q_0, Q_1 both the outputs are equal to 1. So, this way you can have a 3 bit synchronous counter. Now if you have more number of stages like if you have trying to have a 4 bit synchronous counter, then you see that this you will be needing another flip flop here ok. So, you will be needing another flip flop here with J and K the clock signal will be common clock signal there is no problem.

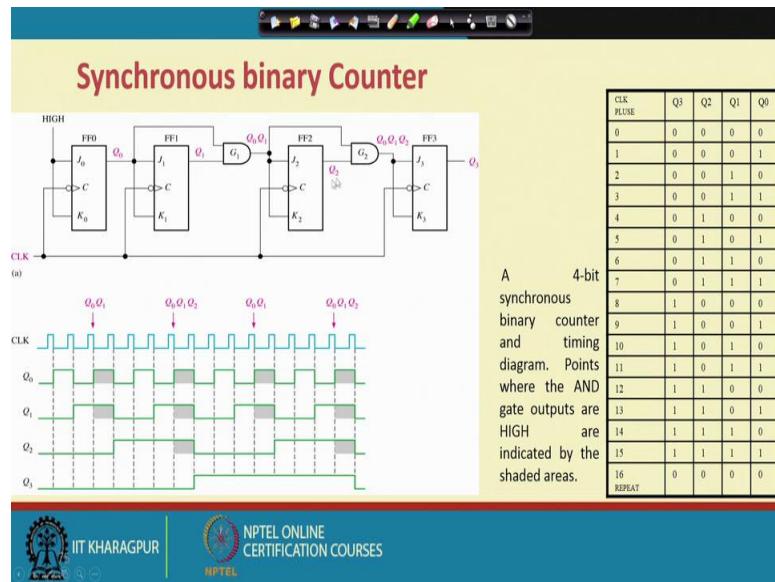
Now this J and K are to be tied together and then that value should come from another AND gate. Now this AND gate should have input from this line ok, it should have input from this line and also, so, this Q_3 , if you say that this line is say Q_3 , then Q_3 will transit only when all this Q_0, Q_1 and Q_2 all of them are 1 ok. So, that is this line should be 1 and this Q_0, Q_1 . So, this line should also be equal to 1. So, you can make a connection like this and then you will be getting this thing.

So, this way, I can have another stage of a counter we can get a 4 bit synchronous counter, but the problem that you can visualize is that you see the farther stages that I am connecting, so, the AND gate, I am forming a chain of AND gates. So, the first, this flip flops says the sees the delay of 1 and gate, but this flip flop sees the delay of these 2 AND gates to get the correct value.

So, if you connect another state will see delay of 3 AND gates in the process or you use AND gates of higher number of input. So, one another possible way of connecting, it is that you do this thing and then you take this output this Q this Q output, this Q output and this Q output and connect it together and then feed it here. So, that may be other possibility, but in that case the AND gate is becoming more complex.

So, you are recording AND gate of higher number of inputs. So, that is the other difficulty. So, this way this synchronous design, they have got this type of problem the circuit becomes complex. Anyway, so, many times depending upon the situation, we choose between synchronous counter asynchronous counter.

(Refer Slide Time: 04:42)



So, here is actually it is shown that we have got this 4 bit synchronous counter and this G_1 is feeding this flip flop 2 by ANDing Q_0, Q_1 , then this G_2 is feeding flip flop 3 by ANDing of this Q_0, Q_1 with Q_2 .

And then a 4 bit synchronous binary counter and timing diagram have been shown. So, the when the AND gate outputs are equal to high, so, they are shown by these shaded portions. So, essentially, it will go through this sequence starting with all 0 going up to all one and then from 16 cycle, it will be repeating. So, it will be again starting with all 0 and continue like this. So, that is a synchronous binary counter type structure that we have.

(Refer Slide Time: 05:32)

The screenshot shows a presentation slide with a yellow background. At the top, it says "Synchronous Counter Design". Below that, it states: "Several methods are available that follow arbitrary sequence. Here we will learn one common method using JK flip-Flops. In synchronous counters all the FF's are clocked at the same time." Underneath this text, there is a section titled "J-K Excitation Table". It says: "Before begin the designing we must know the operation of the J-K FF, let us analysis Truth table for 74LS76 IC (JK flip-flop) and its excitation table." At the bottom of the slide, there are two logos: IIT Kharagpur and NPTEL.

So, how do we design a synchronous counter? So, the example that I have shown you so, it seems that all on a sudden. So, we could land into this particular design, but how do we come to that. So, for that purpose there are several methods that are available for that that can follow arbitrary sequence of counters. So, we will be we will first; we will see some J K flip flop based implementation and since, it is synchronous design. So, all flip flops are clocked at the same time. So, flip flop clocking is we do not have any opportunity to manipulate the clock signal.

So, clock signal is common and it is fed from the same source for all the flip flops. So, all you can do is that you can control this J and K inputs for this flip flops to get your desired sequence.

So, the J K excitation table; so, it is known to us just to recapitulate.

(Refer Slide Time: 06:24)

PRESENT	NEXT	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if the present value is 0 and the next value is also 0, so, what we have to do is this J should be 0 and K can be don't care. Similarly from 0 to 1, if the present value of this flip flop Q output is 0 and the next output we expect to be 1. So, for that I need to give J equal to 1 and K can be don't care. Similarly, from 1 to 0, K should be equal to 1 and J is don't care and for 1 1, I should, this J K value must be equal to 0 and this J value can be a don't care. So, this is the J K excitation table. So, that is known to us.

(Refer Slide Time: 07:03)

TRANSITION AT OUTPUT	PRESENT STATE Q(N)	NEXT STATE Q(N+1)	J	K
0 → 0	0	0	0	X
0 → 1	0	1	1	X
1 → 0	1	0	X	1
1 → 1	1	1	X	0

0 to 0 TRANSITION; FF's Present status is 0 and it should remain in 0 when a clock pulse is applied.
That can be either J=K=0 status or J=0,K=1.

That mean J=0 and K=0 or 1. That is, J=0 and K=X(don't care)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so, for this is again for J K excitation table from 0 to 0; so, present state is 0 and the next state is 0 and J is 0. So, we already explained. So, 0 to 0 transition flip flop's present status is 0 and it should remain 0 when a clock pulse is applied that can be either J equal to K equal to 0 or J equal to 0 K equal to 1. So, it is J equal to 0 K equal to 0 or 1. So, that is J equal to 0 and K equal to don't care. So, that explains this table you have already seen that.

(Refer Slide Time: 07:37)

The slide has a yellow header bar with various icons. The main title is 'Synchronous Counter Design' in red. Below it are three boxes describing transitions:

- 0 → 1 TRANSITION:** The present state is 0 and it has to change to 1. This can happen either $J=1$ and $K=0$ or $J=K=1$.
That mean always $J=1$ and $K=0$ or
 $J=1$ and $K=X$ (don't care)
- 1 → 0 TRANSITION:** The present state is 1 and it has to change to 0. This can happen either $J=0$ and $K=1$ or $J=K=1$.
That mean always $K=1$ and $J=0$ or
 $K=1$ and $J=X$ (don't care)
- 1 → 1 TRANSITION:** The present state is 1 and it has to change to 1. This can happen either $J=K=0$ or $J=1$ and $K=0$.
That mean always $K=0$ and J can be either level
 $K=0$ and $J=X$ (don't care)

At the bottom left is the IIT Kharagpur logo and text 'IIT KHARAGPUR'. At the bottom center is the NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. At the bottom right is a small video window showing a man speaking.

Now, for synchronous counter design; so, you have to see when these transitions are occurring like when this 0 to 1 transition is occurring present state is 0 and it has to change to 1 with J equal to 1, K equal to 0 or J equal to 1, K equal to 1. Similarly from 1 to 0 transition, I can get by K equal to 1 and J equal to 0 or 1 and one to one transition K must be equal to 0 and J can be either of the value 0 or 1.

(Refer Slide Time: 08:06)

Synchronous Counter Design

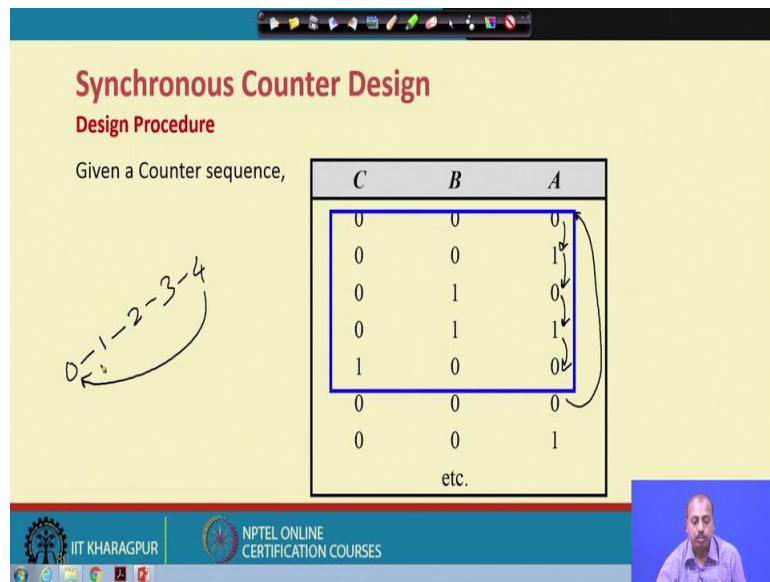
Design Procedure

Given a Counter sequence,

Dashed arrows indicate the sequence: 0 → 1 → 2 → 3 → 4 → 0

C	B	A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
0	0	0
0	0	1
etc.		

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**



So, suppose we are asked to design a counter which goes like this. So, this is the sequence that is given to us that is the it says that it is a 3 bit counter, it will start at 0 0 0 and the next time it will go to 0 0 1, then it will go to 0 1 0, then 0 1 1, then 1 0 0 and after that it goes back to again 0 0 0, suppose, this is the sequence that I need to generate. So, in terms of decimal value; so, the sequence that I need is 0, 1, 2, 3 4 and then again 0 ok.

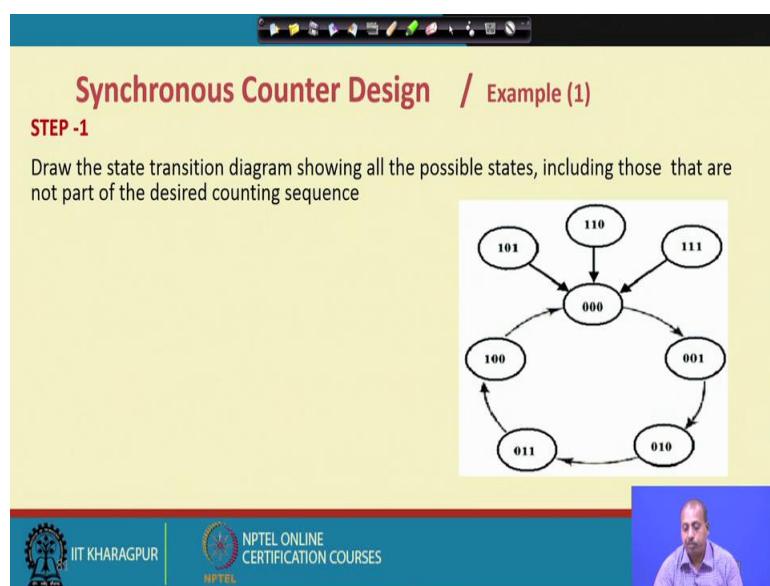
So, how do we design such a counter? So, that is what will try to see.

(Refer Slide Time: 08:54)

Synchronous Counter Design / Example (1)

STEP -1

Draw the state transition diagram showing all the possible states, including those that are not part of the desired counting sequence



So, the first point first step for designing such a counter is to draw something called the state transition diagram for the counter. So, it is like this initially. So, this counter value is 0 0 0, then it goes to 0 0 1 and the next clock pulse, then 0 1 0, 0 1 1, 1 0 0 and going back to 0 0 0, it does not tell anything about what are you going to do for the other states. So, may be, so, if you if you assume like this that if you start at any other state, then if the by chance the counter reaches any of these states 110, 101, 110 or 111, then at the next clock pulse. So, it will come back to this state all 0.

So, it will be starting from because that is the. So, this is the, this is actually the drawing the state transition diagram showing all the possible states including those that are not part of the desired counting sequence.

(Refer Slide Time: 09:50)

	Present state			Next state		
	C	B	A	C	B	A
1	0	0	0	0	0	1
2	0	0	1	0	1	0
3	0	1	0	0	1	1
4	0	1	1	1	0	0
5	1	0	0	0	0	0
6	1	0	1	0	0	0
7	1	1	0	0	0	0
8	1	1	1	0	0	0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, once we are done, we are done with this. So, the next step is to use the state transition diagram to set up a table to list all present and next states. So, you see from 0 0 0, if this is the current state then from this diagram, we know the next state should be 0 0 1. So, that is written here from 0 0 0 to 0 0 1 from 0 0 1 to 0 1 0. So, this is sorry; this is coming directly from this table from this state transition diagram particular. So, when we reach this 1 0 0, then the next state is 0 0 0.

So, from 1 0 0, the next state is 0 0 0 and if from other value like 1 0 1, the next state should be 0 0 0 for 1 1 0, next state should be 0 0 0. So, like that. So, this is actually capturing these 3 state transitions. So, 1 0 1, 1 1 0, 1 1 1 to 0 0 0.

(Refer Slide Time: 10:47)

Synchronous Counter Design / Example (1) ...cont.

STEP -3

Add a column to this table for each J and K input. For each **PRESENT** state, indicate the level required at each J and K input in order to produce the transition to the **NEXT** state.

	Present state			Next state								
	C	B	A	C	B	A	j _C	k _C	j _B	k _B	j _A	k _A
1	0	0	0	0	0	1	0	X	0	X	1	X
2	0	0	1	0	1	0	0	X	1	X	X	1
3	0	1	0	0	1	1	0	X	X	0	1	X
4	0	1	1	1	0	0	1	X	X	1	X	1
5	1	0	0	0	0	0	X	1	0	X	0	X
6	1	0	1	0	0	0	X	1	0	X	X	1
7	1	1	0	0	0	0	X	1	X	1	0	X
8	1	1	1	0	0	0	X	1	X	1	X	1


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Now, so, that table is drawn and since there will be realized by means of J K flip flop. So, each of these bits A, B and C, they will be realized by J K flip flop. So, we try to see what should be the corresponding values of J and K. So, see for this for this one; so, 0 to 0; so, this J bit must be 0 K bit can be don't care from the JK excitation table. So, we know that similarly from say 0 this also 0 to 0. So, this J bit is 0 and K bit is don't care.

So, at this point at this level. So, we have got 0 to 1 transition for that J bit should be equal to 1 and K bit is don't care, similarly, if you take any other points say the this at the transition of the A bit. So, 0 to 1. So, for that this J A that is J input of the a flip flop that should be equal to 1 and K input can be don't care. So, this way, we fill up this part. So, this present state next state. So, this is coming from the counting sequence and once we know that once we know the counting sequence. So, you can derive what should be the control values that should be fed to the J and K inputs of individual flip flops to get the appropriate state transition for the flip flops.

(Refer Slide Time: 12:11)

Synchronous Counter Design / Example (1)cont.

STEP-4

Design the logic expression to generate the level required at each J and K, using K-maps.

Present state			j_A	k_A
C	B	A		
0	0	0	1	X
0	0	1	X	1
0	1	0	1	X
0	1	1	X	1
1	0	0	0	X
1	0	1	X	1
1	1	0	0	X
1	1	1	X	1

$j_A = \overline{C}$

$k_A = 1$

NPTEL ONLINE
CERTIFICATION COURSES

After that the step is you have to make the corresponding combinational logic. So, if the current state is 0 0 0, then in 1 that J_A should be one and K_A should be don't care. So, that way you write down the truth table ok. So, so, 0 0 0 to 1 1 1 and then what is the corresponding expected value of J_A and K_A so, that we can write down and then you put it into a Karnaugh map do minimization and we see that the function that we getting for $J_A = \overline{C}$, $K_A = 1$ ok.

So, $J_A = \overline{C}$, $K_A = 1$ that is the truth table that is that is logic function for the A bit.

(Refer Slide Time: 13:01)

Synchronous Counter Design / Example (1)cont.

STEP-4cont.

Present state			j_B	k_B
C	B	A		
0	0	0	0	X
0	0	1	1	X
0	1	0	X	0
0	1	1	X	1
1	0	0	X	X
1	0	1	X	X
1	1	0	X	1
1	1	1	X	1

$j_B = A\overline{C}$

$k_B = A + C$

NPTEL ONLINE
CERTIFICATION COURSES

Similarly, for the B bit, so, J_B and K_B are to be derived. So, here also we do the same thing that from that previous table, we find out from actually from this big table, you just take out this present state this 3 bits and this J_B and K_B columns. So, that way, we just we are just rewriting it here sorry, here J_B and K_B column and then you do a logic minimization by means of Karnaugh map. So, you see that the function that we are getting is $J_B = A\bar{C}$, $K_B = A + C$ and the same thing; we do for the C flip flop ok.

(Refer Slide Time: 13:40)

Synchronous Counter Design / Example (1)cont.

STEP- 4
.....cont.

Present state			<i>j_C</i>	<i>k_C</i>
<i>C</i>	<i>B</i>	<i>A</i>		
0	0	0	0	X
0	0	1	0	X
0	1	0	0	X
0	1	1	1	X
1	0	0	0	1
1	0	1	X	1
1	1	0	0	1
1	1	1	X	1

$j_C = AB$

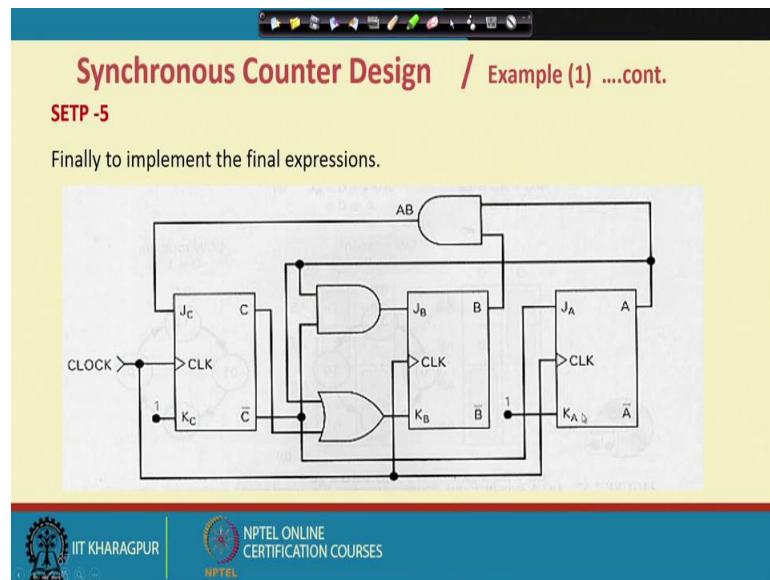
$\overline{B} \ \overline{C}$ \overline{A} A
 $\overline{B} \ C$ X X
 $B \ C$ X X
 $B \ \overline{C}$ 0 1

$k_C = 1$

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

And then we again take out the portions of present state and this J_C , K_C columns and accordingly draw the truth table and make the Karnaugh map and do minimization and $J_C = AC$, $K_C = 1$ turns out to be one once this is done. So, we are now ready to draw the circuit.

(Refer Slide Time: 14:02)



You see that this; if you look into any of them say K_C should be equal to 1. So, where is K_C. So, this K_C is tied high, similarly, this J_C should be equal to AB. So, this J_C this A and B this 2 bits have been taken and they are ANDed and this AB is fed to J_C. Now if you go back, see this K_B equal to A + C. Now, you see this K_B. So, we are ORing this C output and A output. So, these 2 flip flop output are ORed and it is fed K_B. So, this circuit that we are getting so, you see the clock is common to all the 3 flip flops and they are coming from the same source. So, this is the synchronous design and. So, if you start it with some initial value ok. So, it will make transitions accordingly.

So, it will follow this particular sequence that we have. So, start it at any of these values; so, it will be producing this value and if you start at any of these 3 states, it will be first transiting into 0 0 0, then it will continually making transitions in this sequence. So, that way we can design some synchronous counters.

(Refer Slide Time: 15:16)

Synchronous Counter Design / Example (2)

Design a JK synchronous counter that has the following sequence: 000, 010, 101, 110 and repeat. The undesired states 001, 011, 100 and 111 must always go to 000 on the next clock pulse.

STEP -1 : State Transition Diagram

MAX = $\lceil \log_2(n_{MAX}) \rceil$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We can modify the designer bit. So, we want to design a sequence where the sequence is 0, then 2, then 5 and then 6 ok. So, the sequence in terms of decimal, what I am looking for is 0, followed by 2, followed by 5, followed by 6 and then coming back to 0. Now how do you say like how many flip flops will be needed. So, that is determined by this number the maximum count value that you have. So, this is it is 6 for representing 6 in a binary notation we need at least 3 bits. So, that is given by. So, if I say that this is the max value that the counter should produce. So, the number of flip flops needed is given by this $\log_2(\max)$.

So, in this case, this max value is equal to 6. So, this turns out to be equal to 3. So, if you are designing a sequence, if I tell you that you have to design a sequence like this the 0 5, 15, 25, 11 and then 0, then for deriving how many flip flops will be needed. So, you have to look into this value 25 and this max equal to 25. So, you will need a 5 bit, you will need 5 flip flops for designing ok.

So, that way this log value of this max. So, it will tell you how many flip flops will be needed for designing the counter. Now in this particular case. So, we have got we have we have we have to design this particular sequence. So, 0 0 0 followed by 0 1 0, 1 0 1 and 1 1 0 and repeat the undesired states 0 0 1, 0 1 1, 1 0 0 and 1 1 1, they must always go to 0 0 0 on the next clock pulse. So, that is expected. So, it will go to 0 0 0 in the next clock pulse.

So, this state transition diagram; so, it will be like this.

(Refer Slide Time: 17:18)

The slide title is "Synchronous Counter Design / Example (2)cont.". A red caption below the title reads "STEP- 2 : Table to list PRESENT and NEXT status". The table has two sections: "PRESENT State" and "NEXT State".

PRESENT State			NEXT State		
C	B	A	C	B	A
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	0

At the bottom left is the IIT Kharagpur logo, and at the bottom right is the NPTEL Online Certification Courses logo.

And then you can just draw the present state next state type of thing. So, it is 000 to 010, 001 to 000. So, that way you fill up this table. So, as per this sequence from 0 0 0 to 0 1 0 like that so, based on that you fill up this table.

(Refer Slide Time: 17:36)

The slide title is "Synchronous Counter Design / Example (2)cont.". A red caption below the title reads "STEP- 3 : Table indicate the Level required at each J and K inputs in order to produce the transition to the NEXT".

Present State			Next State								
C	B	A	C	B	A	J _c	K _c	J _b	K _b	J _a	K _a
0	0	0	0	1	0	0	x	1	x	0	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	1	0	1	1	x	x	1	1	x
0	1	1	0	0	0	0	x	x	1	x	1
1	0	0	0	0	0	x	1	0	x	0	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x
1	1	1	0	0	0	x	1	x	1	x	1

At the bottom left is the IIT Kharagpur logo, and at the bottom right is the NPTEL Online Certification Courses logo.

And then indicate the level required at each J and K inputs. So, that is similar to what we have done previously this J_c and K_c for the C flip flop J_b and K_b for the B flip flop and

this J_A and K_A for the A flip flop. So, that way we derive the, we write down the transition table.

(Refer Slide Time: 17:58)

inputs			Present State					
C	B	A	J _c	K _c	J _b	K _b	J _a	K _a
0	0	0	0	x	1	x	0	x
0	0	1	0	x	0	x	x	1
0	1	0	1	x	x	1	1	x
0	1	1	0	x	x	1	x	1
1	0	0	x	1	0	x	0	x
1	0	1	x	0	1	x	x	1
1	1	0	x	1	x	1	0	x
1	1	1	x	1	x	1	x	1

IIT KHARAGPUR |
 NPTEL ONLINE CERTIFICATION COURSES

And then we have to go for the logic circuit. So, for the, for this C B A values to be this, this, this, then these 3 values. So, what are the values of J_C , K_C , J_B , K_B and J_A , K_A . So, that part if you write down then you can make Karnaugh map and do a minimization for each of this J_C , K_C , J_B , K_B and J_A , K_A and all.

(Refer Slide Time: 18:20)

Synchronous Counter Design / Example (2)cont.

STEP-5 : Simplify the SOP expression using K-maps

$J_A = BC\bar{C}$

$K_A = 1$

IIT KHARAGPUR |
 NPTEL ONLINE CERTIFICATION COURSES

And so, for $J_A = B\bar{C}$, $K_A = 1$.

(Refer Slide Time: 18:27)

Synchronous Counter Design / Example (2)cont.

$J_B = CA + \bar{C}\bar{A}$

$K_B = 1$

$J_C = \bar{A}B$

$K_C = B + \bar{A}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then this $J_B = CA + \bar{C}\bar{A}$, $K_B = 1$; $J_C = \bar{A}B$, $K_C = B + \bar{A}$. So, now, you can final circuit is not shown here. So, that is just another step. So, you have got 3 counters and then you have got appropriate logic connected to the J and K inputs of individual flip flops.

(Refer Slide Time: 18:53)

Synchronous Counter Design / Example (3)

Design a JK synchronous counter that has the following sequence: 000, 010, 101, 110 and repeat. For undesired states their NEXT states can be DON'T CARES.

STEP -1 : State Transition Diagram

000
010
101
110

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we look into another example where this is again a J K based synchronous counter that counts in the sequence 0 0 0, 0 0 1, 1 0 1, 1 0 1, 1 1 0 and repeat, but all the undesirable states undesirable states the next state can be don't care. So, we do not bother like if we if they start at any of the state which are not stated here then what the system will behave.

So, that is not to be a concern ok. So, that is the user will ensure that those undesirable states will never be reached by this counter; they it will never start at those state.

(Refer Slide Time: 19:35)

PRESENT State			NEXT State		
C	B	A	C	B	A
0	0	0	0	1	0
0	0	1	x	x	x
0	1	0	1	0	1
0	1	1	x	x	x
1	0	0	x	x	x
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	x	x	x

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL



So, this C B A, it can go from 0 0 0, all the 8 combinations can occur, but out of that we are bothered about only these patterns 0 0 0, 0 1 0, 1 0 1 and 1 1 0 for example, we are not bothered about the state 0 0 1. So, at if it if the system is at count value is 0 0 1, what will be the next count value that is we don't bother. So, they are do not cares similarly if the if the current count value is 0 1 1, then what is the next count value. So, that is not relevant. So, that is don't care. So, this way you can introduce a number of don't cares into your design and after that you can do a minimization ok.

(Refer Slide Time: 20:12)

Present State			Next State								
C	B	A	C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	1	0	0	x	1	x	0	x
0	0	1	x	x	x	x	x	x	x	x	x
0	1	0	1	0	1	1	x	x	1	1	x
0	1	1	x	x	x	x	x	x	x	x	x
1	0	0	x	x	x	x	x	x	x	x	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x
1	1	1	x	x	x	x	x	x	x	x	x

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so, if it is 0 0 1 and it is don't care, then all these values are they will turn out to be don't cares. Similarly, we can have these thing, we can have this 1 0 0 and this is don't care so, they are all going to be don't cares. So, this way we can have this don't cares put into this J_a , J_b , J_c , K_a , K_b , K_c columns and accordingly we can do minimization and get the final circuit.

(Refer Slide Time: 20:45)

Present State			Next State					
C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	x	1	x	0	x
0	0	1	x	x*	x	x	x	x
0	1	0	1	x	x	1	1	x
0	1	1	x	x	x	x	x	x
1	0	0	x	x	x	x	x	x
1	0	1	x	0	1	x	x	1
1	1	0	x	1	x	1	0	x
1	1	1	x	x	x	x	x	x

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you can design the logic circuit for this. So, this is we just write down this table again.

(Refer Slide Time: 20:55)

STEP- 5 :Simplify the SOP expression using K-maps

J_a=BC K_a=1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And then do minimization as we have done previously and then we can make the final circuit.

(Refer Slide Time: 21:01)

Objective:
To design a 3 bit counter (D FF) with the following count sequence 7,6,5,4,1. All unwanted stages go to 7.

Output sequence 7,6,5,4,1
In 3 bits format: 111,110, 101, 100, 001

State transition diagram:

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another example that we take is to design a 3 bit counter with D flip flop with the following sequence. So, this is not based on J K, but it is based on the D flip flop. So, it will count in the sequence 7 6 5 4 1 all unwanted states will go to state 7 and this output sequence is 7 6 5 4 1, then it is like this. So, it is starting at 7, then 6 5 4 1, then again 7, it goes like this

and if it starts at other states all unwanted states like 0 0 0, 0 1 0 and 0 1 1, they go to the state 1 1 1 that is 7 ok.

So, we can do, this is the specification of the problem.

(Refer Slide Time: 21:46)

The slide title is "Synchronous Counter Design / Example (4)cont.". Below the title is a caption "D Flip Flop Excitation Table:" followed by a 5x4 table:

PRESENT		NEXT		D	
0		0		0	
0		1		1	
1		0		0	
1		1		1	

At the bottom left is the IIT Kharagpur logo and text "IIT KHARAGPUR". In the center is the NPTEL logo and text "NPTEL ONLINE CERTIFICATION COURSES". On the right is a video frame showing a person speaking.

So, if you want to design it first of all, we need to recapitulate the D flip flop excitation table. So, D flip flop table is very simple if the present state is 0 and the next state is 0. So, D should be 0. So, it is depicted by the next state value that you need. So, accordingly D value should be set.

(Refer Slide Time: 22:05)

The slide title is "Synchronous Counter Design / Example (4)cont.". Below the title is a table:

OUTPUT						INPUT		
PRESENT STATE			NEXT STATE			C	B	A
C	B	A	C	B	A	D _C	D _B	D _A
0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	1	0	0	0	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0

At the bottom left is the IIT Kharagpur logo and text "IIT KHARAGPUR". In the center is the NPTEL logo and text "NPTEL ONLINE CERTIFICATION COURSES". On the right is a video frame showing a person speaking.

So, so, as per our specification from 0 0 0, the next state is 1 1 1. So, this D_A , D_B , D_C . So, that is they are going to be 1 1 1 similar. So, 0 1 so far those state. So, you see here for 0 0 0 and this 0 0 1 in both the cases, the next state is 1 1 1. So, that is done here from 000, it will go to 1 1 1 from 0 0 1 also, it will go to 1 1 1 accordingly, this D_A , D_B , D_C , they should be set to 1. So, it goes like this.

So, this way, we can think about this, we can make this table just we were doing it for this J K based counter. So, we can make this, we can make this D flip flop based counter and get the design.

(Refer Slide Time: 22:57)

The screenshot shows a presentation slide with the following content:

- K-Map:** Three Karnaugh maps for outputs D_C , D_B , and D_A based on inputs \bar{A} and A .

	\bar{A}	A
$\bar{C}\bar{B}$	1	1
$\bar{C}B$	1	1
CB	1	1
$C\bar{B}$	0	1
$D_C = A + C' + B$		

	\bar{A}	A
$\bar{C}\bar{B}$	1	1
$\bar{C}B$	1	1
CB	0	1
$C\bar{B}$	0	0
$D_B = AB + C'$		

	\bar{A}	A
$\bar{C}\bar{B}$	1	1
$\bar{C}B$	1	1
CB	1	0
$C\bar{B}$	1	0
$D_A = A' + C'$		

- Circuit Diagram:** A logic circuit diagram showing three D flip-flops connected in series. The first flip-flop's output Q is connected to its D input. The second flip-flop's output Q is connected to its D input. The third flip-flop's output Q is connected to its D input. The clock inputs of all three flip-flops are connected together.
- Logos and Footer:** IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a small video player window showing a professor.

Then we take it to the Karnaugh map and in the Karnaugh map, we can have this minimization like this we can have this minimization for D_C , D_B and D_A . So, they will turn out to be like this $D_A = \bar{A} + \bar{C}$, $D_B = AB + \bar{C}$, $D_C = A + \bar{C} + B$. So, once we do this minimization. So, you can draw the corresponding circuit. So, that circuit is not shown here, but essentially, what will happen is that you will have this 3 D flip flops. So, these are the 3 D flip flops and the clock will be common to all of them, the same clock will be go to each of them, but this D input. So, this is this is the D input for the A flip flop and A for the A flip flop, it say it is $\bar{A} + \bar{C}$.

So, this is this \bar{Q} line from here and \bar{Q} line for this is the A this is B and this is C. So, this \bar{Q} line and A line. So, they should be ORed and then they should be connected here, they should be ORed and connected here. So, in this way you can you can have the, you can

have the other logic circuit other portion like say. So, for say D_B , it is $AB + \bar{C}$. So, this A and B are output they should be ANDed and ORed with this \bar{Q} output of C that should be connected. So, whatever circuit the way we are drawn the previous circuit. So, we can also make this circuit and we can get the realization in terms of D flip flop.

(Refer Slide Time: 24:42)

The slide title is "Synchronous Counter Design (T Flip Flop based design / Example (5) ...cont.". Below the title is the heading "T Flip Flop Excitation Table:". A 4x3 table follows:

PRESENT	NEXT	T	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player showing a person speaking.

So, you can also design counters using T flip flop. So, the for example, this T flip flop excitation table is like this from 0 to 0 to 0 transition, T should be 0 from 1 to 1 transition also, T should be 0 for 0 to 1 or 1 to 0 transition, T input should be equal to 1. So, that is the excitation table out T flip flop.

(Refer Slide Time: 25:01)

Synchronous Counter Design / Example (5)cont.

T Flip Flop Input Function Table

OUTPUT			INPUT					
PRESENT STATE			NEXT STATE			C	B	A
C	B	A	C	B	A	T _C	T _B	T _A
0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	0
0	1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	0	0
1	0	0	0	1	1	1	0	1
1	0	1	1	0	0	0	0	1
1	1	0	1	0	1	0	1	1
1	1	1	1	0	0	0	0	1

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

So, now, if you go back look into the, our counter specification then for 0 0 0, it has to go to 1 1 1. So, this T_C, T_B and T_A; all the input should be equal to 1. Similarly, from 001 also, you want to go to 1 1 1. Now in this case, you see that A should not transit. So, T_A should be equal to 0 T_C and T_B they should be equal to 1. So, it should go like this, fine.

So, once this is done. So, rest of the thing is simple. So, from this present state, we want to apply T_A, T_B, T_C like this.

(Refer Slide Time: 25:35)

Synchronous Counter Design / Example (5)cont.

K-Map

	\bar{A}	A
$\bar{C}\bar{B}$	1	1
$\bar{C}B$	1	1
CB	0	0
$C\bar{B}$	1	0
$T_C = A'B' + C'$		

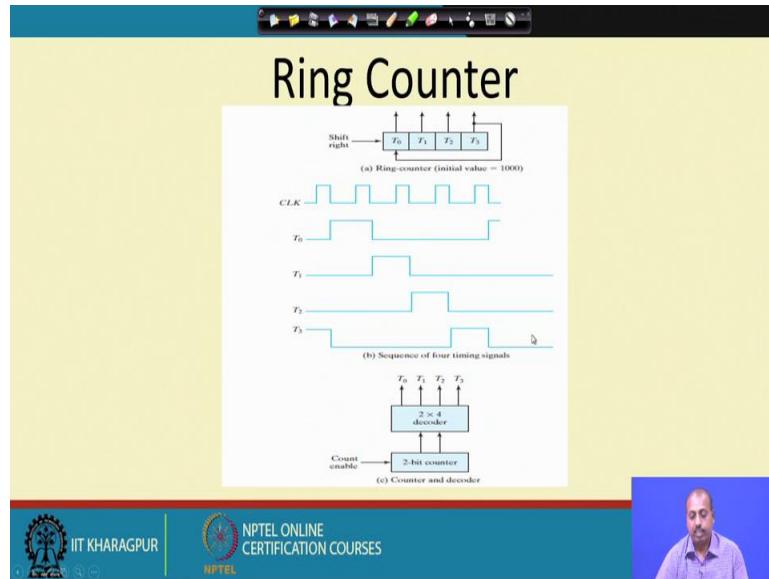
	\bar{A}	A
$\bar{C}\bar{B}$	1	1
$\bar{C}B$	0	0
CB	1	0
$C\bar{B}$	0	0
$T_B = B'C' + A'BC$		

	\bar{A}	A
$\bar{C}\bar{B}$	1	0
$\bar{C}B$	1	0
CB	1	1
$C\bar{B}$	1	1
$T_A = A' + C$		

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

So, you have to derive the corresponding functions and then. So, this $T_C = \bar{A}\bar{B} + \bar{C}$, $T_B = \bar{B}\bar{C} + \bar{A}BC$, $T_A = \bar{A} + C$. So, that way we can have this Karnaugh map for this T flip flop and accordingly, we can make the circuit for the T flip flop based realization.

(Refer Slide Time: 26:00)



So, another important type of counter that we have is the ring counter. So, in ring counter we have got say this initial it is realized by means of T flip flop then this initial value of the ring counter is say 1 0 0 0. So, this is T value is. So, this T₃ value is 1 and these values are 0 0 0, then since this is a T flip flop and this input is 1. So, at the next clock pulse ok. So, this T₀ will be making a transition. So, at the falling edge T₀ makes a transition and it becomes equal to 1, but rest of the otherwise this is a shift register.

So, this that way this is this values are shifted. So, this the T₂ value. So, that was initially 0. So, it comes to this and then you can have this type of shifting ok. So, this is it will be it will be generating the count in the sense it will first 0 0 0 1 and then 1 0 0 0, then 0 0 0 1, 0 0 1 0, 0 1 0 0 and 1 0 0 0. So, you can do the same thing by means of this 2 bit counter and a 2 to 4 decoder so what, this 2 bit counter is a standard counter.

So, it will go from 0 0, 0 1, 1 0 and 1 1. When 00 is given as input, so, T₀ is made high when 0 1 is given as input this decoder will make T₁ equal to high. So, it will go like this. So, that way we can get a decode counter and we can realize it.

(Refer Slide Time: 27:40)

Johnson Counter

- Also known as the twisted-ring counter.
- Same as the ring counter except that the inverted output of the last FF is connected to the input of the first FF.
- Counting sequence:
000 → 100 → 110 → 111 → 011 → 001 → 000
- A MOD-6 counter (twice the number of FFs)

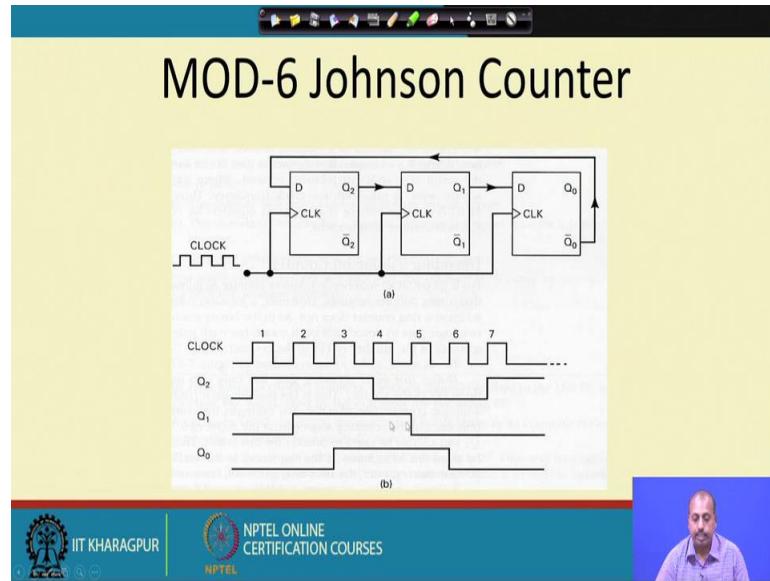
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Another counter is known as Johnson counter. So, this is also known as twisted ring counter because it requires less number of flip flops. So, it is counting in the sequence like say 0 0 0 to, it is counting in the sequence like 000 to 100, 110, 111. So, it generates all the 6 patterns, it goes in a mod 6 fashion. So, the advantage is that number of states that you are getting is twice the number of flip flop. So, we have got 3 flip flops, but using 3 flip flops you can get 6 different patterns ok.

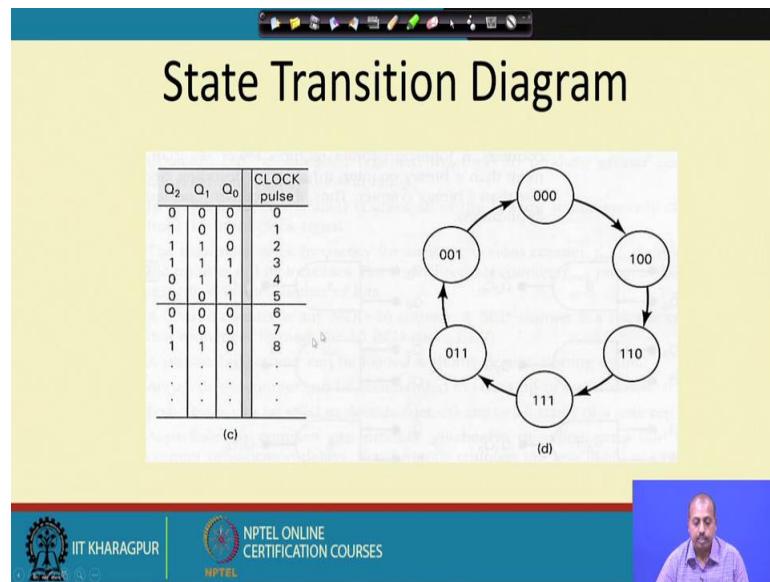
So, that type of, this is known as Johnson counter, it counts in this particular sequence 0, then 4, then 6, then 7, then 3, then 1, then 0.

(Refer Slide Time: 28:30)



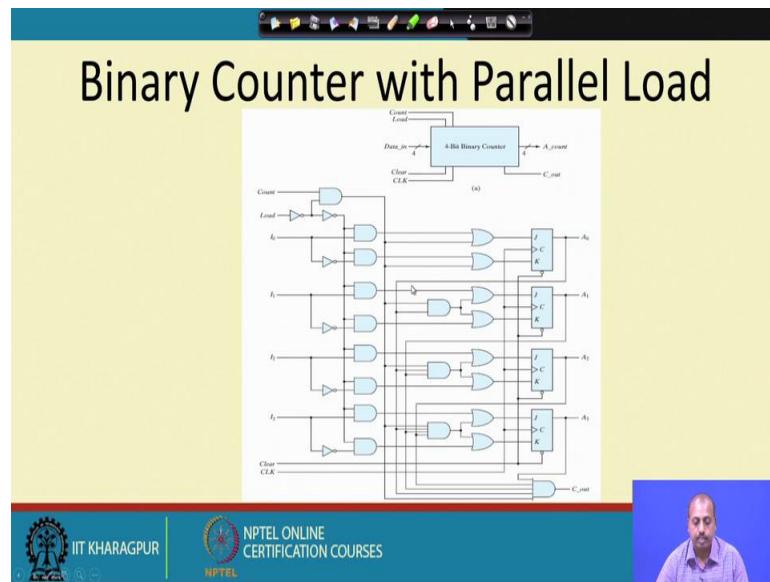
So, this is the corresponding circuit of the Johnson counter. So, you can just trace through this circuit and see that it really generates count in that sequence ok.

(Refer Slide Time: 28:41)



So, the state transition diagram is like this and you can get the corresponding circuitry from there.

(Refer Slide Time: 28:45)



So, finally, in a 4 bit binary counter. So, we can have generic binary counter may have this load facility, count of facility, clear facility which will clear the counter we can load the counter with some initial value and some when this load input is given and this data input is coming. So, this that is loaded into this 4 bit counter and it goes like that. So, this way, we can design a generic counter which has got the parallel load facility also, this is similar to this shift register loading and with that counting facility is added.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 35
Finite State Machine

So, in our course, we will be next looking into another topic, which is Finite State Machine. So, to some extent, we have already seen some finite state machine. As the name suggests, So, this finite state machine, it has got a number of states and the system, it transits through those states as time passes and the number of states that you have in the system is finite, it has got a fixed, finite number of states and that is known a priori. So, you have to design a system that has got a number a known number of states and the transition between the states are also known and accordingly we have to design a circuit.

For example, if we have the talking about a vending machine ok; so, in the vending machine operation initially; so, there are some slots in to which the coins are to be put, after the coins have been put the buyer can select the item that he or she wants to buy and then if the money is sufficient, then the good has to be disposed. So, that goes in a sequence of events.

So whenever we have got this type of application, so they come under the broad heading of finite state machine so, will be looking into some of this finite state machine designs, in this part of the lecture.

(Refer Slide Time: 01:34)

The slide has a yellow background. At the top center, the title 'Finite State Machine' is displayed in a large, bold, black font. Below the title, there is a bulleted list of points:

- An electronic machine which has
 - external inputs
 - externally visible outputs
 - internal state
- Output and next state depend on
 - inputs
 - current state

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, there is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the bar, there is a small video window showing a person's face.

So, a finite state machine is an electronic machine which has some external input externally visible outputs and some internal states. So, output and next state it depends on inputs and current state; so, output and next state. So, both will depend on the input value and currently whatever state the system is in.

(Refer Slide Time: 01:55)

The slide has a yellow background. At the top center, the title 'Abstract Model of FSM' is displayed in a large, bold, black font. Below the title, there is a definition of a machine: 'Machine is $M = (S, I, O, \delta)$ '. To the right of the definition, there is handwritten text: 'S = { s_1, s_2, s_3, s_4 }', 'Output λ : $\lambda(s_i, i_j) = o_k$ ', and ' $\lambda(s_i, i_j) = o_k$ ' with a horizontal line through it. Below the definition, there are four labels with their corresponding descriptions:

- S: Finite set of states
- I: Finite set of inputs
- O: Finite set of outputs
- δ : State transition function

Below these descriptions, there is a bulleted list:

- Next state depends on present input *and* present state

At the bottom of the slide, there is a blue footer bar. On the left side, there are several small icons. In the center, there is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, there is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the bar, there is a small video window showing a person's face.

So, based on that it will depend; so, it is the abstract model of a finite state machine is like this. So, it is a 4 tuple; consider a machine M is A 4 tuple consisting of 4 states 4 quantities S, I, O and λ where S is a finite set of states ok, then the system at any point of time, the

machine at any point of time can be in any of those states. For example, if I say that my state set S is consisting of the state S_1, S_2, S_3 and S_4 ; what it means is that at any point of time, machine will be in one of these 4 states; S_1, S_2, S_3 and S_4 . So, it cannot be in any other state ok. So, they are all excluded.

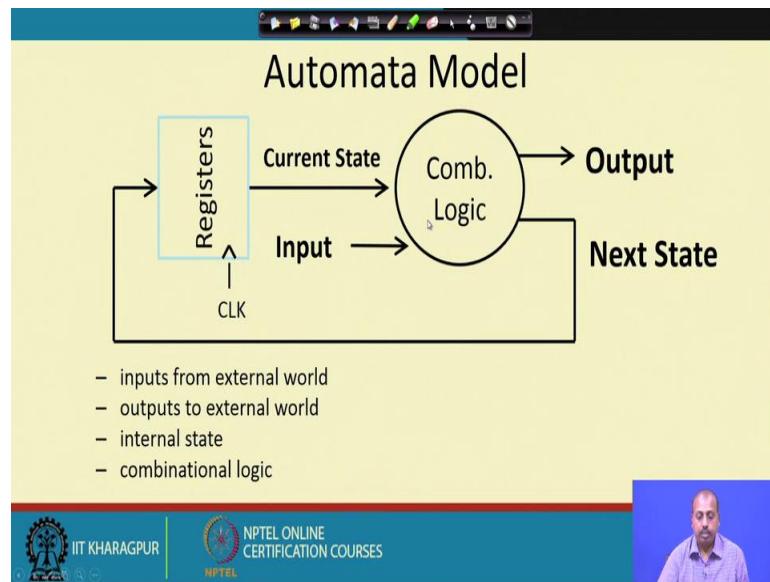
For example, if we if we if your machine is a counter and if you are thinking about a mod 16 counter, then this set S is the set of states 0 to 15. On the other hand, if it is a decade counter, then the set of states S are the values, so, are the states 0 to 9. So, that way depending upon the system, the machine that we are having this set of state will be determining the states in which the system or the machine can be at any point of time.

Then I is the finite set of inputs. So, it interacts the machine interacts with environment accordingly, it get some input from the outside and that is there is a finite set of input, then O is a finite set of output. With respect to that vending machine example, so, when the coins are being put into the system or the user is giving a choice regarding the good that he or she wants to buy, so, they are coming as input to the system and when it is disposing the good by means of activating some switch. So, that is basically the output for that machine. So, you have got this I and O

And this δ so, this is the state transition function. So, that defines how the state is transiting from one state to one state to the next state. So, next state depends on the present input and the present state. So, this state transition; so, this will be this is determined by the present state and the next state of course, there is another function which is not written explicitly, which is the output function. So, output function; so, this is basically it is often represented by λ . So, this λ actually tells. So, lambda over any state say S_i and some input combination say $I_{i,j}$. So, this will be telling us what is the output value say O_k . For all the outputs, so, it will tell you, it will tell me; what is the value of the output.

So, we will come it come to this slowly as we proceed ok.

(Refer Slide Time: 04:56)

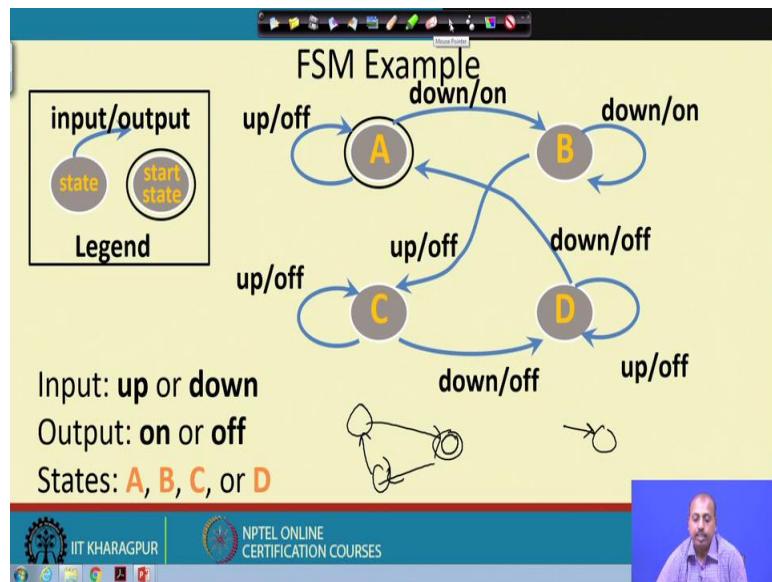


So, so, from the automata theory; so, you can visualize this structure, this machine like this as if we have got some register and this register is clocked and the this register holds the current state of the system. So, this current state comes as input to a combinational logic and the inputs that are there. So, that also come as a combinational logic and this combinational logic. So, it determines what should be the output and also what is the next state.

So, this next state values will be stored again in this register as the current state. At the next clock pulse this next state values will be stored into this register as current state and they will be available as current state. So, this way the system will be go passing through different states and depending upon the input combination. So, it will produce some output also, it will make transitions.

Now, inputs are coming from the external world outputs are also given to the external world, the states are internal and the combinational logic is also there, which is determining these 2 functions.

(Refer Slide Time: 06:04)



So, let us take an example, suppose, we have got a system that has got a switch which may be either up or down and an output which may be either on or off. So, you can think about the electrical switch and a lamp. So, if the switch is in up position switch may be in either the up or down position similarly the lamp may be in on or off situation and the system has got 4 states A, B, C and D. So, when the system is in A state, so, the when the system is in A state, **the**, if you press the switch in the upward direction, then the, the light should be off. So, it is I as long as is the switch is pressed in the, is kept in the up position the light is kept off and it if the system remains in state A.

If you take this switch down ok, then it the light is turned on and if the system makes a transition to state B and as long as the switch is in down position, so, it will be remaining in state B and the light will be turned on. Then if the switch is now put into the off up position, then it comes to state C and puts the light off and as long as in light as long as it is in state C, as long as you are putting this thing switch in the up position. So, it will remain in the off state and from if you put the switch in the down position from C, then the light remains off and similarly, at the D state also, if you put it in the up position the light remains off, if you put it in the down position, then also the light remains off.

So, once you are in state D ok, you see that in all the so, light is always kept off. So, whether you put the switch up or down, then the switch will light will remain off only when you are in going to state A then if you put the switch down, then it will be turned on

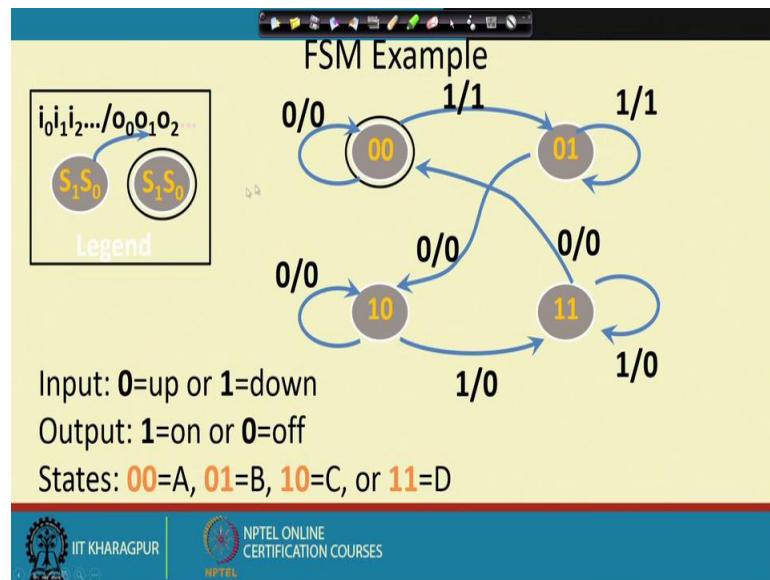
and then after that if your state B then as long as it is down it will be on. So, this is the hypothetical behavior of a system that has got some input switch and some output which can be turned on or off.

And then this input. So, so this is the, an example of finite state machine. So, it has got 4 states A, B, C, D. Now in these notations, there are some symbols ok. So, some we write it like this. So, this state; so, it is normally a state is given like this and if it is put within a it 2 concentric circles like here there is another circle which is surrounding this state name. So, that is the start state ok. So, any start state any arbitrary state transition diagram if you draw then you indicate the start state, if suppose, I have got some arbitrary state transition diagram like this to mean that this is the start states. So, I have to make it like this. So, if there are 2 circles, then that is a start state.

Or you it may be the other symbol is like other symbol sometimes used is like this ok. So, that is the start without any symbol coming without any labeling like that. And whenever a transition is occurring, so, we label it in term like input slash output like. So, everywhere all this transition that you have; so, you see that the labeling is input with, I first write down the input condition that will trigger that transition and then after the slash we write down what is the value of the output signals.

So, that you; so, in this case we have got a single output signal so that is that is why we have got only the one output. So, if there are multiple output signals. So, they can be put here, similarly, if there is a complex input combination condition. So, that will be forming a Boolean expression that can be written as the left part of this, before this slash. So, this way, we can represent a finite state machine by means of some diagram ok. So, mathematically, we can write it in terms of those δ and λ functions in graphical notation, we can draw that diagram and then that also represents the finite state machine.

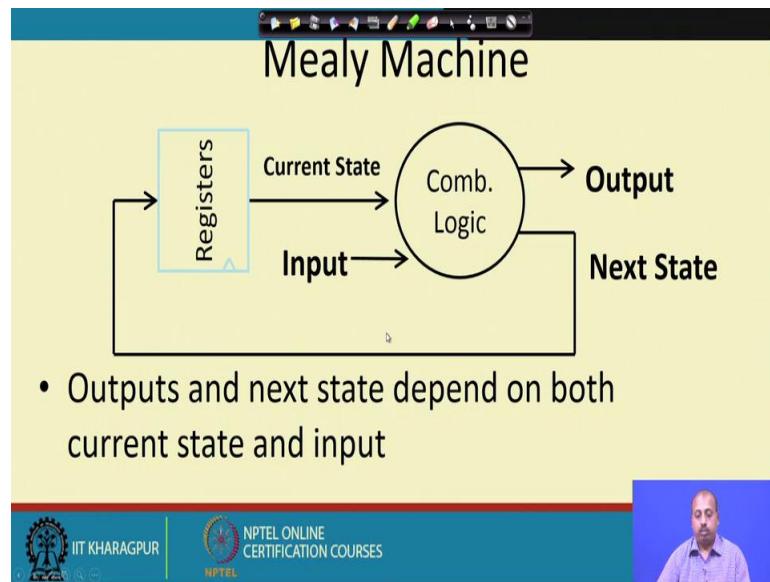
(Refer Slide Time: 10:27)



So, this is a slightly complex example where I have got this i_0, i_1, i_2 , there are 3 input more number of inputs, ok, i_0, i_1, i_2 . So, we can have similarly as I was telling. So, there may be more number of outputs also. So, if this is the input combination. So, this should be the output combination that is the meaning of this labeling similarly for the states. So, instead of showing it by some name; so, you can also show them by means of the state the register content like here, I am writing here 0 0; that means, the state register contains 0 0 and when the state register contains 0 0 if the input is 0 output will be 0 and it will remain in the state 0 0.

Similarly, if the input is 1, then it will make a transition to the such that the state register contain becomes 0 1 and the output is 1. So, it will remain there. So, it goes on like this. So, here this as if this states A, B, C and D, they have been given a 2 bit code, state A is coded as 0 0, B as 0 1, C as 1 0 and D as 1 1. So, this is will come back to this coding again later so, as if these states have been given some name or some code.

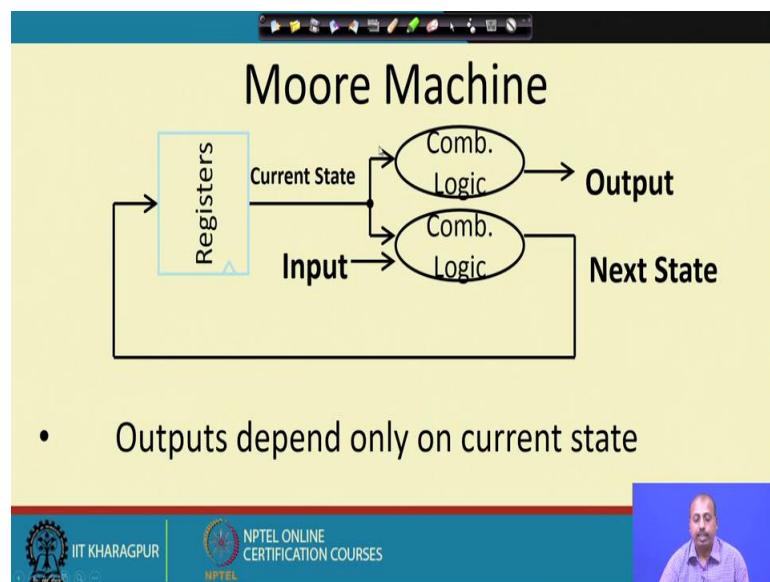
(Refer Slide Time: 11:45)



Now, the type of machine that we were looking so far so, they are known as Mealy machine. So, in case of Mealy machine what happens is that outputs and next state function they depend both of them depend on they depend on the current state as well as the input value. So, this output function and the next state function, they depend on the current state value and the input value. So, this they depend on both the value.

So, this is the standard Mealy machine standard finite state machine that we have seen so far. So, they form the Mealy machine category.

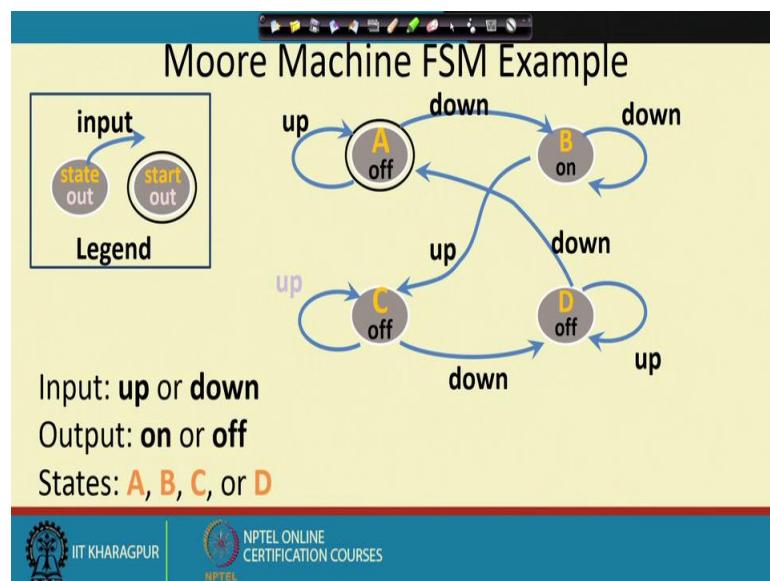
(Refer Slide Time: 12:25)



So, another type of machine that we can think about so, it is Moore machine. So, where the current the, this output depend outputs depend only on the current state. So, the, this input is not determining the current state input is determining only the next state part. So, the current state component and input; so, they are they are determining the next state part whereas, this output is determined only by the current state. The input is not determining the output; output is dependent only on the currents state for this type of configuration. So, it is said to be Moore machine.

So, apparently it seems that the Mealy machine and Moore machine, they are 2 different types of machine. So, may be one of them is more powerful than the other, but that is not the case both the machines are equally powerful. So, whatever computation you want to represent using Mealy machine, so, you can do it using Moore machine and vice versa, but the circuit wise of course, this since this Moore machine the output is dependent only on the current state, it may so happen that it has got more number of states because in case of Mealy machine what will happen is from a particular state for different input combination, so, you can have different outputs. But, for Moore machine that is not possible. So, for all those different output combination; so, maybe we have to have separate states.

(Refer Slide Time: 13:56)



So, that makes Moore machine to have more number of states. So, here an example of this finite Moore machine based FSM. So, here you see that individual states. So, from the edge level; so, we have dropped the output because output is not dependent on the input,

it is dependent on the current state only. So, in state A, the output is always off, at state B, it is always on, similarly, at state C and state D also they are always off. So, this is an example of Moore machine.

Whereas for Mealy machine, so, this is the standard example that we have previously seen. So, output is dependent on the current state as well as the input combination for example, if the current state is A input is up then only the output is off, whereas, the current state A, if the input is down then the output is on. So, that way output is dependent on the current state as well as input which is not the case for the Moore machine.

(Refer Slide Time: 14:58)

Create a Logic Circuit for a Serial Adder

- Add two infinite input bit streams
 - streams are sent with least-significant-bit (lsb) first

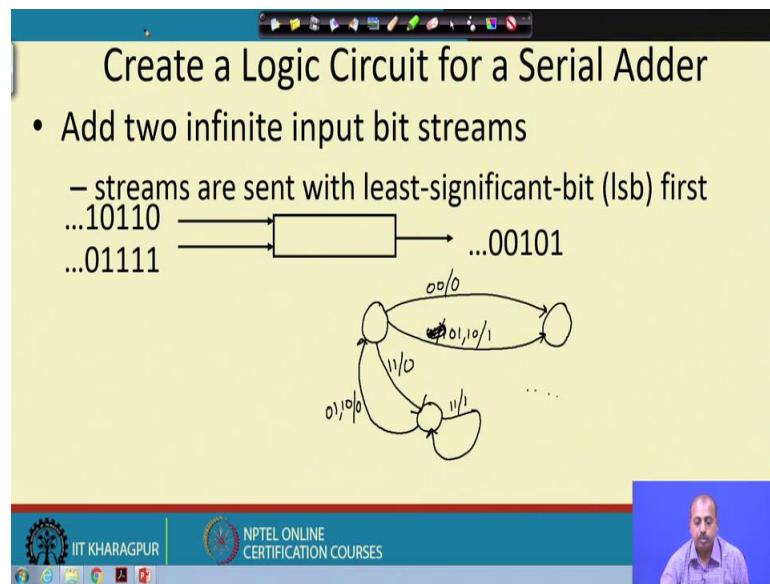
The diagram illustrates a serial adder circuit. It shows two infinite bit streams being added: ...10110 and ...01111. These streams are sent with the least-significant-bit (lsb) first. The streams are input into a full adder. The adder has three outputs: Sum (labeled 00/0), Carry (labeled 01/0/1), and another Sum output (labeled 00/0). The carry from one bit position is passed as the least significant bit to the next bit position's addition.

So, we can try to take some examples to see and understand how this finite state machines can be designed. So, the simple finite state machine that we can think about suppose we are adding 2 infinite input bit streams. So, this is some adder. So, what it does is that it there are 2 infinite bit stream, accordingly, it does this addition. So, it at first these 2 bits; 0 and 1 comes here as a result. So, these 2 bits are added the result is 1. So, it is outputting this one, then this one and one comes. So, sum is 0. So, it is outputting this 0 here and there is 1 carry. So, when the next 2 bits are coming. So, this 1 carry is also taken into consideration as well as a result. So, this bit becomes 1 and then this generates 1 carry. So, this carry is taken to the next bit addition and that is done by 0 that is result is 0.

But the point is that it this at any point of time it just adds 2 bits. So, it remembers what was the result of this previous addition whether it had generated a carry or not and based

on that it is producing the next result. So, if we try to do this do realize this finite state machine. So, it is like this. So, if this is the initial state, then we can say that if the next input combination is say 0 0, then the output is 0 and it goes to a state where the carry was equal to 0, whereas, if the input was say 0 1 or 1 0 in both the cases, output is equal to 1 and it comes to a new state or we can do it like this we can make it slightly better actually we need to distinguish between the 2 cases in one case the carry is generated.

(Refer Slide Time: 17:01)



And in the other case, the carry is not generated. So, so, from this, I can say if it is 0 0, the output should be 0 and it goes to this state if it is say 0 1 or 1 0, if it is 0 1 or 1 0, in those cases, the output should be equal to 1 and it goes to that state ok.

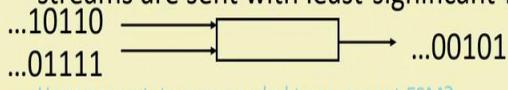
So, in this state, you see that what we have got is the output there, there was no carry generated. Now if in from this state if the input pattern was 1 1, two bits was 1 1, then the result should be 0 and it comes to a state where it remembers that a carry was generated, there was a carry now from this state if the next 2 bits are 0 0, then it has to output a 1 because the carry was generated and we don't have any carry. So, we can say that if the next bit is 0 1 or 1 0 in that case, it has to output a 0 and it goes back to this state on the other hand, if this state is from this state if it is if the next 2 bits are again 1 1, then it has to output a, if the 2 bits are 1, then it has to output a 1 and it should go to a state where it has to generate a carry again.

So, in this way, we can complete this diagram and it will turn out to be a finite state machine by looking into all the combination. So, essentially from a particular state, you have to see the next 2 inputs and accordingly go to a go to the corresponding B. So, this design is slightly complex. So, let's look into another simpler example for our understanding.

(Refer Slide Time: 19:04)

Create a Logic Circuit for a Serial Adder

- Add two infinite input bit streams
 - streams are sent with least-significant-bit (lsb) first



– How many states are needed to represent FSM?

– Draw and Fill in FSM diagram

Strategy:

- (1) Draw a state diagram (e.g., Mealy Machine)
- (2) Write output and next-state tables
- (3) Encode states, inputs, and outputs as bits
- (4) Determine logic equations for next state and outputs

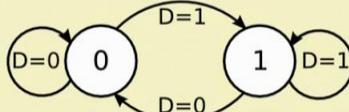
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, this is the exercise. So, I am leaving it for you. So, you just do it.

(Refer Slide Time: 19:12)

Another look at D latch/flip-flop



q_{old}	D	q_{new}
0	0	0
0	1	1
1	0	0
1	1	1

This is an example of a **state diagram**
more specifically a **Moore** machine

$q_{new} = D$

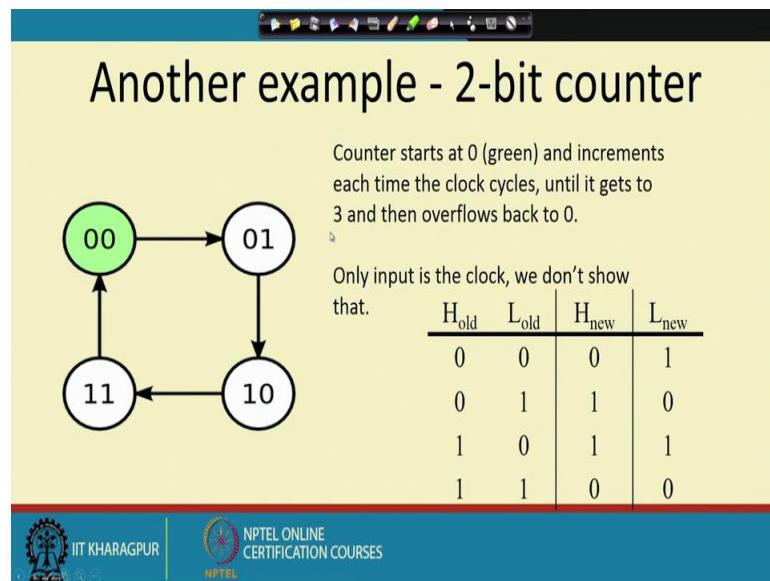
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, let us look into the D flip flop; how does it behave or D latch or D flip flop. So, in terms of finite state machine; so, you see that this is a Moore. So, if I say that at present the Q output is 0, then if the D is equal to 0, then it will remain in these state and output will remain as 0 if D equal to 1, it will come to a state where the output is 1 and as long as D is equal to 1, it will remain in this state when D becomes equal to 0, it comes to this. So, this is the finite state machine that represents the behavior of D latch or D flip flop and this is the Moore machine because output is determined by the state ok.

So, this in state 0 output is 0 and in state one output is 1 and the, if you do a minimization you find that $q_{\text{new}} = D$.

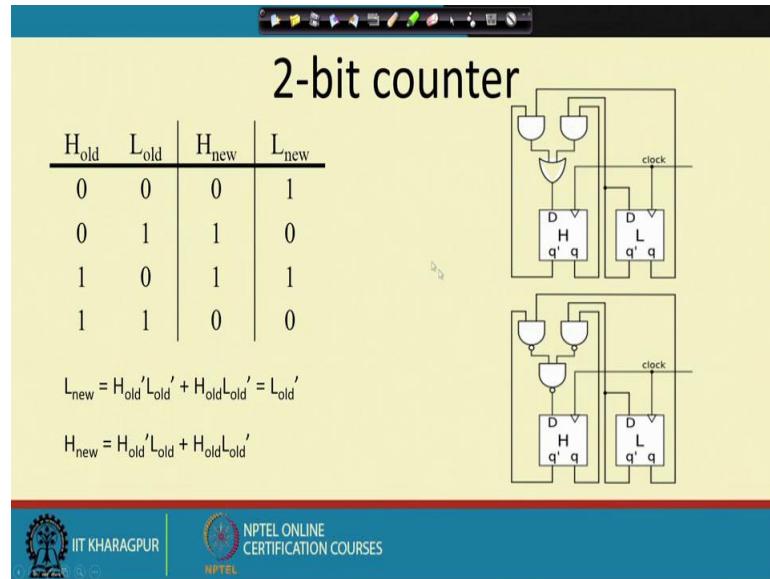
(Refer Slide Time: 20:07)



So, another example a 2 bit counter. So, the counter starts at 0 and at every clock pulse it increments by one. So, first it outputs 0 0, then at the next clock pulse it comes to 0 1, next clock pulse it comes to 1 0, then 1 1, then 0 0, again, this is a Moore machine because the output is determined by the current state. So, this is 0 0 or 0 1, etcetera.

So, if you want to design the corresponding circuit, then what we have to do is we have to find the expression for H_{new} and L_{new} and accordingly you can design the circuit

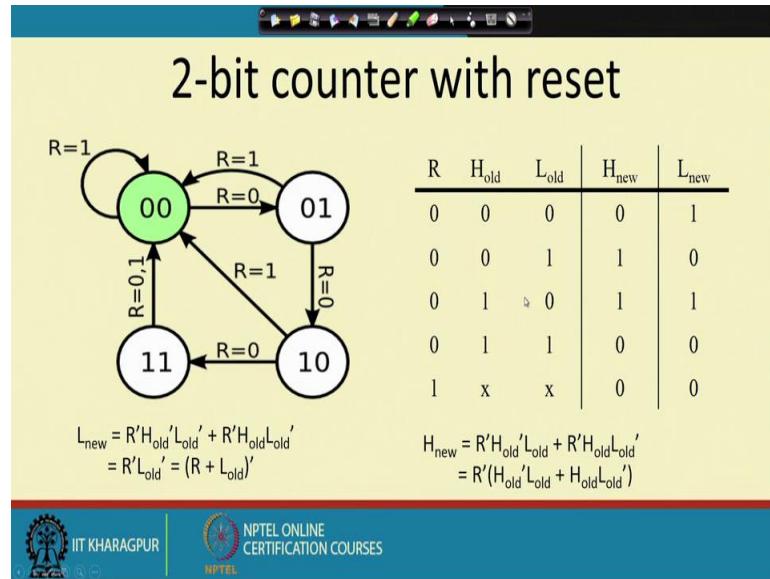
(Refer Slide Time: 20:44)



So, if you derive then H_{new} , L_{new} is given by this expression which is $L_{new} = (H_{old}'L_{old} + H_{old}L_{old}')$ and $H_{new} = R'(H_{old}'L_{old} + H_{old}L_{old}')$. So, essentially what you have to do is that you have to take this H and L, 2 flip flops and then we can do this minimization that is $L_{new} = L_{old}'$. So, this \bar{Q} line is connected here and it is done like this and this one your $H_{new} = R'(H_{old}'L_{old} + H_{old}L_{old}')$. So, this is AND OR gate based realization this is NAND gate based realization otherwise it is same.

So, this way, our finite state machine is this one and from there, we could come to a circuit which realizes the finite state machine.

(Refer Slide Time: 21:41)

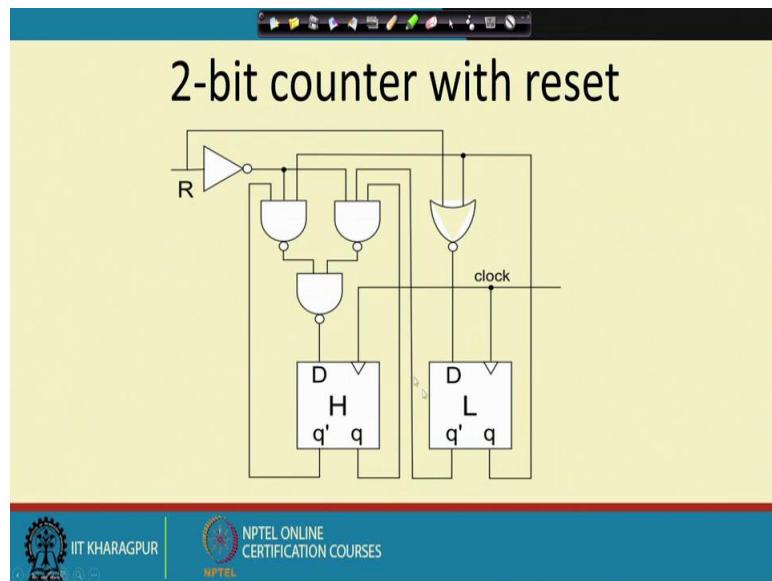


Now, if that counter, that we are talk talking about it has got a reset input like here this counter has got a reset input. So, here this if reset equals to 0, then it behaves in the normal fashion, but whenever this reset input is equal to one whatever be the state in which the counter is, so, it will come to this state 0 0 ok. So, it will be outputting 0 0. So, that is captured here. So, you see whenever R is equal to 1 whatever be the values of this H_{old} and L_{old}, H_{new} and L_{new} will be 0.

Similarly, when R equal to 0 it behaves as a normal counter and it goes like this. So,

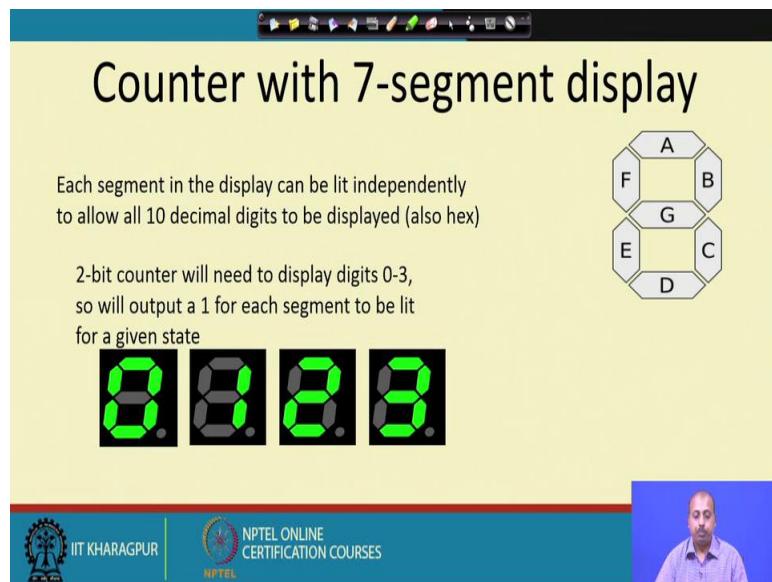
$$L_{\text{new}} = (R+L_{\text{old}})'; \text{ and } H_{\text{new}} = R'(H_{\text{old}}'L_{\text{old}} + H_{\text{old}}L_{\text{old}}')$$

(Refer Slide Time: 22:29)



So, we can make the corresponding circuit. So, this is the corresponding circuit which realizes the 2 bit counter with reset.

(Refer Slide Time: 22:37)



Next example that will look into is a counter with 7 segment display. So, we have got a seven segment display with A, B, C, D, E, F, G and each segment in the display can be lit independently to allow all 10 decimal digits to be displayed ok. So, this is 0, 1, 2, 3 like that and there will be a 2 bit counter will need to display the digits 0 to 3.

So, will output a 1 for each segment to be lit for a given state so, for 0; so, this is for 0. So, this will be output it will turn on this segment for one it will turn on these 2 segment for 2,

it will turn on these segment. So, this way will go in a fashion like 0, 1, 2, 3, 0, 1, 2, 3 like that.

(Refer Slide Time: 23:23)

Counter with output functions												
R	H _o	L _o	H _n	L _n	A	B	C	D	E	F	G	
0	0	0	0	1	1	1	1	1	1	1	0	
0	0	1	1	0	0	1	1	0	0	0	0	
0	1	0	1	1	1	1	0	1	1	0	1	
0	1	1	0	0	1	1	1	1	0	0	1	
1	x	x	0	0	0	0	0	0	0	0	0	

$$A = D = R'H_o'L_o' + R'H_oL_o + R'H_oL_o = R'(H_o'L_o)' \quad B = R' \quad C = R'(H_oL_o)'$$

$$E = R'L_o' \quad F = R'H_o'L_o' = (R + H_o + L_o)' \quad G = R'H_o$$

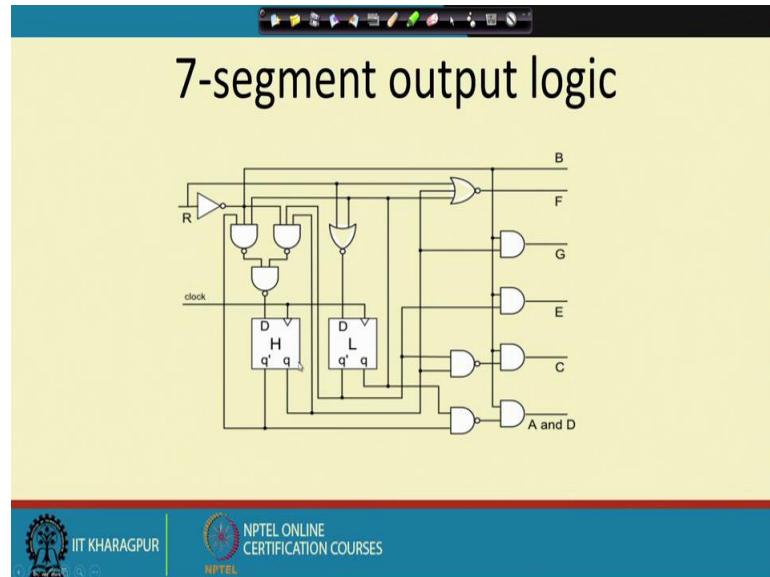


So, from 0 0 from 0 0 so, it has to go to 0 1 from 0 1 it has to go. So, this is the state transition from 0 to 1, 1 to 2, 2 to 3, 3 to 0 and if this reset is 1, then it does not matter whatever be the state. So, it will come to the state 0 0.

Now, we have to see what is the pattern for this 7 segment display A, B, C, D, E, F, G. Now for displaying one, we know that for displaying this G segments for displaying this 0. So, only this G segment should be off and all other segment should be 1. So, that is done here. So, at the current state where 0 0 , it is turning on A, B, C, D, E, F and G is equal to 0. Next at the state 0 1, so, this we have to display 1, accordingly, segments B and C, they should be turned on and the rest of the segments they should be turned off or put 0.

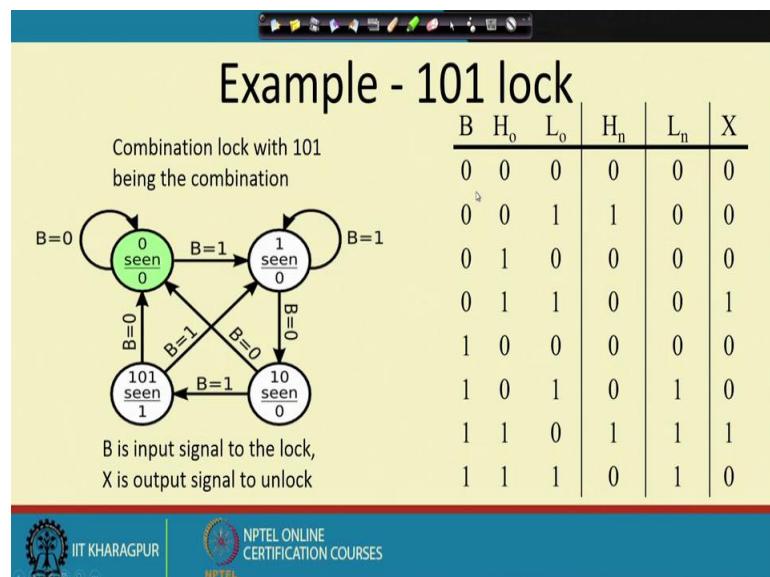
So, this way we can write down the truth table for this A, B, C, D, E, F, G and then do a minimization. So, A becomes A and D are their similar and this is the expression for $A = D = R'H_o'L_o' + R'H_oL_o + R'H_oL_o$, $B = R'$, $C = R'(H_o'L_o)'$ and $E = R'L_o'$. So, this way, you can have this all the expression for this seven segments ok.

(Refer Slide Time: 24:48)



And they can be fed to the individual logic. So, we can have this 7 segment display and this A, B, C, D, E, F, G, those lines are connected to the seven segments. And this counters, so, once it is reset, so, this counter values will become 0 0 from that point onwards. So, it can go on getting the individual values, it can go on getting the individual, individual values.

(Refer Slide Time: 25:13)



Another example that we look into is 1 0 1 lock. So, this 1 0 1 lock is like this. So, we have got this. Whenever this pattern 1 0 1 is observed, so, the lock is said to be open. So, it starts

in a state where we have seen only a 0. Now B equal to 0. So, this is the start state. So, if I if I as long as we see B equal to 0. So, it is we have seen is 0. B equal to 1, so, it comes to a state, which is which we call 1 seen and the output is 0, then if B equal to 0, it comes to the state 1 0 seen and the output is 0 and B equal to 1, it is 1 0 1 seen and this output becomes equal to 1 again it has to look for this. So, 1 0 1; so, again it is 1 seen. So, it is comes to output is 0, it comes to 1 seen, then 1 0. So, like that this finite state machine is made.

So, accordingly, you can draw the corresponding truth table and get this signals realized.

(Refer Slide Time: 26:18)

101 combination lock

$$X = H_o L_o$$

$$H_n = B'H_o'L_o + BH_oL_o'$$

$$\begin{aligned} L_n &= BH_o'L_o + BH_oL_o' + BH_oL_o \\ &= BH_o'L_o + BH_oL_o + BH_oL_o' + BH_oL_o \\ &= BL_o + BH_o \end{aligned}$$

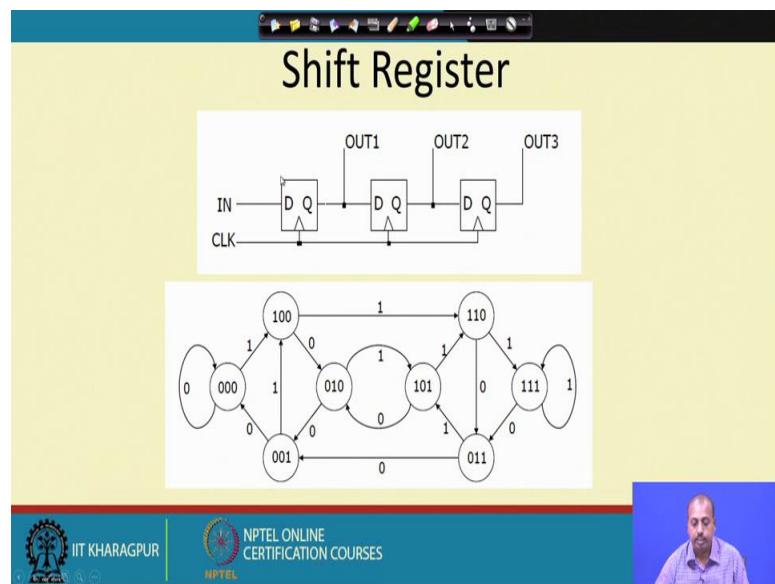
↳


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

So, this 1 0 1 combinational lock can be realized by these functions.

(Refer Slide Time: 26:26)



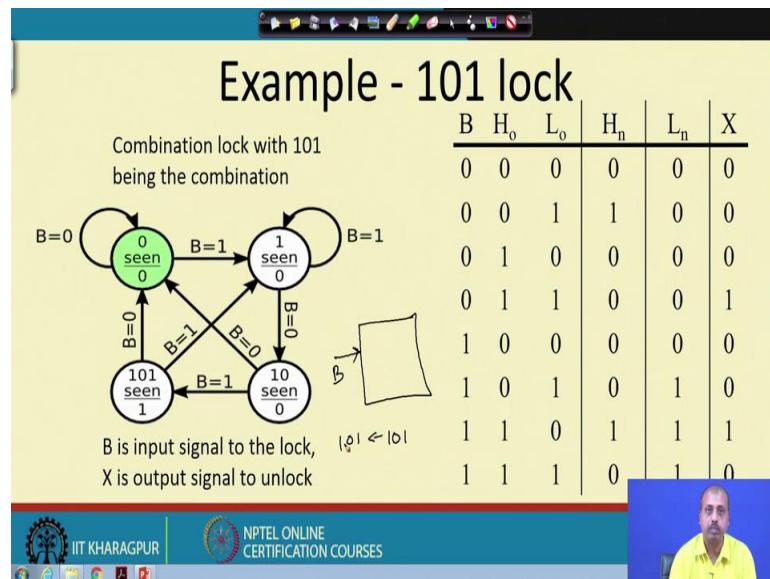
And then it can be converted into a, into some circuit for getting the thing.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 36
Finite State Machine
(Contd.)

So this is the next example that we consider of Finite State Machine. So, in this case we are considering a lock that has got a digital input to unlock it and the input is 101.

(Refer Slide Time: 00:22)



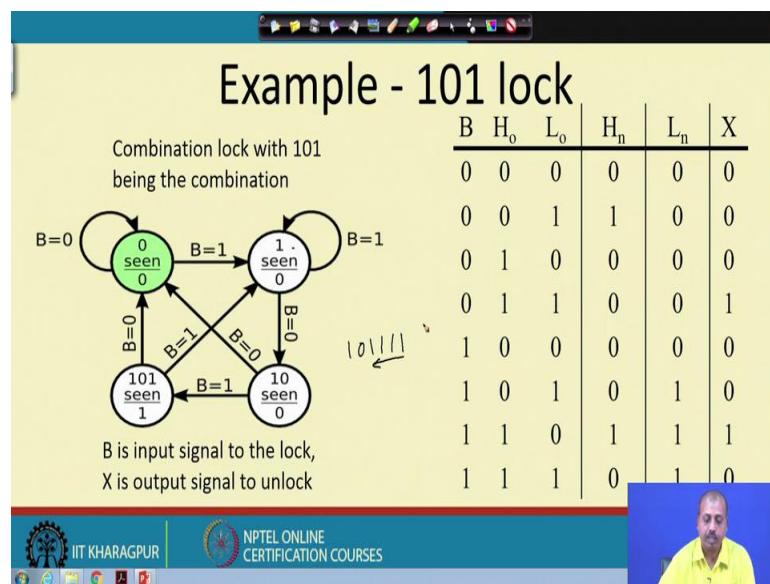
So, that is there is a single input to this lock. So, conceptually you can think of it as a lock like this. And, it has got a B input and this B input is serial like. So, if it sees the sequence 1 0 1, then the lock will be opened. And, once so, every 101 sequence is different like say if I press this sequence 101, then the lock is open.

But, after this lock is opened so, it will go to a lock close mode. And, after that it will require another explicit 101 sequence for turning it on or opening the lock. So, we will see that. So, the state transition diagram that we have is something like this. So, initially we are at in this the first state, where you see that we can say we have seen a 0. The as long as we see 0 we stay in this state. So, we will let us call the name of the state as 0 seen and then when it is in the state 0 seen the output is 0. So, lock is not opened.

So, as long as we are getting B equal to 0 so, we continue in this state. Now, B equal to 1. So, this come it takes us to a state which we name as one seen, that is our lock unlocking pattern is 101. So, we consider that we have seen the first one in that sequence. So, then also the output remains 0 that is the lock is not yet opened. And as long as B remains equal to 1 so, we continue to be in this state.

So, you see that if the sequence may be like this somebody may be entering values the 1 1 1 1 then 0 1.

(Refer Slide Time: 02:06)



So, as long as these ones are going so, it will remain in this state 1 seen. And, then after that when this and the output remains 0. Now, if a B if the B input becomes 0 after that so, it comes to a state which we call 1 0 seen. So, in this 1 0 seen state also the output remains at 0 the lock is closed and then if we get a 1 then it is 1 0 1 seen.

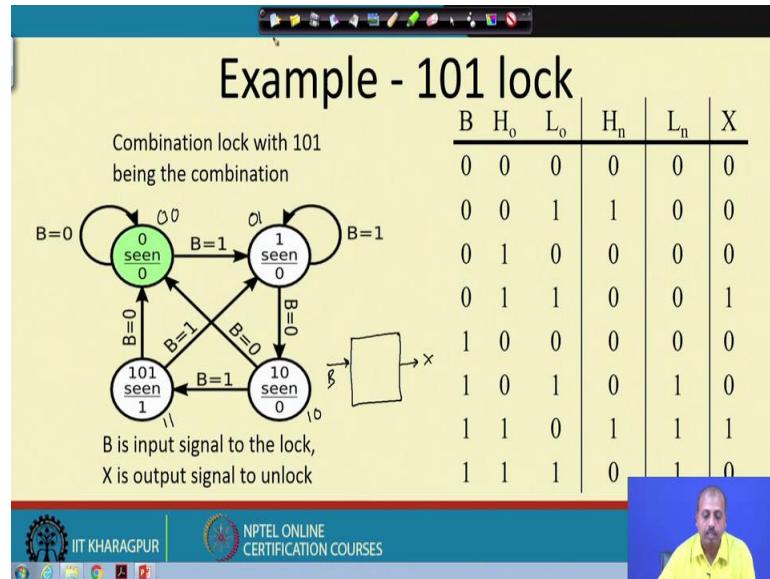
And, when the 1 0 1 seen then the lock output is 1 that is lock is opened. Then, after if you if we get a 0 then it goes to 0 seen state that is lock becomes closed and we are we are in the state as if we have seen a 0 and if B equal to 1. So, this is a new unlocking sequence that I was talking about. So, B equal to 1. So, it takes us to this state.

So, again you need another explicit 1 0 1 for unlocking the unlock the lock. So, now if you press B equal to 1 or B equal to 0 the lock becomes closed. So, lock is; so, we can say

that the as the lock is closed. And then this again another 1 0 1 input sequence is necessary for unlocking the lock.

So, B is the input signal to the lock and x is the output signal of the lock.

(Refer Slide Time: 03:27)



So, you can say that box that we had drawn the lock box. So, B is the input and X is the output. So, whenever we see 1 0 1 in B the lock is turned open. So, we can say that the state transition can be like this. They see if we have if we name these states first state as a 0, the code of the state is or if we make that the code of the state is 0 0 only.

So, this first state it is coded as 0 0, if B equal to 0 the next state is also 0 0 and X remains equal to 0. If, B equal to 0 and the current state is say 0 1, then the next state becomes 1 0 ok. So, if we are at state 1 so, this is this we code as 1. So, we will come to this coding part later. So, there are 4 states. So, in this particular case it is assumed that this state is coded as 0 0 this state is named as 0 1 this is 1 0 and this is as 1 1.

So, if the input state bits are 0 0 and the input is also 0 0, the next states are also 0 0. So, this is the first transition second transition says that if the current state is 0 1 that is this 1 and the input is 0, then it remains in this state only. So, sorry it remains in this state only. So, it sorry no it goes to the state 2 so, 1 0. So, if, if this is 0 1 and B is 0, then it is going to the state 1 0 and the output remains 0. So, that way we can make this full table. And,

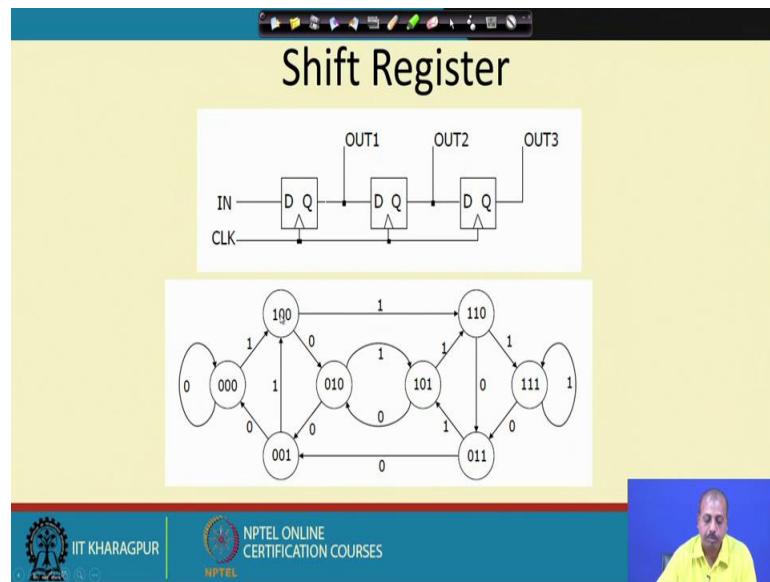
once this table has been made so, we can come, we can do a simplification and get the next the combinational logic for doing this thing.

(Refer Slide Time: 05:12)

So, this $X = H_0L_0$ and this H_n is given by $H_n = B'H_0'L_0 + BH_0L_0'$ and L_n is given by this function. So, we can just make a combinational logic and we can say that we can we can put this combinational logic into some part of the circuit. So, this is the combinational logic. So, it has got B as a 1 input, then H_0 and L_0 as a other 2 inputs and then we have got these 2 flip flops, which holds the H and L which are L_H and L_L values. So, this H next and L next so, they come here and they are fed to let us name this as a H_0 and this as L_0 .

So, this L_0 so, H_0 is fed here and L_0 is fed from this point. And, then this is our X output. So, in this part so, we realize the function for $X = H_0L_0$, then $H_n = B'H_0'L_0 + BH_0L_0'$, and $L_n = BL_0 + BH_0$ this individual functions are realized in this part. So, I am not drawing the combinational part there, but it can be realized using that.

(Refer Slide Time: 06:35)



Next we will be looking into another example, which is a shift register. So, this is the standard shift register that we have. So, as you know that with the input of every clock pulse this input bit gets shifted through this D flip flop chain.

Now, once this input pattern is known. So, you can try to draw the corresponding finite state machine. Now, we really do not know like what is the initial state of this 3 flip flops. So, it can be any of the states 0 to 7. So, if we draw so, if we assume that the initial state is 0 0 0, then if the on the arrival of the next clock with that at that point of time is in equal to 0. So, it remains in this state, if in equal to 1 it goes to this state 1 0 0.

Similarly, if the if the current state is 1 0 0 and the next input is 0, then it comes to the state 0 1 0 because this one gets shifted here and the 0 from the input comes to the first flip flop. So, this way you can complete this total finite state machine which we will so, from the circuit. So, this is this is the other way. So, from the circuit we can come to the finite state machine. So, here you know other combinational logic will be necessary, because these if you simplify this particular this particular machine for the state transition function, and output function you will find that it gives rise to the function. So, it will come in a simple fashion like that.

So, that is why we do not show the circuit separately, but this is nothing, but this connection.

(Refer Slide Time: 08:15)

The slide is titled "FSM design procedure". It lists the following steps:

- Describe FSM behavior, e.g. state diagram
 - Inputs and Outputs
 - States (symbolic)
 - State transitions
- State diagram to state transition table, i.e. truth table
 - Inputs: inputs and current state
 - Outputs: outputs and next state
- State encoding
 - decide on representation of states
 - lots of choices
- Implementation
 - flip-flop for each state bit
 - synthesize combinational logic from encoded state table

On the right side of the slide, there is a hand-drawn state transition diagram showing four states: S0, S1, S2, and S3. Transitions are labeled with binary codes: S0 to S1 on input 00, S1 to S2 on input 01, S2 to S3 on input 10, S3 to S0 on input 11, S0 to S2 on input 01, S1 to S0 on input 11, S2 to S1 on input 00, and S3 to S2 on input 01. Below the diagram is a truth table:

Current State	Input	Output	Next State
S0	00	01	S1
S1	01	11	S0
S2	10	00	S3
S3	11	01	S2

Handwritten notes include:
2-bit coding
 2^2
N no. of states
Min. no. of bits = $\lceil \log_2 n \rceil = n$
 $n!$
 2^n

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player showing a professor speaking.

Now, how do you design an FSM finite state machine? So, first thing that we have by this time we know is the description of the finite state machine behavior. So, which is done in terms of state diagram? So, we can have some specification of a system in some English like language, and then we convert that description, that English language description into a finite state machine. And, that finite state machine has got some inputs and outputs there will be some symbolic steps and there will be state transitions.

So, we can check whether this description that the finite state machine that we have drawn, it really reflects the a finite state machine behavior that was given as input and that check can be manual that check can be based on some formal methods. So, which is beyond the scope of this course, but that verification can be done? Now, once we are satisfied that our state transition diagram whatever we have drawn, truly reflects the behavior that that has been asked for, so, we go to the next step which is a state diagram to state transition table.

So, you where inputs are the current state and the primary input so, they constitute the inputs and the outputs are the primary outputs and the next state. Now, once this from state diagram to state transition table we have drawn. So, next thing is the state encoding. So, every state is given some binary code ok. So, we naturally we need to decide like if there are say 4 binary were 4 states. So, if there are say 4 states say symbolically we represent it as S0, S1, S2 and S3.

Now, for 4 states I will need 2 bit code to the at least a code of at least 2 bit 2 bit length to distinguish them. So, somebody may code it like this that the state S0 is 0 0, S1 is 0 1, S2 is 1 0 and S3 is 1 1. Now, there are other choices also like you see that I somebody may code it like this one as 0 1 this is this is 1 1 this is 0 0 and this is 0 1.

So, there is nothing there is no restriction regarding the choice that you make for the states. Somebody may even do this coding of higher number of bits like in this case so, if I do a 2 bit coding; so if I do a 2 bit coding then you see so, I can have 4 choices. So, it is so, the total number of alternatives that we have like this is alternative 1 this is alternative 2. So, then the number of alternatives that we have is $2^2!$ of that, because this is I can just this 4 values.

So, I can arrange in any order. So, any permutation of those the 4 states that is good enough. So, it can give $4!$ alternatives. And, if there are say N number of states. So, N is the number of states then the minimum number of bits. So, minimum number of bits needed is going to be $\log_2 N$ so, these and the ceiling of that.

So, this is the value of N that we are talking about so, this is the so, this is the small n number of state bits needed. And so, if I do so, so, this is the minimum number. So, there is no restriction on the upper limit. So, we can go for any number of bits which is more than small n ok, but if I choose this small n number of bits then the number of alternatives that I have is $2^N!$ So, you see this is a huge search space that is there and we for each of these combinations. So, it is very much likely that the corresponding circuit that you get will be different.

So, so, basically if we say, that my so, this is the finite state machine and we have got these flip flops ok. So, this is that combinational logic that we have so, that takes the primary input as in primary inputs and it produces the primary outputs, plus it gets lines from these flip flops the next state bits or the present state bits. It gets the present state bits from here and it produces the next state bits sorry let me draw a clear diagram.

(Refer Slide Time: 12:50)

FSM design procedure

- Describe FSM behavior, e.g. state diagram
 - Inputs and Outputs
 - States (symbolic)
 - State transitions
- State diagram to state transition table, i.e. truth table
 - Inputs: inputs and current state
 - Outputs: outputs and next state
- State encoding
 - decide on representation of states
 - lots of choices ✓
- Implementation
 - flip-flop for each state bit
 - synthesize combinational logic from encoded state table

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this is the combinational logic and we have got the flip flops fine. So, the primary inputs are connected to this combinational logic. Similarly, it produces the primary output and these flip flops contain, so, they constitute the present state. So, they constitute the present state and it computes this combinational logic will compute the next state like this. So, this is a standard technique that we have seen. Now, you see that depending upon the coding like individual states what code we give this combinational logic is going to vary.

So, what is the best possible state assignment in terms of maybe the number of gates in the combinational logic or the power consumption of the combinational logic and these or maybe some other factor. So, that is a difficult problem. So, we there are lots of research works that have come up on this particular problem, which is known as the state encoding problem, which you come back to this later. So, I just I wanted to emphasize on this point that there are lots of choices in the state encoding process.

The next part is the implementation part, now these flip flops that I have said. So, I did not explicitly mention which type of flip flop we are using. So, normally while we are doing the design we use D type of flip flop. So, this is the D type of flip flop D and Q. So, D is getting this next state bit from the combinational logic and giving the present state output to the combinational logic.

Now, it is not necessary that we have to use D flip flop only somebody may say I will use a J K flip flop. So, I will use a J K flip flop, accordingly 2 lines should come from the

combinational logic to determine the J input and K input for the, for the flip flops. Now, not just so, that is 1 reason that a why J K is generally not used, because of this reason that when you need to compute this combinational logic that we have so, they need to compute 2 functions per flip flop, one for the J input one for the K input, whereas if the flip flop is D, then the next state logic needs to compute only 1 function for a, for a flip flop.

So, as a result it is, it may be easier that we do the realization using D, but that is not always true it because J K type of flip flop they offer other transition probabilities or the transition properties. So, that way may be for some finite state machine J K is doing better or maybe T type of flip flop may be used instead of this D flip flop.

However, in general in whatever designs that we see so, they use D flip flop because of it's simplicity you can say. And the because of this combinational logic design and the cad tools that have been designed. So, they specifically target it as D flip flop. So, that is why D flip flop has remained as the standard one. Then after this is done so, if after these flip flops have been chosen. So, we synthesize combinational logic from encoded state table. So, that way we do the full synthesis. So, this is the full synthesis process. So, we will take an example and explain like how this is taking place?

(Refer Slide Time: 16:21)

Synthesis Example

- Implement simple count sequence: 000, 010, 011, 101, 110
 $000 \rightarrow 010 \rightarrow 101 \rightarrow 011 \rightarrow 110$
- Derive the state transition table from the state transition diagram

Present State	Next State		
	C+	B+	A+
0 0 0	x	x	x
0 0 1	x	x	x
0 1 0	1	0	1
0 1 1	1	1	0
1 0 0	0	1	0
1 0 1	0	1	1
1 1 0	1	0	0
1 1 1	x	x	x

note the don't care conditions that arise from the unused state codes

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Suppose, we want to implement a counter that counts in this sequence 0 0 0, 0 1 0, 0 1 1, 1 0 1, 1 1 0 and after that it comes back to 000. So, the first point is to derive the state transition table, first step is basically the state transition diagram. So, in this case it is

obvious. So, it starts at, wait, sorry this diagram is something; this sequence is something wrong, I am sorry. So, this sequence is not correct the sequence, that it is doing is basically if you start at 1 0 0 then it goes to 0 1 0, then it goes to 1 0 1 then 0 1 1, 0 1 1 and then 1 1 0 and then it is coming back to 100.

So, that is the sequence that it is doing. So, this is so, derivation of the state transition function is like this that if the present state is 0 0 0, then the next state is don't care. So, the next state for 0 0 0 the next state is don't care, similarly 0 0 1 also the next state is don't care. So, what happens if the counter reaches any of these states? So, that is not an issue with the designer. So, they are that is kept as don't, don't care.

If you start at 0 1 0 then the next state should be 1 0 1. So, this way the present state next state relationship is done. So, don't care conditions that arise from unused state codes. So, that is basically ignored. So, this way we can do this we can do this state transition table part for this example.

(Refer Slide Time: 18:19)

Don't cares in FSMs (cont'd)

- Synthesize logic for next state functions derive input equations for flip-flops

C^+	B^+	A^+																																					
<table border="1" style="border-collapse: collapse; width: 100px; height: 40px;"> <tr><td>X</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>A</td><td>X</td><td>1</td><td>X</td></tr> <tr><td colspan="4" style="text-align: center; border-top: none;">B</td></tr> </table>	X	1	1	0	A	X	1	X	B				<table border="1" style="border-collapse: collapse; width: 100px; height: 40px;"> <tr><td>X</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>A</td><td>X</td><td>1</td><td>X</td></tr> <tr><td colspan="4" style="text-align: center; border-top: none;">B</td></tr> </table>	X	0	0	1	A	X	1	X	B				<table border="1" style="border-collapse: collapse; width: 100px; height: 40px;"> <tr><td>X</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>A</td><td>X</td><td>0</td><td>X</td></tr> <tr><td colspan="4" style="text-align: center; border-top: none;">B</td></tr> </table>	X	1	0	0	A	X	0	X	B				
X	1	1	0																																				
A	X	1	X																																				
B																																							
X	0	0	1																																				
A	X	1	X																																				
B																																							
X	1	0	0																																				
A	X	0	X																																				
B																																							

$$| \delta | \leftrightarrow | \delta |$$

$$C^+ = B$$

$$B^+ = A + B'C$$

$$A^+ = A'C' + AC$$

Now, after that is done. So, we can synthesize the logic for next state functions and derive the input equations for flip flops. So, that way you make the truth table from this you make the truth table, from this graph, from this from this next state table the state transition table.

So, you can make the truth table for C^+ , B^+ and A^+ . And, this is the functionality that we get $C^+ = B$, $B^+ = A + \bar{B}C$, $A^+ = \bar{A}\bar{C} + AC$ and then we can go for realizing the circuit.

(Refer Slide Time: 18:52)

Self-starting FSMs

- Start-up states
 - at power-up, FSM may be in an used or invalid state
 - design must guarantee that it (eventually) enters a valid state
- Self-starting solution
 - design FSM so that all the invalid states eventually transition to a valid state may limit exploitation of don't cares

implementation on previous slide

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Sometimes we are looking for some self-starting FSMs. So, self-starting FSMs so, they have got some startup states so, at power up. So, FSM maybe in an used or invalid state, where design must guarantee that it eventually enters into a valid state.

So, this is self-starting FSMs design the FSM. So, that all invalid states they eventually make a transition to the valid state and this way this limits the exploitation of don't care like say. So, we previously we had say this, while we are writing this is 000. So, we took the next state as don't care, but here that we cannot take so, 000 so, we put it as if you if you start the machine at 0 0 0 it will come to 0 1 1.

Similarly, if you start at 0 0 1 so, it will come to the state 0 1 0. So, that way it is restricted, but it is not arbitrary like in a previous example that we have seen, so, there if you start at some state which is invalid it is unknown. So, what will be the next state? So, that is not known, but in this case it is completely specified. So, this self-starting FSM so, they will always either keep you in this cycle always all one or it will take you to one of these 5 states ok, after some time.

(Refer Slide Time: 20:15)

The slide title is "Self-starting FSMs". Below it is the subtitle "Deriving state transition table from don't care assignment".

Three state transition tables are shown:

- C⁺**:
Inputs: C, B, A
Outputs: C, B, A
State transitions:

0	1	1	0
0	1	1	0
B			
- B⁺**:
Inputs: C, B, A
Outputs: C, B, A
State transitions:

1	0	0	1
1	1	1	1
B			
- A⁺**:
Inputs: C, B, A
Outputs: C, B, A
State transitions:

1	1	0	0
0	0	1	1
B			

Below these is a state transition table:

Present State	Next State				
C	B	A	C ⁺	B ⁺	A ⁺
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	X	X

The bottom of the slide includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo.

And, during by the going by the same mechanism that we have drawn so, you can draw the as state transition table like say from 0 0 0 it is going to 0 1 1. So, 0 0 0 it is going to 0 1 1. Similarly, 0 0 1 it is going to 0 1 0. So, 0 0 1 it is going to 0 1 0. So, rest from 1 1 1 it remains it is 1 1 don't care. So, from 1 1 1, it can either remain in 1 1 1 or it can come to 1 1 0. So, that way it is left to the designer the specification does not tell what will happen to 1 1 1 clearly.

But, it can go to either it can remain either in this state or it can come to this state. So, if that is specified then it has to be 1 1 X, but if we truly if we want to follow this diagram truly then for this 1 1 1 the next state should also be 1 1 1. Anyway so, once we are done with this state transition table so, we can we can make the corresponding circuit and get it done.

(Refer Slide Time: 21:20)

Comparison of Mealy and Moore machines

- Mealy machines tend to have fewer states
 - different outputs on arcs (i^*n) rather than states (n)
- Mealy machines react faster to inputs
 - react in same cycle – don't need to wait for clock
 - delay to output depends on arrival of input
- Moore machines are generally safer to use
 - outputs change at clock edge (always one cycle later)
 - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we make a so, look into a comparison between Mealy machine and Moore machine. So, Mealy machines they tend to have fewer states, because the output is a function of state present state and input. So, as a result so, we many of the Moore machine states so, they are copy they are they can be combined in a Mealy in a Mealy machine state. Because of the reason that for Moore machine for every even the other otherwise the 2 states being same only the output being different. So, they are going to constitute 2 different states.

Whereas for Mealy machine. So, they can be clubbed into one state and we can represent the transitions by means of these inputs also. The input of the inputs primary input they can also be considered as some constraint on the primary output. Second point is the Mealy machines react faster to inputs, because of this reason that in case of Mealy machine what is happening is that suppose this is the current state and there is a transition like this to this state.

So, input slash some say inputs slash some output. So, if in this state if the FSM is in this state then as soon as this input condition is satisfied. For example, if I say that from this state it is going to this state and the condition is $A = 0$. And in that case it will make $B = 1$. So, in if the machine is in this state then whenever $A = 0$ B will be made equal to 1. So, that is possible for Mealy machine, but because some Moore machine what is happening is that this output is a part of the state itself.

So, it will be like this say A equal to 0. So, it will take to this state where it will make B equal to 1. Now, this transition of state cannot occur until the next clock pulse arrives, because the next state function that it will that the combinational logic computes. So, it is put into this flip flop and this, but this flip flop output will come to this point only when this clock has arrived to this flip otherwise this value is not available at the output of the flip flop.

So, as a result even if A equal to 0 till the clock pulse comes thus the machine does not transit from this state to that state. So, B does not become 1, till the transition has taken place or we can say that till the next clock pulse has arrived. So, there is a delay to the output depends on arrival of so, this react in some, react in some in same cycle don't need to wait for clock and delay to output depends on arrival of input. Whereas, Moore machines so, they are generally safer to use because the output will change at the clock edge.

So, always one cycle later; so it can be a bit safe in our design in Mealy machine input a change can cause output change as soon as logic is done. So, sometimes it is a problem particularly if we have got 2 machines that are interconnected with an asynchronous feedback. So, one machine making a transition that can affect another machine's operation; so if it is a Mealy machine type of if it is a Moore machine type of realization then both these machines will transit at the next clock pulse.

So, they are going to behave in a better fashion compared to a Mealy machine where one of the machine makes some transition in between, because the change in the input combination and as a result the output may become that is the interconnection that the interface may become problematic sometimes this is seen. So, though there is no hard and fast rule like which one will be used, but in general Mealy machine has got less number of states. So, for a large machine; so, we will go for Mealy type of realization.

(Refer Slide Time: 25:14)

Implementing an FSM

- 1. Perform state assignment
 - different assignments may give very different results
 - no really good heuristics
 - using an extra bit or two for state works well
 - FPGAs often use a 1-hot encoding
- 2. Convert state diagram to state table
 - equivalent representation
 - mechanical
- 3. State table gives truth table for next state and output functions
 - synthesize into logic circuit
 - e.g. 2-level logic implementation

$S_0 \rightarrow 0001$
 $S_1 \rightarrow 0010$
 $S_2 \rightarrow 00100$
 $S_3 \rightarrow 1000$

$\circ \rightarrow 0$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the state steps for coming back to the steps for implementing a finite state machine. So, first point is to perform state assignment. So, we can make different assignments that can give different results, there is a, as I said that it is an MP hard problem. So, there is we cannot solve the problem optimally with a polynomial time algorithm. So, as a result there is no really good heuristics that exists.

Then sometimes we use some extra bit 1 bit or 2 bit extra that for that makes the combinational logic simpler and the extreme point is with FPGAs where we do one hot encoding. So, one hot encoding is like this like if I have got say 4 states S_0 , S_1 , S_2 and S_3 , then I can say that for S_0 I have a code like this, 0 0 0 1, S_1 is a code the code is like this, S_2 is a code like 0 0 sorry 0 1 0 0 and for S_3 the code is 1 0 0 0.

Now, whenever there is a transition from one state to another state in the finite state machine you see only 2 bits are going to change ok. So, only 2 bits are going to change. So, that makes it interesting. So, that the combinational logic that we have. So, that becomes much simpler since only 2 bits will be changing. So, this type of coding is known as a 1 hot encoding the 1 hot means in the state code only 1 bit is 1 and all are the remaining bits are 0s.

And, this is particularly useful when we have got this flip flop rich architectures like FPGAs field programmable gate arrays; we will see that later. So, this 1 hot encoding is very popular. So, we convert state diagram to state table we can do equivalent

representation or we can do some it is a mechanical process. And, then we can get the state table truth table for the next state and output function and then go for a logic minimization logical implementation.

(Refer Slide Time: 27:15)

- 1. Perform state assignment
 - different assignments may give very different results
 - no really good heuristics
 - using an extra bit or two for state works well
 - FPGAs often use a 1-hot encoding
- 2. Convert state diagram to state table
 - equivalent representation $10\downarrow\downarrow\downarrow$
 - mechanical
- 3. State table gives truth table for next state and output functions
 - synthesize into logic circuit
 - e.g. 2-level logic implementation

(Refer Slide Time: 27:19)

General Machine Concept:

- deliver package of gum after 15 cents deposited
- single coin slot for dimes, nickels
- no change

Step 1. *Understand the problem: Draw a picture!*

Block Diagram

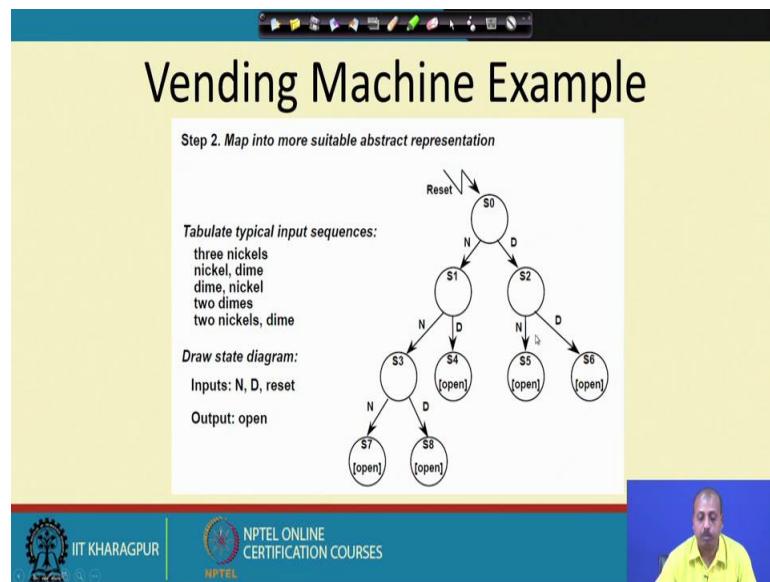
```
graph LR; CS((Coin Sensor)) -- N --> VMFSM[Vending Machine FSM]; CS -- D --> VMFSM; CS -- Reset --> VMFSM; VMFSM -- Open --> GRM[Gum Release Mechanism]
```

So, if we look into an example like we are looking into your vending machine. So, it delivers a package of gum after 15 cents have been deposited. There is a single coin slot for dimes nickels and there is no change no change is given back. So, first stage is first step is to understand the problem. So, this is my vending state machine FSM there is a coin

sensor that senses nickel and dime and there is a reset that will reset the system and there is a clock signal.

So, the when appropriate change appropriate some 15 cents or a more has been given, then this slot opens and 1 gum is released. So, gum release mechanism is given the open signal. So, that is the specification of the problem.

(Refer Slide Time: 28:01)

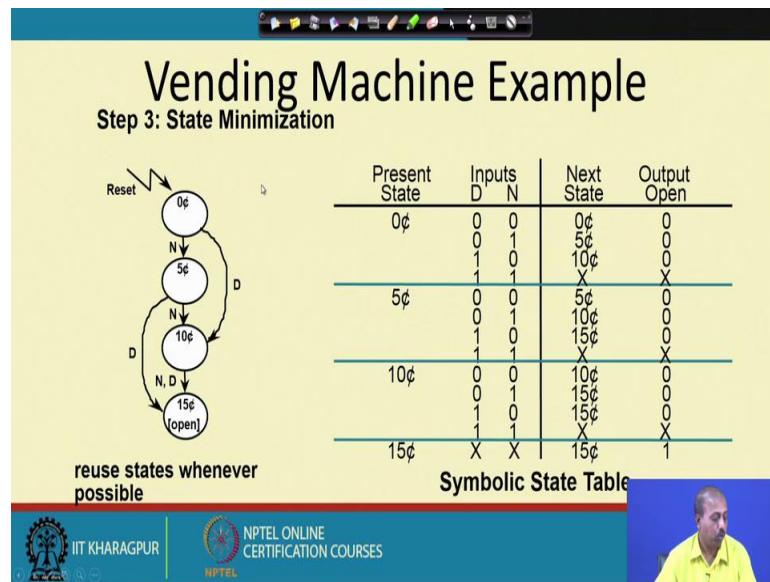


So, you can say that we can so, how can we go to 15? Ok. So, there can be 3 nickels 1 nickel 1 dime so, nickel is equivalent to 5 dime is equivalent to 10. So, this one so, sequence may be a 3 nickels, nickel dime nickel 2 dimes and we do not give the change 2 nickels and 1 dime.

So, that way we do not give the change. So, whenever it becomes just higher than 15 the gum is released. So, it is you start at state S0. So, this sequence is in nickel, nickel, nickel. So, going to the states S0 S1 S3 and S7 and in the state S7 the open signal is equal to 1 or it may be dime, dime going like S2 S6 states.

So, we draw this diagram. After this diagram has been drawn, so, we look into the behavior and get the confidence that this really represents the behavior properly.

(Refer Slide Time: 28:56)



Then we do some state minimization. So, all these states that we have drawn here they are not required and they can be minimized into this thing. So, on reset it enters into a state called 0 cent and then if the nickel is placed given then it goes to 5 cent state as if it has seen 5 cent.

Another, nickel comes to the state called 10 cent and from 0 cent also if they are if a dime is given it comes to the 10 cent state. Similarly, for the 5 cent state if a dime is given it comes to 15 cent and from the 10 cent state if it comes to the state 15 cent. Of course, so, this is from 10 cent if you put a dime also though it is 20 cent, but we do not need to remember that as a separate state because no separate action is necessary. So, that is 15 cent only.

So, after that if we code the states like say the present state is 0 cent. So, 0 cent the then if the input given is 0 0, then it remains a 0 state that is no input is given and output remains 0. Then you if a nickel is given it comes to the state 5 cent and the output remains 0. So, in this way you can draw the symbolic state table.

(Refer Slide Time: 30:12)

Vending Machine Example

Step 4: State Encoding

Present State $Q_1\ Q_0$	Inputs D N	Next State $D_1\ D_0$	Output Open
0 0	0 0	0 0	0
	0 1	0 1	0
	1 0	1 0	0
	X X	X X	X
0 1	0 0	0 1	0
	0 1	1 0	0
	1 0	1 1	0
	X X	X X	X
1 0	0 0	1 0	0
	0 1	1 1	0
	1 0	1 1	0
	X X	X X	X
1 1	0 0	1 1	1
	0 1	1 1	1
	1 0	1 1	1
	X X	X X	X

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

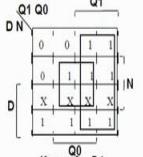


Once you have drawn the symbolic state table the next part is the state encoding. So, this Q_0 present state. So, this 0 cent, 5 cent, 10 cent and 15 cent they are given the code 00, 01, 10 and 11 accordingly, you get this next state function. Once this state of state encoding has been done so, we have got this truth table.

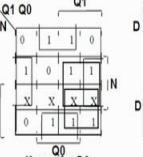
(Refer Slide Time: 30:31)

Vending Machine Example

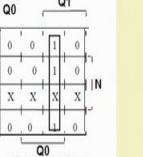
Step 5. Choose FFs for implementation D FF easiest to use



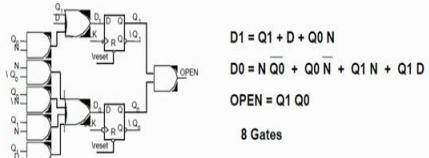
K-map for D_1



K-map for D_0



K-map for Open



$D_1 = Q_1 + D + Q_0 N$
 $D_0 = \bar{N} \bar{Q}_0 + \bar{Q}_0 \bar{N} + Q_1 \bar{N} + Q_1 D$
 $OPEN = Q_1 Q_0$
8 Gates

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, this truth table you can now minimize using some Karnaugh map for individual D_1 and a D_0 and open for the 3 for these 3, 3 functions you can get. So, this is the function that is obtained a $D_1 = Q_1 + D + Q_0 N$. So, like that you can find out that this is the function obtained. And, ultimately we can realize it by means of some combinational logic function like this ok. So, this is the vending machine example.

So, in this way any find given any finite state machine. So, you can start with the behavior come to the corresponding the state transition diagram, then from there the state table and then count encode the states using some codes. And then come to the combinational logic to be realized and then get the final function realized in terms of logic gates and flip flops.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 37
Data Converters

We will next look into a new topic, which is known as Data Converters. Now, as we know that any digital system that, we are designing, so, ultimately it is going to be used in the environment, while it has to interact with the analog signals. Like if we are looking into say a temperature controlling system. So, the temperature is an analog value and there are sensors which will be sensing the temperature. And that sensor, that sensing will convert the physical signal into some electrical signal. So, it may be converting the temperature value into some current or voltage value, but that is an analog value.

But, in case of these computers that we have so; they can process only digital value. So, we have to convert it into some digital value. So, that way there may be some error introduced in the process like, may be when the temperature is forty degree Celsius. So, it output some voltage. So, if it is expected that say from 40 to 45. So, next temperature sensed is 45. So, according the voltage value increases.

So, then that conversion to digital from this voltage to this digital value, so, that should be equivalent if we are going from say 20 to 25 degree change. So, the amount of 5 degree change that we have the corresponding if the sensor that we have so, it gives the good reflection of it in terms of analog signal, analog voltage. And if we assume that the analog signal change is equivalent to the change in the physical signal, then from this analog signal to digital signal conversion that should also be proportional.

So, we need to have some special circuitry that we will be doing this conversion of data like when the computer wants to read some analog value from the environment. So, it has to be some it has to be converted from analog to digital. Similarly, if the computer wants to control some system; so by giving some electrical signal to so, for example, some motor has to be rotated. So, motor has to be given some voltage which it will understand at which speed it has to rotate.

But, this computer cannot output the voltage directly so, analog voltage directly. So, it will output some digital value and then the digital value has to be converted into analog signal before giving it to the motor. So, these type of data converters are necessary, when we are interfacing it is digital system with the environment.

(Refer Slide Time: 02:51)

- Digital-to-Analog Converters (DACs)
- Analog-to-Digital Converters (ADCs)

So, in this part of the course so, we will be looking into 2 such converter mechanism one is called digital to analog converter or DAC, and another is known as analog to digital converter or ADC.

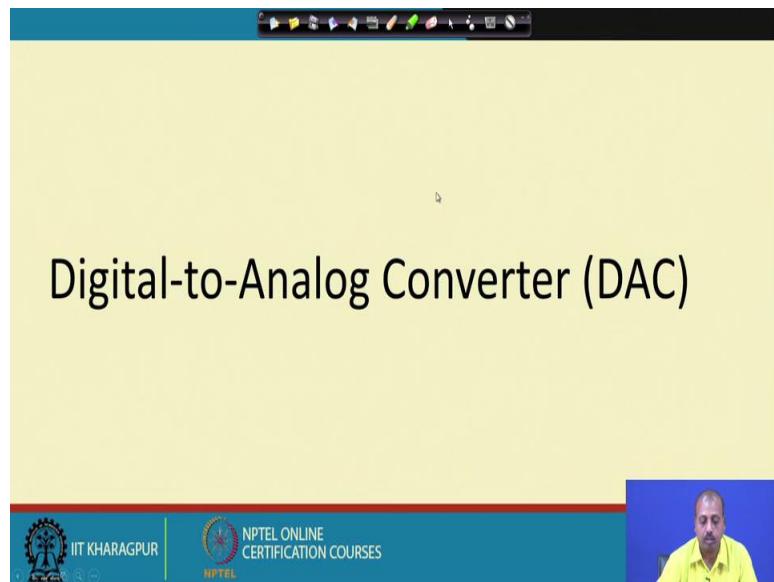
So, we can say that a typical diagram for this entire system is like this that we have got this as the digital system. So, this is the digital system that we have and from the outside world, so, we are getting the analog signal, we are getting the analog signal and it passes through something called analog to digital converter. So, this is the ADC and then these value is the digital. So, this is the digital value that we have and here we have got the analog value.

And, after doing the processing this digital system it outputs some digital value. And, that digital value has to be converted to analog before it is given to the environment. So, this is 1 digital to analog converter and then this is the analog signal. So, this is the analog signal that we get ok. So, this is how this whole system will work. So, we need to understand this 2 circuitry DAC circuitry and ADC circuitry, because they becomes a very important part

of the, this interfacing this digital system to the analog environment. So, how do we do that so that we will see?

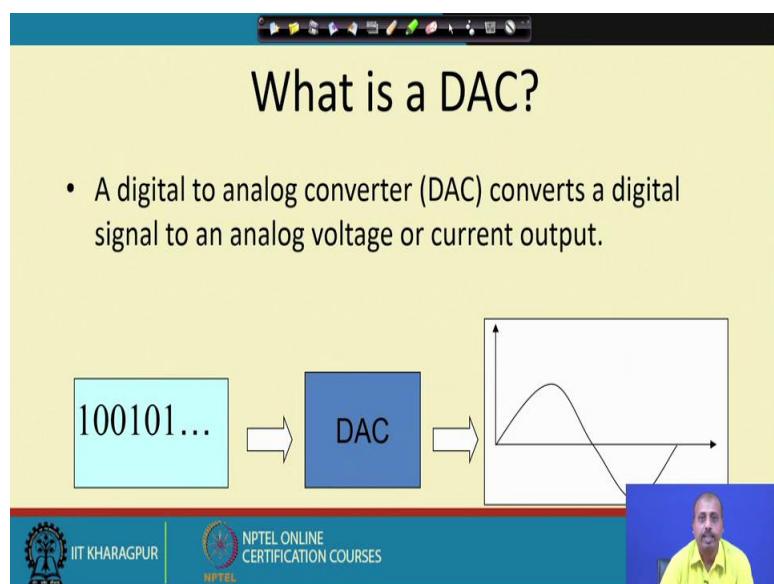
Now, the first they are 2 distinct type of converter.

(Refer Slide Time: 04:27)



So, we will first look into the digital to analog converter which will convert the digital signal to the analog output.

(Refer Slide Time: 04:37)

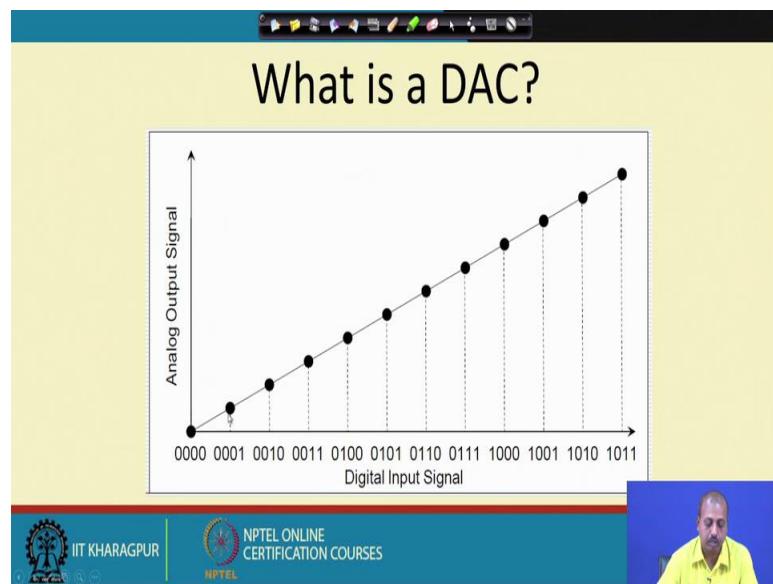


So, what is DAC? A digital to analog converter it converts a digital signal to an analog voltage or current output. So, because voltage and current they are they do not cause much of problem, because if we get a current output also so, we can pass it through a resistance and then measure it the corresponding voltage. So, as a result is current and voltage. So, these 2 outputs any of these outputs are with us.

So, if you give some if this is the digital to analog converter block and you were giving an input like say 1 0 1 0 1 etcetera some bit pattern is fed. So, it is expected that it will be outputting some analog signal. So, it may be some for the sake of understanding, so, I have drawn a sinusoidal signal here, but it may be it is definitely dependent on the bit pattern that your feeding here, accordingly you can generate some analog signal this point.

Of course, you can you can feed this bit pattern in such a fashion, that the output signal is a sinusoidal signal or a square signal or a triangular signal, whatever we want; so whatever control signal is needed for controlling the device that is connected to this analog signal. So, we can produce that signal accordingly.

(Refer Slide Time: 05:51)



So, so, this is the digital input signal. So, there may be we are outputting from 0 0 0 0 to 1 0 1 1 and on the analog output signal. So, it increases like this. So, when it is a 0 0 0 0. So, output is also 0 when the input is 0 0 0 1, so, output is this much, 0 0 1 0 output is this much. So, this is the behavior of the DAC. So, at this count values 0 0 0 0, 0 0 0 1, 0 0 1 0 so, it will be outputting this corresponding voltages. Of course, it is not a continuous signal

because in between what happens so, these points are not known. So, this is basically by means of some approximation we can visualize it as a continuous signal like that.

(Refer Slide Time: 06:34)

Types of DACs

- Many types of DACs available
- Usually switches, resistors, and op-amps used to implement conversion
- Two Types:
 - Binary Weighted Resistor
 - R-2R Ladder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video camera icon shows a person speaking.

Now, if you look into these types of these digital to analog converters. So, there are many types of DACs that are available. And, they are usually switches resistances and operational amplifiers; they are used to do this conversion. And, there are we will be looking into 2 type of DAC one is known as binary weighted resistor and R 2 R ladder type resistor.

So, as I said that the main, main components of a DAC, they are switches, resistances and some operational amplifiers they will be used for getting the analog output.

(Refer Slide Time: 07:14)

- Utilizes a summing op-amp circuit
- Weighted resistors are used to distinguish each bit from the most significant to the least significant
- Transistors are used to switch between V_{ref} and ground (bit high or low)

$V_{out} = -\frac{R_2}{R_1} V_{in}$

The first binary so, first digital to analog converter that we look into is the binary weighted resistor based digital to analog converter. So, it utilizes a summing operational amplifier circuitry. So, what we mean by a summing operational amplifier is so, operational amplifier is some circuit like this I hope your familiar with the this diagram.

So, basically the some input is given it takes a differential input and this amplifier has got a gain of A. So, what is the difference between these 2 voltages if we call it V_{diff} , then the output is AV_{diff} . So, ideal operational amplifier this A is infinity. So, this value will go to infinity. So, if on the other hand we can have some feedback to control the gain and as you know that there are 2 types of inter 2 types of organizations of this amplifier for using op amp.

So, one is called inverting amplifier, another is called non inverting amplifier. So, this they have got particular gain expression which is less than infinity. For example, if we are having this inverting type operational amplifier. So, the circuit diagram is like this.

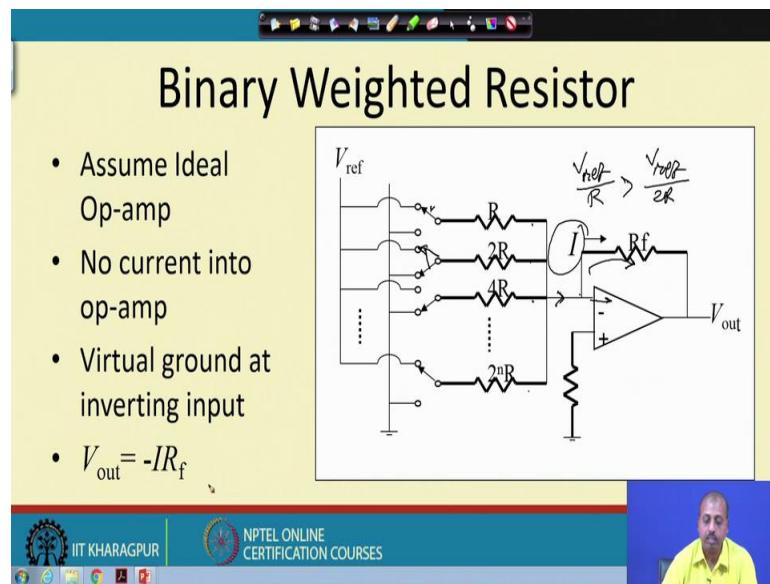
So, if we apply V_{in} here and then if this resistance is say R_1 and this resistance is say R_2 , then this V_{out} value is given by $-\frac{R_2}{R_1} V_{in}$. So, from this infinity gain so, we can restrict we can control this gain of the circuit. Now, this type of so, if I have got a number of voltage sources. So, V_{in1} , then V_{in2} , V_{in3} like that. So, they can be added here, the successive turns

can be added here, and then it give rise to something called this summing operational amplifier. So, we will see some circuitry on that.

However, these weighted resistors are used to distinguish each bit from the most from the most significant to the least significant bit. So, each bit will contribute to some value into the output. So, most significant bit its contribution will be high, that is if the most significant bit is one. So, it will contribute significantly to the output voltage or current. So, if we will go down and look into the successive bits their contribution will be less and lesser and lesser.

So, the least significant bit it will have the minimum contribution and that way it will go. So, transistors are used to switch between V_{ref} and ground, which is bit high and low. So, V_{ref} is the reference voltage, which is which is taken this high value and low is the ground. So, that is taken as the 0 value.

(Refer Slide Time: 10:15)



So, this is the structure you can say. So, each of these bits that we have so, each of this lines so, they constitute 1 bit fine. So, you can see that we have we have got we have got this operational amplifier like this bit. So, if it is a so, this bit. So, if I connect it to this point. So, the current that is drawn by this is V_{ref} / R .

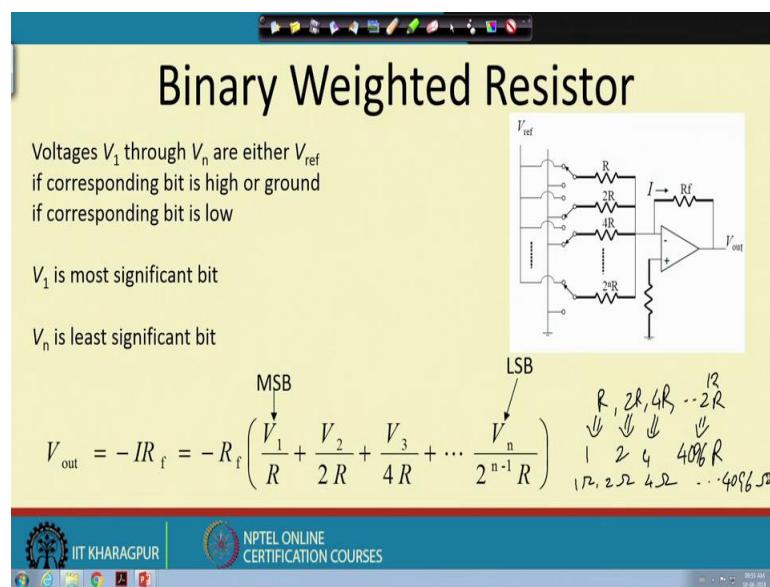
So, similarly when it is when this switch is connected to say this point then the current drawn will be $V_{ref} / 2R$. So, as a result the contribution from the first if this switch is

connected here that the contribution that is coming on to this line is the current value is V_{ref} / R . And, if we connect this line to here if it is connected like this in the contribution is $V_{ref} / 2R$.

So, this contribution is less than this one ok. So, as you go down and down. So, what happens is that this contribution becomes exponentially less. So, whatever be the contribution here current contribution. So, this has got half of that this has got half of that. Finally, the nth bit it will have the contribution reduce by a factor of 2^n . So, if we assume this operational amplifier to be an ideal operational amplifier. So, no current flows into this op-amp. So, whatever current comes like this it passes like this fine.

So, you can say that V_{out} is given by IR_f . So, we can try to figure out what is the value of I and based on that we can find out what is the voltage that is coming as the output. So, let us see.

(Refer Slide Time: 12:12)



So, V_1 is the most significant bit. So, it is and V_n is the less least significant bit.

So, whatever we coming here as voltage; so this is given by

$$V_{out} = -IR_f = -R_f \left(\frac{V_1}{R} + \frac{V_2}{2R} + \frac{V_3}{4R} + \dots + \frac{V_n}{2^{n-1}R} \right)$$

(Refer Slide Time: 13:31)

Binary Weighted Resistor

If $R_f = R/2$

$$V_{out} = -IR_f = -\left(\frac{V_1}{2} + \frac{V_2}{4} + \frac{V_3}{8} + \dots + \frac{V_n}{2^n}\right)$$

For example, a 4-Bit converter yields

$$V_{out} = -V_{ref} \left(b_3 \frac{1}{2} + b_2 \frac{1}{4} + b_1 \frac{1}{8} + b_0 \frac{1}{16} \right)$$

Where b_3 corresponds to Bit-3, b_2 to Bit-2, etc.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Now, if we so, if $R_f = R / 2$. Then this

$$V_{out} = -IR_f = -\left(\frac{V_1}{2} + \frac{V_2}{4} + \frac{V_3}{8} + \dots + \frac{V_n}{2^n}\right)$$

Now, depending upon this $b_0 b_1 b_2 b_3$ this bit pattern so, this contribution will be coming. So, you can say in this diagram you see that if I connect this point to here then V_2 becomes equal to 0. If, I connect this point here then the V_2 becomes equal to V_{ref} ok. So, that way we can say that this V_2 these voltages can be either equal to V_{ref} or equal to 0 and that is controlled by the position of this switch.

So, when the switch is connected to this we can say that the corresponding bit is equal to 1 and when the switch is connected to this position so, we can say the corresponding bit is equal to 0. So, here if I take say b_3 equal to 1 that is the most significant bit is connected to is equal to 1. So, it is getting the contribution $V_1 = V_{ref}$ in that case. So, in the second one if b_2 is not second bit is do not connected. So, b_2 is second bit is second switch is connected to ground so; that means, b_2 is equal to 0.

So, in that situation this b_2 being equal to 0 so, V_{ref} contribution does not have anything to here V_2 becomes equal to 0, so because of this $b_2 \times V_{ref}$, so b_2 being equal to 0 so, this the value of V_2 becomes equal to 0. Similarly if the third bit is equal to 1 then this V_3 becomes again equal to V_{ref} and if this nth if the least significant with b_0 is equal to 0, then this contribution V_4 . So, that will become equal to 0.

So, b_3 corresponds to b_3 b_2 corresponds to b_2 like that. So, that way we can have a binary weighted resistor based structure for getting this V_{out} . So, depending upon the digital value that if b_0 b_1 b_2 b_3 value of V_{out} will get determined.

(Refer Slide Time: 15:58)

Binary Weighted Resistor

- Advantages
 - Simple Construction/Analysis
 - Fast Conversion
- Disadvantages
 - Requires large range of resistors (2000:1 for 12-bit DAC) with necessary high precision for low resistors
 - Requires low switch resistances in transistors
 - Can be expensive. Therefore, usually limited to 8-bit resolution.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, what are the advantages? So, it is a simple construction and analysis. So, simple because you just need a couple of resistance resistances and then connect it in some fashion. Conversion process is quite fast, because it is all the comparisons are being done simultaneously. So, the conversion process is also fast, but there are disadvantages because requires large range of resistors like for 12 bit DAC. So, for 12 bit DAC, so, we can understand if we look into this diagram. So, this is going from. So, it is going from R , $2R$, $4R$ up to $2^{12}R$ ok. So, 2^{12} is $4096R$.

So, if you take even if you take R equal to 1 ok, if you take R equal to 1 then this is equal to 2. So, this is equal to 4. So, you need all these resistors like 1 ohm resistor, 2 ohm resistor, 4 ohm resistor so, like that up to 4096 ohm resistors. So, getting resistances of this exact values is a problem or getting resistance is in this ratio is also a problem. So, that makes it difficult for this realization of this binary weighted DAC ok.

So, with necessary high precision of low resistors so, it is difficult requires low switch resistances in transistors. So, these transistors that, we have taken. So, these resistances actually coming so, this switches they are actually realized by some transistors like if it is a MOS transistor.

(Refer Slide Time: 17:46)

Binary Weighted Resistor

Voltages V_1 through V_n are either V_{ref} if corresponding bit is high or ground if corresponding bit is low

V_1 is most significant bit

V_n is least significant bit

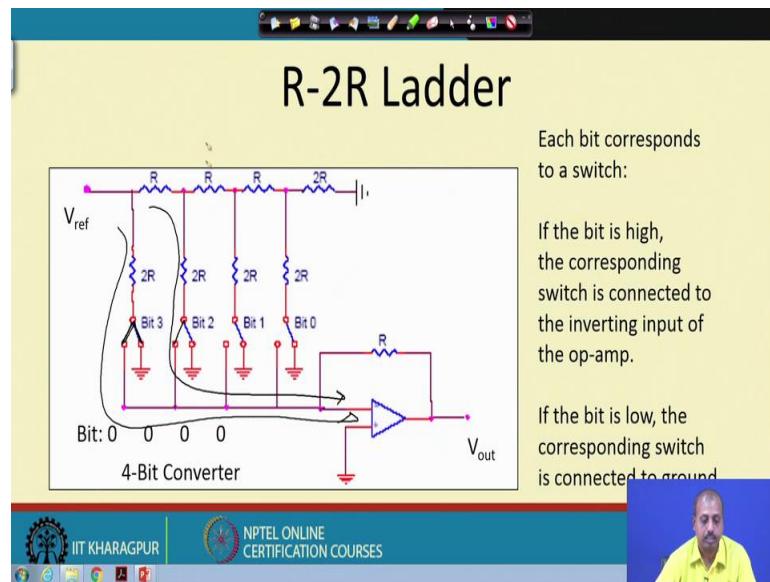
$V_{out} = -IR_f = -R_f \left(\frac{V_1}{R} + \frac{V_2}{2R} + \frac{V_3}{4R} + \dots + \frac{V_n}{2^{n-1}R} \right)$

So, this is if the connection is like this. So, this point is connected to V_{ref} this is connected to ground and we have got a switch. So, this is connected to the digital bit V .

So, if b equal to 1 then V_{ref} gets connected here if b equal to 0 then this ground gets connected here. So, if the problem is that this channel resistance that also comes in the series in series with the resistance. So, here I have got that R resistor. So, this is the R resistor finally, going to this side and finally, going to that op-amp. So, this channel resistance is they will come in series with this R resistance. So, with this channel resistance it should also be pretty low. So, that we do not have we do not have problem in getting the proper resistance values implemented.

So, that is the point. So, this switch resistances should be low. And, it can be expensive because of this we have to getting this resistances of good precision particularly, the resistors which have got low values getting them is good high precision may be costly. Therefore, so, it is usually limited to 8 bit resolution you do not go for 12 bit and all. So, normally keep it within 8 bit resolution.

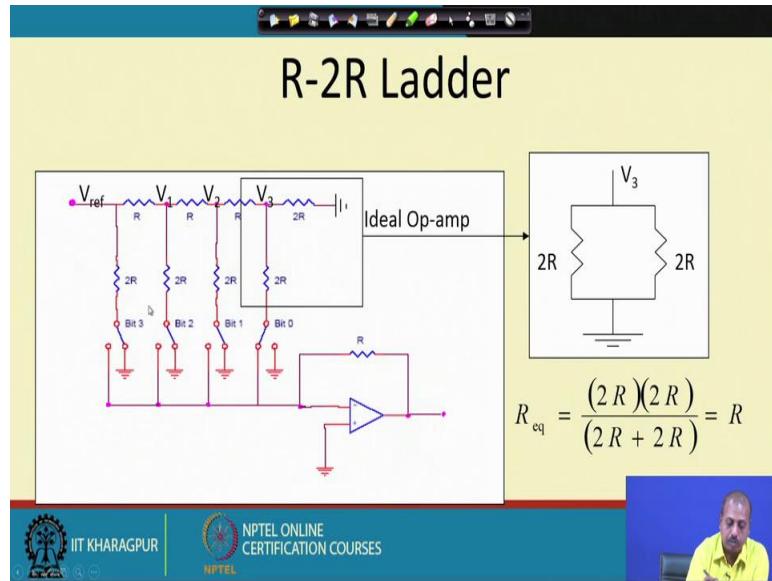
(Refer Slide Time: 19:16)



Another type of DAC, which is pretty common, which is known as R 2 R ladder; so here you need only 2 types of resistors 1 is of value R, other is a value 2 R. So, unlike the previous case where you were requiring a series of resistance values so, here that is not required. So, you require R 2 R ladder type of R and 2 R resistances and they are connected in this fashion. So, from V_{ref} we make it R, R, R and finally, there is a resistor 2 R. And so, so this is a 4 bit converter. So, we have got 3 such resistance R in series and then 2 R resistor in series and then from this point. So, we take down take the 2 R resistance and take it down.

Now, this depending upon the bit configuration so, either this is connected to ground or it is connected to this one this. So, this is a point which is it may be connected to this point or it may be connected to ground. So, these switch so, either it is connected here.

(Refer Slide Time: 20:31)



So, this sorry; so this point so, it is either connected to this or it is connected to this. So, when it is connected to this; that means, is 0 is coming here, if it is connected to this then the contribution, the current flow contribution, actually so, either the either the connection is like this or the connection is like this.

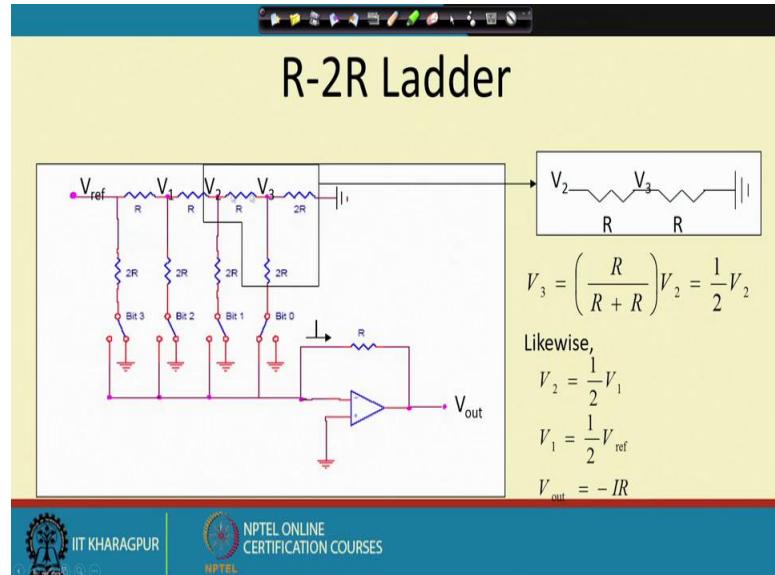
So, if it is connected like this then there is no contribution of this bit. And, if it is connected like this then, it is this current flows in this fashion. Similarly, for the second bit may be it is connected like this. So, as a result this does not have any contribution no current contribution is coming, but if it is connected in this fashion then again a current contribution like this will come.

So, then it is a summing type of structure. So, that is all right. So, each bit corresponds to a switch if the bit is high the corresponding switch is connected to the inverting input of the operational amplifier, if the bit is low the corresponding switch is connected to the ground ok. So, it is in this fashion it is connected ok.

So, so, this is the switch. So, either it is connected to the inverting input of the op amp like this or it is connected if the bit is low then it is connected this is switch is connected to the ground. So, this way this connection is made. Now, so, let us see how it works, fine? So, so, if we look into say this part then what is happening is that this V_3 voltage? So, there is a $2R$. So, these $2R$ resistances are there. So, they are in parallel. So, R equivalent is equal to R for this part. So, the equivalent when this bit is connected like this. So, some part of

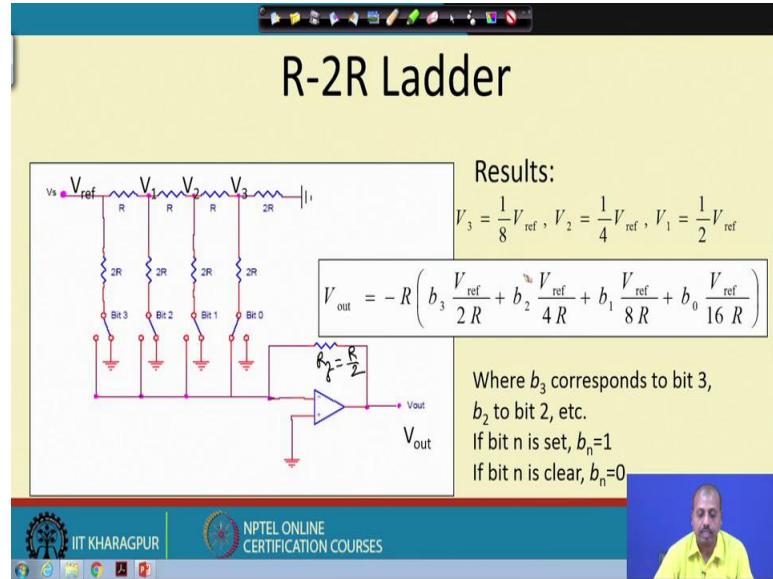
the current will flow like this some part of the current will flow like this. So, you will get R equivalent equal to R.

(Refer Slide Time: 22:38)



So, now if you consider this V_2 ok; after that we have got this R then V_3 then this part is equivalent to R as we have seen previously so, this part is equivalent to R . So, this equivalent will the V_3 becomes half of V_2 . In this way if we continue then V_2 will turn out to be half of V_1 , V_1 will turn out to be half of V_{ref} and $V_{out} = -IR$. So, that is there. So, once you have got these values of V_1 , V_2 and V_3 . So, you can find out what is the value of I and accordingly you can get the V_{out} value.

(Refer Slide Time: 23:22)



So, for example, in this case we have got $V_3 = \frac{1}{8} V_{\text{ref}}$, $V_2 = \frac{1}{4} V_{\text{ref}}$, $V_1 = \frac{1}{2} V_{\text{ref}}$. So, this way we get it like this. And then when this summing is done when this summing is done then it

$$V_{\text{out}} = -RV_{\text{ref}} \left(b_3 \frac{1}{2R} + b_2 \frac{1}{4R} + b_1 \frac{1}{8R} + b_0 \frac{1}{16R} \right)$$

So, in this structure, so, it is assumed that this feedback resistance this $R_f = R / 2$, that is why this expression becomes like this ok. So, b_3 is connected to so, you so, b_3 is connected to b_3 b_2 is bit 2 etcetera. So, if bit n is set then b_n equal to 1 and if bit n is clear then b_n equal to 0.

So, in this fashion we can have an R 2 R ladder implemented and it gives the corresponding digital output.

(Refer Slide Time: 24:51)

R-2R Ladder

For a 4-Bit R-2R Ladder

$$V_{\text{out}} = -V_{\text{ref}} \left(b_3 \frac{1}{2} + b_2 \frac{1}{4} + b_1 \frac{1}{8} + b_0 \frac{1}{16} \right)$$

For general n-Bit R-2R Ladder or Binary Weighted Resister DAC

$$V_{\text{out}} = -V_{\text{ref}} \sum_{i=1}^n b_{n-i} \frac{1}{2^i}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we have got this V_{out} is this is the final expression that we have. So,

$$V_{\text{out}} = -V_{\text{ref}} \left(b_3 \frac{1}{2} + b_2 \frac{1}{4} + b_1 \frac{1}{8} + b_0 \frac{1}{16} \right)$$

So, for general n bit R 2 R ladder and or binary weighted resister and DAC so, the equation becomes $V_{\text{out}} = -V_{\text{ref}} \sum_{i=1}^n b_{n-i} \frac{1}{2^i}$. So, this is the final expression for any DAC circuitry that we have which converts from digital to analog value.

(Refer Slide Time: 25:38)

R-2R Ladder

- Advantages
 - Only two resistor values (R and $2R$)
 - Does not require high precision resistors
- Disadvantage
 - Lower conversion speed than binary weighted DAC

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, why should we go for R 2 R ladder? So, advantage is the only 2 resistor values are necessary which is R and 2 R and does not require high precision resistors.

So, only thing is that you need for getting 2 R. So, what we can do we can take 1 resistance 1 type of resistance are connect 2 of them in series. So, we get 2 R. So, ideally we can say only single resistance is sufficient and you can just sometimes connect two of them sometimes connect one of them, to get the R 2 R configuration or you can get the exactly double of the resistance 1 k and 2 k and that way we can connect it.

So, disadvantage is the low conversion speed than binary weighted DAC.

(Refer Slide Time: 26:29)

The slide has a yellow background and a blue header bar. The title 'Specifications of DACs' is centered in the yellow area. Below the title is a bulleted list of six items. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Resolution
- Speed
- Linearity
- Settling Time
- Reference Voltages
- Errors

Because, of this current this voltage being decided by individual resistances and then so, it becomes lower compare to this binary weighted DAC. When, we are talking about specification of a DAC. Now there are several DAC that may be available in the market, like if we are looking for a particular type of DAC. So, what are the factors that you should talk about?

One thing is the resolution, then the speed, then, resolution will come to this term, then speed of conversion, then how much linear it is in the settling time reference voltage and error.

(Refer Slide Time: 26:56)

Resolution

- Smallest analog increment corresponding to 1 LSB change
- An N-bit resolution can resolve 2^N distinct analog levels
- Common DAC has a 8-16 bit resolution

$$\text{Resolution} = V_{LSB} = \frac{V_{ref}}{2^N}$$

where $N = \text{number of bits}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, will look into this issues, then resolution it says the smallest analog increment corresponding to 1 LSB change.

So, the minimum amount of change that DAC can do is by changing the LSB. So, if you change the least significant bit from 1 to 0 or 0 to 1 there will either be increase or decrease in the analog outputs. So, when you do this change then what is the corresponding analog voltage change? So, that is the resolution. So, you cannot change the analog voltage by a value which is less than this one. So, that is the resolution.

N bit resolution can resolve 2^N distinct analog levels and basically from 1 bit once a bit sequence to the next bit sequence next digital count binary sequence, if you go then what is the minimum change in the analog voltage. So, we cannot go change beyond that. So, if we have got the reference voltage as V_{ref} then if no, that there are 2^N distinct analog levels that you can have.

So, the resolution is given by $\frac{V_{ref}}{2^N}$. So, which is also known as the voltage corresponding to LSB, V_{LSB} so, this is the expressions for the resolution of the DAC, where N is the number of bits. So, common DACs they have got 8 to 16 bit of resolution because for higher resolutions I have to go for more number of bits and then the constructions of the DAC becomes difficult. So, normal DACs we go for 8 to 16 bit of resolution. And depending upon the V_{ref} we can get the final analog voltage value by which it can change.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 38
Data Converters
(Contd.)

So, the speed is another factor that determines the selection of this DAC. So, rate of conversion of a single digital input to its analog equivalent, so, that is called the rate of conversion, that is, you give a particular input pattern, so, after how much time the analog signal comes to the proper value. So, that gives the rate of conversion for the digital input.

(Refer Slide Time: 00:36)

The slide has a title 'Speed' in large font. Below it is a bulleted list of four items. To the right of the list is a small diagram of a DAC symbol. At the bottom of the slide is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

- Rate of conversion of a single digital input to its analog equivalent
- Conversion rate depends on
 - clock speed of input signal
 - settling time of converter
- When the input changes rapidly, the DAC conversion speed must be high.

Then conversion rate it will depend on the clock speed of the input signal. So, because if the input signal is changing at a particular rate, then the conversion must be done before that next change occurs. So, if we have got this DAC if we have got this DAC operating this input is coming at some rate then this it is coming at some frequency f , then the conversion must be faster than that frequency otherwise it is not acceptable. So, depending upon the rate at which the processor is doing the computation so, we have to convert at that rate. So, clock speed of the input signal becomes an important point.

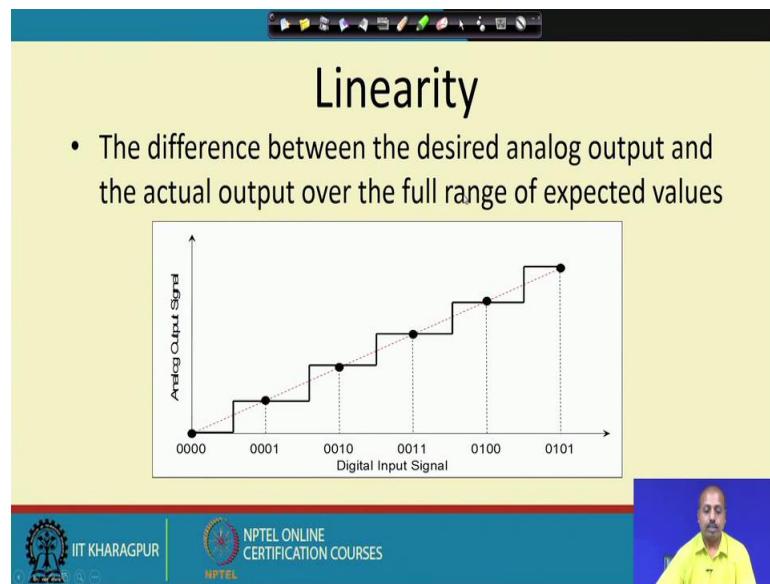
Second point is the settling time of the converter that is after you have given a particular pattern. So, it does not if the final analog value voltage is say V_A then it takes some time

for the analog output to get settle to V_A . So, sometimes it goes into an as a damped oscillation oscillatory behavior. So, as a result it takes some time to settle to V_A .

So, this clock speed of the of the input this settling time of the converter so, that is this is another issue that is after you have finished applying this pattern how much time it takes to settle to the value. So, before that this input should not change, ok, otherwise this behavior will be unpredictable. So, that is why this conversion rate will also depend on the settling time of the converter.

And, when the input changes rapidly, so, DAC conversion speed must be high. So, this is particularly true when we have some high end microprocessor or microcontroller which can produce output at a very high rate. So, then this DAC that we take so, that should also be able to cater to that particular speed. So, that is why this choice of DAC is application dependent and we have to be careful there.

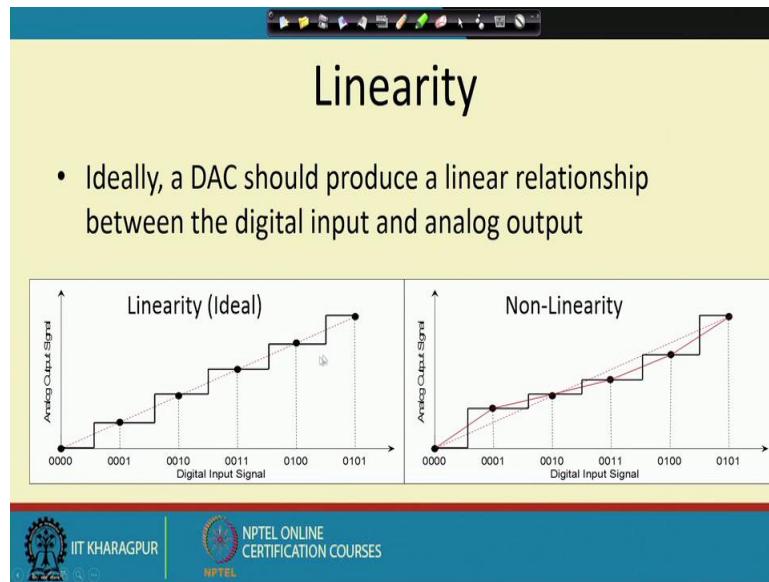
(Refer Slide Time: 02:37)



The next important issue that we have is the linearity. So, difference between the desired analog output and the actual output over the full range of expected values. So, you see that if this is say the digital input signal is like this. So, 0000, 0001, 0010 then the corresponding analog output values that we expect is at this point we expect the analog value to be this, then the next one are you expected values. So, this is the ideal behavior that we have. So, this is the, this should get so, here the step size is more the step size is same the analog voltage is not increasing rapidly. So, it is following a linear pattern. But, since this is

ultimately some circuit so, these resistances and this op amp so, it will have its own properties, as a result you may not get this linear behavior always. But, however, we want this linear behavior.

(Refer Slide Time: 03:36)



So, ideally so, this is the, this is what I was talking about. So, this is the linear behavior, this is the expected behavior on the other hand. So, we can get a non-linearity. So, it may so happen that when you change from 0000 to 0001 the voltage analog voltage increases from this value to this value, ok. So, it becomes high like this. Then when you change from 0001 to 0010 the output does not change that much. So, it increases only by this small amount of step.

Similarly, here from 0010 to 0011 it increases by this much; whereas, at later point it increases at a different rate. So, ideally we want that this step size should be same, but as I said that depending upon the circuit parameters. So, you may not get this a straight line this as say equal step size behavior so, you can there may be problem so, you can get irregular sizes and this analog output may become irregular. So, the linearity the linear relationship between digital input and analog output may not be maintained.

(Refer Slide Time: 04:41)

Settling Time

- Time required for the output signal to settle within +/- 1/2 LSB of its final value after a given change in input scale
- Limited by slew rate of output amplifier
- Ideally, an instantaneous change in analog voltage would occur when a new binary word enters into DAC

The graph shows a blue curve representing the actual output voltage over time. A straight horizontal line represents the expected output. The vertical axis has markings for +1/2 LSB and -1/2 LSB. The curve starts at the origin, rises above the +1/2 LSB mark, then falls below the -1/2 LSB mark, and finally settles back towards the expected output line. A horizontal arrow labeled t_{settle} indicates the time interval from the start until the output first reaches within +/- 1/2 LSB of the final value.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next issue that we have is the settling time. As I was telling that so, we have got say the some analog or some digital value change as a result so, this is the expected output. So, this straight blue line so, this is the expected output. However, the analog value it does not settle to this one immediately. So, it increases like this overshoot this point then it again comes down undershoots, then it goes like this. So, after some after some time it settles to the value.

So, if this is the amount of change that is the V_{ref} divided by this LSB voltage change. So, this is the plus due to this $+\frac{1}{2}V_{LSB}$ and $-\frac{1}{2}V_{LSB}$ change so, whatever be the difference in these 2 voltages. So, from your digital circuit there from your DAC circuit that we have so, we can compute what will be the analog voltage change. So, if we go from $-\frac{1}{2}V_{LSB}$ to $+\frac{1}{2}V_{LSB}$; that means, 1 LSB change.

So, if you take that region as the region of compromise then we can say that when the output becomes within confined within that range so, we are happy. So, then it will be then it settles down. So, so, this time is that t settling time. So, starting from the change in the digital input to this settling time so, this is known as t_{settle} . So, this is the time required for the output signal to settle within $\pm\frac{1}{2}V_{LSB}$ of its final value after a givens change in the inputs input scalar there, there is the input digital value. So, that is that that is the settling time for the DAC.

It is limited by the slew rate of the output amplifier. So, amplifier it will have some slew rate and if you look into the operational amplifier literature. So, you can find this thing. So, this is the, this is actually gives us that timely how fast it settles to the proper value. Ideally, an instantaneous change in the analog voltage would occur when a new binary word enters into DAC. This is the ideal behavior that is the DAC becomes produces the output immediately, but it will not happen.

So, it will have this t_{settle} . So, as I was telling that if your processor is giving digital output which are faster than this t_{settle} then naturally the DAC will not be able to convert all the digital values to their proper analog ones. So, that way there will be problem. So, while choosing the DAC to be used so, we have to be careful and we have to select this t_{settle} we have to select that DAC properly. So, that the rate at rate of data change for the application is can be accommodated within this t_{settle} time.

(Refer Slide Time: 07:24)

Reference Voltages

- Used to determine how each digital input will be assigned to each voltage division
- Types:
 - Non-multiplier DAC: V_{ref} is fixed
 - Multiplier DAC: V_{ref} provided by external source

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL



Next important issue is the reference voltages. So, it is used to determine how much each digital input will be assigned to each voltage division. So, we have so, this is basically. So, as I said that the voltage the, this with a change in the bit pattern. So, number of levels that we have is the $\frac{V_{ref}}{2^N}$. So, this n will determine one value. So, this individual steps ideas again $\frac{V_{ref}}{2^N}$.

So, if your V_{ref} is high enough then this accordingly you will get significant change with the step in the step size, ok. So, with us we each individual with each change in the digital count value so, you get a significant change in the output; whereas, if V_{ref} itself is low then this change is not that much significant so, that may not be able to activate the actuator that we have at the output of the DAC. So, that becomes a problem.

So, it is used to determine how much digital input will be assigned to each voltage division. So, there are two types of this reference voltages; one is non multiplier DAC where V_{ref} is fixed and this multiplier DAC where the V_{ref} will be provided by some external source. So, some of the DAC chips you will find that this V_{ref} value is fixed by the designer, some of the some of the DAC is that we have some DAC chips you will find that there is a V_{ref} pin that is available and there is a range that is specified. So, you can apply some V_{ref} in that range and as a result it will convert the voltage value accordingly.

So, many times what happens is that you need both positive and negative voltages for the for the analog output, so, in that case your V_{ref} can be both positive and negative. On the other hand if you want if you are looking for only positive output then this V_{ref} can be set accordingly. So, many chips they have got this type of configurations.

(Refer Slide Time: 09:26)

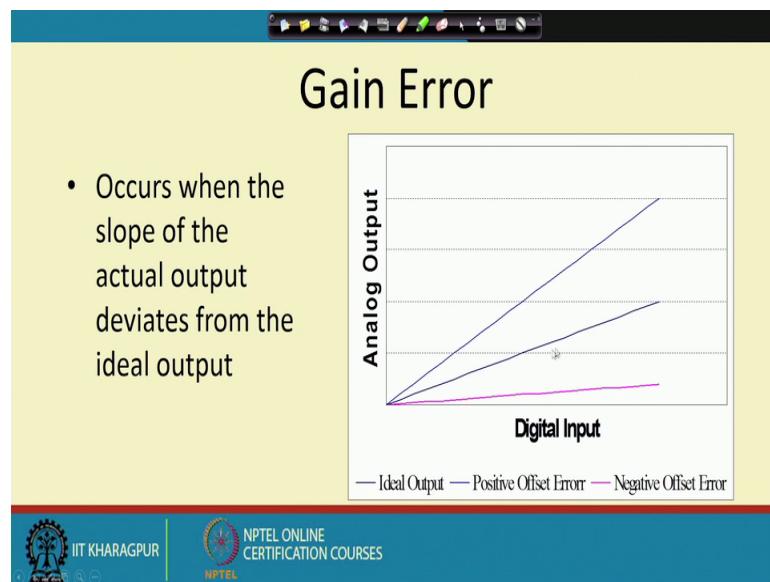
Types of Errors Associated with DACs

- Gain
- Offset
- Full Scale
- Resolution
- Non-Linearity
- Non-Monotonic
- Settling Time and Overshoot

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

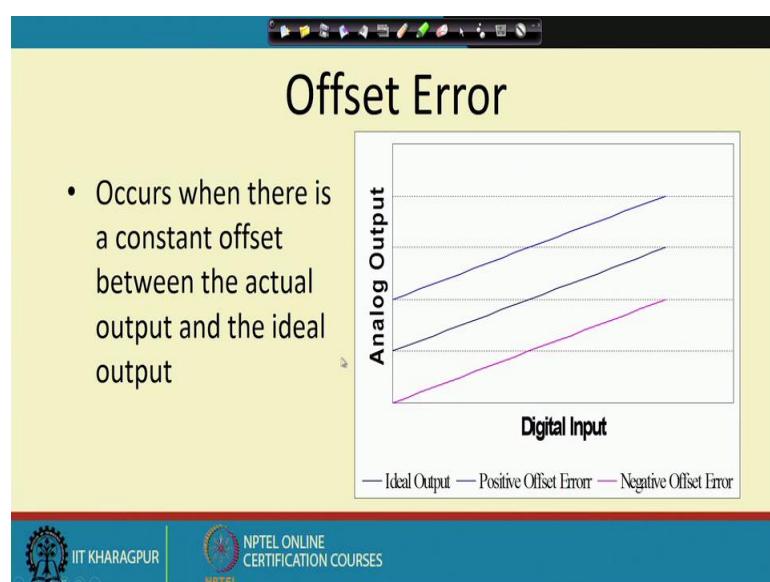
So, the types of errors that we have with DAC is the gain part, offset part, then the full scale full scale range that you get, resolution, non-linearity, non-monotonicity and settling time and overshoot. So, these are the different types of errors that we can have.

(Refer Slide Time: 09:48)



Gain error: so, it is like this. So, this is my ideal output, ok, so, depending upon my DAC design so, I expect the output to be like this then, but it in reality it may so happen that you have you get the output like this pink line or this blue line. So, this is the positive offset error and this is the negative offset error. So, this occurs when the slope of the actual output deviates from the ideal output. So, both of them are straight lines this blue line as well as this purple line both are both are linear both are going in straight lines, but they are deviating from the, the actual output, the ideal output. So, this is known as the gain error. So, we naturally we would expect that the gain error to be less.

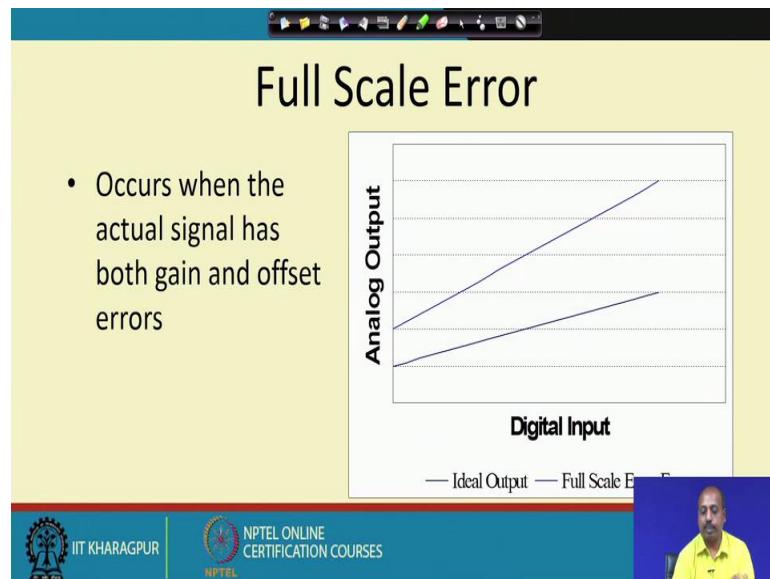
(Refer Slide Time: 10:36)



Then, there is an offset error so, offset error says that, it occurs when there is a constant offset between the actual output and the ideal output. So, this is the ideal output and always there is an offset, ok. So, this offset is fixed whereas, for gain error, so, gain is not fixed, but this offset error so, there is this difference is fixed, ok.

So, so, it may be possible that offset error we may add some constant value and get that or some positive or negative value and that way that offset gets nullified. So, this occurs when there is a constant offset between the actual output and the ideal output. So, that is the offset error for the DAC. Again, we will like that it should be reduced.

(Refer Slide Time: 11:18)



And full scale error, so, this when the output has got both gain and offset errors so, then that is the full scale error. So, if there is only a gain error then this it will be following the similar pattern like this, but here you see that the gain is also changing and offset is also changing. So, none of them are fixed. So, as a result so, you get this gain and offset error. So, this is known as full scale error. So, this is the ideal output and this is the full scale error output. So, that way we can think about, we definitely we like to have DAC where this full scale error is minimum, ok.

(Refer Slide Time: 11:59)

Resolution Error

- Poor representation of ideal output due to poor resolution
- Size of voltage divisions affect the resolution

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Then there is resolution error. So, resolution error is like this; suppose, we have to realize this sinusoidal wave. So, we want to output this sinusoidal wave and that DAC that we have. So, DAC it has got only the voltage levels that we have is this is one level and this is another level only these two levels are available. So, to approximate the sine wave this part so, what the DAC does is that the, it first outputs this, the digital pattern that we feed so, it first output this voltage and then it output this voltage.

Now, if we average over time the behavior becomes more of a triangular type. So, if you just put it on some oscilloscope so, you will find something like say a triangular type of behavior because so, if you average it over the period, ok, so, then you if you integrate over this period, so you will be getting a triangular type of behavior.

Now, instead of having only these two levels, so if we have got more number of levels like here in this case we have got 1 2 3 4 5 levels. So, if the DAC is made to produce all the voltages at these 5 levels then you get a better approximation of the sinusoidal wave. So, this is the resolution error. So, we say that in the second case the resolution is more because I for the same maximum voltage I have got more number of digital or the analog voltage levels that can be produced by the DAC.

So, it is a poor representation of the ideal output due to poor resolution. So, this is the resolution error. So, resolution is poor. So, resolution expression is given by $\frac{V_{ref}}{2^N}$, as we know. Now, if we have do not have many levels available so, if the resolution value is low

then naturally we it may be the case that we do not have so many voltage levels available. So, it will give a poor resolution. The size of voltage divisions this will affect the resolution ok. So, that we have already explained.

(Refer Slide Time: 13:59)

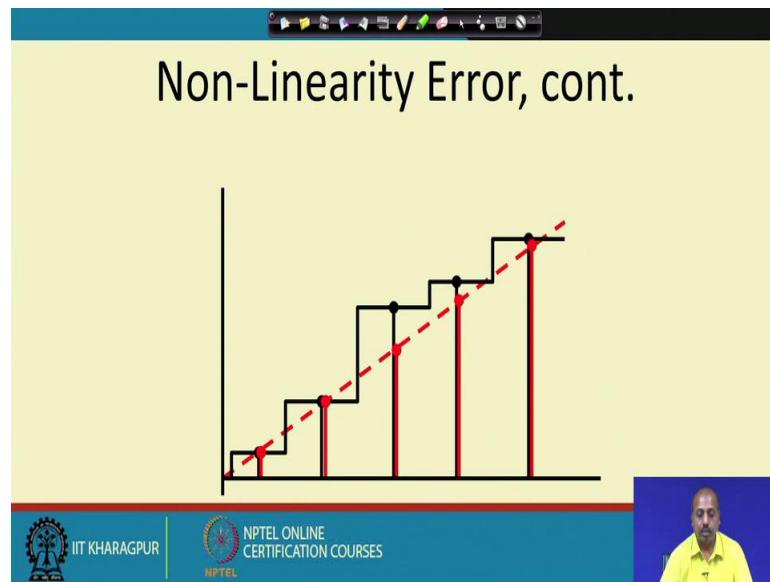
The slide has a yellow header and a blue footer. The title 'Non-Linearity Error' is in bold black font. The footer contains the IIT Kharagpur logo, the NPTEL logo, and a small video thumbnail of a person speaking.

- Occurs when analog output signal is non-linear
- Two Types
 - Differential – analog step-size changes with increasing digital input (measure of largest deviation; between successive bits)
 - Integral – amount of deviation from a straight line after offset and gain errors removed; on concurrent bits

Then there is a non-linearity error. So, occurs when the analog output signal is non-linear. So, this is so, as I said that the analog output that you get we may not be following a linear relationship. So, there were two types of non-linear behavior; one is the differential where the analog step size changes with increasing digital input that is if you between successive deviation so, between successive bits so, you can measure the largest deviation. So, that is the differential nonlinearity error.

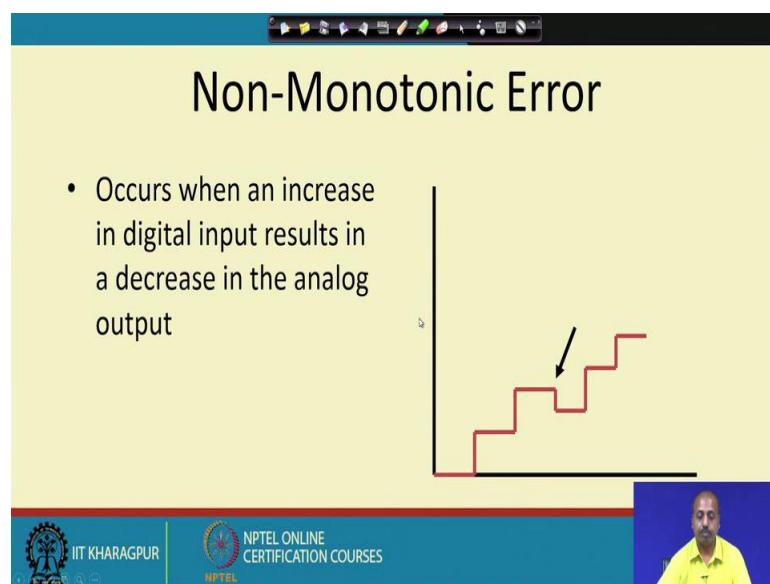
And, the integral errors the amount of deviation from the straight line after offset and gain errors removed so on subsequent bits. So, if we see this so, thus this is the differential type of integral type of non-linearity error. So, both of them can be measured; one is the differential type, that is if you change if you change from one position to another position so, what is the maximum amount of change; ok. So, measure of largest deviation between successive bits so, that is the differential change and the integral changes, so, if you so from the so, if you so, if you take this offset and gain errors so, if you remove this offset and gain error so you will get the ideal behavior, then from that how what is the deviation on concurrent bits.

(Refer Slide Time: 15:20)



So, this is the nonlinearity error as I was telling that this is the ideal behavior that if you put this digital value. So, you get this you should get this analog signal. So, this is the ideal behavior, but due to this non-linearity this step size is different. So, you get the behavior like this, so, it produces different amounts of error. Whereas; the ideal behavior should have been this red dotted line, so, what you get is this black line. So, that is a nonlinearity error.

(Refer Slide Time: 15:52)



Then there is a non monotonic error. So, monotonicity means as it is increasing or decreasing in a particular sequence in a so, as you are increasing the digital input. So, it is expected that the analog output is also increased and if you decrease the digital input the analog output should decrease. So, we are not bothered about linearity here. So, we are suppose we accept that this linearity is not a concern, so, we just want that with increasing digital value the output should increase and decreasing digital value output should decrease.

But, it may so happen where that for some bit patterns some so for some digital inputs output instead of increasing it decreases. Like here; so, when you put this digital value, so it produces this voltage then you then you put this digital value so, it is the analog corresponding analog value. So, like this now after this when this digital value was given the output was like this, after that when we give these digital values output reduces like this. So, though the second digital value that we have given is higher than the previous digital value, but the analog value has reduced.

So, occurred, so, in an increase in digital input results in decrease in the analog output; so, this non-monotonicity error occurs; so, naturally this is not the, this is not desirable. So, my DAC should not have this type of behavior, but sometimes if we know that it has got a non-monotonicity at some points of values. Then we can avoid those values by the processor, processor may avoid those values to be outputted to the DAC or some other care may be taken, but this ideally this should be avoided.

(Refer Slide Time: 17:35)

Settling Time and Overshoot Error

- Settling Time – time required for the output to fall within $\pm \frac{1}{2}V_{LSB}$
- Overshoot – occurs when analog output overshoots the ideal output

The graph illustrates the settling process of an analog output. The vertical axis is labeled 'Analog Output' and 'Ideal Output'. The horizontal axis is labeled 'Time'. A vertical dashed line marks the 'Ideal Output' step change. The 'Actual Analog Output' is shown as a solid line that rises sharply, overshoots the ideal value by $+1/2*V_{LSB}$, and then oscillates as it settles back towards the ideal output level. The time interval during which the output settles within the $\pm \frac{1}{2}V_{LSB}$ range is labeled 'Settling Time'.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then settling time and overshoot error. So, we have already explained to some extent so, let us just revisit it. So, this is time required for output to fall within the $\pm \frac{1}{2}V_{LSB}$. So, this is the as I said that ideally this is this is the behavior. So, it should come to this value, but the analog output becomes like this. So, it shoots up like this and then it comes down so, it goes like this so, from this after this point so, you have got this output contained within the $\pm \frac{1}{2}V_{LSB}$ voltage. So, as a result we say. So, we say that when it comes within this range so, that is the settling time.

So, the time needed from the change in the digital input to the change to the point till this analog output comes with the $\pm \frac{1}{2}V_{LSB}$. So, that is the settling time for the DAC. And overshoot, it occurs in the analog signal analog output overshoots the ideal output. So, this is the overshoot. So, whether it is desirable or not, that is a question. So, sometimes if we if the circuit is too sensitive then we may not like this overshoot to be there, but it cannot change instantaneously. So, normally we get a behavior like this only.

(Refer Slide Time: 18:57)

The slide has a yellow background and a blue header bar. At the top, it says 'Applications'. Below that is a bulleted list of five items. At the bottom, there's a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Digital Motor Control
- Computer Printers
- Sound Equipment (e.g. CD/MP3 Players, etc.)
- Electronic Cruise Control
- Digital Thermostat

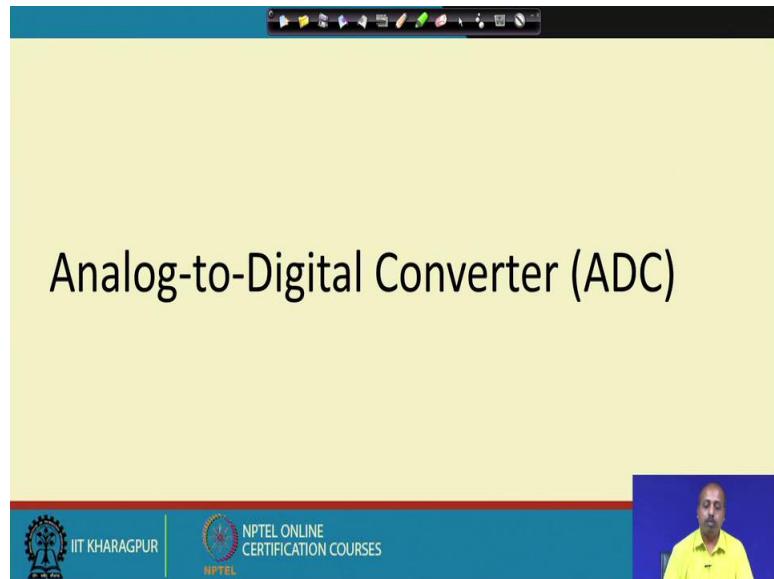
So, where are we going to apply these DACs? One possible application of DAC is the digital motor control. So, digital motor control; so, motor would require some analog voltage so, as you increase the voltage then its speed of operation increases. So, as you reverse the polarity of the voltage then at the motor rotates in the reverse direction like that. So, from a digital signal so, if you are going to control the motor so, then you need a DAC, so that this digital value is converted to analog and then it is applied to the motor. Computer printers so, printers you have to activate some of the keys and then accordingly the key should go up and it should get printed depending upon the printing technology that we have. So, there also we have got applications of this DAC.

Sound equipment like these audio players basically. So, the volume or the pitch or so, that the signal that is controlled by the processor and then when it is given to the player so, before that this digital value has to be converted into analog. So, most of the songs and media data that we have a digital in nature; so, when you are trying to put it onto a player. So, we have to convert it into analog values.

Then cruise control like you so, you have to control some vehicle or a missile in some direction. So, there also we have we go for digital control because of this error tolerance and all, but then how to control that cruise, so, that becomes and that, so, for that we need this digital to analog converter and digital thermostat so, basically these temperature

control mechanism. So, there also you need this digital to analog converter for doing controlling the thermostat.

(Refer Slide Time: 20:46)



Next we will be looking into another module which is known as analog-to-digital converter or ADC. So, this is the other way. So, digital to analog converter, so, it was converting a digital signal to analog signal. This analog to digital converter it will convert an analog signal to a digital signal and once it is from the environment the signals that are coming there analog in nature and then we are converting them into digital value and then the this digital value is being fed to the processor for processing.

(Refer Slide Time: 21:21)

Background Information

- What is ADC?
- Conversion Process
- Accuracy
- Examples of ADC applications

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

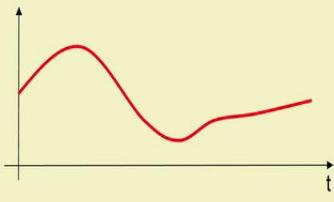
So, we will be discussing about this point like what is ADC? What is the conversion process? What is the accuracy of an ADC and some examples of ADC application. So, we will go in this sequence.

(Refer Slide Time: 21:31)

Signal Types

Analog Signals

- Any continuous signal, that is a time varying variable of the signal, is a representation of some other time varying quantity
 - Measures one quantity in terms of some other quantity
 - Examples
 - Speedometer needle as function of speed
 - Radio volume as function of knob movement



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, just go back so, if we go back into the types of signals so, at the beginning of our course so we have seen that there are general there are one type of signal which is which is the coming from the nature so, it is the analog signal. So, most of the signals that you get are analog in nature for our ease of processing we convert them into digital, but the

actual signal that we get from the environment or the actual signal that we need to produce to the environment for controlling it some equipment or something. So, that is going to be an analog signal.

So, any continuous signal that is that can be represented by a time varying variable so, is a is an analog signal and it is representation of some other time varying quantity. So, the time is varying, so, the time is one parameter and this time varying quantity is what we want to represent. So, for example, if we are representing the distance covered by a train so, with the passage of time so, we can we can see you can find out what is the distance covered and then the, if the distance that is covered is coming is sensed by some sensor and then it is coming to us. So, that way it is going to be some voltage value that is there.

So, basically or say the temperature monitoring system so, with the passage of time temperature is varying, now the temperature is the time varying variable that we have. Now, the temperature value so, it by means of temperature sensors so, they are converted into electrical signal and as a result that electrical signal becomes a time varying signal and that represents the original time varying signal which is the temperature. So, that is why the definition is like this that it represents a time varying signal in terms of another time varying electrical signal.

So, measures one quantity in terms of some other quantity. So, for example, speedometer needle as function of speed. So, speedometer needle so, it represents the speed of the vehicle. So, this is the needle is moving so, that is a physical thing that we have physical signal and then this actual physical signal that is changing is the speed of the vehicle. So, that way one signal is representing the, another time varying variable.

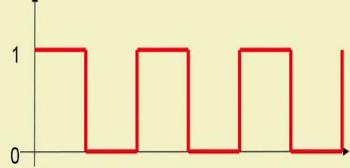
Then radio volume as function of knob movement. So, if we say there is a volume knob of some device and if we just rotate that volume knob, then the volume is either increasing or decreasing depending upon the turning. So, the turning is the actual mechanical variable that is changing and then that is going to affect the volume. So, volume is becoming the signal that we see from the electronic side.

(Refer Slide Time: 24:20)

Signal Types

Digital Signals

- Consist of only two states
 - Binary States
 - On and off
- Computers can only perform processing on digitized signals



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another type of signal that we know is the digital signal that consists of only two states binary states or on-off states. So, they so the so the either 0 either low or high often represented as 0 or 1 or on or on and off like that. So, computers can only perform processing on digitized signals. So, that is the problem. So, we have got this the digital signals only in the computer system.

(Refer Slide Time: 24:45)

Analog-Digital Converter (ADC)

- An electronic integrated circuit which converts a signal from analog (**continuous**) to digital (discrete) form
- Provides a link between the analog world of transducers and the digital world of signal processing and data handling



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this analog to digital conversion; so, this is an electronic integrated circuit which converts a signal from analog to digital form. So, analog is continuous and digital is a

discrete form and it provides a link between the analog world of transducers and the digital world of signal processing and data handling. So, transducers so, they are actually devices which can sense some physical quantity and convert it into electrical signal. So, that is the transducer and once we have that once we have some physical data converted into electrical signal. So, that physical data may be say temperature, may be pressure, may be some gas or whatever. So, there can be different types of physical quantity that we would like to that we like to measure and that is converted into some electrical signal by means of the transducers.

And, then they are converted once if they are converted into digital signal. So, they are now handled as by this is the computers or other digital signal processing equipments and the data is handled accordingly in the digital form.

(Refer Slide Time: 25:54)

Analog-Digital Converter (ADC)

- An electronic integrated circuit which converts a signal from analog (continuous) to digital (**discrete**) form
- Provides a link between the analog world of transducers and the digital world of signal processing and data handling

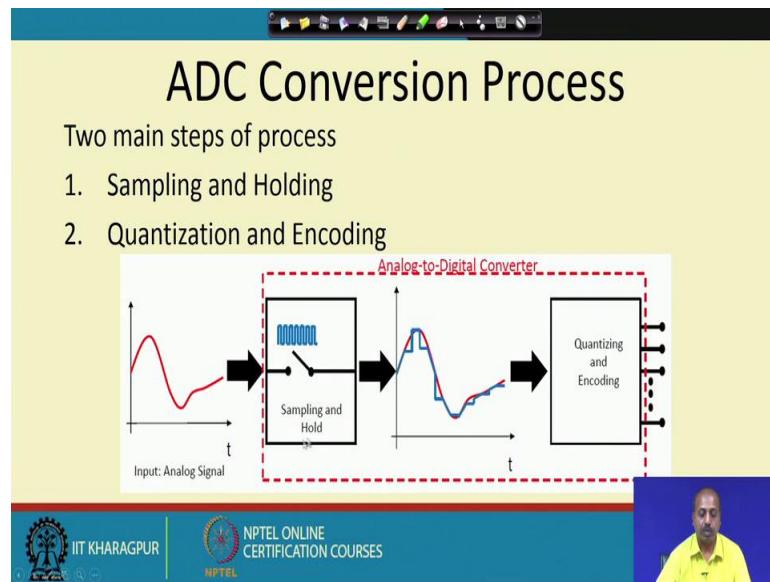


IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES



So, an electronic integrated so, this ADC is this ADC is an electronic integrated circuit which converts the signal from analog to digital form and it provides so, this is basically this signal that we have. So, maybe we just look into this signal at some discrete points of time and we see that its values are like this. So, this is basically a sampling of the signal at different time instants.

(Refer Slide Time: 26:21)



So, the ADC conversion process so, it has got two main steps, one is called the sampling and hold sampling and holding and the second part is the quantization and encoding. So, what is happening so, this is the actual analog signal. So, there is a sampling and hold circuitry which will sample this signals at some intervals of time. So, this so, though the analog signal is continuous, but to convert into digital signal so, we need some value at some fixed point of time. So, that value can be converted.

So, I cannot design a circuit that can go on converting these analog values continually. So, instantaneously it cannot convert into the value, because the circuit will have its own delay. So, if the input is not stable for that much period of time. So, I cannot convert it into any other form. So, the so, digital, any digital circuit it will require the input to be constant for some amount of time.

So to ensure that the input is constant for some period of time so, the input is sampled at some periodic intervals of time; so, you can think of this sampling as a switching as a switch and which has got this is. So, this sampling is done when the switch signal value is high, the switch is closed you can say, when the switch is closed then the, this value comes here and when this switch is open that is a low then here you don't get the output. So, the previous whatever value was there so, that continues to be available here.

So, that is if this is the red line is the analog signal, then depending upon the pulse that we are getting so, maybe if during this time period signal the switch is closed, as a result you

get this part of the output, but from this point onwards the sampling switch is open. So, you don't get any further value. Again, at this point the sampling volt switch is closed. So, again you get this part of the red signal again it is closed on this point. So, you don't get this red part again you get it from this portion.

So, that way whenever the switching gate is closed so, you get the analog signal fragment available at the at this output point and whenever it is closed whenever it is open so, it does not get the thing. So, when this signal when this switch is open then the digital circuit is entrusted with converting the analog sample that it has got into the digital value and the sampling rate it should not be very fast. So, basically what we need is that the digital circuit should be given enough time for getting it converted into the getting the analog value converted into digital value and then only the next sample should come.

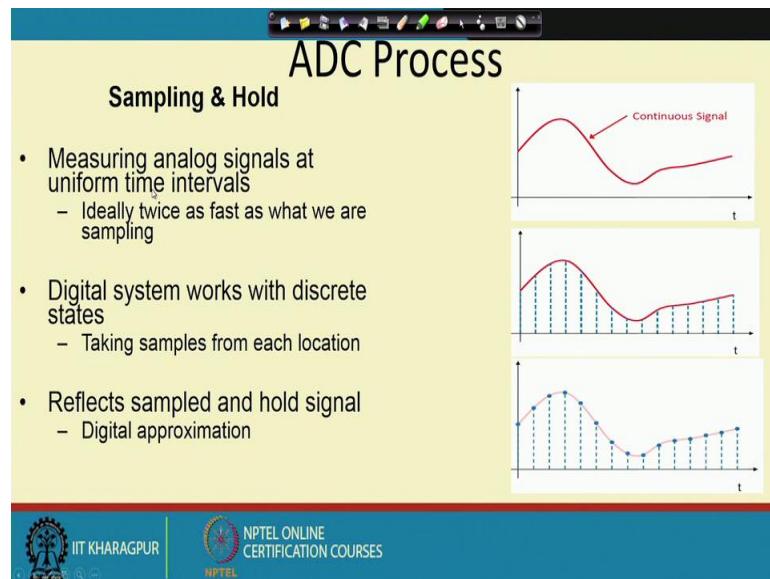
So, you can understand that if you are having a very fast varying signal then this analog to digital converter it is also going to it is also needed to be very fast and after it is converted into these digital the these samples and then this analog this sample analog samples they are quantized and encoded to get the final digital value. So, we will explain this as we proceed in this portion. So, essentially there are two parts. So, one is called the sampling and hold part and another is the quantization and encoding part. So, we will first look into the sampling and hold circuitry and then we will go to this quantization and encoding part.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 39
Data Converters
(Contd.)

So, this ADC process, this analog to digital conversion process. As I said the first step is the Sampling and Hold part.

(Refer Slide Time: 00:25)



So, it is for measuring analog signal at uniform time intervals. So, ideally, it should be twice as fast as what we are sampling. So, this is, this comes from the Nyquist sampling theorem which says that if you are sampling some analog signal at t_0 and later on you want to reconstruct it from the samples. Then, we need at least the sampling rate should be at least twice the frequency of the input signal.

So, you can look into that reference in the communication books, the signal processing books. But anyway, so if we follow by following that property, so ideally, we should do this sampling twice as fast as what we are sampling. So, digital systems work with the discrete states. So, we it takes the samples of each location like this is the continuous signal that we have. So, if this is taking as samples at this steps so, we get the sampled, we reflect

the sampled and hold signal. So, this is the thing that we get. So, you get these values at these points. So, this blue dots, so these are the sampled values that you get.

So, for reconstructing the signal, so we have to start with this and do the reconstruction, but reconstruction is not the objective of this particular course. So, we would like to see what is the, we would like to convert this values into some digital values may be at the receiving end. So, once it is digital values are received, it will be converted back on to analog and finally, it has to be converted into continuous signal to get back the original analog signal. So, that is why the sampling should be at least twice as fast as the input signal frequency, input signal frequency.

(Refer Slide Time: 02:12)

The slide has a yellow background. At the top center, the title 'Sample and Hold Circuit' is displayed in a large, bold, black font. Below the title, there is a bulleted list of five items, each describing a characteristic of the sample and hold circuit. At the bottom of the slide, there is a red footer bar. On the left side of the footer, there is a logo for IIT Kharagpur and the text 'IIT KHARAGPUR'. In the center, there is a logo for NPTEL and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

- The time during which sample and hold circuit generates the sample of the input signal is called **sampling time**.
- Similarly, the time duration of the circuit during which it holds the sampled value is called **holding time**.
- Sampling time is generally between **1µs to 14 µs** while the holding time can assume any value as required in the application.
- Capacitor is the heart of sample and hold circuit. The capacitor charges to its peak value when the switch is closed, i.e. during sampling and holds the sampled voltage when the switch is open.

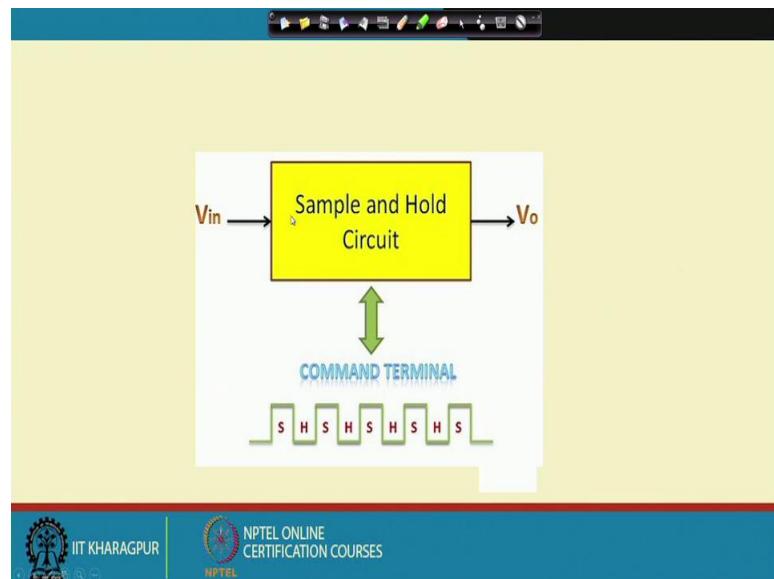
So, we will look into this sample and hold circuit. So, time during which the sample and hold circuit generates the sample of the input signal is called the sampling time. So, as the name as the, has got two parts sample and hold; so, naturally, so it has got two functions; one is the sample, sample, function. So, the circuit gets the sample from the input signal and then there is a hold time the time duration of the circuit during which it holds the sampled value is called the holding time. So, we have got sampling time and holding time.

So, typically the sampling time varies between 1 microsecond to 14 microseconds and the holding time can assume any value as required in the application; So, normally, the samplers that we design, so they have got 1 microsecond to 14 microseconds sampling time. So, capacitor is the heart of the sample and holds circuit because once you get the sample,

after that the sample has to be held for some time and capacitor is the instrument that we have, capacitor is the device that we have that can hold the value electrical voltage or current value for some time. The capacitor charges to, it is peak when the switch is closed and during that is during the sampling period and holds a sampled voltage when the switch is open.

So, that is the behavior that we will explained that capacitor will be charge when the sample and switch is closed. So, it will sample the input. So, capacitor will charge to the value equal to the peak value that you get on the analog signal and then it will be not leaky. So, during the hold period should be the capacitor should be able to hold the charge for the hold period. So, that is how the sample and holds are it is expected to behave.

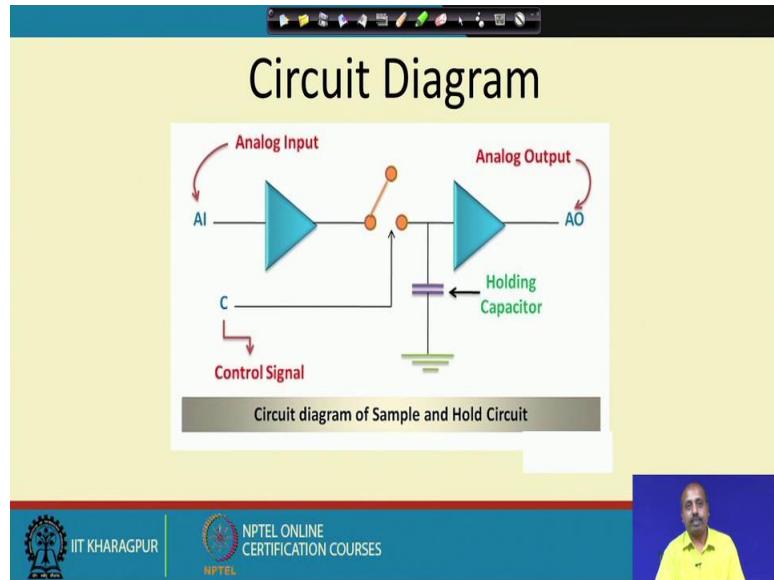
(Refer Slide Time: 04:03)



So, pictorially you can represent it like this. So, as if this input voltage V_{in} is coming and there is a command terminal for the sample and hold circuit. So, it a so when this signal is high, so this is the sampling period and when the signal is low, this is the hold period. So, it alternate between sample and hold period, sample and hold like this. So, during the sample periods this V_{in} value gets connected to this V_o and during this hold time it is expected that the sample and hold circuit will be hold the V_o value. So, it will not change ok. So, then by means of there has to be some capacitor here which will hold the value when this sampling say when the sampling interval is on, then the capacitor gets charged, so that during this hold time the capacitor will can provide the last sample value in the V_o .

line. Then again, the next sample period sampling period come. So, again the next sample value whatever is coming, so, accordingly the capacitor will get charged to up to that point. So, this way the sample and hold circuit is expected to behave.

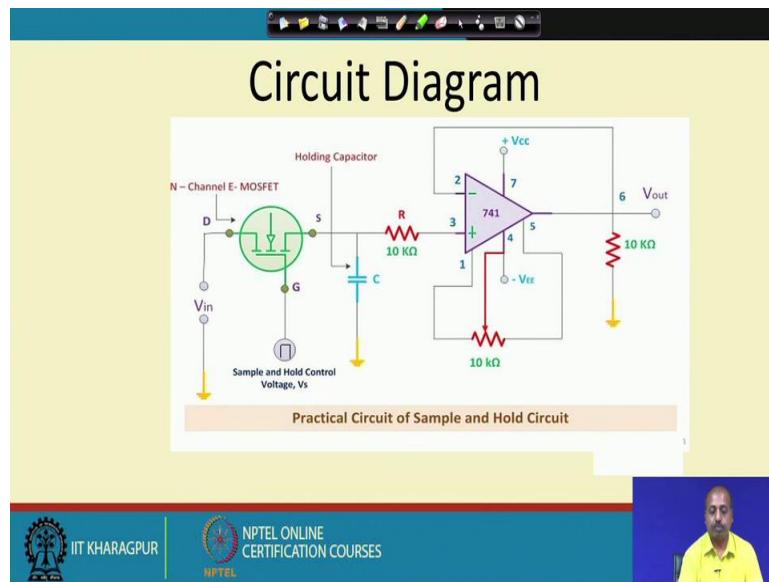
(Refer Slide Time: 05:08)



So, how do we make such a sample and hold circuit. So, basically, we have got this analog input. So, that is amplified because of this, that will be ensuring that the noise and all those effects to be taken care of and there is a control signal C which will control the closing and opening this particular switch ok. So, this is the, so when this when the sampling period is on, so the switch will be taken to this position as a result this analog signal will be coming here. So, it will be charging this holding capacitor. And then, when this analog signal when the control signal is in the hold time, then sig switch will be open as a result this capacitor does not have any discharging path. So, it will hold this value for some time.

So, this, so this leakage of this capacitor should be small enough, so that this charge does not change the significant the, voltage does not change significantly over the hold period. And then, this value will be available at the analog output. So, then this analog output value which is basically a sampled version of this analog input, so, that will be used by the analog to digital converter circuit to convert to digital value.

(Refer Slide Time: 06:22)



So, this is a typical circuit that we have. So, practical circuit, so that uses one n channel MOSFET for this n channel enhancement mode MOSFET for this conversion. So, this gate of this MOSFET, so this is fed this sample and hold control voltage. So, this is basically a clock signal you can say that contains a, during the on time of the clock period, the sample is, the sample is collected and during the off time of the clock period, the sample the input is not disconnected from this sampled value. And, so that way this sample can input can reach the point s only when this gate is high

So, this, this transistor ensures that, so this is basically when this sampling period is on. So, this G is equal to 1, 1. So, this V_{in} that you connected to the V_d , the drain terminal of this n channel MOSFET. So, the voltage available at the s point or the source point and then that will be charging this capacitor, it will be charging this capacitor. So, it will be capacitor will get charged to the V_{in} value.

And then, when this sampling hold period signal goes low then, this transistor is now open. So, this is the, it is the, it is not connected it gets disabled. So, as a result this capacitor will not have, this capacitor will not be charged any more. Then it is connected to this voltage follower type of circuit where it will be it will be transferring this voltage value to this point and the, you will get the sampled value at V_{out} . So, that is the operation of the circuit ok. So, this is the sample and hold circuit.

(Refer Slide Time: 08:12)

The slide contains a bulleted list of 11 points describing the behavior of an N-channel Enhancement MOSFET as a switch:

- The N-channel Enhancement MOSFET is used as the switching element.
- Input voltage is applied through its drain terminal and control voltage to its gate terminal.
- When the positive pulse of the control voltage is applied, the MOSFET will be switched to ON state - it acts as a closed switch.
- On the contrary, when the control voltage is zero then the MOSFET will be switched to OFF state - acts as an open switch.
- When the switch is closed, the analog signal applied to it through the drain terminal is fed to the capacitor.
- The capacitor charges to its peak value.
- When the MOSFET switch is opened, then the capacitor stops charging. Due to the high impedance operational amplifier connected at the end of the circuit, the capacitor experiences high impedance due to this it cannot get discharged.
- Charge is held by the capacitor for a definite amount of time - **holding period**.
- Time in which samples of the input voltage is generated - **sampling period**.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window showing a person speaking.

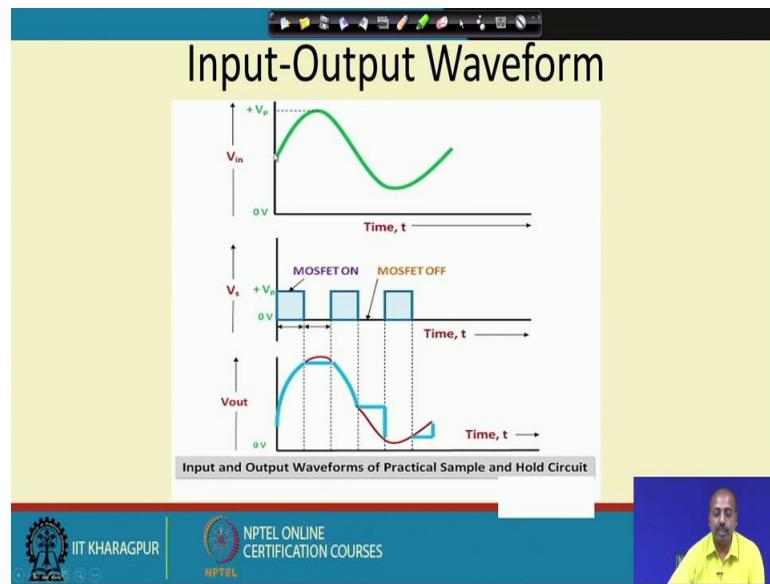
So, overall operation is like this, the n channel enhancement MOSFET is used as the switching element. Input voltage is applied through it's drain terminal and control voltage to it's gate terminal. When the positive pulse of the control voltage is applied the MOSFET will be switched to ON state and it acts like a closed switch. On the contrary, when control voltage is 0 the MOSFET will be switched to OFF state and act as an open switch.

So, that is the behavior of the MOS switch that we have seen. And when the switch is closed, the analog signal applied to it through the drain terminal is fed to the capacitor. The capacitor charges to it is peak value whatever be the input voltage coming. So, it will charge to that and when the MOSFET switch is open, then the capacitor stops charging and due to the high impedance operational amplifier connected at the end of the circuits, the capacitor experiences high impedance due to this, it cannot get discharged.

So, this circuit, so you see that this is connected to this op amp and op amp resistance is, ideal op amp is infinity, so, practical op amp also this input resistance will be infinity. So, close to, will be very high. So, as a result, the capacitor will not get chance to discharge much except in some amount of leakage.

So, charge is held by the capacitor for a definite period of time amount of time. So, that is called the holding period and time in which samples of the input voltage is generated is known as the sampling period. So, we have got a sample operation and a hold operation.

(Refer Slide Time: 09:50)

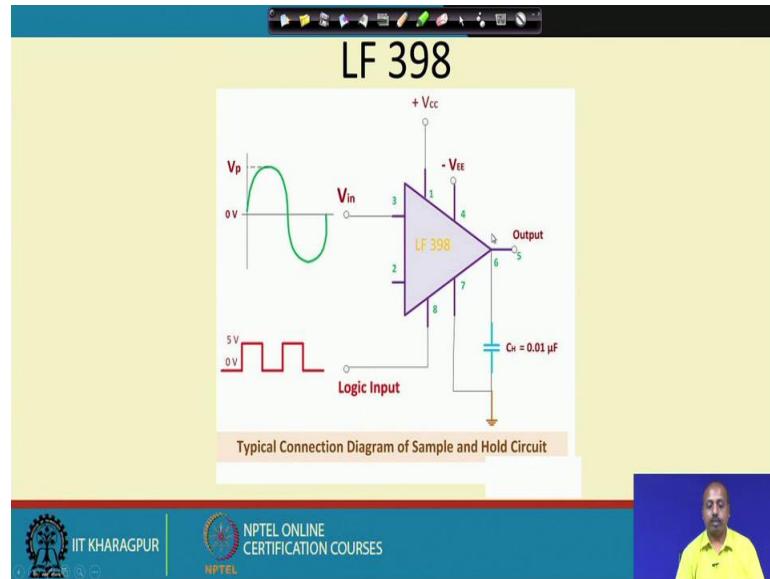


So, pictorially, if this is the input voltage that we have and this is the sampling control signal that we get, so the this is this time this MOSFET is on and this time it is off again, this time it is on, this is off. So, during this time thus capacitor gets voltage like this. So, it charges to this value during hold time. So, it holds the value. So, it holds the value like this and then during this next on period of the MOSFET. So, this capacitor will get this input the capacitor will be getting this input signal connected as a result it will at the end of this sampling period, the capacitor will hold this value and it will hold this value till the next, for the entire next holding period.

Then, the capacitor will again the next sampling period will come and capacitor will be looking into the next sample value next signal value. So, it will go like this. So, in this way, so again it sees this portion and it holds the value. So, there is a problem in the mistake here actually, this portion of the diagram it should shift to this. So, this is this blue line, it should start form here and go like this; so, that that is not shown here correctly.

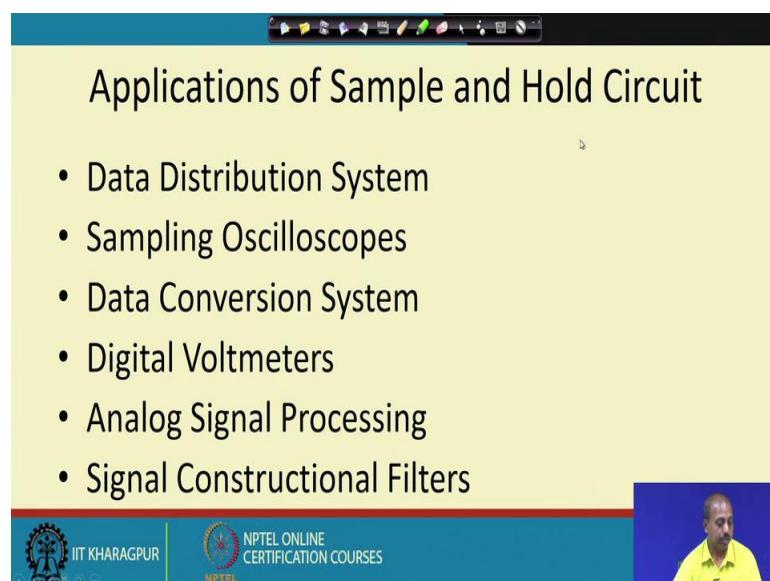
So, what I mean is that because at this point, it has got this part. So, it will be it should be like this. Then it should be after that it will be holding this value. So, it will be going like this. So, this part should be this part is missing. So, this should this should be there.

(Refer Slide Time: 11:46)



Anyway, so next we will be looking into how this circuit can be made to operate. It is a practical chip like LF 398. So, this is a chip that is available using op amp. This is a sample and hold circuit. So, here this capacitor has to be connected externally, the holding capacitor has to be connected externally, but rest of the thing are already there. There is a terminal to give this logic input which is the sampling signal and then the signal to be sampled is applied to this pin number 3 and then V_{CC}, V_{EE} those connections are to be there and this capacitor has to be connected externally like this. So, this way we can have this LF 398 connection for this which is a sample and hold circuit.

(Refer Slide Time: 12:29)



Now, where are you going to use this sample and hold circuit. One application is the data distribution system, so where you need to distribute some analog data to various places, so you first collect the analog data and for collecting the data. So, you have to go for a sampling and hold type of circuit, so that we get the analog samples collected then sampling oscilloscopes there are some oscilloscopes where we need to where the input signal is occurring only once and we need to display that signal. So, for displaying that signal, so it has to collect the samples from the input signal; So, they, so this sampling oscilloscopes, so they collect the input signal samples and then reproduce it on the, on the screen.

Then, the Data Conversion System, so this, so one format to another format or one level to another level if you want to change, then you have to do it. Then digital Voltmeters, so this is another case. So, you have got this analog values available from the, from some circuit, the voltages have various nodes and that is converted to be to, that that is to be converted to digital value. So, you need a need an analog to digital converter. So, at the first stage of that conversion process we have got the sample and hold circuitry.

Analog Signal Processing will definitely need this ADC because this analog signal has to be converted into digital form and for doing the conversion at the very fast step you need a analog to digital converter. Then Signal Constructional Filters, there many we need to construct many signal and accordingly some filters are necessary. So, these filter design, so they also use this sampling and hold circuitry.

(Refer Slide Time: 14:15)

ADC Process

Quantizing

- Separating the input signal into discrete states with K increments
- $K=2^N$
 - N is the number of bits of the ADC
- Analog quantization size
 - $Q=(V_{max}-V_{min})/2^N$
 - Q is the Resolution

Encoding

- Assigning a unique digital code to each state for input into the microprocessor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next once we have done this sampling and some this sampling and hold operation, so we have got the analog samples of data available now that is to be converted to digital value and that goes into goes through two stages; one is called quantization another is called encoding. So, what has happened is that, after you have sampled an analog signal. So, you might have, so if this is the sinusoidal signal that you have that is there may be you have got the samples like this. You have got samples, suppose after doing the sampling. So, we have got the samples like this. So, this portion I have sampled, again in this portion I have sampled, again this portion I have sampled. So, we have got so many voltages.

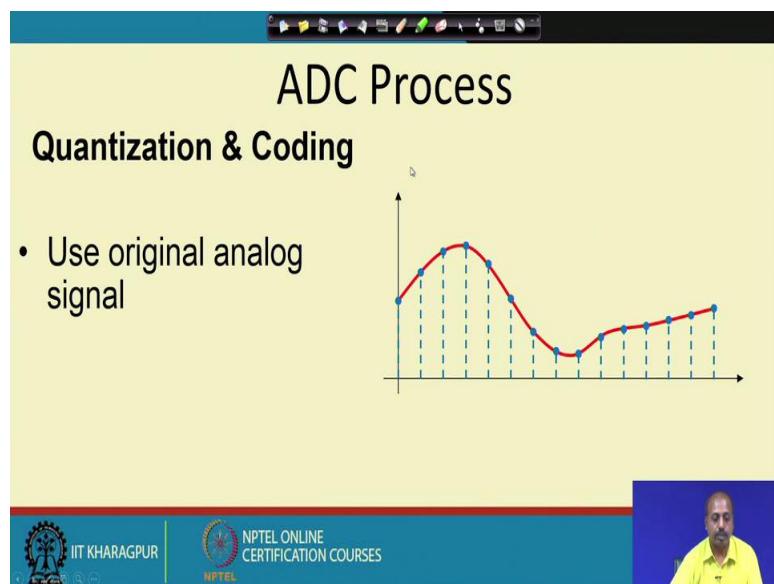
Now, this capacitor accordingly has got charge to say this value, this value, this value. So, it has got charge to all these values. Now, while outputting to, while converting to digital, so my converter may not have so many levels, may not have the capacity to converts so many levels. So, it might have capacity to convert only say, this is the maximum value may be in this range. So, it may it the converter may be able to produce only four such, may be able to consider only four such levels.

So, if it is that, then I can say that this whole thing is divided into four such levels. So, whatever falls in this region for that, it will be outputting this level. Similarly, whatever false in this region it will be outputting this level. So, that way it goes ok. So, I can say that we can separate this input signal into discrete states with k increments. So, at k equal

to 2^n , n being the number of bits of the ADC, so this quantization size q is given by $\frac{V_{max}-V_{min}}{2^n}$. So, and this is known as the resolution of the, this is known as the resolution of the ADC, ok.

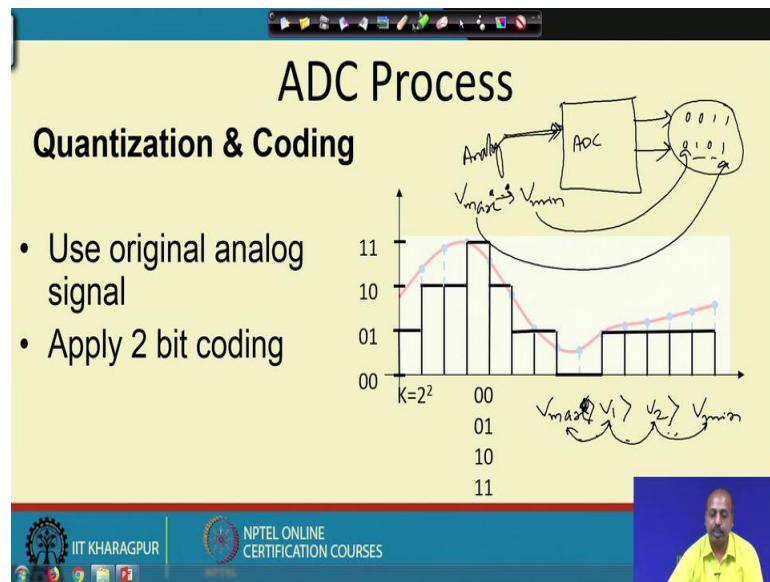
So, this is, so this is like this thing. So Q , so this Q is given by $\frac{V_{max}-V_{min}}{2^n}$. So, if you change one if you want to if the, so this minimum change that it has to, that it, that it can do is basically given by this Q . So, if you want to get a digital change in the digital output for a voltage change less than this, so it is not possible. And then, comes encoding which will be giving some unique code to individual stages.

(Refer Slide Time: 17:16)



So, we will see some case then it will be more clear. So, quantization, so, what we have done in the ADC process. So, this red signal is the original analog signal that we have and then after the sampling and hold operation, so we have we have sample data at some periodic intervals of time and accordingly we have got these samples. So, at this point when I sampled, I got this value. At this point, I got this value. At this point, I got this value. So, these are the analog sample values that I have got at different time instants, fine.

(Refer Slide Time: 17:50)



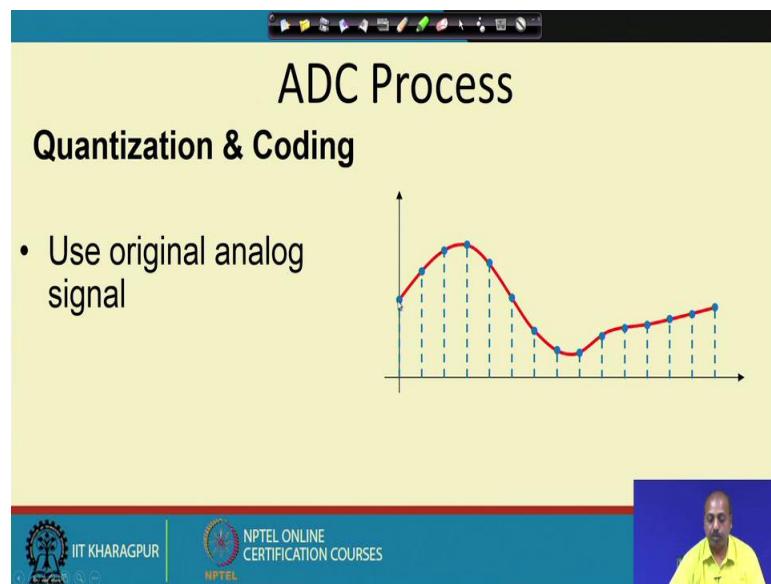
Now, next what is done is that this quantization is applied. Suppose, I have got this 2 bit coding, so 2 bit coding, so I have got four possible levels. So, if this see if this is my ADC, if this is my ADC and it has got 2 bit as output and if this is the analog voltage that I have, so this is the analog voltage that is coming. Then, depending upon analog voltage, so this ADC can either see 0 0, 0 1, 1 0 and 1 1. So, it cannot produce any other input combination. Had it been a 3 bit ADC, then it can produce 8 such alternatives. But with 2 bit ADC, so it can be only this 4, 4 possible alternatives.

Now, if your analog input voltage is in the range say V_{\max} to V_{\min} , V_{\max} to V_{\min} , then for V_{\min} it can output a this one, for V_{\max} , it can give this 1 1, but in between it cannot represent any arbitrary voltage now, all arbitrary voltages because there are only two more levels available. So, what I have to do is I have to decide two points in between which it will be representing. So, finally, I have got this V_{\max} , then value V_1 which is slightly less than V_{\max} , then we have got V_2 and then we have got V_{\min} with the relationship that sorry, with the relationship that V_{\max} is greater than V_1 , V_1 greater than V_2 , V_2 greater than V_{\min} .

Now, for all practical purposes what we would like to do is that this distance between this V_{\max} and V_1 , similarly V_1 to V_2 and V_2 to V_{\min} , they should be same ok. So, this distances, so basically whatever input range I have, so, I need to distribute it equally between all this points all this say 0 0, 0 1, 1 0, 1 1 these outputs.

So, I can say that if this is my, if this is my input range, then, so from, so this is my total input range the. So, this is the minimum value that the signal can produce and this is the maximum value that the signal can produce, accordingly I can say that ok I will be my digital output only 4 level. So, I divide this entire range into 4 levels. So, for 0 0, it will be outputting the minimum value; for 1 1, it will be outputting say this level. Now, in between, I put this 0 1 and 1 0 at some regular intervals. So now, if you consider the sample, so initially initial sample was somewhere here.

(Refer Slide Time: 20:39)

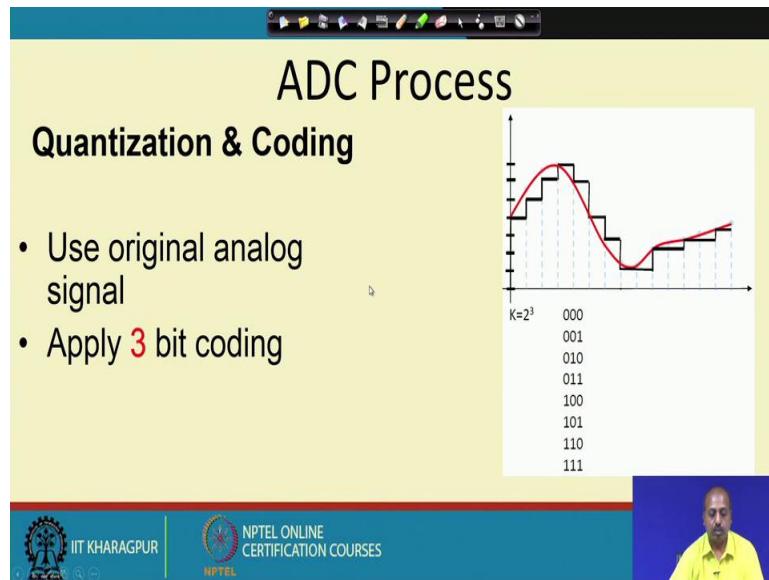


So, for that I can say that it is more than, so it was to the initially sample was here actually in this diagram. So, this is between 0 1 and 1 0. So, it has not reached 1 0. So, I output is 0 1. The second sample that I had is somewhere here. So, that is between 1 0 and 1 1. So, I output 1 0, similarly the third sample is also here. So, there I will show a output a 1 0 because it is not more than 1 1.

The next sample is more than 1 1. So, it is, 1 1 is outputted, then the next one next sample is between 1 1 and 1 0 as a result the output becomes some this level 1 0 and it goes like this. So, these are the voltage levels that I have corresponding to this four possible digital outputs. So, this is the original analog signal is used and accordingly, we have got we have produced like this. So, in reality what has happened is corresponding to this analog voltage we have outputted say this much. Similarly, for this analog voltage we have outputted this much. So, that way, so this quantization process, so it is dividing this, it is dividing this

range into a number of sub ranges and the, this distance, so this distance between the actual value and the value outputted by the quantizer, so, this is called the quantization error. So, it introduces some error into the conversion process. So, this is the 2 bit encoding that we can do for this analog to, so this is by the quantization process.

(Refer Slide Time: 22:22)



So, if I have got 3 bit encoding, then you see the same signal. So, now, I can have eight possible alternatives so, 0 0 0 to 1 1 1, now you see say. So, I can divide this range into 8 possible levels; now this 0 0. So, initial signal value was here. So, I can make it a more faithful representation. Similarly, second sample that I got is somewhere here. So, I make it this one. So, you see that this signal that I output by means of by the quantization process. So, this much more faithfully represents the situation compared to this.

So, that way if I have got more number of quantization levels, then the signal will be approximated better by the quantizer state naturally. So, if I increase the number of quantization levels, then the number of bits needed for encoding will also increase. So, this ADC's number of bits needed will also increase. So, in this case we have got this 3 bit ADC. So, we can we need eight possible steps. So, if you want to make it even more fine, then you have to go for say 16 number of levels. So, that will require 4 bit encoding. So, that way the cost of the ADC will go up as the quantization, number of quantitation levels you are going for high.

(Refer Slide Time: 23:49)

ADC Process

Quantization & Coding

- Use original analog signal
- Apply 3 bit coding
- Better representation of input information with additional bits
- MCS12 has max of 10 bits

$K=2^3$	000	$K=16$	0000	$K=...$
001		0001		
010		0010		
011		0011		
100		0100	1111	
101		0101		
110		0110		
111		0111		

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is basically the thing that is we have got this K equal to 2^3 . So, if we go for K equal to 2^4 then, we can go for even better approximation. So, like this. So, use original analog signal apply 3 bit coding. So, better representation of input information with additional bits. So, MCS 12, so this is one ADC that has got a maximum of 10 bits, so that is quite fine; so, we have to it has got 2^{10} possible steps. So, here actually in this case, we show the situation where K equal to 8 and it changes from this point to K equal to 16 as a result we got a better representation, so of the sample values.

(Refer Slide Time: 24:36)

ADC Process-Accuracy

The accuracy of an ADC can be improved by increasing:

Sampling Rate, T_s

- Based on number of steps required in the conversion process
- Increases the maximum frequency that can be measured

Resolution (bit depth), Q

- Improves accuracy in measuring amplitude of analog signal

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the accuracy of the ADC can be increased by increasing first of all the sampling rate. So, sampling rate if you increase, then it will be this distance between two successive samples will less. So, as a result you will be getting better, you will be get a better input as far as the input signal is concerned. So, like in the, so if you are sampling at this point and this point then in between whatever values are coming, so they are getting ignored. So, they are not faithfully, they are not faithfully produced that the output.

So, if you can reduce this sampling rate then we can make it better. So, that is by increasing the sampling rate. So, based on number of steps required in the conversion process, so that can be increased and increases the maximum frequency that can be measured. So, if you are doing it closer means you are, you are sampling at a higher rate. So, by Nyquist criteria, so, this is this is also going to improve the signal frequency that you can sample. Second possibility is by accuracy improvement is by improving the resolution. So, improving the resolution means you reduce the distance between two points, two successive voltage levels that this quantizer can output.

So, so naturally, so, it as we have seen that giving a 8 level quantizer or 16 level quantizer is doing much better than 4 level quantizer. So, but the point is that as you are going for more and more levels of quantization the accuracy will improve, but this digital number of digital outputs will also increase. So, that is, that is another problem.

Anyway, so we can go for this resolution high resolution, so which is non as the bit depth that improves accuracy in measuring the amplitude of analog signals. So, if you are trying to go for a better ADC, so it may be better in terms of sampling rate, it may be better in terms of the resolution.

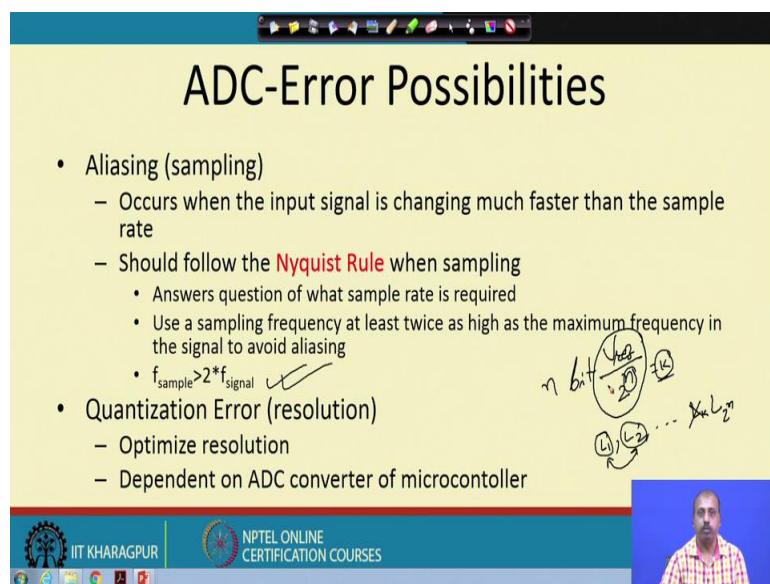
So, we will see this ADC designs as we proceed further and we will see that there are different types of ADCs giving rise to different sampling rates and resolution. As a result, cost of those ADCs will also vary and a high resolution high the sampling rate ADC will be very costly. So, that will also determine depending upon the applications. So, we can choose between the types of ADCs and accordingly you can come to some proper ADC that can be used for your purpose.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 40
Data Converters (Contd.)

So in case of ADC Analog to Digital Converters so, there are several errors that can come up. So, 2 major error are aliasing error and quantization error.

(Refer Slide Time: 00:19)



ADC-Error Possibilities

- Aliasing (sampling)
 - Occurs when the input signal is changing much faster than the sample rate
 - Should follow the **Nyquist Rule** when sampling
 - Answers question of what sample rate is required
 - Use a sampling frequency at least twice as high as the maximum frequency in the signal to avoid aliasing
 - $f_{sample} > 2 \cdot f_{signal}$
- Quantization Error (resolution)
 - Optimize resolution
 - Dependent on ADC converter or microcontroller

Hand-drawn diagram: A bit stream represented by a sequence of circles containing numbers. The first circle has 'bit' written above it. Arrows point from the numbers in the circles to the right, with some numbers crossed out. Below the stream, there are more circles with numbers and arrows pointing to the right.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, aliasing error so, or sampling errors so, this occurs when the input signal is changing much faster than the sampling rate. So, we have seen previously while discussing on sample and hold circuit, that the sampling rate should be sufficiently high so that signal can be reconstructed at the receiving end.

So, here also when we are converting one analog signal to digital signal, the sampling rate should be sufficiently high so that we have enough information about the signal and later on when that digital signal is processed. So, this digital signal is having enough information about the analog signal so that later reconstruction maybe possible. So, there is a theorem called Nyquist rule which is to be followed for this sampling.

So, this sampling in Nyquist rule so, it answers question of what sample rate is required. So, it is required that a sampling frequency should be at least twice as high as the maximum

frequency in the signal to avoid aliasing. So, maximum frequency of a signal in a signal means, in the in the analog signal for example, if there is a say music that is playing. So, there are several frequencies that are there

So, if you if you want to get a true picture about the music so, it you have to find out what is the maximum frequency in that music and the sampling rate should be twice that of the highest frequency. So, if you do not do that so there will be aliasing, so, later on the reconstruction maybe problematic so, you may not get the actual the original signal back in its true form.

So, the third point here is actually telling us that; what is the formula. So, this one the sampling frequency should be twice the signal frequency or you can say the maximum frequency in the signal so, this is the aliasing error. Another type of error that can come we have seen in the quantization process. So, quantization as we have discussed in the last class so, we the while converting from analog to digital signal so, we do not have all the levels of analog signals available for the, for to be so, converted into digital form.

And if I have got an n bit ADC, then the number of different digital pattern is 2^n . So, this $\frac{V_{ref}}{2^n}$ so many different voltage levels are only possible to be recognized by the analog to digital converter. So, if this levels are named as a $L_1 L_2$ up to say L so, this quantity being equal to say k so, this L_k .

So, between L_1 and L_2 there is a change in voltage. So, any signal which lies between L_1 and L_2 any analog signal. So, that is quantized as L_1 . Similarly anything lying between L_2 and L_3 will be quantized as a L_2 . So, naturally it introduces error. So, this error is the quantization error. So, if we want to reduce this quantization error so, we have to increase the resolution. So, if I want that this value of k should be large, then this number of levels should be large then what we have to do is that we have to. So, we have; so, there are 2^n levels so, if we want that to be large than the value of n should be large.

So, as a result this is the voltage difference. So, this is not L_k . So, this is L_1, L_2 up to L_2^n . So, if I want that we I have large number of levels, then this value of n should be sufficiently large so; that means, that this converter will be more complex.

So, this is dependent on the ADC convert of the microcontroller. So, they have they very often. So, the underlying processor or the microprocessor or microcontroller so, they have

got built in ADC, and this ADC, ADC for the ADC the resolution is fixed. So, we cannot change it so, error is dependent on that. So, these are the 2 sources of error in case of ADC applications.

(Refer Slide Time: 04:44)

ADC Applications

- ADC are used virtually everywhere where an analog signal has to be processed, stored, or transported in digital form
 - Microphones
 - Strain Gages
 - Thermocouple
 - Digital Multimeters

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, ADCs are used virtually everywhere, where an analog signal has to be processed, stored or transported in digital form like microphones, strain gages, thermocouple digital multimeters. So, everywhere you will find some application of ADC; so, ADC.

(Refer Slide Time: 05:02)

Types of ADC

- Successive Approximation A/D Converter
- Flash A/D Converter
- Dual Slope A/D Converter

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will be look into different types of ADCs that are available. So, of course, this is this list is not exhaustive. So, in fact, this analog to digital converter design by itself is a research area.

So, there so, even you today you can find lots of research papers on this analog to digital converters, but the purpose of this course is to understand the basic analog to digital converter designs. So, we will be looking into three alternative designs. So, one is known as successive approximation AD converter, another is flash type AD converter, another is dual slope AD converter.

So, successive approximation AD converter so, this is a serial converter. So, this we will try to approximate it will it will try to guess like what is the value of the analog signal, accordingly it will set some bits and then if the signal value is higher than the guessed value, then we will be further turning on some more bit. So, it will go like this, so, we will see how this is done.

So, in a step by step fashion so, if there are say n bit for an n bit ADC so, it will be doing in n steps by setting and resetting the bits. On the other hand this flash type ADC is they are pretty fast so, their. So, this it uses a number of comparators and these comparators they work in parallel. And accordingly this flash ADC will be setting this comparator outputs to high such that we can get the corresponding digital bit pattern to understand like what is the corresponding digital value. And this dual slope AD converter so, this uses some reference signal and with respect to that it tries to see like what is the amount of what is the level of the given analog signal so, that way it is doing a dual slope AD converters.

(Refer Slide Time: 06:57)

Successive Approximation ADC

- Elements
 - DAC = Digital to Analog Converter
 - EOC = End of Conversion
 - SAR = Successive Approximation Register
 - S/H = Sample and Hold Circuit
 - V_{in} = Input Voltage
 - Comparator
 - V_{ref} = Reference Voltage

SAR: Successive Approx Reg.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, we will see them one after the other. The first one is the successive approximation AD converter. So, here what we have is that so, this is there is a successive approximation register this SAR. So, the SAR stands for Successive Approximation Register. So, this successive approximation register so, if I have got an n bit ADC then the size of this successive approximation register is also n.

So, interestingly it uses one digital to analog converter as a module in this ADC. So, while designing this ADC we need a DAC to be implemented DAC, to be available. So, the there are a few signals like clock signal means that in every clock. So, it will be doing some operation. So, this SAR registers so, it has got a clock so, that at every clock output at the clock active point, so, the bit pattern will be available on these lines. There is an end of conversion signal or EOC. So, this tells the outside world that the conversion is over.

And also we have got a sample and hold circuit. So, this sample and hold circuit the input signal is applied here. So, this will be sampler the value will be sampled and held for some time. It will be doing a comparison and then the comparator output will be fed to this SAR. So, that the SAR will be changing in the bits so, we will see it how it changes. Now after n such steps this EOC signal becomes high, telling the outside world that for this current sample the conversion is over.

So, whatever be the value of this D_0 to D_{N-1} so, that gives me the digital equivalent of the analog input signal V_{in} fed here. So, you have got this DAC which is the digital to analog

converter, end of conversion signal successive approximation register SAR, the sample and hold circuit SH, input voltage V_{in} , a comparator and a reference voltage. So, this comparator is simply an operational amplifier, which is connected in open loop configuration and you see that if this signal is more than the output from the DAC is more than V_{in} , then this will be high and this will go to $+V_{CC}$.

So, if we assume that this has got supply voltage of connected to V_{CC} and ground then what will happen is that. So, when this value is higher than this then the output will be a high, and later on when this V_{in} becomes less than this V_{in} becomes higher than this DAC output then it will become a loop. So, that way this is a comparator that works in this is an operational amplifier that gives only 2 levels of output $+V_{CC}$ and ground ok. So, that is and V_{ref} is the reference voltage. So, how does it operate? So, let us try to understand.

(Refer Slide Time: 10:05)

The slide title is "Successive Approximation ADC". Below the title is a section heading "Algorithm" with a bullet list. To the right of the list is a hand-drawn diagram of a digital-to-analog converter (DAC). The diagram shows a binary number $0\ 1\ 0\ \underline{1}\ 0\ 0\ 0$ with a circled '1' under the fourth bit. An arrow points from this number to a box labeled "DAC". Another arrow points from the "DAC" box to the formula $V_{ref}/2$. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

- Algorithm
- Uses an N-bit DAC and original analog results
- Performs a binary comparison of V_{DAC} and V_{in}
- MSB is initialized at 1 for DAC
- If $V_{in} < V_{DAC} (V_{REF} / 2^{(N-1)})$ then MSB is reset to 0
- If $V_{in} > V_{DAC} (V_{REF} / 2^{(N-1)})$ successive bit is set to 1
- Algorithm is repeated up to LSB
- At end DAC in = ADC out
- N-bit conversion requires N comparison cycles

So, this SAR the successive approximation type ADC, it uses an N bit DAC and original analog results. And it performs a binary comparison between voltage coming out from DAC V_{DAC} and V_{in} . So, this is the V_{DAC} coming out of the digital to analog converter. So, this is the V_{DAC} and V_{in} they are compared, and then this is compared. So, V_{DAC} is compared with V_{in} initially the most significant bit is initialized to 1 for the DAC

So, this successive approximation register what is done is that, the most significant bit that is D_{N-1} is set to one and rest of the bits are set to 0 as a result this DAC will be outputting a voltage, which is we are equal to $V_{ref} / 2$. So, this $V_{ref} / 2$ is compared with V_{in} and if V_{in}

is higher than $V_{ref} / 2$ that means, this output will be this comparison output in that case a. So, V_{in} is less than a V_{DAC} ($V_{ref} / 2^{(N=1)}$) so, then MSB is reset to 0. So, if V_{in} is less; that means, the value that we have guessed through this successive approximation register, that voltage value is higher than the V_{in} value. So, as a result in the digital counts the MSB should not be 1 so, it should be 0 so, MSB is reset to 0. On the other hand if V_{in} happens to be greater than V_{DAC} which is in this case in this case it is n equal to 1.

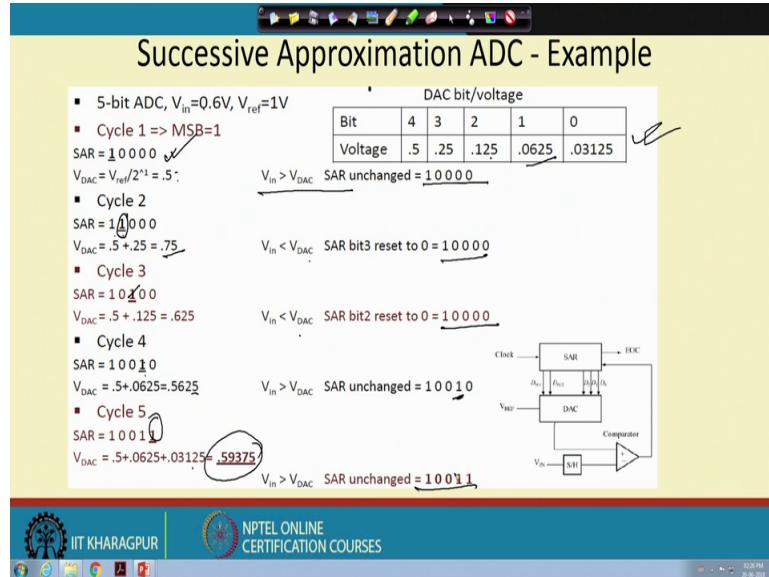
So, it this actually if this comparison fails a V_{in} is not less than V_{DAC} so, this is say a bit confusing here. So, we should better say that if V_{in} is greater than V_{DAC} , where n equal to 1, n equal to 1. So, that means, in this previously we have said this MSB to be equal to 1 in the SAR register and all other bits were 0. So, with that we have got the output from this DAC which is $V_{ref} / 2$. Now if this V_{in} happens to be more than $V_{ref} / 2$ that is a second condition this condition is true.

So, this bit must continue as one now I have to try to set this bit ok. So, that is that is what is done. So, if V_{in} is greater than V_{DAC} then successive bit is set to 1. So, this bit I try with setting equal to 1 now, these 2 bits being 1 so, V_{DAC} value will be even higher and then again the comparison will be done. So, if now it happens that this is less in that case this will be reset to 0 and then this will be turned to 1 so, it will go like this.

On the other hand if we if it is the initial point itself V_{in} is less than V_{DAC} . So, this bit is reset to 0 and we set this second bit to 1 and it proceeds like this. So, this way the algorithm is repeated till the least significant bit position at the end DAC input equal to ADC output. So, that so, their ADC output is that one and this for n bit we need some n comparison cycles for this purpose.

So, this way this conversion takes place.

(Refer Slide Time: 13:43)



So, we have got an example that will tell how this is going to happen. Suppose we have got a 5 bit ADC with V_{in} equal to 0.6 volt, the analog voltage value is 0.6 volt and V_{ref} reference voltage is 1 volt. So, in the first cycle MSB is set to 1. So, this is the 5 bit pattern that we have. So, this is the 5 bit pattern that we have so, there this MSB is set to one.

So, V_{ref} or the V_{DAC} value is $V_{ref} / 2$ that is 0.5. So, when we compare 0.5 and 0.6 so, V_{in} is greater than V_{DAC} . So, as a result the successive approximation register content remains unchanged in the second cycle this SAR the second bit of the SAR is set. So, it becomes 11000. So, DAC bit per voltage. So, if you for every bit, so in this table we have listed the voltage value corresponding voltage value.

So, in cycle 2 when these 2 bits are set so, 0.5 plus 0.25, so that will be the V_{DAC} output so, that is 0.75. So, 0.75 is more than 0.6 volt. So, in this case V_{in} is less than V_{DAC} as a result this bit will be reset. So, it the DAC content becomes this SAR content becomes like this then we come to the third bit. So, third bit is turned 1 and third bits configuration is 0.125.

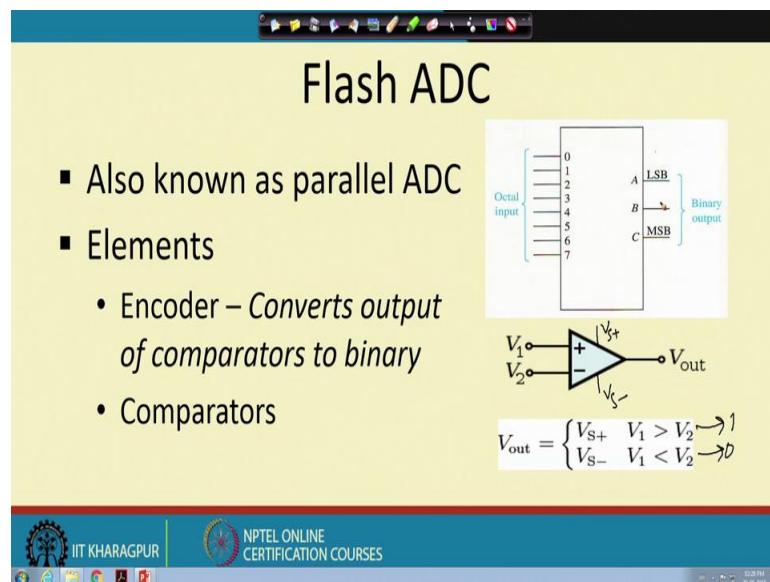
So, 0.125 is added. So, it becomes 0.625 again V_{in} happens to be less than V_{DAC} so, this bit will be this set so, we again remain at this position. In the cycle four we will be trying out the fourth bit. So, this bit is turned one and then so, the contribution is 0.0625. So, it

becomes 0.562. So, 0.5625 is less than V_{in} . So, V_{in} is greater than V_{DAC} . So, this bit survives. So, this bit remains as 1; in the fifth cycle.

So, in the fifth cycle so, we turn on the LSB and then the contribution is added. So, the value becomes 0.59375 and since V_{in} is greater than V_{DAC} . So, this bit remains unchanged. So, this bit remains one only. So, this way through this so, through an iterative step; so every individual bits from the MSB side is turned on and then if the content becomes if the DAC output becomes more than the input voltage input voltage then the bit is reset.

If the if the DAC output is less than this input voltage then the bit is retained to 1; of course, for if the V_{in} and this reference this V_{DAC} . So, these 2 are equal then also we have to retain the value and in fact, in that case we do not need to proceed with further bits rest of the bits will always be 0, but anyway. So, that is not shown here as a case, but you can always do that exercise. At any point of time V_{in} becoming equal to V_{DAC} whatever be the SAR content so, that is the digital output and we can terminate the conversion process there though it is difficult to detect that situation. So, normally we do for all the n cycles and get take the DAC output after end cycles only.

(Refer Slide Time: 17:05)



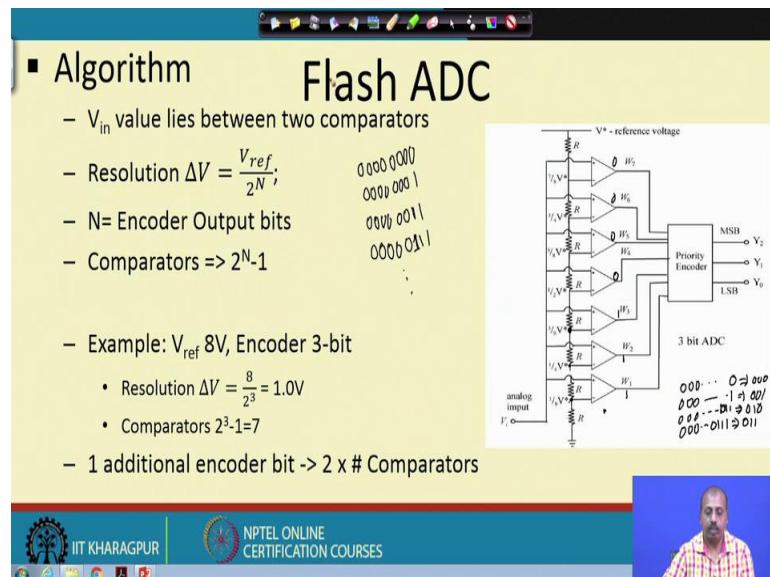
So, next we will be looking into flash type ADC. So, in flash type ADC so, as the name suggests the conversion is done in a flash. So, so all the all the bits are set simultaneously and this is a parallel type of ADC.

So, the elements that we have here is encoder that will convert output to of comparators to binary, and we have got a set of comparators. So, any comparator is like this. So, the if this is a comparator circuit, so, you see this V_1 and V_2 so, these 2 are input to the comparator and so, it has got V_{S+} and V_{S-} as the supply voltages. So, this is $V_{\text{Saturation}+}$ and $V_{\text{Saturation}-}$ we tell that way.

So, if this V_1 is greater than V_2 then this V_{out} is equal to V_{S+} , if V_1 is less than V_2 then V_{out} is equal to V_{S-} . So, that is this is basically logic high and this is logic low. So, these are the 2 things now what is done? We apply some. So, this is the encoder circuit; so encoder circuit. So, it is an 8 by 3 encoder. So, if you give an 8 bit pattern here. So, it will be converted into 3 bit binary output.

So, we will see how it operates.

(Refer Slide Time: 18:33)



reference voltage we have divided into a number of intermediary values $\frac{1}{8}V$, $\frac{1}{4}V$, $\frac{3}{8}V$ up to $\frac{7}{8}V$.

And the input voltage is applied to all these comparators simultaneously and this voltage is also applied. So, this first comparator this is for the LSB so, this will be doing a comparison. So, it depends upon the voltage value that you have given V_i value that you have given. So, this comparator output may be 0 or 1 depending upon the voltage whether the voltage is more than that the voltage that we are getting here.

So, if the V_i is greater than this $\frac{1}{8}V^*$, then this bit will be set to one similarly at this point if it is more than $\frac{1}{4}V^*$ then this bit is also set to 1. So, it may so, happen that at this point it becomes low V in becomes low. So, this becomes 0 so, rest of the bits are 1 and all this bits will be 0 from that point onward.

So, so then it goes into a priority encoder. So, priority encoder it will see that these three bits are 1. So, accordingly it will be during an encoding and it will be producing a 3 bit output. So, how many bits are 1? So, this least significant three bits are 1 so, you can say the count value that we have is basically. So, this is. So, if you think in terms of count values.

So, the first count value is 0 0 0...0 so, that is outputted as three zeros then 0 0 0..1. So, that will be outputted as 001 then 0 0 0then. So, it the one will start at some point and then it will continue. So, if you get after that if you get the pattern which is 2 ones. So, this is 2 output will be 0 1 0 and it the example that we have taken.

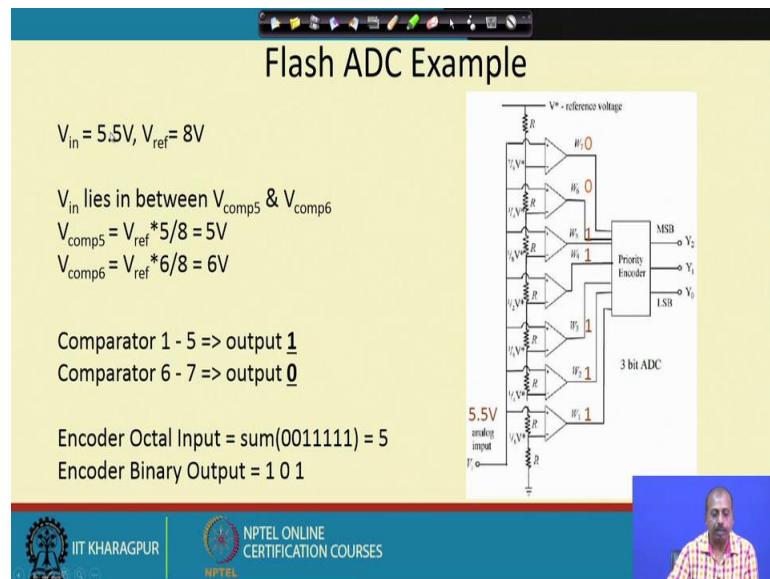
So, here it is 111 so, this is then this bit is the bit before that is 0. So, this is coded as 3. So, it is it will be coded as 011. So, this basically counts. So, at this output of this comparators, so, we have got the count value, but in some unary number system you can say. So, it can produce all 0 to all 1, but the pattern that it can produce is 0 0 0 0 0 0 0 then 0 0 0 0 0 0 0 1 then 0 0 0 0 0 0 1 1 then 0 0 0 0 0 1 1 1 so, it can go like this ok.

So, this 1 so, once a bit becomes one the bits towards the right of it must be 1 and the bits towards the left of it must be a 0. So, it is important to find out where how many ones are there. So, accordingly this priority encoder circuit it can do a encoding and output the corresponding sequence number.

So, in this particular example we have got V_{ref} equal to 8 volt and that the 3 bit encoder. So, resolution is given by $8/2^3$ that is 1 volt ok. So, number of comparators needed is 2^{N-1} so, that is 2^{3-1} that is 7 so, we need 7 such comparators and. So, one additional encoder bit will be a. So, if we say that instead of a 3 bit instead of having 3 bit ADC, we will take a you will design a 4 bit ADC then the number of encoders will become 16.

So, every additional encoder bit number of comparators will double. So, that is the cost of flash ADC so, the number of comparators needed is pretty high, but this conversion is immediate. So, this whenever this input is given accordingly this unary pattern will be set here, and this priority encoder will encode it into one of the binary pattern. So, that way this flash ADC works.

(Refer Slide Time: 23:34)



So, another example suppose V_{in} equal to 5.5 volt and V_{ref} equal to 8 volt; so V_{in} lie so, 5.5 volt. So, V_{in} will lie between V_{comp5} and 6 volt. So, this is. So, the bit pattern will be like this. So, this is between 5 and 6 so, up to this. So, this a seventh and eighth bit so, there are sixth and seventh bits. So, they will be 0 and the up to bit number 5 they are all 1.

So, then because $V_{comp5} = V_{ref} \times \frac{5}{8}$ that is 5 volt and $V_{comp6} = V_{ref} \times \frac{6}{8}$ that is 6 volt. So, comparator 1 to 5 will be equal to 1 comparator 6 to 7 will be equal to 0. So, this is the unary coding that I get; so 0 0 111. So, that there are 5 ones in it. So, the binary output of this priority encoder is 101 so, that is the operation of the flash ADC.

(Refer Slide Time: 24:32)

Dual Slope A/D Converter

- Also known as an Integrating ADC

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Now, next we will be looking into another analog to digital converter, which is known as dual slope analog to digital converter or dual slope ADC. So, basically the circuit is like this. So, there is a reference voltage and this capacitor is charged for some time. And then it is connected to this supply voltage and they this input signal and then we try to see in comparison to this how much time this capacitor needs to get discharged or charged so, like that.

(Refer Slide Time: 25:10)

Dual-Slope ADC – How It Works

- An unknown input voltage is applied to the input of the integrator and allowed to ramp for a fixed time period (t_u)
- Then, a known reference voltage of opposite polarity is applied to the integrator and is allowed to ramp until the integrator output returns to zero (t_d)
- The input voltage is computed as a function of the reference voltage, the constant run-up time period, and the measured run-down time period
- The run-down time measurement is usually made in units of the converter's clock, so longer integration times allow for higher resolutions
- The speed of the converter can be improved by sacrificing resolution

$$V_{in} = -V_{ref} \frac{t_d}{t_u}$$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, I will explain it how does it work. So, an unknown input voltage is applied to the input of the integrator and allowed to ramp for a fixed amount of time t_u . So, initially sorry so, initially the input voltage the V_{in} is the input voltage. So, it is connected to this, and this circuit is kept on for some amount of time. So, that is controlled by this control logic. So, this gives this start and stop pulses.

So, this control logic will be telling the, will be noting down the time and for t_u amount of time this which remains in this position, as a result this integrator output. So, this is an integrator configuration so, integrator output will be increasing like this. So, integrator output increases like this. So, it is charged it is allowed to ramp for a fixed amount of time t_u . Then a known reference voltage of opposite polarity is applied to the integrator and is allowed to ramp until the integrator output returns to 0.

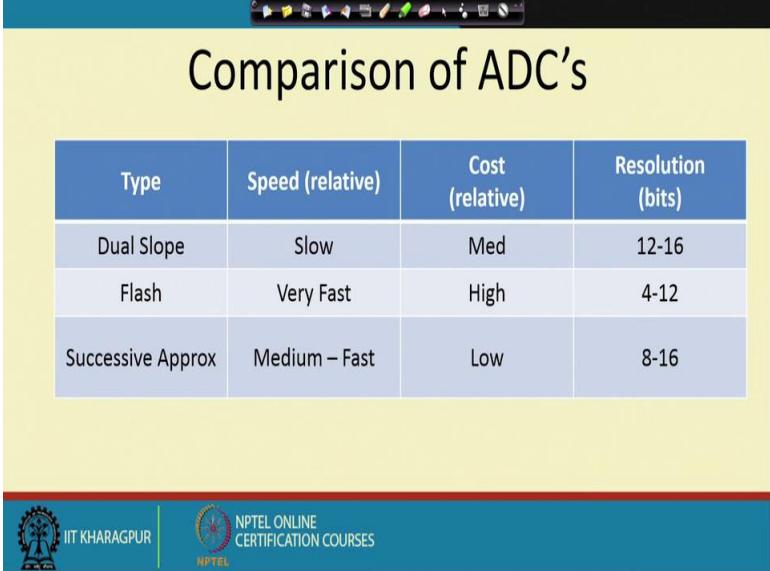
So, at this point what is done? So, the sorry at this point this switch is taken down to the V_{ref} position and it is ensured that the polarity of V_{ref} is the inverse of polarity of V_{in} then when you do this so, capacitor will start discharging ok. So, because this is a negative voltage so, it will if the V_{in} is positive V_{ref} is negative so, it will start discharging. So, we see the depending upon the value of V_{ref} so, and the amount of charge this capacitor has. So, it will take some amount of time to get discharged.

So, this is happening here that a known reference voltage of opposite polarity is applied to the integrator and is allowed to ramp until the integrator output returns to 0 so, this is the time t_d . The input voltage is computed as a function of the reference voltage the constant run up time period and the measured run down time period. So, these values are. So, how is it computed. So, $V_{in} = -V_{ref} \frac{t_d}{t_u}$.

So, minus comes because V_{in} and V_{ref} are of opposite polarity and this t_d by t_u . So, this ratio will tell me; what is the ratio between V_{ref} and V_{in} so, that is how we measure the input voltage. So, this run down time measurement is usually made in units of converter clocks. So, longer integration times will allow higher resolution. So, then what is what is happening is that we in this circuit we have got a counter. So, we count the amount of time for which it has to come down. So, the time needed for t_d time is fixed the counter value will be fixed the t_u time will be computed by looking into this counter when this capacitor is discharging.

And then this t_d/t_u ratio will be computed by this control logic and it will be telling; what is the corresponding bit pattern. So, that part is not explicitly shown this control logic will finally, output the bit pattern depending upon this t_d/t_u ratio.

(Refer Slide Time: 28:11)



A slide titled "Comparison of ADC's" showing a table of ADC types, their speed, cost, and resolution.

Type	Speed (relative)	Cost (relative)	Resolution (bits)
Dual Slope	Slow	Med	12-16
Flash	Very Fast	High	4-12
Successive Approx	Medium – Fast	Low	8-16

IT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if you compare between the ADCs, then this dual slope ADCs are a bit slow, flash ADCs are very fast and successive approximation are medium fast; cost dual slope medium cost, flash ADC high cost, and this successive approximation is low cost resolution dual slope is 12 to 16 bit, flash ADC is 4 to 12 bit and successive approximation 8 to 16 bit.

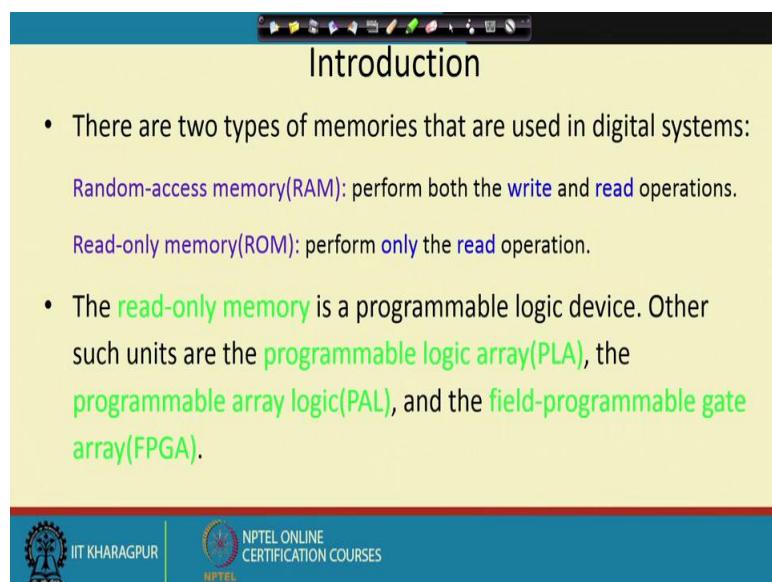
Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 41
Memory

Semiconductor memories, so, these are one of the very important components like if you look into any digital system, because we have seen that if you want to store some information in a digital system, then one possible storage that we have seen at the registers. But you see registers are quite costly because for making one register one register so, you will need a number of flip flops and some control logic like that.

So, if you are willing to store large amount of information, and then you want to access those contents, then we need some mechanism by which it can be done in a much better fashion, not via some circuitry to access individual flip flops. So, the semiconductor memory design so, this will tell us like how these things are done and where can what are the functionalities of this memory and there are many memory chips available so different types. So, what are the general control lines that you should expect in a memory chip and how is the memory organized in a system, so we will be looking into these aspects.

(Refer Slide Time: 01:24)



The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the word "Introduction" is centered in a bold black font. Underneath the title, there is a bulleted list of text. The text is color-coded: some words are in blue, while others are in green. The blue words include "two types of memories", "Random-access memory(RAM)", "Read-only memory(ROM)", "read operations", "only the read operation", and "field-programmable gate array(FPGA)". The green words include "memories that are used in digital systems", "programmable logic device", "programmable logic array(PLA)", "programmable array logic(PAL)", and "read-only memory".

- There are two types of memories that are used in digital systems:
 - Random-access memory(RAM): perform both the **write** and **read** operations.
 - Read-only memory(ROM): perform **only** the **read** operation.
- The **read-only memory** is a programmable logic device. Other such units are the **programmable logic array(PLA)**, the **programmable array logic(PAL)**, and the **field-programmable gate array(FPGA)**.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

So, there if you look into this memory there are 2 broad types of memories if that are used in digital system, one is known as random access memory or RAM and another is read only memory or ROM so, in random access memory so, you can perform both write and read operation, and in read only memory so, you can perform only the read operation.

So, though this name random access memory is somewhat misleading so, it means that there is some as if there is some other memory which is not random access which is sequential axis. This is true, but sequential axis memories are there, but they are no more used in the computer systems now, but the both random access memory or RAM and read only memory ROM so, they are they act they are actually random access so, if there is no order in which you have to axis the locations in this chips.

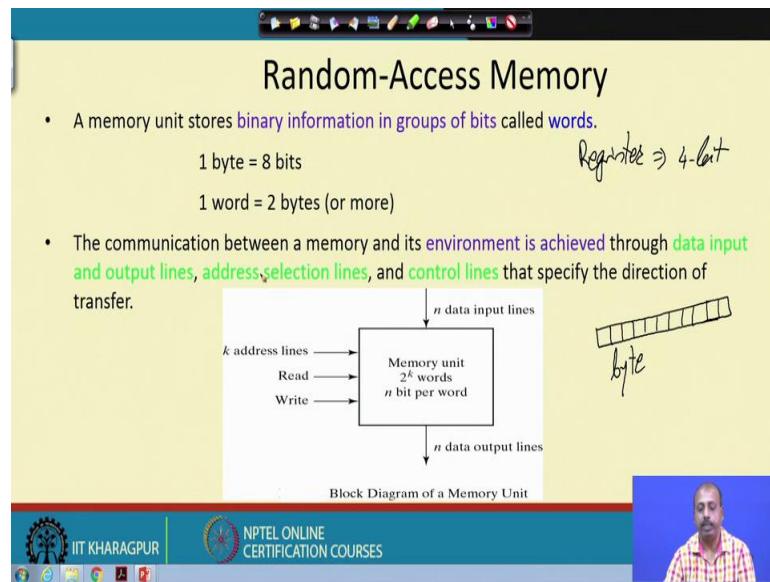
So, you can tell the location address and the memory will be able to access that particular location. So, the basic difference between a RAM and a ROM is that the in case of RAM so, you can do both write and read operation so, either that is you can modify the content of a location in the RAM or you can read the content of a location in the RAM. Whereas, for read only memory so, it is you can only to read operations so, normally if the information is not going to change. For example, say the some part of the operating system which is the memory resident part of the operating system, which is not going to change which is commonly known as the basic input output sub system ok.

So, those or the monitor program so, they are actually kept in the ROM because they are not going to change. Similarly if you are thinking about some other application where say some filtering application where you have got a set of filter coefficient. So, this filter coefficients may be kept in a ROM and then we can do the operation so, the filter coefficients are not going to change, so, they are they can be kept in a ROM.

So, this read only memory is a programmable logic device so, this is programmable logic device means you can program it, and you can put some content there. So, random access memory so, you can the user can change the content at any point of time, whereas, this programmable logic device based read only memory so, you can change the content, but only with some restriction so, you cannot change the content while the system is in operation; so, either you have to take out the chip or you have to stop the system functionality, and do some modification to the programmable device.

So, apart from this read only memory which can act as programmable logic device so, there are other programmable logic devices like programmable logic array, we have seen some part in our discussion previously the programmable array logic or pal and filled programmable get array is or FPGAs. So, will be looking into this devices also to some extent.

(Refer Slide Time: 04:21)



So, to start with will be looking into the random access memory, a memory unit it stores the binary information in groups of bits called words. So, as we know that in case of in case of a register so, if I say that it is a 4 bit register, so, 4 bit register then we know that this 4 bits may be contained in four flip flops. So, the same concept we have in this random access memory, so, we have got a few locations of this memory chip which are accessed together ok.

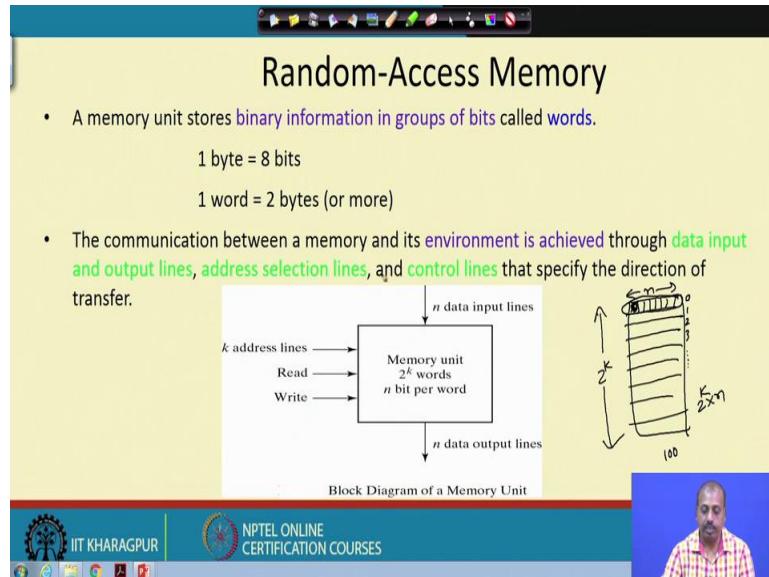
So, these are this may be there may be 8 such bits which are access together so, that constitutes a byte. So, this 8 bit will constitute a byte and depending upon the memory design in one memory access, so, you may get the content of one you may get the content of one byte or may be more than one byte. So, this is determined by this lines like if you say if you see that I have got a memory unit like this, that has got n data input lines and n data output lines; so, if you are accessing a particular location in this memory so, it will be you will be getting data in terms of n bits.

So, if you trying to modify something so, you have to give n bit data here which will be written at some a set of n bit locations or if you are trying to get the content of this memory for a particular location so, you will get this n bit data. So, this individual bits of memory so, they are accessed in a group so, that will be call that will call a word.

So, so, that is known as the word size of the memory. So, word size is normally equal 2 bytes or 4bytes like that so, there is no harden first rule like what should be word size, but word size is never less than one byte of course. So, so you can in your computer system so, you may have different word sizes.

So, communication between a memory and its environment is achieved through data input and output lines address selection lines and control lines so, you can think about this memory as if it is having 2^k words and each word is n bit.

(Refer Slide Time: 06:46)



So, you can you can think that as if my memory is like this so, it has got 2^k such words and each location is consisting of n bits so, this side you have got n and this side you have got 2^k .

So, if you can think of this memory as a collection of cells, and number of cells is equal to $2^k \times n$. So, this is individual cells you have got $2^k \times n$. But the point is you cannot access a single cell alone so, if you trying to if you are trying to access a single cell so, you have to access this entire location so, the I can so, there are some so, every location has got an

address so, if this may be address 0, this may be 1, this may be 2 this may be 3 so, it goes like this.

So, for example, if I ask for the content of location 100 so, it will give me an n bit pattern that was that is the content of that location. So, we have got these memory unit that consist of 2^k words, and each word is n bit quite.

So, this k address lines had to be given to tell which location we are trying to access so, then this n bit data input and output lines are there, if you are doing a read operation then the content of that address location will be available on this n bit data output line. On the other hand if you are doing a write operation so, whatever value is available on this n data input lines will be stored in the corresponding location.

So, these control lines the read and write so, they specify the direction of the transfer. So, this is the idea of this basic random access memory.

(Refer Slide Time: 08:32)

The slide has a title 'Content of a memory'. Below the title is a bulleted list of two points:

- Each word in memory is assigned an identification number, called an address, starting from 0 up to $2^k - 1$, where k is the number of address lines.
- The number of words in a memory with one of the letters K= 2^{10} , M= 2^{20} , or G= 2^{30} .
Handwritten note: $64K = 2^{16}$ $2M = 2^{21}$ $4G = 2^{32}$

Memory address		Memory content
Binary	decimal	
000000000	0	101101010101101
000000001	1	1010101110001001
000000010	2	0000110101000110
	:	:
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Content of a 1024 × 16 Memory

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, each word in memory is assigned an identification number called an address. So, these are the addresses that the each locations so, this is address 0 this is address one this is the address 2 like that so, this is the up to address 1023. So, in this particular case we have got 1024 locations 1 k 1 kilo memory locations one kilo memory address so, if the k equal to 10 in this case so, the address will go from 0 to 2^{k-1} so, addresses go from 0 to 1023, k is

the number of address lines and number of words in a memory we write with one letter that is K write it in the K which is kilo which is 2^{10} , mega M 2^{20} or giga 2^{30} like that.

So, 64 k that is equal to 2^{16} so, K is 2^{10} and this 64 is 2^6 so, that gives us 2^{16} , similarly 2 mega it is 2^{21} . Now if you write whether it is byte or word etcetera so, that will be written here like if I write like 64 KB so, if I write like 64 KB. So that means, individual locations are having 8 bits ok.

So, this way we can tell the number of locations in the memory chip.

(Refer Slide Time: 09:55)

Write and Read operations

- Transferring a new word to be stored into memory:
 1. Apply the **binary address** of the desired word to the **address lines**.
 2. Apply the **data bits** that must be stored in memory to the **data input lines**.
 3. Activate the write input.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so, if you do, try to do write and read operations so, first of all will looking into the write operation, which is to transfer in new word to be stored into the memory. So, for that what we have to do is that, we have to apply the binary address of the desired word, to the address lines and apply the data bits that must be stored in the memory to the data input lines and you have to tell the write inputs.

So, we have to so, if you look into this diagram for writing, we have to give the address lines we have to give the data value and we have to keep the write control signal so, read control signal should not be activated. So, that is how this write operation will take place

(Refer Slide Time: 10:42)

The slide has a yellow background with a title 'Write and Read operations'. It contains the following text and diagram:

- Transferring a stored word out of memory:
- 1. Apply the **binary address** of the desired word to the **address lines**.
- 2. **Activate the read input.**
- Commercial memory sometimes provide the **two control inputs** for **reading** and **writing** in a somewhat different configuration.

A small diagram shows a square box with 'Read' and 'Write' arrows pointing to it.

Control Inputs to Memory Chip		
Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

At the bottom, there are logos for IIT Kharagpur and NPTEL Online Certification Courses, along with system status icons.

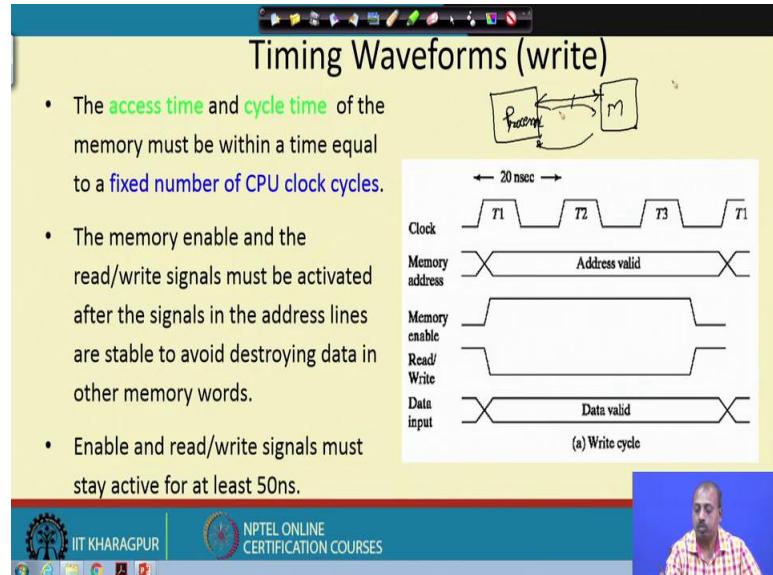
Similarly, we can do this read operation which is basically transferring a stored word out of memory so, here also it is similar so, apply first the binary address of the desired word to the address lines, activate the read input, and so then the content will be available on the data output lines. So, commercial memory is they sometimes provides 2 control input so, reading and writing in a somewhat different organization. So, basically the problem that we have with the configuration that we have taken like say we have said that we have got a memory chip memory like this and it has got 2 controls one is read another is write.

Now, in this situation how do you tell that what a which operation your trying to do. So, it may be the situation that I don't want do a either read neither read nor write operation so, I do not want do any of those operations ok. So, somehow they should be a mechanism by which I should be able to distinguish between the situation that say I do not want to do any of the operation. So, this one possibility is like this memory chips they come with an enable input. So, with this enable line is 0 then whatever be this read write value so, no operation will be done. If enable line is 1, then if the read write bit is 0 so, that is a write operation and if enable is 1 and read write is 1 that is the read operation.

So, in many processors we see that it is written has $READ/\overline{WRITE}$ so, so, the control lines will written has $READ/\overline{WRITE}$. So, with this enable is one and $READ/\overline{WRITE}$ is equal to 0; that means, we are trying to do a write operation similarly if you this line is one; that

means, you are trying to do a read operation. So, this is the organization of this control signals memory read write operations and all.

(Refer Slide Time: 12:44)



So, typically this memory access can be represented by means of a timing diagram so, what happens is that this memory chips they are connected to some processor for this which will be doing the operation. So, in our from our school days so, we know that if I have got a processor then this processor and memory if they are connected by means so, so they are connected by means of some lines and then whenever processors needs to write something on to the memory so, it will send the data to the memory.

Similarly, when it wants to read so, it gets the value there. And this line that I have shown here so, this is not a single line so, we will have the situations like this.

(Refer Slide Time: 13:34)

Timing Waveforms (write)

The diagram illustrates the timing waveforms for a write cycle. It shows the interaction between a Processor (P) and a Memory (M). The memory address is valid during T1, T2, and T3. The memory enable signal is high during T1, T2, and T3. The read/write signal is low during T1 and high during T2. The data input is valid during T2 and T3. The data output is valid during T3.

• The access time and cycle time of the memory must be within a time equal to a fixed number of CPU clock cycles.

• The memory enable and the read/write signals must be activated after the signals in the address lines are stable to avoid destroying data in other memory words.

• Enable and read/write signals must stay active for at least 50ns.

So, this lines they can be divided into 3 D at least 3 different set of lines so, this is this side is the processor this side is the memory. The first set of lines they are the address lines, then we have got the data lines these are the data lines and there are some control lines, these are the control lines so, with basically read write or this enable so, they who has the control lines.

So, this processor how does it operate is that so, processor operate are some clock so, it so, this suppose this is the clock signal, where the duration of this clock signal period this 20 nanosecond so, this is just an example. Now this so, at the at T 1 so, this the processor will put the address on to the address bus so, address will be valid for the entire time T 1 T 2 T 3 or the three cycles and then after putting the address on to the address bus this memory enable signal is made high.

Then this *READ/WRITE* line so, this is made low because this is write operation so, this *READ/WRITE* is made low, but before that so, data is put on to the data bus. So, you see that these 2 points, so, they occur simultaneously that is the memory address is made a valid address is put on to the address bus, valid data is put on to the data bus after that the memory signal memory enable signal is given, and this read write control is made low so, that this is the write operation then the memory will be doing the operation

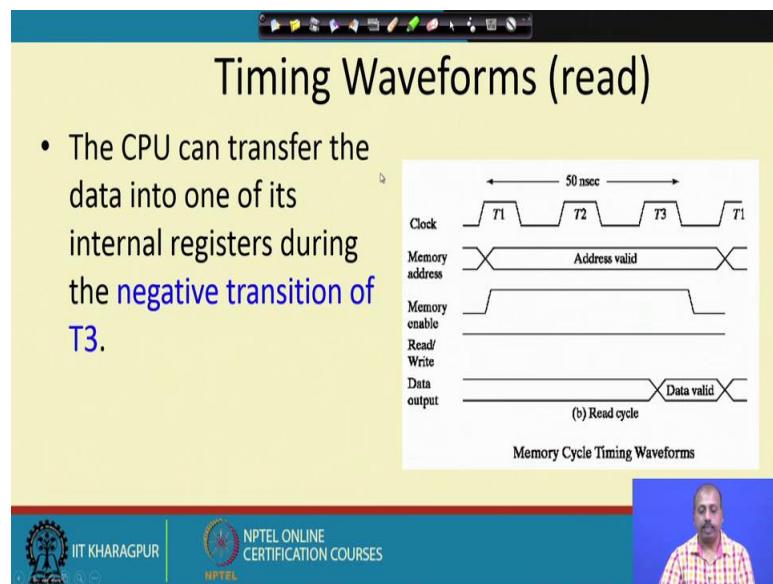
So, how much time this memory takes to respond with the data so, that is given by the cycle time. So, this is this memory at the access time and cycle time of the memory must

be within a time equal to the fixed number of CPU clock cycles. So, access time means how much time it takes to access the memory so, for how much time this read write this address input then this read write control input should be valid. So, that memory will understand it and cycle time means after we have put the address data and say control line after how much time the operation will be done by the memory there is a cycle time.

So, this memory enable and read write signal must be activated after the signal in the address lines are stable, to avoid destroying data in other memory words. Because otherwise what will happen is that if these signals are given before they become valid so, they are having some wrong values. And if you given enable and this read write signals before that, then the memory will access some other location and that locations content will be lost, that location content will get destroyed.

So, this is the problem so, it is always advisable that we first put this make this address and data lines valid, and then only we give this control signals. So, this enable and read write signals must be the must stay active for at least 50 nanoseconds. So, this is say typical situation like if your clock CPU processor clock is of period 20 nanosecond, then this read write signals are expected to be active for about 50 nanosecond.

(Refer Slide Time: 16:48)



So, next will be looking into the read operation so, in case of read operation also the CPU can transfer data into one of its internal registers using this negative transition or T3. So, here also the operation is like this the address valid address is put, memory enable signal

is made active and this *READ/WRITE* line is made active so, this is made high so, this is read operation is done. So, so after 50 nanosecond so, if I the memory cycle kind is 50 nanosecond so, after 50 nanosecond the data is available on to the data output so, this data valid comes here.

So, processor should take the data at this point so, it has started the write operation at this point, but it should take the value at this point. So, this negative transition of T3 so, this falling edge of T3 so, this should be used for storing the data into the CPU register because at that time only the data is valid so, data is definitely valid at this point of time so, this is the way this read operations will be done.

(Refer Slide Time: 17:50)

The slide has a yellow background. At the top, the title 'Types of memories' is centered. Below the title, there are two bullet points:

- In **random-access memory**, the word locations **may be** thought of as **being separated in space**, with each word occupying one particular location. To the right of this text is a circular diagram representing a magnetic disk with several concentric tracks and data blocks.
- In **sequential-access memory**, the information **stored in some medium is not immediately accessible**, but is **available only certain intervals of time**. A **magnetic disk or tape unit** is of this type. To the right of this text is a small video player window showing a man speaking.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a series of small icons.

So, if you are going back to the types of memories so, in random access memory the word locations may be thought of as being separated in space with each word occupying one particular location. So, as we can think of it logically as if we have got a in case of this random access memory so, we have got a set of locations, so all these are locations, and you can access any of them randomly so, there is no fixed ordering in which you have to access them. On the other hand this sequential access memory information is stored in some medium is not immediately accessible, but is available only certain intervals of time.

So, basically this is typically this is true for magnetic disk or tape type of devices, so, they are what happens is that we have got these type of disk and there are some tracks in it there

are some tracks which are further divided into sectors like this and there is a read write head. So, there is a read write head so, suppose this is a read write head.

Now, so, this disk rotates and accordingly whichever comes under this read write head so, that value is read at that point so, so, since this disk is rotating so, after you have accessed say this location, then only you can access this location if I assume that the disk is rotating like this. So, you cannot access randomly so, it cannot be the case that first you access this then you access this then you access this so, that cannot happen. Because if your disk is rotating like this if the initial read write head is at this point then it will read this content first then this content then this content.

So, that is a sequential access so, this semiconductor memory they are mostly random access type, and this secondary storage like say disk and all so, they are of this sequential access type.

(Refer Slide Time: 19:54)

Types of memories

- In a **random-access memory**, the **access time is always the same** regardless of the particular location of the word.
- In a **sequential-access memory**, the time it takes to access a word depends on the position of the word with respect to the reading head position; therefore, the **access time is variable**.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, in random access memory access time is always same regardless of the particular location of the word so, it does not matter which location you are accessing so, location accessing location 100 or location 200, they take same amount of time.

Whereas in a sequential access memory the time taken to access a word depends on the position of the word with respect to the reading head position. As I was telling previously that since the disk is rotating, and if the word that you are trying to access is close to the

read write head then it will be access must faster compare to the situation where when this actual data point is will be come much later so, it is not adjacent to the current head position.

So, in that case it will take time for the disk to rotate and the head get aligned to that location. So, this is the problem so, all the sequential access and the access time becomes variable so, in case of random access memory. So, depending upon the technology we can immediately find out what is the access time, but in case if sequential access memory. So, it depends on the location that your trying to access accordingly the access time will vary.

(Refer Slide Time: 21:02)

The slide has a yellow background. At the top center, the title 'Static RAM' is written in a large, bold, black font. To the right of the title is a small hand-drawn diagram of a memory cell, showing a central node connected to four lines labeled 'D', 'C', 'R', and 'W'. Below the title is a bulleted list of features:

- SRAM consists essentially of **internal latches** that store the binary information.
- The stored information remains valid as long as power is applied to the unit.
- SRAM is easier to use and has **shorter read and write cycles**.
- **Low density, low capacity, high cost, high speed, high power consumption.**

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, the IIT Kharagpur logo and the text 'IIT KHARAGPUR' are visible. In the center, the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' are displayed. On the right side of the bar, there is a small video window showing a man speaking.

So, static RAM or SRAM so, this is one type of random access memory, it consist of essence essentially it has got latches that store the binary information. The stored information remains valid as long as power is applied to the unit input so, SRAM is easier to use and have shorter read and write cycles, low density, low capacity, high cost, high speed high power consumption. So these are the features of this SRAM. So, this low density means per unit area how many such how many memory cells you can put.

So, low density means so, you cannot make the density high so, you cannot put large number of memory cells in a short, in a small area. Low capacity so, if the density is low then naturally capacity will also be low because how many such cells, you can put into the chip so, that way the capacity will be low cost is high.

Cost is high because you are putting less cells there so, the cost is going to be high. But the advantage that you get is the high speed so, speed of operation is high the access time is faster, power consumption is also high. So, these are the problems with static RAM, but still they are used because of this the speed high speed of property.

(Refer Slide Time: 22:29)

Dynamic RAM

- DRAM stores the binary information in the **form of electric charges on capacitors**.
- The **capacitors** are provided inside the chip by **MOS transistors**.
- The capacitors tends to discharge with time and must be periodically recharged by **refreshing the dynamic memory**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There is another variant of this RAM which is known has dynamic RAM or DRAM so, it stores the binary information in the form of electric charges on capacitors. So, in case of static RAM what happens is that so, conceptually a static RAM is like this so, we have got 2 inverters which are connected back to back so, which are connected back to back.

So, once this is say equal to 1 so, this is equal to 0 since this is equal to 0 so, this is equal to one as a result, this point will always remain as 0. On the other hand if this value is sense 0 then this will be 1 so, this will be one and this will be 0 and that way this point will always remain at one. So, this static RAM so, very simplistic structure so, that will be that can be formed by connecting two back to back inverters, but that way the cost of the system goes high there are other constructions of this SRAM so, we will not going to that.

But in case of dynamic RAM what is done is that, you do not have such back to back inverters rather we have we take a capacitor. So, the capacitor is charged and this capacitor is actually holding the charge. So, if you what trying to store a 1 so, this capacitor is charge, and so, naturally so, it will be so, if you sense the value here so, you will get a 1. On the other hand after some time of course, we cannot make a perfect capacitor so, this capacitor

will leak so, after some time the voltage level will come down as the charge leaks from this capacitor so, it will go towards 0.

So, this is the so, what is what is required is that we need some sort of refreshing. So, this DRAM or dynamic RAM its stores binary information in the form of electric charges on capacitors, the capacitors are provided inside the chip by means of MOS transistors. So, we have seen CMOS logic so, the MOS transistors we have seen there so, they are the gate of it acts as the, acts as the capacitor. So, this so, this gate value the gate part of it acts as a capacitor to hold the bit pattern, bit value.

And the capacitors as the capacitor they try tend to discharge with time so, periodically we must recharge the dynamic memory. So, this compared to static memory the which does not require any extra circuitry for do a for holding the value so, in case of dynamic memory periodically you have to read the content of the memory and write it again so, that way we should have a refreshing circuitry.

So, it is so, apart from this if this is your D RAM chip, then along with that I should have a refreshing circuit. So, what this refreshing circuit does is that, it reads the content of individual locations and write the same value there. So, as since it is rechecked periodically so, these capacitors will get recharged and then the value will be restored to their original one.

So, this is the idea of this dynamic RAM. So, of course, we do not go into the circuit part.

(Refer Slide Time: 25:44)

The slide has a yellow background with a black header bar containing icons. The title 'Dynamic RAM' is centered in a large, bold, black font. Below the title is a bulleted list of advantages:

- DRAM offers reduced power consumption and larger storage capacity in a single memory chip.
- High density, high capacity, low cost, low speed, low power consumption.

At the bottom of the slide, there is a red horizontal bar. On the left side of this bar are the logos for IIT Kharagpur and NPTEL, followed by the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the bar is a small video player window showing a man in a pink shirt.

So, in case of dynamic RAM so, it offers reduced power consumption so, that is one of the largest advantage that we have with DRAM. Storage capacity is high because only a single capacitor is needed for making a DRAM chip sorry a DRAM cell, so, that way this whereas in case of your SRAM cell so, standard SRAM cell it required 6 transistors whereas, standard DRAM cell it is requiring only a single transistor.

So, that the way capacity is much higher and high density, high capacity, low cost, low speed, low power consumption. So, this low speed because it requires this refreshing periodic refreshing so, speed is not that high as this a SRAM; SRAM configuration, but it is still used because very much because of this high density and high capacity and low power consumption.

(Refer Slide Time: 26:44)

Types of memories

- Memory units that **lose stored information** when power is turned off are said to be **volatile**.
- Both static and dynamic, are of this category since the binary cells need external power to maintain the stored information.
- Nonvolatile memory, such as magnetic disk, ROM, retains its stored information after removal of power.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, memory units that loose stored information when power is turned off is are said to be volatile. So, all this chip static RAM or dynamic RAM that I am talking about so, if you with the power is turned off then the content is lost so, this type of memory is there known as volatile memory. Both static and dynamic are of this category since the binary cells need external power to maintain the stored information.

There are nonvolatile memory is like magnetic disk and ROM, that retains its stored information even after removal of power. So, in a ROM so, the information is stored permanently so, you can you can switch of the ROM and, but still get the content there, when later on also the content will be there, but for RAM it will get destroyed.

(Refer Slide Time: 27:34)

Memory decoding

- The equivalent logic of a binary cell that stores one bit of information is shown below.
Read/Write = 0, select = 1, input data to S-R latch
Read/Write = 1, select = 1, output data from S-R latch

(a) Logic diagram

SR latch with NOR gates

(b) Block diagram

Memory Cell

IIT KHARAGPUR | NPTEL ONLINE
NPTEL CERTIFICATION COURSES

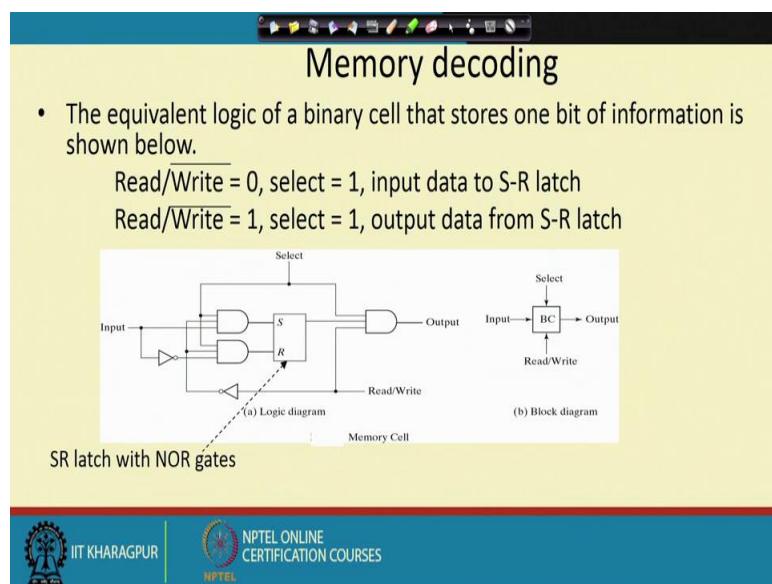
Next we start with memory decoding; so memory decoding process. So, this will try to figure out like how this addresses will be selecting a particular location, and then. So, here the we can logically we can think of as if there is an SR flip flop and then this SR flip flop is getting selected by means of this read write lines. So, this is just a logical diagram. So, ideally it does not happen like this so, that is a there are separate courses on memory design. So, that will be talking about this cell design part, but for our course so, will be looking into this logical diagram and try to understand the philosophy of operation.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 42
Memory (Contd.)

The equivalent circuitry for a logic circuitry for a binary cell that stores 1 bit of information can be like this.

(Refer Slide Time: 00:19)



So, this is as I said that this is just an equivalent circuitry, so actual RAM cells are not like that. So in that case I have told you that individual SRAM cells they require only 6 transistors, but here you see that I have put an SR flip flop. So SR flip flop means you have got some 4 NAND gates so a 4 NAND gate each NAND gate takes a 4 transistor so that will take 16 transistors.

So, this SR flip flop itself will take 16 transistors, then we have got all this AND gates etcetera inverter etcetera they will take more number of transistors. So this is just a logical diagram do not take it as the actual memory diagram, but from the digital logic point of view so this is an equivalency of, equivalent of what is happening in the memory cell.

So, the equivalent logic of a binary cell that stores 1 bit of information is like this, that if *READ/WRITE* is equal to 0 then it will select and select equal to 1, then the input data is

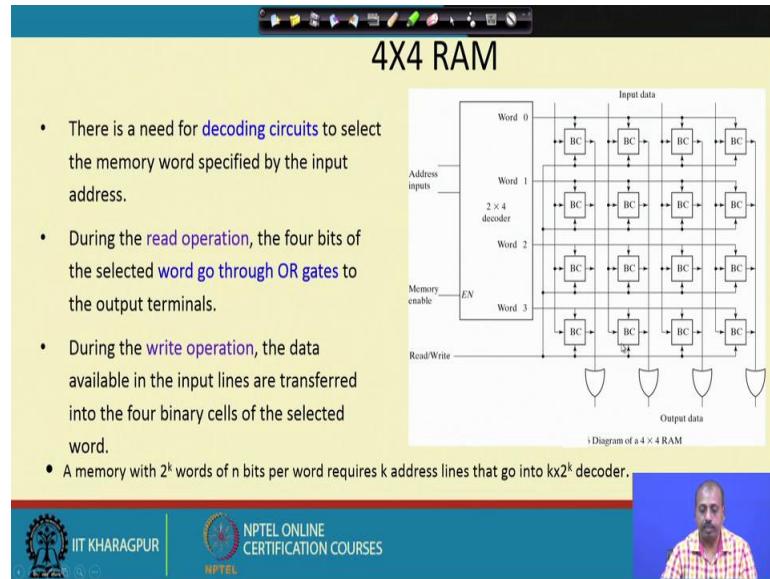
put into the SR latch. So input is coming so if $READ/\overline{WRITE}$ is 0; that means, we are trying to do a write operation. So a select line is actually the enable line of the memory chip you can say so at this location I am trying to write the value, so whatever is coming at the input they should be stored here. So if you look into this diagram so when this $READ/\overline{WRITE}$ is equal to 0.

So, this line so this and gate so it has got this select equal to 1, this $READ/\overline{WRITE}$ equal to 1 and a input is whatever be the input value that will be coming here. And you see since this $READ/\overline{WRITE}$ line is equal to 0 so since this is equal to 0, so this will be coming as this will be coming here. So this will be coming as 1 here and the input bar will be coming to this point this reset point.

So, in the S I have got input and in the R, I have got \overline{input} . So as a result whatever be the value of the input that will get stored here so if input value is equal to 1 in that case I have got S equal to 1 and R equal to 0 as a result this flip flop will be set to 1. On the other hand if input value is equal to 0, then I will get here S equal to 0 and R equal to 1 as a result the value of this Q output will be equal to 0.

So, this way this value input is getting stored in this SR latch, and then for the output part so we have got this if the select line is 1, then the whatever be the output that will go to a whatever be the out flip flop value here the latch value here. So that will come to the output provided this read line is made equal to one so that is a read operation. So, conceptually we can say that if this is the basic cell so this whole thing is the basic cell so it has got the inputs like select, input, read write and output the output line and the output so that way this cell works. Of course, it should as I have already said that this is just a logical diagram not the physical one.

(Refer Slide Time: 03:36)



Now, how can we construct a memory array? So I have talked about in the previous slide the design of this Basic Cell or BC.

Now, if you suppose we are trying to construct a 4 by 4 RAM that is there will be 4 locations or 4 address locations, and each location is consisting of 4 bits. So I have take 4 by 4 that is 16 basic cells arranged in a 4 by 4 fashion, and then there is a so this address lines, so this is there are 4 such memory locations so I will need 2 address lines so 2 address lines are coming here so, and I use a 2 to 4 decoder.

So this memory enable line is connected to the enable line of the decoder, so if this memory enable is equal to 0; if this memory enable is equal to 0 as a then all this decoders all this outputs will be 0 so none of them will be none of the location values will be available at the output. On the other hand if enable equal to 1 so depending upon this address lines depending upon this if enable is equal to 1 so, depending upon this address lines so one of this word lines will be selected.

Suppose for example, if the address line is say 0 1 then what will happen? So this will be 0, this will be 1. This will be 0 and this will be 0. So the content of this basic 4 basic cells so they will be available here; so the other otherwise since the select line is 0 so will be so this BC lines will be this basic cells outputs will be 0.

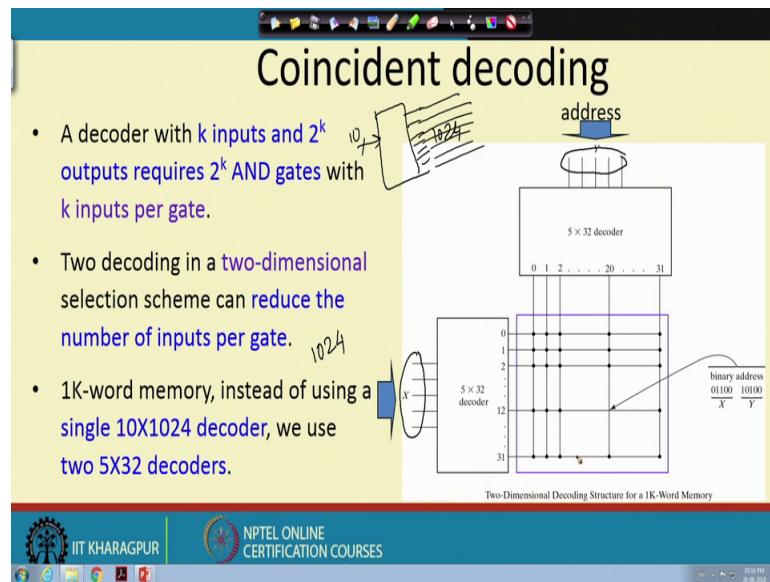
So, they are this basic cell outputs in column wise so they are ORed at this point so then you get the 4 bit output data. So we need some sort of decoding circuitry from the address lines so to select the appropriate row of this array. And then, this individual columns of this array so they are ORed to produce the individual the output part of from the chip and for the input part so they will be connected here so they will come to the basic cell. So there is a need for decoding the circuit there is a need for decoding circuits to select the memory words specified by the input address; during the read operation the 4 bits of the selected word go through the or gates to the output terminals as I explained.

And during the write operation, the data available on the input lines that is these points so they will be who available on the input lines and they transfer to the 4 binary cells that you have selected. And then so that way the whatever input data will be coming and whatever address line are given so based on that a particular row will get selected and this input data value will be stored in those binary cells.

So, a memory with 2^k words of n bits per word will require k address lines, and that will go to a go into $k \times 2^k$ decoder. So this decoding circuitry becomes a major concern because if you are having a large sized memory. So you have got a large number of rows in that memory and so this decoder has to have so many outputs. So this is a simple 2 to 4 decoder when you have got say 4 memory words.

So, if there are 1 kilobyte of 1 kilo memory word. Then this decoder size should be **10 by 2^{10}** so that way it will go on increasing significantly, exponentially the requirement of the decoder will go on so that is a major concern.

(Refer Slide Time: 07:06)



So, to reduce the complexity of this decoder to some extent, so what is done? We have got some structure which is known as coincident decoding so, this coincident decoding is like this.

So a decoder with k inputs and 2^k outputs require 2^k AND gates and k inputs per gate, so this is a standard design of the decoder that we have seen previously. The 2 decoding in a 2 dimensional selection scheme can reduce the number of inputs per gate. So what you have to do in this particular case suppose I have got a memory array that has got 1 kilo word of memory.

So, 1 kilo word of memory so that is there are 1024 locations so 1 kilo word means there are 1024 locations. So if you are thinking about a simple design, then I should have a decoder like this that has got 10 number of inputs and this 1024 number of outputs; so individual outputs going to individual arrays individual rows of that cell array.

However, you can do it like this, so you can divide this 10 lines into a row address and column address part. So first you put this so this x and y are divided 5 bit 5 bit they are divided and then so you take 5 by 32 decoder, so it selects this one of this 32 locations, one of this 32 rows and then we have got a 5 by 32 decoder on the column side that will select again 5 by 32 that that will again select one of the 32 columns.

So, this is the 2 dimensional organization of 1 kilo word of memory, that way reducing the decoder complexity.

(Refer Slide Time: 09:03)

Address multiplexing

- DRAMs typically have four times the density of SRAM.
- The cost per bit of DRAM storage is three to four times less than SRAM. Another factor is lower power requirement.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, normally these DRAMs they have typically 4 times the density of SRAM; so this DRAM density is pretty high and the cost per bit or DRAM storage is 3 to 4 times less than SRAM. So this is one of the basic advantage of DRAM over SRAM so the cost per bit is less and other factor is the power requirements; so DRAM power consumption is also low compared to this SRAM.

(Refer Slide Time: 09:31)

Address multiplexing

- Address multiplexing will reduce the number of pins in the IC package.
- In a two-dimensional array, the address is applied in two parts at different times, with the row address first and the column address second. Since the same set of pins is used for both parts of the address, so can decrease the size of package significantly.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

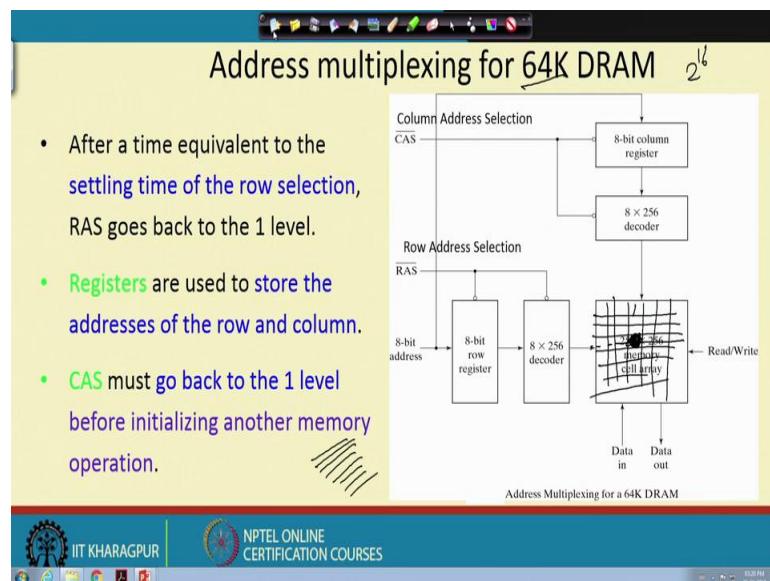
Address multiplexing technique so that that can reduce the number of pins in the IC package. Because this what you can do so as I said that the total 10 bit address is divided into 2 parts 5 and 5 bit and 5 bit so 5 bit row address and 5 bit column address.

The address what we can do instead of by applying this 10 address bits simultaneously, we can apply 5 bit address first and then again 5 bit address. So if I have got this memory chip and then I can have a 5 bit address lines instead of 10 bit, first I apply the row part of the address and then after the row has been selected, we can apply the column part of the address.

So, we can we can divided into we can divide the address part into row part and column part and apply them one after the other. So the advantage that we get is if you have to apply 10 bit address, then this memory should have 10 address lines. So instead of that I should have only 5 address lines and some extra control by which I can tell whether it is the row address or a column address. So in a 2 dimensional array the address is applied in 2 parts at a different times with the row address first and column address second.

Since the same set of pins is used for both the parts of the address, we can decrease the size of the package significantly.

(Refer Slide Time: 10:57)



So, how is it done we will see in the next slide. So what is done so I have got a 256 by 256 memory cell array, so initially the so the 8 bit address is given. So 8 bit address is stored in this 8 bit row register and then from this row register it goes to an 8 by 256 decoder.

So, this that way this particular row is selected and then we apply the column part of the address. So the same 8 bit address lines is used to feed the column part of the address to the memory chip. So this is giving me this is giving me this is actually selecting the corresponding column; so this is give this is stored in the 8 bit column register, then it goes to 8 by 256 decoder and then it selects one of the 256 alternative columns.

So, this memory that we have here is 64K sorry so memory that we have here is 64K so that is 2^{16} locations should be there. So what we have done, this 2^{16} locations instead of organizing them in a serial fashion so what we have done? We have organize them in 2 dimensional fashion so such that in each so we have got 256 such rows and 256 such columns. So once the row selection has been done, so this row has been selected then we select a particular column say we select say this particular column then ultimately so this location gets selected. And this location itself may be your an 8 bit location or 64 bit or 16 bit location depending on the word size of the memory, but this particular memory cell will get selected. So that way I do not need to apply a 16 bit address simultaneously, I apply 8 bit at a time and it will be reducing the complexity in term in the number of pins.

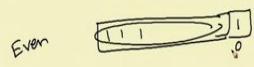
However, I need to tell like when I am applying the row address and when I am applying the column address. So for that purpose so there are 2 special pins \overline{RAS} and \overline{CAS} ; so load address or row address selection and column address selection. When I am giving the row address this \overline{RAS} line should be made equal to 0, so that the value the row address is stored latched on to this row address register. And this decoder can enable the corresponding row and then this column address after that the column address is put and then at that time this \overline{CAS} line is made low.

And then the value that I have that is available on this 8 bit register is a latch 8 bit address is latched on to this 8 bit column register, and then it goes to this decoder and by it selects a one of the 256 columns. So that way finally, one particular memory location or memory word gets selected. So this CAS must go back to the 1 level before initializing another memory operation, so that has to be that is the requirement that first this column selection and the row selection will be done.

And it must go back to 1 and then only it can become 0 or any, to ensure that the operation is done properly.

(Refer Slide Time: 14:18)

Error detection and correction

- It is protecting the occasional errors in storing and retrieving the binary information.
- Parity can be used to check the error, but it can't be corrected.
Even 
- An error-correcting code generates multiple parity check bits that are stored with the data word in memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will be discussing about another very important topic, which is known as error detection and correction, like when you are transferring data or storing data in the memory locations. So it since the data is coming from outside the memory, so it may so happen that in the transfer process some bits got corrupted ok.

So, how to detect that situation and what to do like if we see that some bits have got corrupted? So this error detection and correction mechanism so they are for protecting the occasional errors in storing and retrieving the binary information. So one possibility is the parity; so parity can be used to check the error so we have seen parity separately previously where if I have got say an 8 bit data, then I can just add another bit parity.

So, I count the number of ones here and if the if I say that finally, I should have an even parity, then if the number of ones in this part is ORed then this bit will be turned 1 on the other hand if this is even then this bit will be turned 0. So that we have seen in the x and you know that this can be done very easily using the some XOR circuit. Now this is so parity can check the errors so it can detect the error, but it cannot correct it.

So, this error correcting codes so they can generate multiple parity check bits that are stored with the data word in memory and that can lead to a better their possible correction of the this corruption. So we will see how this is done.

(Refer Slide Time: 15:56)

The slide has a yellow background and a blue header bar with various icons. The title 'Hamming Code' is centered at the top. Below the title is a bulleted list:

- One of the most **commonly used** in RAM was devised by R. W. Hamming (called Hamming code).
- In Hamming code:
 - k = parity bits in n -bit data word,
 - forming a new word of $n + k$ bits. Those positions numbered as a power of 2 are reserved for the parity bits.
 - the remaining bits are the data bits.

On the right side of the slide, there is a small diagram showing two horizontal bars representing binary words. The top bar has a '1' at position 3 and a '0' at position 5. The bottom bar has a '1' at position 1 and a '0' at position 3. To the right of these bars is a small drawing of a person's head and shoulders.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', and the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. There is also a small video player window showing a person speaking.

So, one commonly used code for this error detection and correction is the Hamming code. So this is this was in way divided by R W Hamming and that is why it is called Hamming code so it is say in a hamming code so for n bit of a data word we have got k parity bits ok.

So, for n bit data there are k parity bits and these k parity bits are attached with this n data bits. So total number of a total word size becomes $n + k$ bits and these positions are numbered as a power of 2 are the positions which are powered of power of 2 they are reserved for the parity bits. So it is not that I have got an n bit data and this k bits are computed. And they are just attaching a so it is not done like that but this bits are actually inserted into the total $n + k$ bit may be I have got 1 bit here, another so the 1 parity bit P_1 here another parity bit P_2 there so like that. So it says that the positions are numbered as a power of 2 are reserved for the parity bits while the remaining bits are the data bits. So that is how this organ organization is done.

(Refer Slide Time: 17:11)

The slide has a yellow background. At the top center, it says "Hamming Code". Below that, a green text box contains the following text:
Ex. Consider the 8-bit data word 11000100. we include four parity bits with it and arrange the 12 bits as follows:
Bit position: 1 2 3 4 5 6 7 8 9 10 11 12
P₁ P₂ 1 P₄ 1 0 0 P₈ 0 1 0 0

Below this, there are four equations in purple:
P₁ = XOR of bits(3,5,7,9,11) = 1 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 0 = 0
P₂ = XOR of bits(3,6,7,10,11) = 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 = 0
P₄ = XOR of bits(5,6,7,12) = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1
P₈ = XOR of bits(9,10,11,12) = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1

At the bottom left, there is a logo for IIT Kharagpur and NPTEL Online Certification Courses.

So let us take an example; suppose we have got an 8 bit data word 1 1 0 0 0 0 1 0 0 and we want to include 4 parity bits and then we get a 12 bit pattern. So this is the 12 bit pattern now this bit positions are the since this is a power of the so this is a bit position 1 that is 2⁰, this is bit position 2 so 2¹, so this is bit position 4 which is 2² so, at this powers of 2 so we put the parity bits.

So P₁ is put as bit 1, P₂ is put as bit 2, P₄ is put as bit 4, and the remaining bit 1 1 0 0 so they are coming here this is 1 this is 1 0 0 then 0 at bit number 9 so that way it goes and how to compute this P₁? So this is fixed so P₁ is given by XOR of bit 3 5 7 9 and 11, so if you do this the bit is turning out to be 0, P₂ is given by XOR of 3 6 7 10 and 11 so this is 0.

So, this XORing pattern is fixed, so P₄ is the XOR of 5 6 7 12 so that becomes equal to 1. So this way this P₁ P₂ P₄ and P₈ are calculated and once this is done, the data is stored in the memory together with the parity bit as 12 bit composite word. So this is the thing that is stored in the stored in the memory so 12 bit word that is there, now when you are reading the data from memory so we get this 12 bits. And then we check whether the parity bits are correct or not. If there is an error then the parity bit this parity bits will differ. We compute the check bits based on the similar formula like P₁ was done computed using XOR of 3 5 7 9 11.

(Refer Slide Time: 19:03)

The slide has a yellow background. At the top center is the title "Hamming Code". Below the title is a bulleted list of points. To the right of the list is a table showing bit positions from 1 to 12 and their corresponding binary values. Further down are four equations defining check bits C₁, C₂, C₄, and C₈. The bottom of the slide features a red footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man speaking.

- The data is stored in memory together with the parity bit as 12-bit composite word.

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	1	1	1	0	0	1	0	1	0	0

- When read from memory, the parity is checked over the same combination of bits including the parity bit.

$$C_1 = \text{XOR of bits}(3,5,7,9,11)$$
$$C_2 = \text{XOR of bits}(3,6,7,10,11)$$
$$C_4 = \text{XOR of bits}(5,6,7,12)$$
$$C_8 = \text{XOR of bits}(9,10,11,12)$$

So, here we compute C₁ which is the XOR of the red values of a 3 5 7 9 11. Now if P₁ differs from C₁; that means, there is some error in any of these bits.

Similarly, if C₂ differs from P₂ there is some error in this bit. So if your P₁ and P₂ both are creating a, if C₁ and C₂ both are not with P₁ and P₂ then you know that possibly there is an error in bit number 3, 7 or 11. So this way you can figure out a few locations so which are which may be wrong.

(Refer Slide Time: 19:43)

The slide has a yellow background. At the top center is the title "Error-Detection". Below the title is a bulleted list of points. To the right of the list is a table showing the calculation of C = C₈C₄C₂C₁. The bottom of the slide features a red footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man speaking.

- A 0 check bit designates an even parity over the checked bits and a 1 designates an odd parity.
- Since the bits were stored with even parity, the result, $C = C_8C_4C_2C_1 = 0000$, indicates that no error has occurred.
- If $C \neq 0$, then the 4-bit binary number formed by the check bits gives the position of the erroneous bit.

So the how do you do a check? So a 0 check a 0 check bit designates an even parity over the check bits, and 1 designates an odd parity. And since these bits was stored with even parity, the results C indicates that there is no error. So since we have followed even parity so the if it is if this $C_1 C_2 C_4 C_8$ is 0 0 0 0 there is no error, but if C is not equal to 0; that means, the 4 bit number formed by the check bits gives the position of the erroneous bit.

So, this so that this patterns have been selected in such a way this positions have been selected in such a way that this property holds that, the 4 bit number that you get here gives the position of the erroneous bit.

(Refer Slide Time: 20:31)

The slide has a title 'Example' and a table for error detection. The table shows bit positions 1 through 12 and their corresponding values. It includes three rows of binary data and a row for evaluating XOR to find check bits. The bottom part of the slide features logos for IIT Kharagpur and NPTEL, and a video player window showing a person speaking.

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	0	0	1	1	1	0	0	1	0	1	0	0
	No error											
	1	0	1	1	1	0	0	1	0	1	0	0
	Error in bit 1											
	0	0	1	1	0	0	0	1	0	1	0	0
	Error in bit 5											
<ul style="list-style-type: none"> Evaluating the XOR of the corresponding bits, get the four check bits 												
	C_8	C_4	C_2	C_1								
For no error:	0	0	0	0								
with error in bit 1:	0	0	0	1								
with error in bit 5:	0	1	0	1								

We will take an example, suppose this is the bit that we have this is the so this is the no error so this is the actual one. Now if there is a error in bit number 1, if there is an error in bit number 1 that is this bit has got transform from 0 to 1, but rest of the bits are not.

Then you will get the this when you compute this $C_1 C_2 C_4 C_8$ you will get this particular pattern, so that tells the bit number 1 there is some problem. On the other hand if there is an error in bit number 5; so bit number 5 was 1 here it has become 0 there, and rest of the bits are then after you have computed $C_1 C_2 C_4 C_8$ you will find that the pattern coming is 0 1 0 1. So this identifies the bit at which the error has occurred so you can just flip that particular bit on the received pattern, and then you can, you get the corrected version of the bit.

So, this way that that is the beauty of this hamming code, so it is XORing in such a fashion that this when you check it, so you can figure out like which particular bit has created problem.

(Refer Slide Time: 21:42)

Range of Data Bits for k Check Bits	
Number of Check Bits, k	Range of Data Bits, n
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

$$2^{k-1} \geq n + k$$

So, the hamming code can be used for data words of any length, and total bit in a hamming code should be $n + k$ bits, the an syndrome value C consists of k bits and has a range of 2^k values between 0 to 2^{k-1} .

And range of k much be equal to a larger than $n + k$ so this is the requirement. So if you are using an n bit code and k extra bits or check bits, then the requirement is that

$2^{k-1} \geq n + k$, then only this hamming code will work. For example, if number of check bits is 3, then the data bits that you can that that you can be in the range of 2 to 4.

So, if you have got data bit of length more than 4, then 3 check bits are not sufficient. If you have got 5 to 11 data bits, then you should have check bit k equal to 4, 4 check bits will be necessary. So this is coming from the theory of hamming code so we do not have a spoke to discuss on that, maybe you can look into some coding theory course which will be talking about this hamming code in more detail.

(Refer Slide Time: 22:55)

Single-Error correction, Double-Error detection

- The Hamming Code can detect and correct only a single error.
- By adding another parity bit to the coded word, the Hamming Code can be used to correct a single error and detect double errors. Becomes $001110010100P_{13}$.

$001110010100 P_{13} \rightarrow 001110010100 1$
 $P = \text{XOR}(001110010100 1)$

if $P = 0$, the parity is correct (even parity), but if $P = 1$, then the parity over the 13 bits is incorrect (odd parity).
the following four cases can occur:

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you are trying to detect single error so single error and correct it, so this hamming code can detect and correct only a single error. So if you if you want to have this situation single error correction and double error detection like as I have said that in the, this pattern that the that check pattern that we generated was such that it identifies one particular location as erroneous.

Now, if there are multiple bits which are erroneous, then this check will not able to pin point to the correct errors. So it is just an assumption that in a transmission or a at most one error has occurred so hamming code can correct it. So but if you want to detect the situation in which if there are multiple if there are multiple bits which are wrong, so you should be able to detect it, but you may be happy if correcting only a single bit.

So, single bit error correction and double bit error detection if you are looking into that, so for that purpose we have to add another parity bit and to the coded word, the so the we add this word P_{13} here, so P_{13} is connected here. And then P_{13} is so this P_{13} is computed here, so P_{13} is becoming a 1 at this point, so that is by means of taking XOR of all these bits.

So, this P equal to XOR of this with 1, so that naturally so if you take XOR so if P equal to 0. That means, the parity is correct even parity, but if P equal to 1 then the parity over this 13 bits is incorrect the odd parity.

(Refer Slide Time: 24:36)

Single-Error correction, Double-Error detection

1. If $C = 0$ and $P = 0$, no error occurred
2. If $C \neq 0$ and $P = 1$, a single error occurred that can be corrected
3. If $C \neq 0$ and $P = 0$, a double error occurred that is detected but that cannot be corrected
4. If $C = 0$ and $P = 1$, an error occurred in the P_{13} bit

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

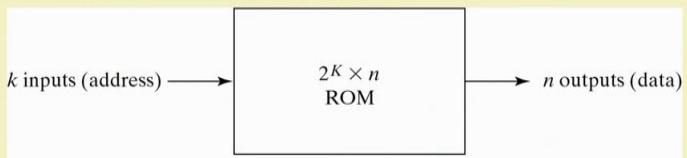
So, in that case we can have different situations C equal to 0, P equal to 0 no error has occurred; $C_0 = 0$, $P = 1$ a single error has occurred and that can be corrected.

If $C_0 = 0$ and $P_0 = 0$, then double error has occurred it is that can be detected, but cannot be corrected if $C = 0$ and $P = 1$ and error occurred in the P_{13} bit. So these are the inferences from this C and P bits. Again it is not possible to proof these results in our class so you I will suggest that you look into some discussion on this coding theory, for getting more details about this type of coding.

(Refer Slide Time: 25:17)

Read-Only Memory

- A block diagram of a ROM is shown below. It consists of k address inputs and n data outputs.
- The number of words in a ROM is determined from the fact that k address input lines are needed to specify 2^k words.



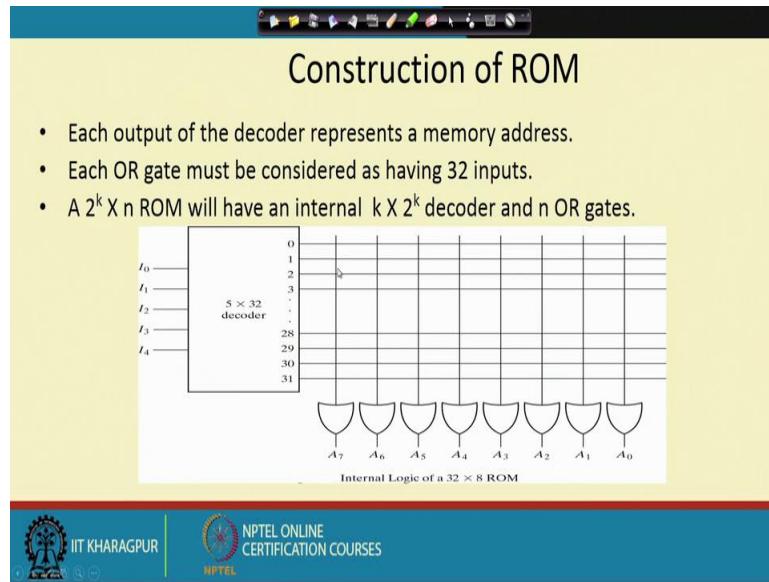
The diagram shows a central rectangular box labeled "2^K × n ROM". An arrow points to it from the left with the label "k inputs (address)". Another arrow points away from it to the right with the label "n outputs (data)".

ROM Block Diagram

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Next we look into read only memory. So read only memory so if the block diagram is like this so you have got k input lines, and we have got n output lines. So this the number of words in a ROM is determined from the fact that the k input address lines are needed to specify 2^k words. So 2 to the power k words so are there so and a k bit input address is necessary then there are n output lines from there.

(Refer Slide Time: 25:47)



Now, how do you construct a ROM? So logically a ROM is constructed like this that, each output of the so we have we take a 5 to 32 decoder, so if I have got say their 32 location ROM that we want to make. So we take this a 5 to 32 decoder address lines they come to this $I_0 I_1 I_2 I_3 I_4$ and then this 32 lines are coming out of the decoder.

Now, we have got this columns and this columns are summed in the OR gates so each OR gate is considered is having 32 inputs, so $2^k \times n$ ROM will have an internal $k \times 2^k$ decoder and n such OR gates. But essentially so whatever design that we have shown here is just a logical diagram physically it is never done like this, but logically you can think of it that these are the components in the ROM.

(Refer Slide Time: 26:51)

Truth table of ROM

- A programmable connection between two lines is logically equivalent to a switch that can be altered to either be close or open.
- Intersection between two lines is sometimes called a cross-point.

ROM Truth Table (Partial)													
Inputs					Outputs								
I ₄	I ₃	I ₂	I ₁	I ₀	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
0	0	0	0	0	1	0	1	1	0	1	1	0	
0	0	0	0	1	0	0	0	1	1	1	0	1	
0	0	0	1	0	1	1	0	0	0	1	0	1	
0	0	0	1	1	1	0	1	1	0	0	1	0	
1	1	1	0	0	0	0	0	0	1	0	0	1	
1	1	1	0	1	1	1	0	0	0	0	1	0	
1	1	1	1	0	0	1	0	1	0	1	0	0	
1	1	1	1	1	0	0	1	1	0	0	1	1	

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

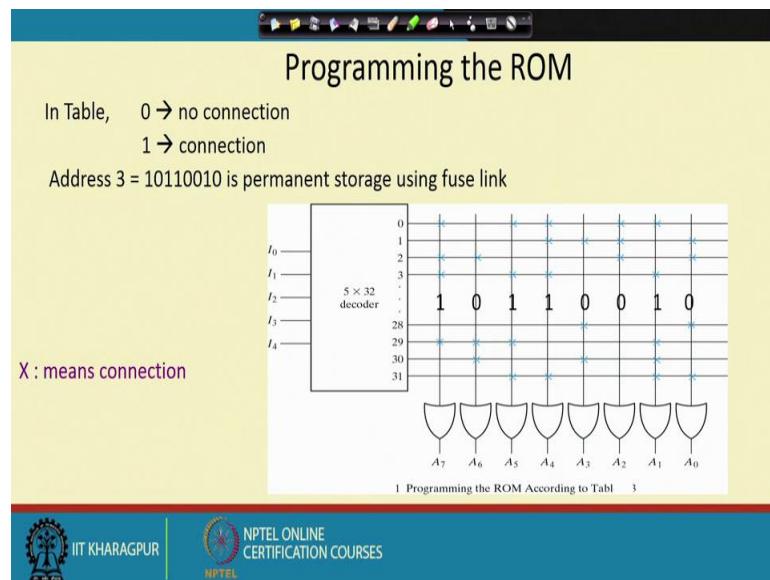
So, you can program the ROM like say a programmable connection between 2 lines is logically equivalent to a switch that can be altered may be either closed or opened. So for example, we can say that say this for this I₀ so if this address is 0 0 0 0 0 at that location this is the output pattern that I expect similarly, at say 1 1 1 0 0 so this is the input pattern that I expect.

So if we have got separate if we if in this in this point so you have got some connecting switches at these junctions, we have got connecting switches then those switches can be programmed in this fashion, so that it can talk about it can store the corresponding pattern that we need for the ROM location. So this can be done by means of putting some transistors and that will act as switch. And, there are switch the transistors can be programmed to either 1 or 0 so that the location content becomes 1 or 0.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 43
Memory (Contd.)

(Refer Slide Time: 00:18)



So, this programming of this ROM so, suppose we follow this convention that a 0 will be represented by the situation when there is no connection, and one will be represented by the situation when there is a connection. So, this address location 3 so, you see these crosses are there so, that means, these points are connected. So, they are so, this one is connection so, this if you look into these crosses. So, they are representing the pattern 1 0 1 1 0 0 1 0.

So, when this particular line is selected, since this is a decoder so, at one point of time only one of these 32 locations will get selected. So, when this if the input pattern applied here is corresponding to the address 3, then this line will get selected, and this line will be since the crosses are here, so, so this is one so, this A_7 will be equal to 1, A_6 will be equal to 0, A_5 will be equal to 1.

So, at the output we will get the pattern that is stored or at the location 3. So, this way we can program the ROM so that we can, so, for some permanent content there so, we can the

permanent storage. So now, even if the power goes so, this fuse, these fuses remain, so, these contacts remain.

So, as a result you will be getting the content as it is when you put the ROM into the circuit, next time so, the value does not change.

(Refer Slide Time: 01:43)

Combinational circuit implementation

- The internal operation of a ROM can be interpreted in two ways: **First**, a memory unit that contains a fixed pattern of stored words. **Second**, implements a combinational circuit.
- Previous figure may be considered as a combinational circuit with eight outputs, each being a function of the five input variables.

$$A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \dots, 29)$$

Sum of minterms

In Table, output A_7

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you can use the ROM for realizing some combinational circuit, ok. So, **is a** internal operation of a ROM can be interpreted in 2 way, first as a memory unit that contains a fixed pattern or stored words that we have already seen. The second is for implementing a combinational circuit.

Let us take an example, suppose we want to store, in the previous example that I have taken. So, this we can be considered as a combinational circuit with 8 outputs each being a function of 5 input variables. So, for example, here this A_7 is a 5 input function, A_6 is another 5 input function, A_5 is another 5 input function, A_0 is another 5.

So, I can simultaneously realize 8 such 5 input functions by programming the ROM. Why? Because if I apply a particular pattern here, then this A_7 to A_0 , so, they will be containing the content of this so, I can ideally visualize it like this that if I_0 to I_4 . So, they represent some input for the function A_7 , and since this A_7 this is equal to 1 so, this function output

say 1 here, but for the function A 6. So, this is not the case so, this is equal to 0. So, A 6 will be output will be 0.

So, so if I say that this function A 7 is like this. So, this function A 7 so, it has got the minterms 0, 2, 3 some other min terms are there and 29 is there suppose this is the A 7.

(Refer Slide Time: 03:35)

Example

- Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

Derive truth table first

Truth Table for Circuit :

Inputs			Outputs						Decimal	
A ₂	A ₁	A ₀	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀		
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	1	0	0	0	4
0	1	1	0	0	1	0	0	1	0	9
1	0	0	0	1	0	0	0	0	0	16
1	0	1	0	1	1	0	0	1	0	25
1	1	0	1	0	0	1	0	0	0	36
1	1	1	1	1	0	0	0	1	0	49

IIT Kharagpur | NPTEL Online Certification Courses

Then we can, it can be realized like this. So, if you look back so, this so for this is. So, A 7. So, 0, 2, 3 and 29 so, if you look into this table then you see that 0 for the 0 row. So, this cross is there, one we don't have the cross 2, I have the cross 0, 2, 3 and 29. So, at these points I have got the cross.

So, if this input pattern is such that these rows 0, 2, 3 or 29 get selected, then the A 7 output will be equal to 1, if the pattern is such that some other output is getting selected from the decoder, then the output of A 7 will be 0. So, in some sense I can say that this A 7 is realizing the combinational function whose minterms are 0, 2, 3 and 29. So, this is what is shown here.

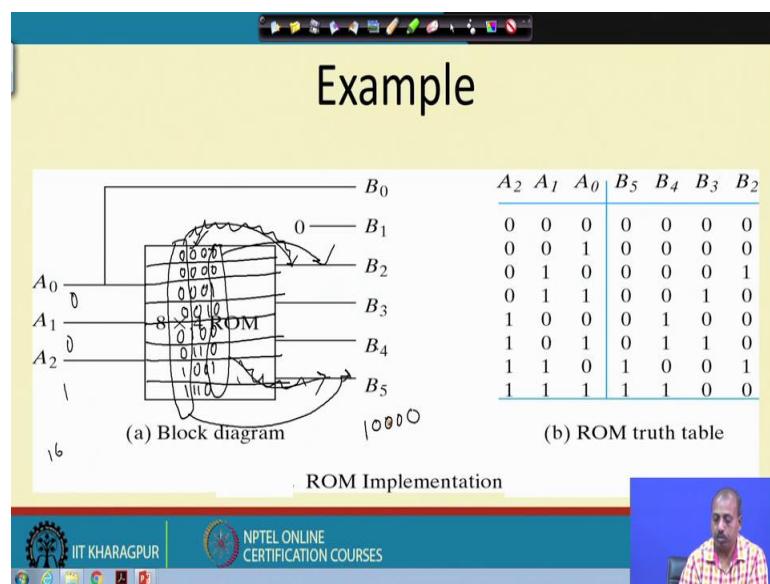
So, let us take another example so, we want to design a combinational circuit. The circuit will accept a 3-bit number, and generate an output binary number equal to square of the input number. So, I have got this 3 bit input so, A 0, A 1, A sorry, sorry they should be A 2 this should be A 2 not A 1. So, A 0, A 1, A 2 and I want the square of that.

So, this number is 0 so, square of that is 0. So, basically I am looking for a circuit where I have got these 3 inputs A 0 A 1 A 2. And it outputs 6 bits so, B 0, B 1, B 2, B 3, B 4, B 5 so, B 0 to B 5. So, these are the outputs, ok, I am trying to design this circuit such that this circuit it makes the square of these 3-bit number.,

Now, this 0 0 0 square of that is 0, 0 0 1 square is 1, 0 1 0 square is 4 so, this is 4. 0 1 1 that is 3 square is 9. So, 0 0 1, 0 0 1 this way I make this circuit, ok. I have to I have this is this is that functional table that I have. Now the knowledge that we have gathered in our combinational circuit design it says that, you have to design 5 different Karnaugh maps or 6 different Karnaugh maps and do minimizations, and then you have to draw the circuit using logic gates to realize the function.

So, this ROM based realization is somewhat simpler.

(Refer Slide Time: 06:08)



Because what we do? We take one 8 by 4 ROM and these address lines we feed A 0, A 1, A 2 and this in this data line so, we look into this thing more carefully, if you look into this part very carefully you see that B 1 is always 0. So, I do not need any logic to realize B 1. So, B 1 is always 0, and B 0 is always equal to A 0. So, B 0 and A 0 are same, if A 0 is 0, B 0 is 0. So, this that is no point storing this B 0 and B 1 lines.

So, I can directly take it B 0 B 1 equal to 0, and B 0 equal to A 0. But for B 2, B 3, B 4 and B 5, they are dependent on the values of this A 0, A 1 and A 2. So, what we do? We

take 1 A so, this is the final truth table that I need to store. So, I need to store the truth table for this B 2, B 3, B 4 and B 5, because I know B 1 is equal to 0, and B 0 is always equal to A 0, B 0 is always equal to A 0.

So, from this table so, if we just draw this one so, it says that for 0 0 0, the output should be 0 0 0 0, 0 0 1 it is 0 0 0 0 like that so, this is the output. So, what I do? I take one 8 by 4 ROM so, it has got A 0, A 1, A 2 as the output so, the if I take the 8 locations of this ROM.

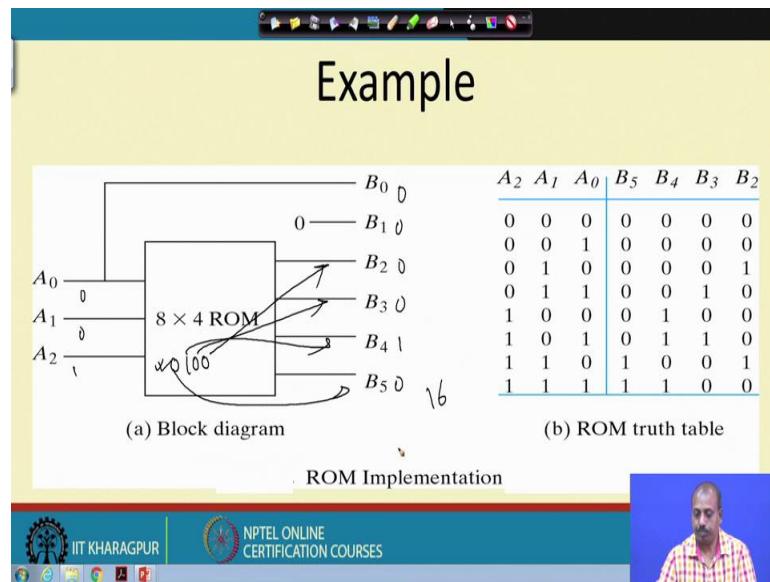
In the first location, it is a 8 by 4 ROM so, the there are 8 locations the first location there are 4 bits so, I store 0 0 0 0. Second also I store 0 0 0 0, third I stored 0 0 0 1, then 0 0 1 0, then 0 1 0 0, then 0 1 1 0, then 1 0 0 1 and 1 1 1 0.

So, this first bit so, that is this bit is transferred to B 2, similarly the second bit is transfer second bit is transferred to B 3, third bit is transferred to B 4 and this last bit will be transferred to B 5. So, you will be getting this function realized. Now, suppose A 0, A 1, A 2 comes as say 1 0 0, suppose it gets 1 0 0. So, if it gets 1 0 0, then my pattern is sorry this is slightly wrong. So, this is this is not connected here, rather this bit is connected to B 5, and this bit is connected to B 2. As it is, so, you see that B 5 is the most significant bit. So, B 5 is connected so, the first bit that I have stored here should be connected to B 5 and this bit should be connected to B 2.

So, so if I apply the pattern 1 0 0 then the square of that is so, this is 4 so, that is 16. So, that is so, that is that is 16. So, 16 is the pattern is given as 1 0 0 0 0, ok. So, if I just look into this table, if I look into this table so, this is the so, say 1 0 0. So, 1 0 0 the output is 0 1 0 0 0 so, that should be the output. So, B 4 is one and rest of the thing should be all 0.

So, in this circuit in this case what is what will happen? If I apply 1 0 0 then this B 5 will be equal to 0. So, in this I have stored this pattern 0 1 0 0 here. I have stored the pattern 0 1 0 0 here 0 1 0 0 at that location, at location 4.

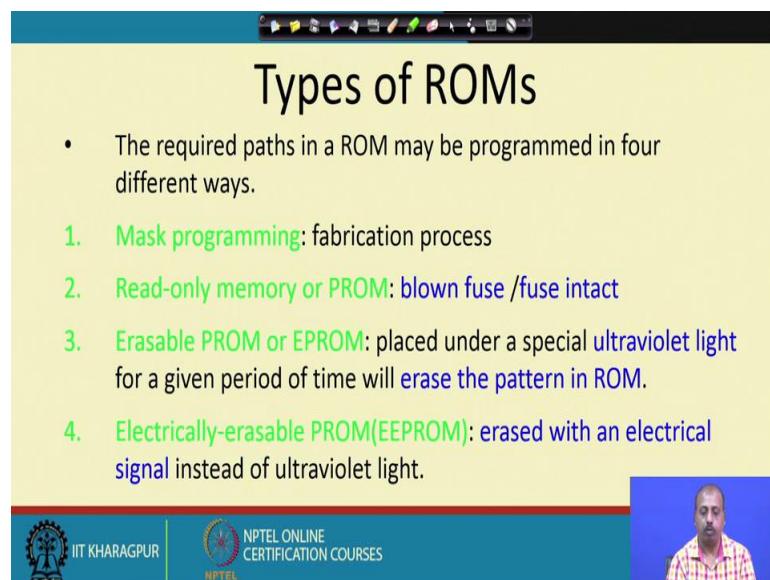
(Refer Slide Time: 10:32)



So, when I give the pattern 1 0 0 here so, this location gets selected, and as a result this 0 comes to B_5 , this one goes to B_4 , this one goes to B_3 and this one goes to B_2 ok.

So, you will be getting like so, 0 1 0 0 and so, this is always, all these are always 0. So, which is basically equal to 16, ok. So, this way you can use some ROM to get the combinational functions realized.

(Refer Slide Time: 11:11)



Next will be looking into other types of ROMs so, there are different types of ROMs like the very simplistic type of ROM their known as mask ROM. So, they are at the fabrication

time itself so, this is fixed then there are some read only memory or PROM programmable read only memory. So, they are so, if what is so, here normally we have got a programmer available is there it is a hardware circuitry.

So, you can put the ROM there in a, in a slot, and apply some high voltage accordingly the ROM will get programs. So, so the different locations of the ROM, it will get different values, then there is erasable PROM or EPROM. So, that is so, you can program it by means of applying high voltage, but for you can also erase the content in case of PROM. So, you cannot erase the content so that is one time programmable. Whereas, this erasable PROM or EPROM.

So, you can erase the content also by exposing the chip to the ultraviolet light for some period of time and this electrically erasable PROM or EEPROM. So, they can be erased with an electrical signal instead of ultraviolet light.

So, that way so, you can you need not take the chip out of the board. So, if the chip can be there in the circuit board itself, only we apply a high voltage so, the chip content will be erased. Whereas, for EPROM you have to take the chip out and expose it to ultraviolet light separately for erasing it. So, these are the different types of ROMs that we have.

(Refer Slide Time: 12:44)

The slide has a yellow background and a blue header bar with various icons. The title 'Combinational PLDs' is centered at the top. Below the title is a bulleted list of four items:

- A **combinational PLD** is an integrated circuit with **programmable gates** divided into an **AND array** and an **OR array** to provide an **AND-OR sum of product** implementation.
- **PROM**: **fixed AND array** constructed as a decoder and **programmable OR array**.
- **PAL**: **programmable AND array** and **fixed OR array**.
- **PLA**: both the **AND** and **OR arrays** can be programmed.

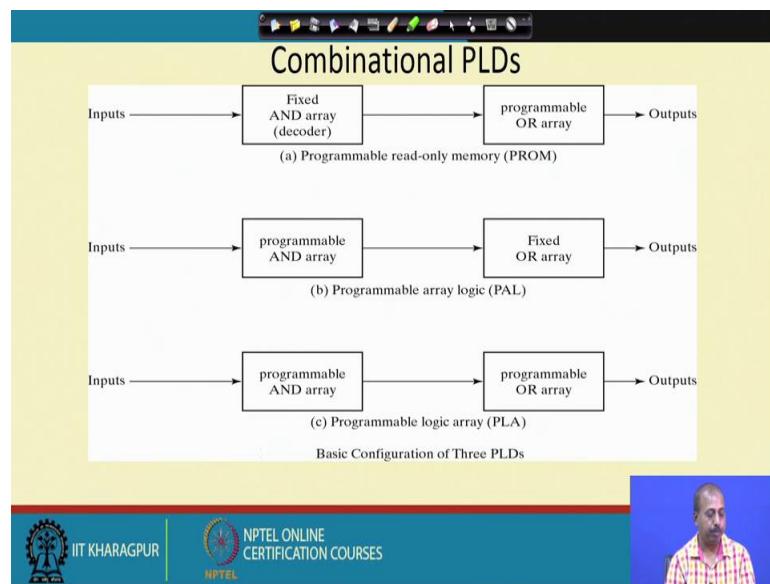
At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL. On the right side, there is a small video window showing a man speaking.

There are some other programmable logic devices. So, they come under the broad heading of combinational PLD's or programmable logic devices. A PLD is an integrated circuit

with programmable gates divided into an AND array and an OR array to provide AND OR sum of product type of realizations.

So, in case of PROM we have got fixed AND array constructed as a decoder and a programmable OR array. We have got PAL or programmable array logic. So, you have got programmable AND array and fixed OR array and the PLA where the AND OR both are programmable.

(Refer Slide Time: 13:25)



So, this is the PROM type of PLD that we have, where the AND part is fixed. So, you apply the input so, one of the outputs will get selected. So, that is the fixed AND then we have got this programmable OR so, you connect some of these devices.

So, while discussing decoder based combinational circuit design, you have seen this type of structures where we have got this decoder. And some of the decoder outputs are odd together. So, for whatever a min terms, the circuit output should be one. So, we have odd those outputs of the decoder to get the programmable output.

So, the same thing is here so, if you are using a decoder and then the AND part becomes fixed and you have got the programmable OR part to get the PROM. So, we have got this PLD or this PLA or programmable array logic where the AND part is programmable, but OR part is fixed. And we have got this PLA where the both AND plane and OR plane they are programmable.

(Refer Slide Time: 14:37)

Programmable Logic Array

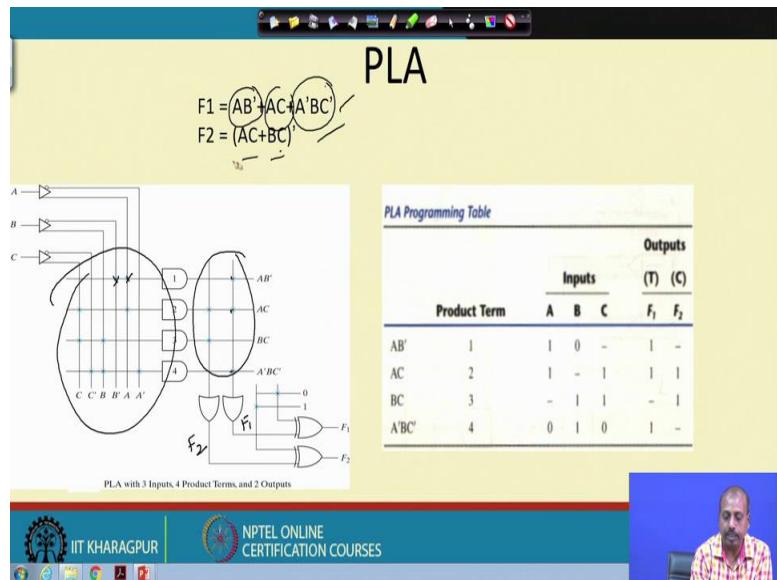
- The decoder in PROM example can be replaced by an array of AND gates that can be programmed to generate any product term of the input variables.
- The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.
- The output is inverted when the XOR input is connected to 1 (since $x \oplus 1 = x'$). The output doesn't change and connect to 0 (since $x \oplus 0 = x$).

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES



So, we will be looking into this PLA programmable logic array. So, in case of programmable logic array, decoder in the PROM example can be replaced by an array of and gates and that can be programmed to generate any term any product term of the input variables. I think there is an example here.

(Refer Slide Time: 14:56)



So, suppose I have to, got, I have to realize these 2 functions. So, F_1 equal to this and F_2 equal to this.

Now, so, also if first we will look into the product lines that I need. So, I need the line $A\bar{B}$, then AC , $\bar{A}B\bar{C}$, then again AC , BC and this whole bar so, what is done? So, this AB , AC , $\bar{A}BC$ and BC . So, these are the 4 product terms that I need. So, this is so, this is the AND plane the programmable and plane so, this switches that I have that I shown here.

So, they all these points are programmable, out of that so, these connections have been made, as a result on this line you will get $A\bar{B}$, on this line you will get AC , this line you get BC and this line you will get $\bar{A}B\bar{C}$, and then that this so that is about the so, this is the and plane, and this side is the or plane. In the OR plane, I am connecting the product terms product lines in such a fashion, that their OR will give me the individual function.

So, this gives me F_1 and this gives me sorry, this gives this gives us F_1 this gives us F_2 . So, this $A\bar{B}$, AC and $\bar{A}B\bar{C}$ so, these 3 points so, they are connected to this or gate. So, their connections are fused so, you get this term here. And then there is an XOR gate so that we can do some inversion if needed.

Like see here F_2 you see, that in F_2 we said it should be $\overline{AC + BC}$. So, after getting $AC + BC$ I need to do a complement. And that complementing is done by this additional XOR gate that I have, and here there is a problem there are 2 programmable points or 4 yeah.

So, you can select either the 0 to come to the XOR gate or the 1 to come to the XOR gate. Since for F_1 , we do not need the complemented output direct F_1 is necessary. So, this is connected to 0, for F_2 I need this complemented output so, this is connected to 1 so, you get this F_1 and F_2 , so, in the programmable PLA programming table. So, we first find out what are the product terms, what are the inputs and I could what are the outputs. So, this is known as the PLA programming table.

(Refer Slide Time: 17:21)

The slide has a yellow header bar with a toolbar icon at the top. The main title 'Programming Table' is centered in a large black font. Below the title is a numbered list of four steps:

1. First: list the **product terms** numerically
2. Second: specify the **required paths between inputs and AND gates**
3. Third: specify the **paths between the AND and OR gates**
4. For each **output variable**, we may have a **T(true) or C(complement)** for programming the XOR gate

At the bottom of the slide, there is a blue footer bar containing the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man speaking.

And once this is done, we can, we can go for the actual realization. So, for this programming table formulation, first we list down the product terms numerically, then we specify the required paths between inputs and AND gates. Then in the third phase we specify the paths between AND gates and OR gates.

And then for each output variable we may have a true output or a complemented output for the programming the XOR gate. So, that is how this is done for the programming table part.

(Refer Slide Time: 17:53)

The slide has a yellow header bar with a toolbar icon at the top. The main title 'Simplification of PLA' is centered in a large black font. Below the title is a bulleted list of two points:

- Careful investigation must be undertaken in order to reduce the number of distinct product terms, PLA has a finite number of AND gates.
- Both the **true and complement of each function** should be simplified to see which one can be **expressed with fewer product terms** and which one provides product terms that are common to other functions.

At the bottom of the slide, there is a blue footer bar containing the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man speaking.

Sometimes it is better that we do a more careful investigation to see whether I can reduce the PLA size further. So, it may so happen that if I compliment one of the output then there will be there may be more commonality between the product terms of different functions. Since the AND plane is realizing all the product terms so, if I can get a situation where this product terms are less so, I will be getting a better realization.

(Refer Slide Time: 17:25)

So, this, what is done? Is, it is like this suppose I have to realize these 2 function, F_1 which is which has got the mean terms 0, 1, 2, 4 in it and F_2 has got ABC which is 0, 5, 6, 7. The two functions are simplified in the maps as shown here, now while doing the simplification, we try for both F_1 and \bar{F}_1 , ok.

So, if our F_1 so, if you if you would group with, 1 element so, you will get $\bar{A}\bar{B}$ like if you if you combine these 2, so, you will get $\bar{A}\bar{B}$, then this $\bar{A}\bar{C}$. So, combining this one and this one you will get $\bar{A}\bar{C}$, and you will get $\bar{B}\bar{C}$ by combining these 2.

On the other hand, if you group in terms of 0's so, which you are getting AB like say here, this is A equal to 1 and B equal to 1. So, these 2 term you can get. So, if you are so, this AB will be coming like so, this is basically the complement of this. So, if you complement it, then this function will come so, this is, the 0 elements are taken into consideration then you will be you will be getting this like here $\bar{A}\bar{B} + \bar{A}C + B\bar{C}$, that is the function we will get.

And similarly here for F_2 for \bar{F}_2 also you will get $AB + AC$, and if you are doing it taking \bar{F}_2 that is then you will be getting the other function $\bar{A}\bar{C} + \bar{A}B + A\bar{B}\bar{C}$. Now what we do is that we compare all these functions and try to figure out the common terms. And we see that if I choose this complemented form of F_1 , and true form of F_2 then this AB and AC so, these terms are becoming common.

So, I will, so, they were as these terms will become common so, I can just take them together and get it realized. So, the less number of product terms will be sufficient for realizing this thing.

(Refer Slide Time: 20:31)

PLA table by simplifying the function

- Both the **true** and **complement** of the functions are simplified in **sum of products**.
- We can find the same terms from the group terms of the functions of F_1, F_1', F_2 and F_2' which will make the minimum terms.

$$F_1 = (AB + AC + BC)'$$

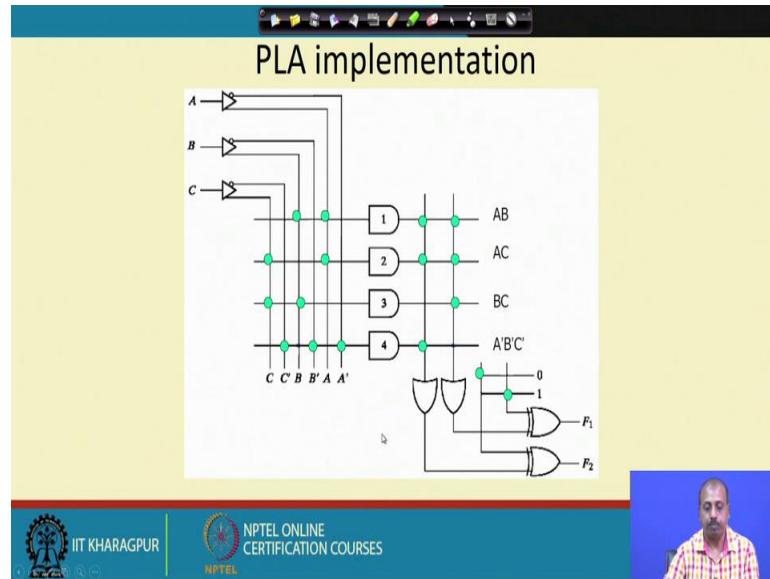
$$F_2 = AB + AC + A'B'C'$$

Product term	Inputs			Outputs	
	A	B	C	(C)	(T)
AB	1	1	1	-	1
AC	2	1	-	1	1
BC	3	-	1	1	-
$A'B'C'$	4	0	0	0	-
	F_1	F_2			

So, we do it that and the both true and compliment of the functions are simplified in sum of product form, and we can find the same terms from the group from the group terms of the functions F_1, \bar{F}_1, F_2 and \bar{F}_2 . which will make the minimum number of terms.

So, we find that F_1 should be taken in complemented form and F_2 it should be taken in true form to get the minimum sized minimum number of product terms. So, that is done here.

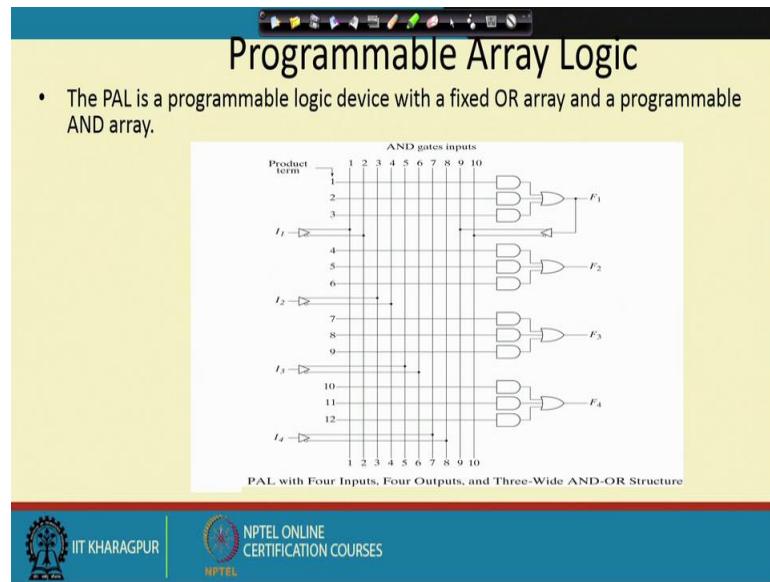
(Refer Slide Time: 20:59)



And final PLA implementation turns out to be like this; that we take these terms so, that this is simple. So, this term realizes the product term AB. So, this A and B so, those 2 lines are switches are bond, so, this AB comes here.

Similarly, AC comes here, BC comes here, and then after doing the XOR after doing the OR plane here, then we select this XOR programming point so that for F₁, I get \bar{F}_1 . and F₂ comes directly. So, this way we can have PLA implementation of this circuit.

(Refer Slide Time: 21:34)



So, then we will this programmable array logic so, this is another programmable device, where we have got this AND plane fixed so, this AND, sorry, the OR plane is fixed and the AND plane is programmable so, then the, in the OR plane we have we can you can get OR of these 3 AND gates.

So, so, this so, you cannot choose any other one whereas, while drawing this AND gates so, you can program all these points to select the inputs that should come for the, for the first AND gate. So, this OR gate is fixed so, this is the OR of these 3 inputs, these 3 AND gates. Similarly, second OR gate is the, is the OR of these 3. Now you can select so this first 2 lines they correspond to I_1 and \bar{I}_1 . Second 2 lines correspond to I_2 and \bar{I}_2 .

So, sometimes we take this F_1 back, and this F_1 and \bar{F}_1 . that is 2 additional inputs that comes back here. So, that is sometimes helpful, if there is a sharing of function between F_1 and something else, then you can pass it through these lines and use it on the on the lines 9 and 10 and that can be used for AND gate connection so, we will see that.

So, essentially in the in the PAL device so, this OR part is fixed and the AND part is programmable.

(Refer Slide Time: 23:00)

PAL

- When designing with a PAL, the Boolean functions must be simplified to fit into each section.
- Unlike the PLA, a product term cannot be shared among two or more OR gates. Therefore, each function can be simplified by itself without regard to common product terms.
- The output terminals are sometimes driven by three-state buffers or inverters.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, Boolean functions must be simplified to fit into each section. So, you see that if you in a case so, if you if you need F_1 as a function which requires more than 3 product terms,

then I cannot fit it. Similarly, if F_2 requires more than 3 product terms, I cannot fit it standalone.

Of course, what you can do possibly if you is that for realizing one function if you require more than more product terms, then you can break it down into 3 product term cases and then additionally use some OR gate at the output of those 2 and or them together. But that is costly that is some additional gate will be necessary.

So, that way it is restrictive unlike PLA, a product term cannot be shared among 2 or more OR gates. Therefore, each function can be simplified by itself without regard to common product terms. So, this is this form from the simplification point of view. So, this is easy so we do not need to compare between the product into between the functions to see whether there is a commonality of product terms between the functions. So, we can simply take individual functions separately, and we can minimize them so that they go to one of the alternative.

And the output terminals are sometimes driven by 3 state buffers or inverter so, that is also done. So, that we can get the inverted output.

(Refer Slide Time: 24:24)

The screenshot shows a presentation slide with the title "Example". Below the title, four Boolean functions are defined:

$$w(A, B, C, D) = \sum(2, 12, 13)$$
$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

Below the functions, the text "Simplifying the four functions as following Boolean functions:" is followed by the simplified forms:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D = w + AC'D' + A'B'C'D$$

The slide footer includes the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer, there is a small video window showing a man speaking.

So, let us take an example suppose w $ABCD$ is this one so, w x y z so, these are 4 functions. So, we try to simplify the 4 functions like this w equal to $AB\bar{C} + \bar{A}\bar{B}\bar{C}D$. X equal to $A + BCD$, y equal to $\bar{A}\bar{B} + CD + \bar{B}\bar{D}$ and z equal to $AB\bar{C} + \bar{A}\bar{B}\bar{C}D + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$. Then

after doing this, we find that first 2 terms of z are same as w. So, these we rewrite it as $w + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$. So, after doing this rewriting now a so now, we are satisfied because individual, individual functions they are having at most 3 product terms, ok. So, previous so, the, this particular expansion of z so, that was problematic, because it was requiring 4 product terms. So, but here it is requiring 3 product terms only. So, if it was, if some line is requiring more than 3 product terms, then we can take a dummy output, and the dummy output can be fed as input for the next one so, I will come to that.

(Refer Slide Time: 25:31)

PAL Table

- z has four product terms, and we can replace by w with two product terms, this will reduce the number of terms for z from four to three.

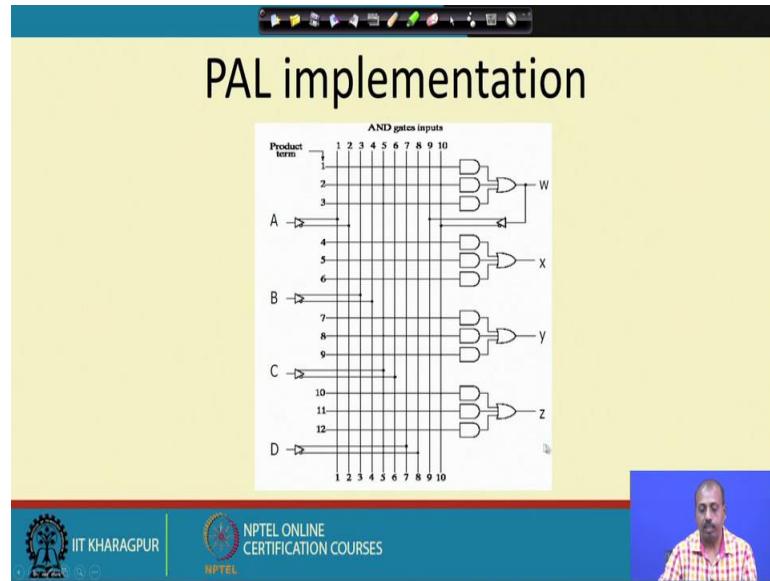
Product Term	AND Inputs					Outputs
	A	B	C	D	W	
1	1	1	0	—	—	$w = ABC'$ + $A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A$ + BCD
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B$ + CD
8	—	—	1	1	—	
9	—	0	—	0	—	+ $B'D'$
10	—	—	—	—	1	$z = w$ + $AC'D'$ + $AB'C'$
11	1	—	0	0	—	
12	0	0	0	1	—	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL



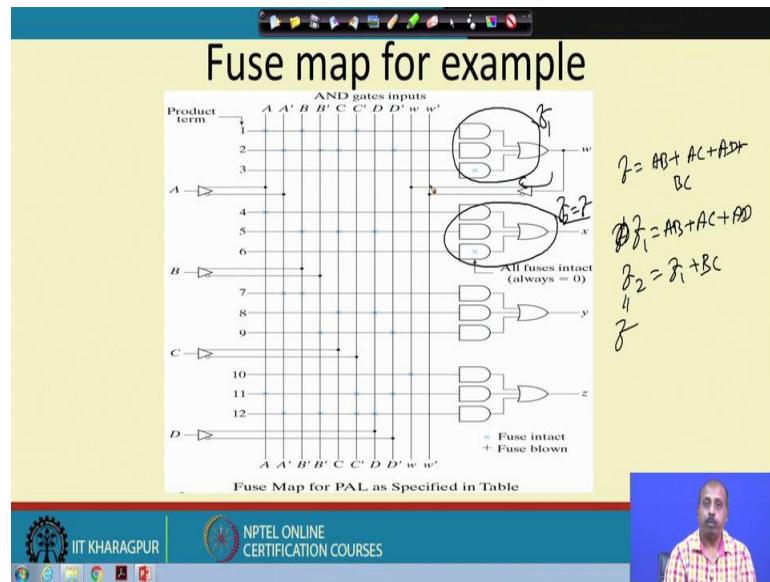
First of all, so, this realization so, z has 4 product terms and we can replace by w so, here like this. So, this way we can make it simplified.

(Refer Slide Time: 25:41)



And then the realization becomes so, w line is connected back. And it is connected to this this 9 and 10 lines they carry w and \bar{w} lines, and they can be used for connecting to z.

(Refer Slide Time: 25:56)



So, this way we can have a PAL implementation. So, this fuse map that tells like how are you going to connect. So, this product line so, this is $AB\bar{C}$. So, that is connected to this AND gate. Similarly, this $\bar{A}\bar{B}C$ so, that is connected to this AND gate, and this AND gate is not used at all. So, this is it had only this function had only 2 product lines. So, this AND gate is not necessary so, this is not programmed.

So, all these connections are opened. Then for the second one again the same thing I have got connections where third AND gate is not necessary, all fuses are intact. So, it is always equal to 0. So, this way we can do the connection and the w output so, it is fed back to this w and w' lines. And this w and w', this w line is used here by the z output as another AND input to this OR gate. So, that way we can do this realization.

Now, if there are if you get some function which is the more, having more number of product terms, then what you can do is you can take a dummy output, like suppose I have got a function f consisting of the product terms like $AB + AC + AD + BC$, say something like this. Then what you can do? You can have A_1 such you can take a dummy output. So, f' , let us call it f_1 . So, f_1 is say $AB + AC + AD$, and then f_2 is $f_1 + BC$.

So, you can break it down like this and use this type of feedback connection for realizing this. So, for this first by this first set you realize f_1 , and this f_1 is fed back here and then by the second set you realize f_2 that is equal to f . So, f_2 is equal to f and this $f_1 + BC$ is realized by that. So, this way you can utilize this back connections.

So, that you can have functions with larger number of product terms realized, but of course, it is restrictive in the sense that the device that you are using may not have a large number of such connection, back connections available. So, you may not be able to realize very large functions with this type of logic.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 44
FPGA

The family of logic devices next we will look into another very prominent structure that you will find in today's digital design field, which is known as field programmable gate array or in short FPGA. So, this as the name suggests so it is field programmable means the programming of the device can be done at the customer end. So, just like we can program a PROM that is programmable read only memory or say PLA, so here also we can do the programming. And the programming can be done at your end that that you can just buy the device and connect some you have to have some sort of burner and through that burner so the control program can be loaded on to this FPGA chip. And then the FPGA chip will be customized to your particular application.

So, it does not need detailed semiconductor manufacturing phases, so the last phase that is the final connection of the module so that can be done at the customer point. So, that drastically reduces the design turnaround time, so that is why it has become very popular. And also it is many cases, so it is reprogrammable so you can change the design. So, if there is an error so you can very easily correct it or if there is some modification or upgradation of the design, those things can be carried out very easily.

(Refer Slide Time: 01:46)

The slide has a yellow header bar with the title 'Evolution of implementation technologies'. Below the title is a bulleted list of technology milestones:

- Logic gates (1950s-60s)
- Regular structures for two-level logic (1960s-70s)
 - muxes and decoders, PLAs
- Programmable sum-of-products arrays (1970s-80s)
 - PLDs, complex PLDs
- Programmable gate arrays (1980s-90s)
 - densities high enough to permit entirely new class of application, e.g., prototyping, emulation, acceleration

To the right of the list, a vertical black arrow points downwards, labeled 'trend toward higher levels of integration'.

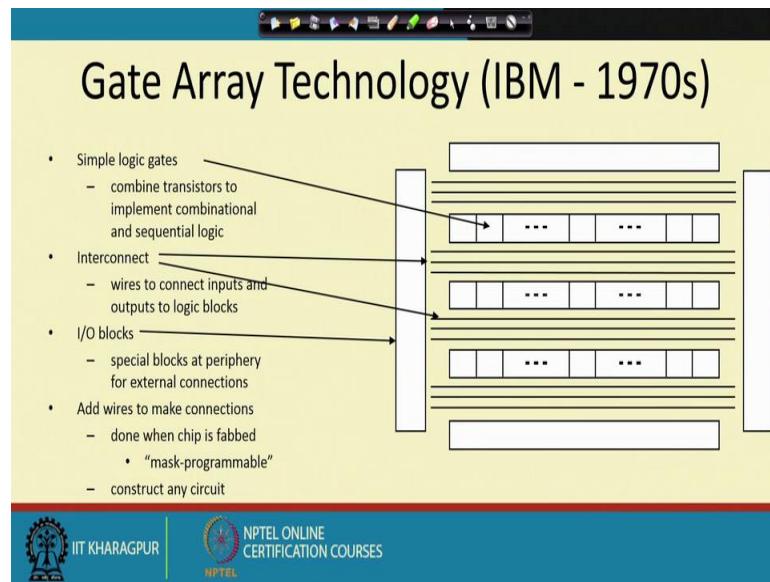
The footer of the slide includes the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

So, if we look into the evolution of this implementation technologies, so started with logic gates in 1950 and 60 then there was regular structures for 2 level logic gate 1960-70. So, 2 level logic means so you have got an AND level and a OR level, so this to a PLA type of structures then from there it evolved to more complex programmable devices which are known as PLDs programmable logic devices or CPLDs complex programmable logic devices that was 1970s and 80s.

Then came the programmable gate array so programmable gate arrays means you have got an array of gates of similar type and then you can just connect them in some fashion to get the overall, overall functionality implemented. So, in programmable gate array so density is high enough to permit entirely new class of application for example prototyping, emulation, acceleration. So entire design entire hardware design you can prototype on to this programmable gate arrays and you can get the thing done. So, you can have a basic idea about whether your design is going to work or not by doing a proper simulation or emulation of the system.

So, if you as you go from this logic gates towards this lower side programmable gate arrays. So, these higher levels of integration can be seen so more and more number of more and more number of gates can be clapped on to one device, so you can as a result you can implement more complex designs. So, if we just look into the gate array, so that was gate array so it was.

(Refer Slide Time: 03:43)



So, this is gate array technology so it came from IBM in 1970. So, the overall structure is similar to this, so we have got a say simple logic gates. So, like this so these are logic gates implemented in some layer and then there are some interconnects running in between.

So, this interconnect so they are running between the layers, so between the we can say between the rows of these logic elements and then these logic elements they are implementing some function. So, if you want to establish a connection from here to here say for example, so you have to make a connection like this and then at this point you have to make a connection like this.

So, what I mean is so maybe output of this module has to be fed to the input of this module, so you can use this interconnect to do the connection like this. So, this way these gate arrays can be combined through these interconnects to implement complex logic. Apart from that there are some IO blocks. So, normally this IO comes from the outside world. So, for driving these logic gates you need some inputs and those inputs are given through this input block and outputs are taken from the output block and as so to have protection against over voltage or to have a higher drive current etc., this integral, this IO blocks are designed in a much to be are design in such a fashion that they are capable of driving more load or they are capable of withstanding noise margin sufficiently, so that way this IO blocks are design.

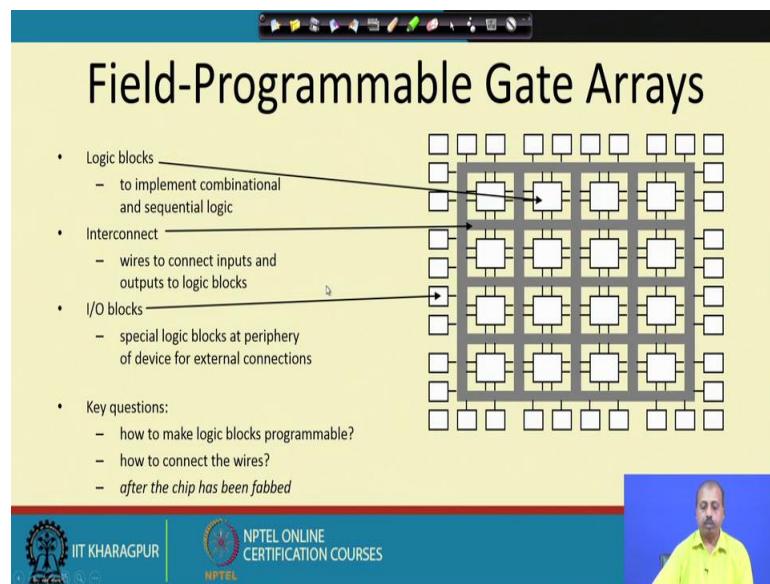
So, they are special blocks at the periphery of our external connections, then we can add wires to make connections. So, as I was showing there so you can have some extra wire connected and between these. So, these wires are these vertical wires are not shown here, but so there are connections from this and all those connections are programmable.

So, if you need to establish a connection then you need to burn out some of the points and as a result the connections will get established and the output of 1 block will be connected to input of another block. So, that way complex logic can be implemented. So, as a result so this is done when the chip is fabricated.

So, we can or the mask it is called mask programmable because, we can, we can put a mask for this type of connection and then we can do the connection or it may be. So, this is mask programmable, because this is not even, not yet in the customer side. So, this has to be done the final stage of connection has to be done at the fabrication house itself, but before the design has arrived to the fabrication house so the remaining part is fabricated.

So, only when the final connection have, when the design comes and the connections are determined, so the final metallization stage is carried out and it establishes the connection between the logic elements. So, it can be used for constructing any circuit.

(Refer Slide Time: 06:57)



Next we will be looking into field programmable gate array, so in field programmable gate array so it is otherwise similar. So, we have got logic blocks to implement combinational

and sequential logic. So, there are some logic gates and there are some flip flops which are part of this block logic block that can be used for implementing combinational and sequential logic. Then there are interconnects running both horizontally and vertically so these in the gray region.

So, they are for running interconnect and you can have some switches at this junction point, so these junction points are not shown explicitly. So, you can have some switch here and that switch will determine the connection between the horizontal and vertical lines. So, we have got interconnects that wires to connect inputs and output logic blocks and the IO blocks for the input output operation. So, these are special logic blocks at periphery of device for external connection.

So, the question that we have is first of all how to make this logic blocks programmable. So, I said that you can there may be some sort some sort of logic elements there, but those logic elements may be programmed to have different combinational and sequential function implemented. So, how to make this logic blocks programmable, how to connect the wires like as I was telling that I need to connect the output of this logic block to the input of this logic block, so how do we establish the connection and naturally when we say it is field programmable, so, this field means it has to be done after the chip has been given to the customer. So, at the customer side I we should be able to do this programming part. So, the design never goes to the fabrication house, so it is always design is with the customer only and then the getting on FPGA chip the customer should be able to program it, so that it gives the proper function.

(Refer Slide Time: 09:01)

CPLD vs. FPGA

- CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers.
- Results in less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio.
- The FPGA architectures are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for.
- In practice, the distinction between FPGAs and CPLDs is often one of size as FPGAs are usually much larger in terms of resources than CPLDs.
- Typically only FPGAs contain more advanced embedded functions such as adders, multipliers, memory and other hardened functions.
- Another common distinction is that CPLDs contain embedded flash to store their configuration while FPGAs usually, but not always, require an external flash

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are 2 types of logic devices that comes under this field programmable capability one is known as CPLD or complex programmable logic device. And another is called FPGA or field programmable gate array out of these 2 CPLD has a somewhat restrictive structure consisting of 1 or more programmable sum of products logic array feeding a relatively small number of clocked registers.

So, in CPLD the basic structure will be something like a 2 level structure. So, sum of product type of logic array will be there AND OR planes will be there and they will feed some number of clocked flip flops so. And overall CPLD capacity wise it is less capacity is less compared to FPGA. So, as a result the CPLD it has got less flexibility with the advantage of more predictable timing delays and a higher logic to interconnect ratio.

So, it is less flexible, but at the same time you this timing delays that will occur is predictable. So, even if this logic is implemented in the programmable part, so the distance between those points or the lengths of the wires are predictable. So, as a result the timing delays are predictable and higher logic to interconnect ratio means you have a there are more number of logic elements compared to interconnect.

So, it is likely that we can push in more amount of more number of logic gates on to the CPLD structure. On the other hand these FPGA architectures they are dominated by interconnect. So, they have got large number of interconnects in them, so that you can establish connection between logic elements very easily logic elements and flip flops

which is not the case for CPLD. So, this availability of interconnect now this makes them far more flexible in terms of the range of designs that are practical for implementation within them, but also for far more complex to design for.

So, that gives us the flexibility that you can have very wide range of designs implemented there, but at the same time it makes it very complex also. So you normally nobody does and FPGA design manually, so normally the cad tools are used for doing the design. In practice that distinction between FPGAs and CPLD it is often 1 of size as FPGAs are usually much higher in terms of resources than CPLDs. So, the total amount of logic elements that we have in FPGA total amount of interconnects that we have in FPGA, so they are much more compared to CPLDs. So, these FPGAs they are going to be much larger than CPLDs typically only FPGAs contain more advanced embedded functions such as adders, multipliers memory and other hardened functions.

So, what I mean is that if you look into advanced FPGAs now, so you will find that there are built in adder available built in multipliers available and memories are also available because, if you are if you are trying to make a design. So, if you have trying to make a system so normally what we will need is apart from this logic element processing you will require some amount of memory and many a times we need this addition, subtraction, multiplication division type of operations.

So, they if we have built in modules for that, then the implementation will be much more efficient compared to the customer making the implementation by himself. So, that way FPGAs advanced FPGAs they will contain this adder multiplier memory, and other functions which are not available in case of CPLDs. Another common distinction is that CPLDs contain embedded flash to store their configuration while FPGAs usually, but not always require an external flash.

So, CPLD devices they have got a flash memory within them, so as a result even if this power is even if it is switched off then thus the flash will contain the configuration program. So, when it is turned on so the CPLD functionality remains same; whereas, in case of FPGAs, in many FPGAs you will find that it will require an external flash. So, whenever you power on the FPGA chip or you give a reset so it will load the program from the external memory and then it will some external flash and they it will into an internal ram and then it will start operating in that fashion.

So, the point is that this embedded flash in CPLD that is good because, you do not have need this external flash; but at the same time it stops you from getting the reprogram ability. So, if I have got an external flash for this programming purpose, so I can just change the configuration program in the external flash. And we can get a new functionality realized by the same FPGA chip which is not true for CPLDs.

(Refer Slide Time: 14:08)

Programmability of FPGAs

- *User programmability* of CPLDs and FPGAs is achieved via user-programmable switch technologies.
- For CPLDs, floating-gate transistors are used like EPROM or EEPROM. On the otherhand, FPGAs normally use SRAM (static RAM) or antifuse technology.
- Properties of the switches, such as, *size*, *on-resistance*, and *capacitance* dictate trade-offs in architecture.
- In SRAM based FPGAs, there is an SRAM bit corresponding to each of the programmable points within the device.

So, if you look into this program programmability feature of FPGAs. So, user programmability of CPLDs and FPGAs achieved via user programmable switch technologies. So, both CPLD and FPGAs they use some sort of switch technology, for CPLDs we have got floating gate transistors like EPROM or EEPROM. So, floating gate transistors you know that it is these are transistors so if you apply a control voltage then the gate will appear.

So, as a result the communication, this device will be turned on and if you remove the voltage then this or if you apply some negative voltage or to are to remove that gate terminal. So, basically this is a transistor so this is, so, you know that this is a transistor. Now in this is a normal gate now there is another gate in case of CPLD and this floating gate thing. So, you have got another transistor another gate that will appear, so when we apply the voltage this additional gate appears as device gets turned on even if you remove this voltage now this gate will remain. So, as is the so this CPLD or FPGA using this floating gate technology so it will remain programmed in the same status.

Now, to remove this, remove this floating gate you have to apply some other voltage and then this will be going out. So, these are this CPLDs they used normally this floating gate philosophy of this like EPROM or EEPROM. FPGAs they normally use static ram or anti fuse technology. Of course, there are FPGAs that use a floating gate as well, but many of them use this static ram or this anti fuse technology.

So, properties of the switches such as size, on resistance and capacitance dictate tradeoff between the trade-offs in the architecture. So, these switches they are coming as extra so we need them for programming the device, but they do not participate in forming the logic part of the design.

So, as a result they are overheads you can say so they are size on resistance size will determine the, what is the amount of logic that you can put there. On resistance will determine the delay in the communications array on resistance and capacitance they will detect the dictate the delay that you will have. In the design in SRAM based FPGAs there is an SRAM bit corresponding to each of the programmable points within the device.

So, device has got many programmable point like between to whenever we have got 2 interconnect, so those 2 interconnects are connected to each other or not, so that is controlled by 1 SRAM bit similarly. So, the programmable device or wherever you can think about programmability so it is implemented by 1 SRAM bit.

So, there are large number of SRAM bits and that actually constitutes the configuration program and let when this the FPGA SRAM based FPGA they are switched on, so we get this configuration program is loaded on to the FPGA. So, that all those programmable points are now programmed to have proper values and then the chip behaves accordingly. So we will look into this SRAM these programming techniques slightly later.

(Refer Slide Time: 17:39)

- When the device is powered-on or reset, it reads a configuration program from an off-chip memory and loads it into on-chip SRAM.
- The configuration program defines the logic function realized by individual logic blocks and interconnections.
- Devices using SRAM based switching can be reprogrammed easily by just changing the configuration program.
- FPGAs belonging to *Xilinx*, *Plassey*, *Algotronix*, *Concurrent Logic*, *Toshiba*, etc. are SRAM based.
- SRAM provides fast reprogrammability at the cost of large area (at least five transistors for cell and one for switch).

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Now, so this is the thing that when the device is powered on or reset SRAM based FPGAs it reads configuration program from an off chip memory and loads it into on chip SRAM. So, the chip the FPGA chip will have got an SRAM which will have the configuration program.

So, when the chip is switched on or reset it is the configuration program will be loaded from off chip memory to the SRAM, the configuration program defines the logic function realized by individual logic blocks and interconnections. So, that will be determining the logic that is implemented by the logic blocks that we have and the interconnection, so that will determine the overall functionality of the FPGA chip.

Devices using SRAM based switching can be reprogrammed easily by just changing the configuration program, so that is very good, because if I normally what happens is that when you are in the early stage of you design so there will be designed bugs. So, if you go for directly chip fabrication then there is a possibility that many of those bugs are not detected at the time of design.

So, what we can do we can put this design onto an FPGA chip and we can check for the functionality whether the chip is working properly or not. If we find some bug there then nothing is lost because, I can just change the configuration program. So, I can correct my design as a result it will gives a new configuration program and that new configuration program can be loaded on to the again that SRAM based configuration memory and that

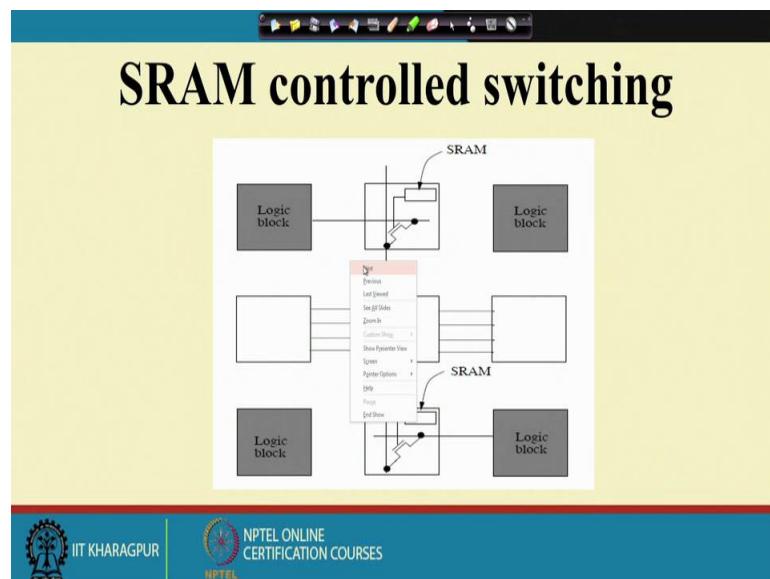
gives us the corrected chip and also if you are going for some upgradation like you have got some functionality.

So, you want to add a new function then also these FPGAs are going to be helpful because, all that you need is to change the configuration program. Now FPGAs belonging to Xilinx, Plassey, Algotronix, Concurrent logic, Toshiba, so they all use this SRAM based FPGAs SRAM based programming technologies. In fact, there are large numbers of vendors for this FPGAs. And these are some of them. Out of these Xilinx is the most popular because of its capabilities as we will see later others are also there so they are also used in many applications, SRAM provides fast reprogram ability at the cost of large area.

So, reprogram, as I said the reprogram ability becomes very easy because, you can very easily change the configuration program. However, the area occupied is more because each SRAM based will require 5 transistors ok. So, that way if I have got say 10000 programmable points, so there will be 50000 transistors just needed for making the configuration memory so that makes it that makes it very costly.

Anyway, so, that is a trade off so if you are asking for flexibility we have to pay for the extra space that is needed extra area that is needed for the FPGAs.

(Refer Slide Time: 20:53)



So, this diagram it shows how this SRAM can control the switching, so these are suppose some logic blocks within the FPGA and we want to establish a connection. So, output of this logic block should feed the input of this logic block.

So, how to do that so if these interconnect ok, so these interconnect comes to this is a switch box so this is a switch box. So, it has got this line and this line as 2 wire lines there and there is a programmable transistor here. So, this SRAM bit is going to control this transistor, so if this bit is equal to 1 in that case this transistor is on as a result a connection is established between this line and this line. So, with this the signal that is available that is given here will be available at this point.

Similarly, if this point is equal to 1 this SRAM bit is equal to 1. So, connection is established between this line and this line this transistor is turned on and these 2 lines are now shorted. So, as a result the output of this logic block will come to the input of this logic block so that will happen. And similarly whatever like here you see that there are 4 lines coming as input 4 lines going there out as output and similarly 4 more lines will come 4 more columns will come vertically and go out vertically.

So, between all these points so we have got this type of programmable transistors and by putting the proper SRAM bit equal to 1 so we can establish the connection. So, if we do not want the connection then the SRAM bit is 0. If we want the connection then the SRAM bit is equal to 1, this way we can use this controlled SRAM to control the switching.

(Refer Slide Time: 22:37)

The screenshot shows a presentation slide with a title 'Antifuse based programming'. Below the title is a bulleted list of facts about antifuses. At the bottom of the slide, there is a footer bar containing the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and various system icons.

Antifuse based programming

- Antifuses are originally open-circuit, offering very high resistance. However, on programming (applying a 11-20V across terminals), the resistance becomes very low, thus, establishing electric connections.
- Antifuses can be made very small using modified CMOS technology, thus offering very high device density, compared to SRAM.
- However, once programmed, they cannot be reused. Thus, the device is one-time programmable.
- The structure is commonly known as PLICE (Programmable Low-Impedance Circuit Element).
- PLICE uses Poly-Si and n+ diffusion as conductors and ONO (silicon diOxide - silicon Nitride- silicon diOxide) as an insulator.
- The advantages include small size (little more than the cross-section of two metal wires) and low series resistance.
- It has disadvantages, such as, large size of programming transistors, need of isolation transistors, and one-time programmability.
- FPGAs from *Actel, Quicklogic, Crosspoint, etc.* support antifuses.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Major problem with this SRAM based control is that the area that you need ok. So, another type of programming technology that is popular with FPGAs is known as Antifuse based programming, so Antifuses they were or they are originally open circuit. So, it is like this so conceptually you can view it a like this, so these are the two points and in between we have got a Antifuse. So, this Antifuse on this is open normally it is open circuit and that as a result the resistance between these two points is very high resistance between these two points is very high.

Now, if we apply a programming voltage so 11 to 20 volt across the terminals then as if so this connection gets shortened ok. So, this resistance becomes very low and that establishes an electric connection and now even if you withdraw the voltage the programming voltage the connection remains ok. So, that that is how this is going to be useful. So, wherever you need the connections or programming so we have to apply proper voltage across the terminals. So, that the resistance becomes low, the on resistance becomes low, so we have the connection gets established.

So, Antifuses, advantage that we have with Antifuse is that they can be made very small ok. So, they can be made very small using some modified CMOS technology thus offering very high device density compared to SRAM. So, SRAM, SRAM is 6 transistor shall I said. So, compared to that these Antifuses can be made very small, so that is why you can accommodate large number of Antifuses within small chip area. So, that is why it is a very popular, but the difficulty with Antifuse is that once it is programmed they cannot be reused.

So, it is for 1 time use only so this cannot be reused; so the device is 1 time programmable. So, we do not get the flexibility that I can use the same chip again and again and the other advantage that we had like say if there is a design error, so we can use, we can correct the error and use the same FPGA chip for corrected design. So, that is not possible for Antifuse based FPGAs, but still they are popular because of their low cost and this size of the Antifuse being small.

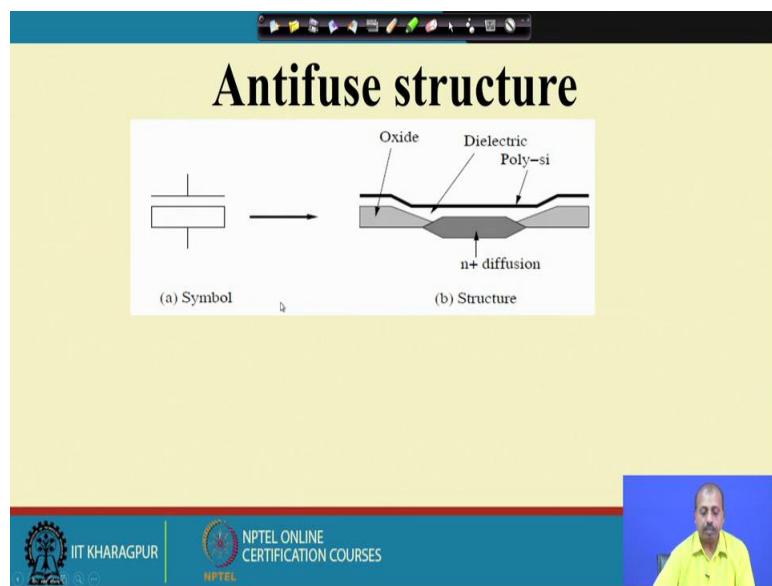
So, this Antifuse structure so they are now commonly known as PLICE programmable low impedance circuit element so and this PLICE it uses as far as manufacturing is concerned it uses poly silicon and n plus diffusions as conductor and ONO that is silicon diOxide silicon Nitride silicon diOxide as an insulator. So, the advantages that we have

that includes small size little more than the cross section or 2 metal wires, so as that is why it is size is pretty small and it has got low series resistance. So, there is resistance of this Antifuses is very small when it is programmed at disadvantage.

So, large size of programming transistors, so this 11 to 12, 20 volt that I say it has to be applied. So, that volt has to be generated across the across those programming elements. So, so programming transistors become large and need of isolation transistor, so normally that 11 to 12, 20 volt. So, if it comes across some other circuit element that circuit element may get damaged, so they need to be applied specifically to these programming terminals.

So, that way we need some isolation transistors and it is 1 time programmable, so that I have already said that it is you can program it only once. So, FPGAs from Actel, Quick logic, Cross point they support these Antifuses. So, these are some of these standard FPGAs that we have that support this Antifuse based programming.

(Refer Slide Time: 26:56)



So, this is the antifuse structure so symbolically it is represented like this. So, whereas, the structurally so you have got this oxide layer then there is a poly there is a n^+ diffusion and this is another oxide layer and we have got this poly silicon layer.

So, this is the dielectric that we have, so this dielectric if we apply a potential then this dielectric will break it will establish a connection between this poly silicon and this n^+

diffusion. So, that is how the connection gets established and we get this Antifuse based structure.

(Refer Slide Time: 27:33)

Floating gate

- FPGA devices from Altera, Plus Logic, AMD, etc. use floating gate programming technology.
- While Altera and Plus Logic use ultraviolet erasable EPROM, AMD uses electrically
- erasable EEPROM.
- It contains a control gate and a floating gate. The transistor can be disabled by applying a high voltage between control gate and drain. This injects charge on the floating gate, increasing the threshold voltage of the transistor – disabling it.
- Charge can be removed by exposing floating gate to ultraviolet light or by erasing electrically. It provides reprogrammability and unlike SRAM, no external memory is needed to program the chip on power-up.
- However, EPROM technology requires additional processing steps, high ON resistance and high static power consumption due to pull-up resistor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Other programming technology that we have is the floating gate technology. So, FPGAs from Altera, Plus logic, AMD etcetera they use this floating gate programming technology.

So, this Altera and plus logic they use ultraviolet erasable EPROM so and AMD uses electrically erasable EPROM. So, you know that there is EEPROM and EPROM. So, this Altera and plus logic they use EPROM technology and this AMD uses EEPROM technology. So, as I said that it contains a control gate and a floating gate a transistor can be disabled by applying a high voltage between the control gate, and drain these injects charge on the floating gate increasing the threshold voltage of the transistor and it will get disabled.

So, threshold voltage of the transistor if it is increased means to apply to turn on the transistor, so you have to apply more voltage.

(Refer Slide Time: 28:31)

The diagram illustrates a floating gate memory cell. It features a vertical 'Word line' on the left and a horizontal 'Bit line' at the top. A transistor is connected between these two lines. The gate terminal of the transistor is labeled 'Floating gate'. A small spring symbol is shown above the Bit line, indicating the physical presence of the floating gate. The drain terminal of the transistor is connected to ground.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the diagram so if you apply some voltage here then this floating gate appears, so as a result the threshold voltage of this transistor goes up. So, transistor becomes normally off sort of thing, whereas, if this is not there then the threshold voltage is low so then this you have got this transistor on.

So, this floating gate that we can insert. The charge can be removed by exposing the floating gate to ultraviolet light or by erasing it electrically. It provides reprogram ability and unlike SRAM no external memory is needed for program to program the chip on power up. So, this is reprogrammable because, you can remove that floating gate by applying either ultraviolet light or this a high voltage, But unlike SRAM, so it does not need this off chip, this reprogrammable off chip memory for hold the control program.

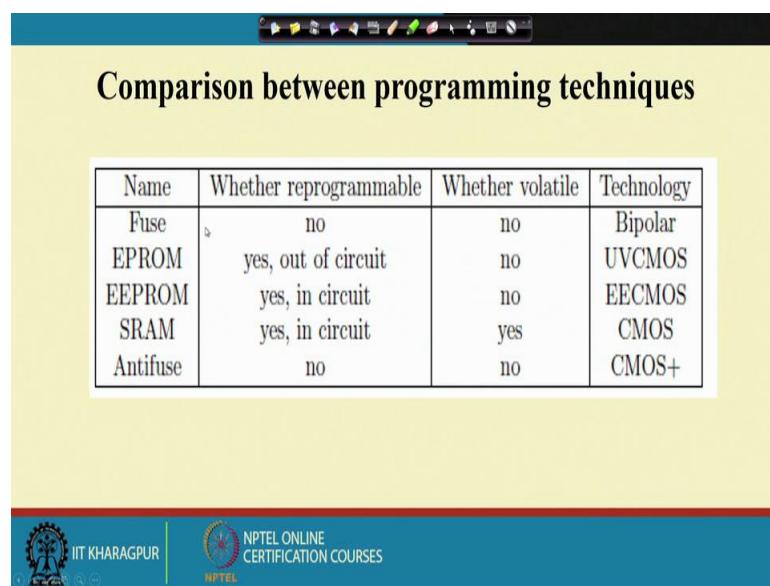
This EPROM technology it requires additional processing steps of course, the high on resistance and high static power consumption due to pull up resistor. So, this is the problem with EPROM technology, so that is there with the floating gate based this FPGAs. So, this is the floating gate programming I said so if you want to apply a particular if you apply a voltage then this gate will appear here and as a result this will be this threshold voltage will increase. Whereas, if you do not apply the voltage the threshold this floating gate will not be there, so threshold voltage will be low. So, the transistor will be treated as on so that way we can establish connection or not.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 45
FPGA (Contd.)

So if we compare between these programming techniques. So, this the programming technology, if it is a fuse based programming.

(Refer Slide Time: 00:22)



The slide has a title 'Comparison between programming techniques' and a table comparing five memory technologies: Fuse, EPROM, EEPROM, SRAM, and Antifuse. The table includes columns for Name, Whether reprogrammable, Whether volatile, and Technology.

Name	Whether reprogrammable	Whether volatile	Technology
Fuse	no	no	Bipolar
EPROM	yes, out of circuit	no	UVCMOS
EEPROM	yes, in circuit	no	EECMOS
SRAM	yes, in circuit	yes	CMOS
Antifuse	no	no	CMOS+

Then it is not reprogrammable because once the fuse is blown so, we cannot restore it. So, it is not volatile like even if you switch off the power this blown switch will not come back to its original value. So, that is not programmable, not volatile and a technology that uses is the bipolar technology.

Normally, it is a TTL type of technology Transistor Transistor Logic type of technology that is used in the fuse; then EPROM. So, EPROM it is also programmable definitely, but you have to take it out of the circuit, because for reprogramming. So, we have to take it out of the circuit and expose it to ultraviolet light and then only the content will be reset and then you can program it again.

So, that way it is EPROM is reprogrammable, but it is out of circuit, it is not volatile because, the content will never be lost. The technology that uses is UBCMOS ultraviolet

CMOS technology. Then we have got EEPROM technology for a programming technique. And this is programmable and it is in circuit programmable, because you can just program this we can apply a suitable voltage. And then the then the PROM content will be modified.

So, it is also not volatile and the technology is electrically erasable CMOS technology. Then this SRAM, so, this is also reprogrammable and it is in circuit reprogrammable. So, that is fine it is volatile so if you switch off the content will be lost. So, the FPGA chip has to be programmed again and the technology it uses is the CMOS and this anti fuse. So, anti-fuse it is not programmable it is not volatile and it uses technology it is CMOS+ that we have said that a silicon oxide nitride type of connections.

So, this is some advanced masking will be required over and above this normal CMOS fabrication. So, this is the summary of comparison between these programming techniques. Next, we will be looking into this FPGA logic blocks.

(Refer Slide Time: 02:29)

FPGA Logic Blocks

- There are wide variations in the logic block structure of FPGAs available from different vendors.
- They vary in number of inputs and outputs, amount of area consumed, complexity of logic functions that they can realize, total number of transistors needed, and so on.
- The logic blocks can broadly be classified into the following two categories – Fine Grain, Coarse Grain

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, logic blocks are the heart of this FPGA chip. So, they are the, they are actually the modules which will be implementing the logic function that we are going to have. So, there are wide variations in the logic block structure of FPGAs available from different vendors. And here actually there are you can find the different FPGA vendors, they are come up with different type of logic block structures.

And they vary widely they vary in the number of inputs and outputs amount of area consumed complexity of the logic functions that, they can realize, total number of transistors needed etcetera. So, that way there is the wide amount of variation.

The logic blocks can broadly be classified into 2 categories. One is called fine grain logic block, another is called coarse grain logic block. So, as the name suggests so, you can understand that when we have got very fine very simple logic block consisting of less number of logic gates in them. So, they will give rise to this fine grain logic block and when we have got this complex far logic block. So, having individual logic block having more number of functionality implemented within it so, they will be called coarse grained logic block.

Now, we will see that trade-off between them and which one is superior. So, it is difficult to answer so, each one have got their own advantage and disadvantage so, but we will. So, the choice is depends on many things like first of all the availability, then the familiarity of the designer right, if I am familiar with the Xilinx FPGA more, so, maybe in my future design I will be using Xilinx FPGAs only.

So, that way it is like that. So, choice is more determined by availability, familiarity and in some cases, the price also.

(Refer Slide Time: 04:25)

Fine Grain Logic Block

- The block contains a few transistors that can be interconnected via programming.
- Crosspoint* FPGA uses a single transistor pair for each Boolean variable in the logic block.

$f = \overline{ab} + c'$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, first we look into this fine grain logic blocks so, this is a logic block from cross point it. So, this logic block contains a few transistors that can be interconnected via programming.

So, for example so this, cross point FPGA so, it uses a single transistor pair for each Boolean variable in the logic block. So, this is the thing like so, you have got a P type transistor and an N type transistor. So, they that is used for 1 Boolean variable, you have got a chain of such transistors and you can establish these connections. So, you can you can do metallization, so that this programming so, that these gate lines so, they can be connected they can be connected to some inputs.

Similarly, these drain and source lines they can be connected to some they can be connected in some fashion. So, that can happen so, here is a small example suppose, we want to realize the function $f = ab + \bar{c}$. So, you see so, from this function you can understand. So, if c is equal to 0 then the functional the function must output a1. So, if you look into this design if you see if c if I put c equal to 0 then what will happen?

So, this transistor will be off so, this transistor will be off because this is N type transistor and giving 0 to the gate. So, this will be off and this upper transistor will be on because this is a P type transistor and I am giving a 0 to the gate. So, this transistor will be on so, this up arrow means. So, it is connected to the supply voltage V_{CC} equal to 5 volts. So, we normally call it V_{DD} in case of CMOS.

We in the MOS circuit, so you called it is drain voltage V_{DD} . So, that is maybe equal to 5 volt so, what will happen? So, this is this point is at 5 volt and this transistor is on. So, you will get f equal to 5 volt. So, that is equal to logic one. So, you see if I put c equal to 0, I am getting f equal to 1. Now, this ab part. So, ab part is realized by the first this section.

So, this realizes the product term ab how? Suppose, a equal to 1? Suppose a equal to 1 and b equal to 1. So, both of them are equal to 1 then what will happen? Since a equal to 1 so, this transistor is on this is off because this is N type transistor and if I give this gate voltage equal to 1. So, the transistor will be on. Similarly, this b being equal to on, this is b being equal to 1. So, this transistor is on this transistor is off fine.

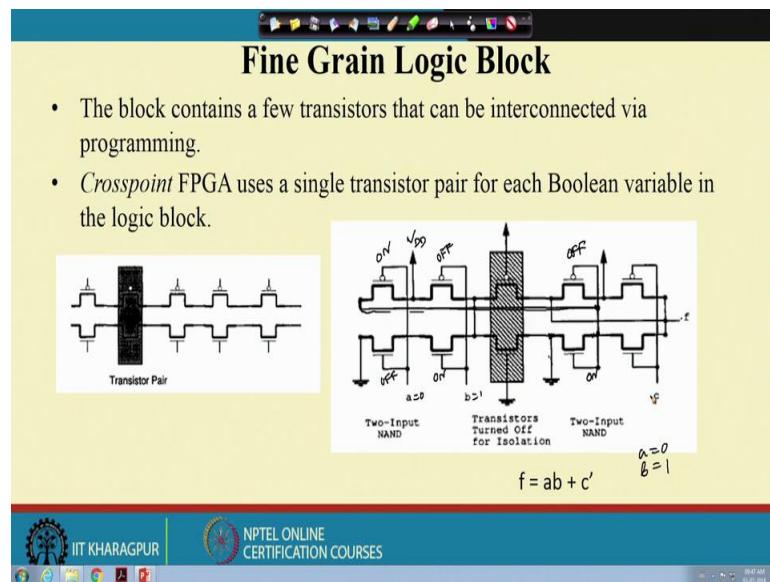
So, what will happen? So, this point is grounded. So, you see that this ground potential, ground voltage is available here and this ground through this line. So, it will be coming so

this line is made equal to 0 and you get a 0 voltage here. So, in this 0 comes so as a result so, this transistor this is a p transistor with gate value 0.

So, this is turned on and this is an n transistor with gate value 0. So, this is turned off. So, you see that this transistor being on. So, this voltage this V_{DD} that we have is available at this point. Because, this transistor is also this drain and source will be at the same potential.

Since, the drain is at 5 volt, so, this source will also become at 5 volt. So, as a result this line will be become becoming equal to 1 again. So, on the other hand if I have this say a equal to say 0.

(Refer Slide Time: 08:19)



So, a equal to 0 and b equal to say 1. So, a is 0 b is 1 so as a result, this transistor is on this is off this is off and then I have got this transistor as off, this transistor is getting on so, this is off and this transistor lower transistor N transistor so this is on.

Now since, this transistor is on so you get this one value coming here. So, these so this is also connected to this is actually whenever, we have got this up arrow means it is connected to V_{DD} so it is connected to V_{DD} .

So, this is these 2 points potential are same so I get a 5 volt here, and that 5 volt get transferred to this point. So, I when I get 5 volt here so this transistor will be on and this transistor will be off as a result this ground value 0 will be coming to this point and that will be so the ground value will be 0 and this 0 will be communicated to this point, this f .

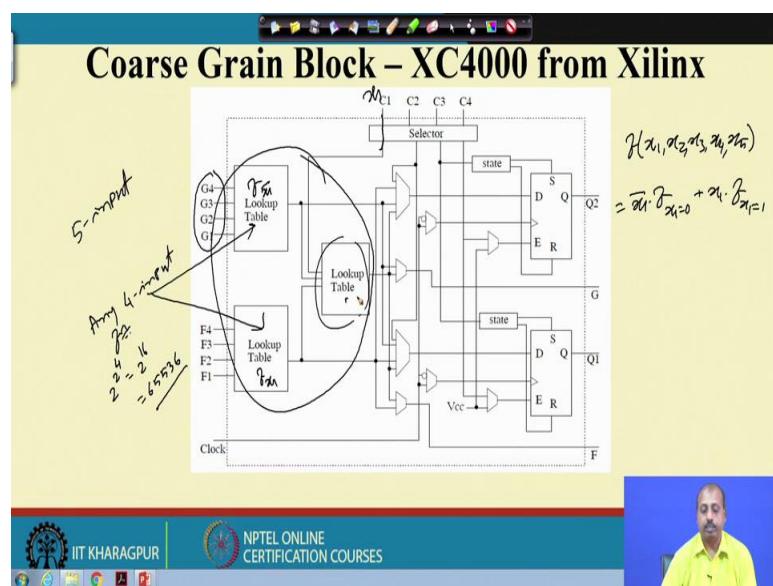
So, in this fashion so we can have different input combination as a result, this individual stages of this transistor. So, they will be propagating the signal or they will not be propagating the signal. In between, we have got this isolation stage.

So, this isolation stage so this will be this is for separating the 2 product transfer. So, this $ab + \bar{c}$ they are separated by this isolation states. So, this is this you see that this is an N type transistor and I have connected it is gate to ground, the P type transistor I have connected is gate to high. So, as a result both the transistors are off.

So, this actually establish an isolation between this ab term and the c term and. So, the logic is transferred from one side to another side by means of these interconnects that we have between these transistors arrays. But, otherwise the transistors are not connected to each other by means of the isolation, isolation stage so, that is ensured.

So, this way we can have this cross point FPGA. So, they realize this logic functions and you see the logic block is very simple. So, you have got only 2 transistors per Boolean variable ok. So, next we will be looking into a more complex logic block structure which is from Xilinx, which is Xilinx has got a family of FGPAs.

(Refer Slide Time: 10:54)



So, this is one of the initial families I should say. So, today if you look into the advanced families of Xilinx. So, they are very complex we will see some of them. So, this is the

XC4000 series from Xilinx, Xilinx FPGA. So, what it has this is a coarse grain block because, you will see that the logic function that that you can get is quite complex. So, compared to only 2 transistors, that we had in cross point FPGA so here, we have got this type of lookup table. So, lookup table means so this has got 4 inputs G_1 G_2 G_3 G_4 .

So, it can realize any of any 4 variable function. So, this lookup table can realize any 4 input functions, any 4 input function. Similarly, this lookup table also can realize any 4 input function. So, this so I can so if I if I look took take them separately then I can realize two 4 input functions directly from the lookup table.

So, number of 4 input Boolean functions that you can have is 2^{2^4} that is 2^{16} . So, it is 65536.

So, this is the total number of 4 variable functions that we have that is the total number of 4 variable functions that are possible. So, you see this is a huge number of functions that you can realize using this for input look up tables plus there is another lookup table after this.

So, there the output of these 2 look up tables are fed and also this from this C_1 C_2 C_3 C_4 through this selector, you can select one of these inputs to come to this lookup table. So, if I take these 3 look up tables together if I take these 3 lookup tables together then I can realize any 5 input Boolean function.

Any 5 input Boolean function because if I have. So, if I have got a 5 input Boolean function $f(x_1, x_2, x_3, x_4, x_5)$. So, we know that by Shannon decomposition. So, I can write it as $\bar{x}_1 f_{x_1=0} + x_1 f_{x_1=1}$. So, this x_1 line can be connected to this C_1 . Now, this can realise so, this lookup table can be used to realize $f_{x_1=0}$ and this lookup table can be used to realize $f_{x_1=1}$ and this selector can be programmed such that it comes here ultimately.

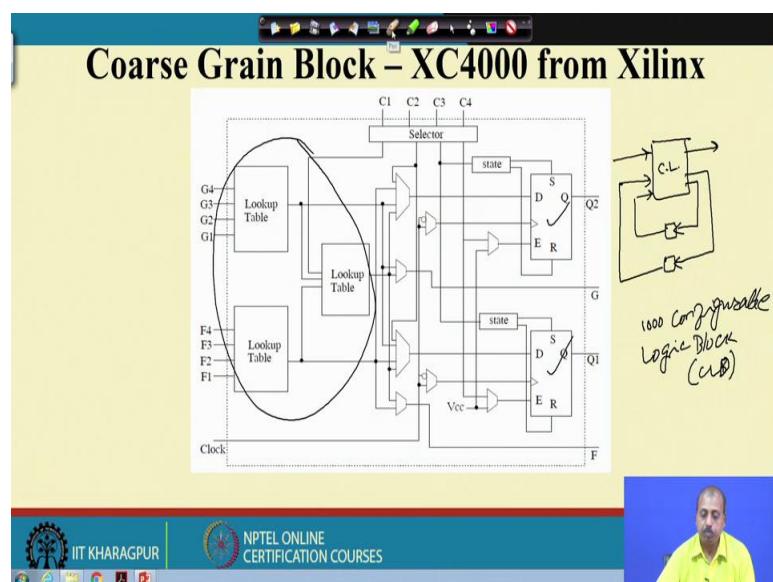
So, this lookup table realizes the 3 variable function consisting of the variables x_1 f x_1 equal to 0 and f x_1 equal to 1. So, these 3 variable these 3 functions can be 3 these 3 variable function can be realized by this lookup table. As a result, combining these 3 lookup tables you can get any 5 input function realized. So, that is very that is very powerful.

So, after the lookup table after the basic combinational logic function that we can have. So, we see that there are other elements for example, there is a multiplexer here and this

from this multiplexer you can either pass one of these lines through this to or you can pass this output of this G or you can pass the output of this f lookup table or you can pass this output of this lookup table. So, this is a 4 input multiplexer so, you can pass any of these 4 inputs to this to this D flip flop S.

So, this is similarly to this D flip flop. Similarly, for the lower side also we have got another d flip flop and this you can. So, you can pass this flip flop values, you can pass this combinational logic into these flip flops and. So, as a result the value will be stored. So, as you know that whenever we are trying to realize this finite state machine type of structures, so, what we have is we have got some combinational logic and that combinational logic feed some flip flops.

(Refer Slide Time: 15:02)



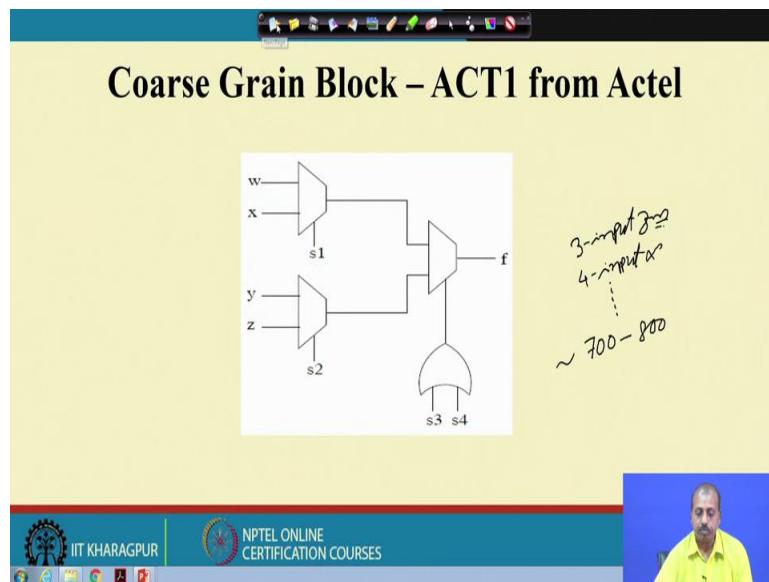
So, they feed some flip flop for the, this next state values. So, this is the structure of a finite state machine. Now, you see if you look into this structure. So, it is very much likely that this combinational logic, I will be able to realize using these lookup tables and I have got 2 flip flops. So, this 2 flip flops. So, these 2 flip flops can be used to do that.

So, you see a sizable amount of finite state machine can be realized by a single FPGA logic block of Xilinx and of course, there are large number of such logic blocks available. So, if you in your design if there are thousand in this family XC 4000s family, so, if they are there are thousands such configurable logic blocks. So, they are called configurable

logic block or CLB. Then these thousand such CLBs, so, they will ensure that you have got so there are large number of flip flops in the design in the chip.

So, normally we can use them for realizing any complex finite state machine with large number of states and all. So, that can be done very easily. So, this makes this Xilinx family of FPGAs quite popular because with this coarse grain logic block. So, you can put good amount of logic into individual blocks and of course, we can connect them with one another.

(Refer Slide Time: 16:42)



Next we will be looking into another coarse gain logic block from Actel which is known as ACT 1 logic block.

So, compared to Xilinx FPGAs so, they are not that much that much logic rich I can say, but it is ah, but it is quite simple. So, we have got 3 multiplexers connected in this fashion. So, this 3 multiplexers are connected and there is one OR gate. So, these 3 multiplexers and this OR gate so they can realize some functions.

So, this multiplexers can be programmed like say. So, overall logic block has got this **3 plus 3 plus 2** total 8 inputs. So, this overall logic block is of 8 input and 1 output.

So, it of course, it cannot realize all 8 input functions, but if you analyze this particular block. So, it can be shown that it can realize up to all 3 input functions. So, up to all 3 input functions can be realized by this logic block. So, it can be proved similar for 4 input so it

cannot realize all. So, it is not all, but many of the 4 input functions can be realized so this way if you enumerate.

Then, there are around 700 unique 700 to 800 unique functions that can be realized by this ACT 1 logic block.

So, that is also quite large number. So, because of the advantage that this individual logic blocks are pretty small, you can have more number of logic blocks within the FPGA chip, but compared to this Xilinx FPGAs. So, they are much simpler.

(Refer Slide Time: 18:33)

The slide has a yellow background with a black header bar containing icons. The title 'Trade-offs' is centered in a bold, black font. Below the title is a bulleted list of nine points. At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt.

- A large logic block can implement more logic within a single block, requires lesser number of logic blocks to realize a given functionality on the FPGA.
- On the other hand, a large logic block consumes more space of FPGA.
- 4-input look-up table gives best result in terms of logic synthesized and area consumed.
- A higher granularity level results in lesser delay between system input and output.
- With the increase of granularity level, average fanout increases, number of switches also increases as each block has more pins.
- Also, the length of wires increases with increase in size of logic block.

So, next we will see how this what are the trade-offs between this coarse grain logic block and the fine grain logic block whenever we are looking for this coarse grain logic block a larger logic block and implement more logic within a single block and requires lesser number of logic blocks to realize a given functionality on FPGA.

So, individual logic blocks are having more number of functionality that can be implemented there. So, I will require less number of such logic blocks. So, the number of interconnects that will be needed will be less. So, it will require that delay of the design will be low because the interconnects, they contribute significantly towards that delay and with the increase in the length of the interconnect the delay increases by the square of that. So, as a result so if the length of the interconnects are small then I will the delay of the design will be less.

So, a large logic block can implement more logic function within a single block. So, it requires lesser number of logic blocks to realize a given functionality on FPGA. The other hand, a large logic block consumes more space on FPGA as I said that say for example, these Xilinx XC4000 logic block that we have seen. So, this is a quite powerful no doubt it has got large number of, large number of flip flops within it. So, we have got say we have got say 10 one thousand logic blocks and each are having 2 flip flops in them. So, 2000 flip flops.

So, are we going to use really these 2000 flip flops in our design? So, if you are not using that then of course, there is a wastage and there is no way to tell that I will not use these flip flops. So, they will remain whether you use it or not. So, they remain in the FPGA chip so, so that is there so, there that that is a wastage of space. So, a large logic block consumes more space on FPGA. Now, there are there is a study that shows that among these lookup tables that we have.

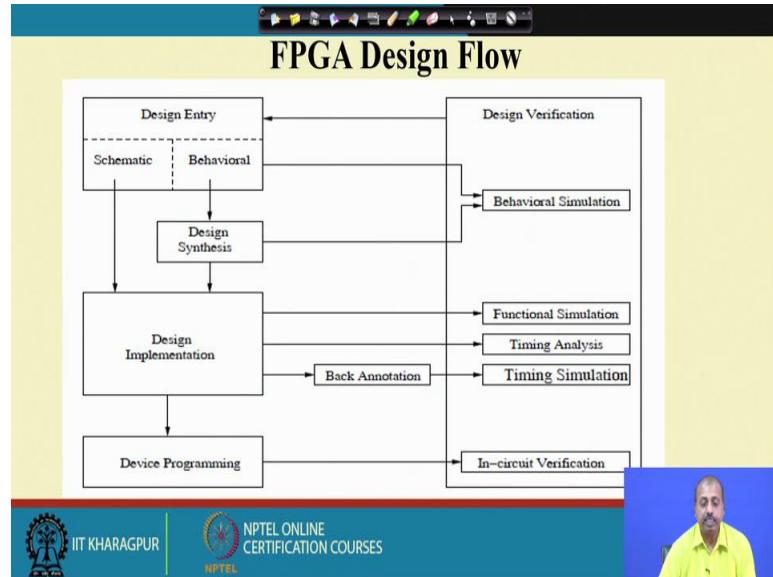
So, 4 input lookup table they gives best result in terms of logic synthesized and area consumed. So, if you make a trade off plot so, some early researchers they have shown that that this 4 input is the ideal value 3 input if you make. So, that is also not good 5 input also we cannot they are not very much useful, but these 4 inputs are better.

A higher granularity level results in lesser delay between system input and output this point I have already said that we have got this interconnect length will be small. So, as a result that delay will be less with the increase of the granularity level average fan out increases and number of switches also increases as each block has more pins. So, one per one output may be one output of a logic block may be going to 10 different places.

So, as a result the fan out will be high. So, that way again that is another point so, if fan out is high so, we have to put some buffer in between. So, as a result the overall logic requirement will also go up and length of the wires increases with increase in the size of the logic block so, the logic block is, so, if you look into this FPGA chip as we have shown you that if these are the logic blocks and say this is this is another row of logic block. So, we have got interconnects running between them. So, if these logic blocks sizes are large then naturally this interconnects length is also going to be large. So, that is why we have got that is another problem.

So, if you have got larger sized logic block then the length of the wires will also increase. So, that way it may be better that we go for smaller size logic block or finer logic blocks.

(Refer Slide Time: 22:18)



So, the overall FPGA design flow is like this. So, no FPGA based designs are done manually. So, the first of all, it goes through a design entry phase. The Design Entry phase it can be either by a Schematic Entry or it may be a Behavioural Entry. So, Schematic Entry means. So, you draw your circuit in terms of AND OR NAND NOR gates and say flip flops and all.

So, you do draw your design in terms of that. Normally FPGA vendors they provide you some logic libraries. So, we take the elements from that logic library and draw our schematic in terms of that then. So, other possibility is that you describe your design in some language like VHDL or Verilog type of language and then from there you go through a synthesis stage.

So, that will convert your description into circuit. So, whatever you do whether you directly enter the schematic of your design. So, normally for smaller designs and for more if we are looking for optimized design so, we go for this schematic based our strategy.

So, that the overall control is in the hand of the designer or we can have this behavioural entry. So, behavioural entry is normally there when we have got the entries from more

complex systems. So, you want to it is difficult to draw the circuit directly. So, go for this behavioural design so whatever and design synthesis.

So, these are the cad tools so that will convert this description on to this implementation. So, after we have got this implementation so, implementation is by means of this logic elements that the library supports and the interconnections that we do. So, we do some timing verification and all so we will come to that and then it goes to device programming. So, once this design has been implemented so this device programming so, it generates the control program.

For the FPGA, and that control program is used for say if it is an a SRAM based FPGA. So, it will be used it will be downloaded onto the FPGA chip for functioning. If it is anti-fuse or this one time programmable FPGAs then these device programs. So, they will be used to burn the appropriate fuses or anti fuses and that way this connection the FPGA will become functional. So, this is the design process.

So, design whenever we are doing the design digital design the verification also comes into question. So, if you are having this behavioural specification or so now, we can do a simulation of it. So, we can we can put some sort of we can put some sort of tested vectors and see like whether it is giving me proper result or not, that is via behavioural simulation.

Similarly, for the Design Synthesis also so, we can go via some simulation and do that. Once the design has been implemented, so, you can go for functional simulation, so, to see whether functionally it is giving correct result or not. You can do a timing analysis so now may be in your design entry.

So, we have put output of 1 block to connected to the input of another block but, while you have done the implementation so, these 2 blocks are placed far away on the FPGA chip and they are going through a number of interconnect stages for doing the for getting the connection between the 2 points.

So, as a result the delay of that wire will be more so, that will be captured in the timing analysis phase. So, the timing analysis it will find out whether there is any timing violations etcetera and then it will be giving all the wire delays and those wire delays will be back penetrated into the original design. So, that is the stage of back annotation. So, the delay values are put onto the design and then we do another timing simulation.

So, after timing simulation, so, if we are satisfied with the timing simulation then only this device programming is done and once the program control program has been loaded on to the FPGA device, we can go for in circuit verification. So this in circuit verification, so, they will be it will be it can verify if there is any problem in the actual running of the system or not.

Normally, if your design is through this timing simulation so, we can be sure that the design will work in the final implementation as well. You look into the modern FPGAs, so, in addition to the basic blocks such as logic blocks IO blocks and interconnects.

(Refer Slide Time: 26:50)

The slide has a yellow header bar with a toolbar icon. The main title is 'Modern FPGAs' in bold black font. Below the title is a bulleted list of six points about modern FPGAs. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt.

- In addition to the basic blocks (such as, logic blocks, I/O blocks and interconnects), modern FPGAs have additional units that make the design process simpler and more efficient.
- The two major system components, difficult to implement in FPGAs are embedded memories and blocks for arithmetic calculations.
- Amongst the various calculations, multiplication is the most widely used one. Most of the modern FPGAs contain embedded logic blocks for multiplication and memories to hold data. DSP functionalities are highly facilitated by the availability of these.
- In many applications, FPGAs need to communicate with microprocessors. This has motivated many FPGA vendors to embed soft processor cores within FPGAs. This reduces the latency of communication between the microprocessor and the FPGA.

Modern FPGAs have additional units that help in the design process to make it simpler. The 2 major components that we have that we do not have in FPGA the what FPGA this logic blocks and all one is embedded memory. Another is block for arithmetic calculation. So, the various calculations multiplication is the most widely used one because FPGAs may be used for some signal processing type of applications, so multiplication will be very important.

And most of the modern FPGAs they contain embedded logic blocks for multiplication and memories to hold the data. So, DSP functional digital signal processing functionalities are highly facilitated by the availability of this multiplication and multiplier and memory. In many applications FPGAs need to communicate with microprocessors. So, the FPGA

chip that needs to communicate with the some outside chip and many FPGA vendors they have embedded some soft processor codes within FPGAs.

So, this processor codes are there. So, we can you can very easily use the communication mechanism supported by those soft course of the processors that, we have that is embedded into the FPGAs. For example, these Xilinx they have got power pc embedded on to this FPGA chip. So, that way we can have this microprocessor it can establish a communication between the microprocessor and the FPGA.

(Refer Slide Time: 28:20)

Xilinx Virtex-6 and Virtex-7 FPGA

- Each CLB of a *Virtex-6 FPGA* can be configured as one 6-input LUT or two 5-input LUTs.
- The LUT can also be used as a 64-bit RAM or two 32-bit RAMs.
- Apart from this, every *Virtex-6 FPGA* has 156-1064 (depending upon the subfamily) dual port block RAMs, each storing 36 Kbits.
- They also possess many dedicated, full-custom, low-power DSP slices. Each slice contains 25, 18-bit, 2's complement multiplier and a 48-bit accumulator.
- Each *Virtex-6* device has a 17-channel, 10-bit ADC and 8-72 Gbps transceiver.
- The next advanced version, *Virtex-7* is a 3D IC with many improved features.
- The peak transceiver speed varies between 12.5-28.05 Gbps with 36-96 transceivers.
- It can perform 2756-5314 giga multiply accumulates (GMACS) and contains 46.5-85 Mb block RAM, PCI express bus interface, and upto 1200 I/O pins.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, Xilinx, Virtex 6 and Virtex 7 series of FPGAs so, just to have a brief idea the CLB of Virtex 6 can be configured as 6 input LUTs or two 5 input LUTs. So, these are the advancement from 4 input it comes to 5 and 6 input. So, LUT can be used for 64-bit ram or two 32 bit rams so, that is. So, that way it gives to gives you to store. So, some RAM space there.

So, Virtex 6 FPGA has 156 to 1024 dual port block RAMs each storing 36 kilobits. So, the huge amount of space is provided, dedicated full custom low power DSP slices are there. So, that can do digital signal processing calculations. It has got 17 channel 10-bit ADC, 8 to 72 Gbps transceiver. So, 10-bit ADC 17 different analog signals can be fed into this 10-bit ADC for doing the analog to digital conversion and this transceiver is there for communication. Virtex 7 is 3 D IC. So, 3 dimensional integrated circuit so, again it has got much higher capabilities.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 46
VHDL

So, in the next part of our course, we will be looking on a different topic which is known as VHDL which is a hardware description language. So, the way that we have learnt in this particular course, we have seen that if we want to design a circuit, we have to start with logic gates and draw the circuit and then to get a confidence on whether the circuit is working properly or not, we do some analysis of the circuit and try to see whether, try to get the confidence that my design is correct.

But if the circuit is very large ok so or if the design is very complex, then it is very difficult to do that type of approach like if you are trying to do a complex design. So, there can be 2 ways in which you can do it, one is known as the top down approach another is known as the bottom up approach.

So, whatever we have learnt so far. So, we have learnt about the bottom up approach. So, we can start with the simplest modules that we have in the design and then we can after designing those and understanding that they are correct. So, we can go on connecting them in some fashion to connect to see that if the complex functions are realized.

But, another approach that can be there is top down approach where we start with the top most level of the design and then go on breaking them up and coming with the further and simpler and simpler levels of implementation.

Now, this other this top down and bottom up approaches, so, both of them; so, they becomes complex when the overall design is quite complex and in that way we need a better way to represent the represent our design or our circuit. So, this VHDL is a hardware description language that will help us in doing that and there are 2 such hardware description languages which are quite popular one is this VHDL that we are going to discuss and the another is Verilog.

So, both of them are equally powerful; so, there is no reason to believe that one is better than the others, it is just for an example that we have taken VHDL. So, you can very well do a, learn, learn Verilog language.

So, will be looking into a introduction of this VHDL language. So, this how does it work? What are the essential features of it? So, will we will try to go look into it by means of an example.

(Refer Slide Time: 02:45)

The slide has a yellow background with a blue header bar at the top containing various icons. The title 'Introduction' is centered at the top. Below the title is a bulleted list:

- Hardware description languages (HDL)
 - Language to describe hardware
 - Two popular languages
 - VHDL: Very High Speed Integrated Circuits Hardware Description Language
 - Developed by DOD from 1983
 - IEEE Standard 1076-1987/1993/200x
 - Based on the ADA language
 - Verilog
 - IEEE Standard 1364-1995/2001/2005
 - Based on the C language

On the right side of the slide, there are two small hand-drawn sketches. The top sketch shows a complex logic gate with multiple inputs and outputs, with a curved arrow pointing to one of the output lines. The bottom sketch shows a simpler logic gate with three inputs and two outputs, also with a curved arrow pointing to one of the output lines.

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo on the left and the text 'NPTEL ONLINE CERTIFICATION COURSES' in the center. On the far right of the footer bar, there are some small icons and the text '2014-15'.

So, these hardware description languages, they come under the broad heading of HDLs language to describe hardware and so this is the, so, they why not the standard languages that we have in software like C, C++, Java, Pascal, Fortran, etcetera. So, those languages, why they are not suitable for this hardware the basic difference between the hardware and the software is that the hardware is always parallel in nature whereas, software is always sequential in nature.

What I mean is like this that if I write a program in say C language and these are the several the statements of the C language, then these statements are executed one after the other first this one is executed then this one. So, it goes like this until and unless there is a branch. So, it goes in that sequence and even if there is a say branch. So, it goes to the target and then again from the target it goes sequentially.

Whereas in a, in the hardware so if I have got one functional modules, here another functional modules there another functional module here like this. Now all these functional modules so they are always active. So, whether their outputs are meaningful to the user or not that is a different question like if I have got one AND gate here and say 1 OR gate, here they are getting inputs. Now may be that the, their outputs will be say NANDed further to get the final output.

So, whatever be the inputs here this A, B, C, D values. So, these gates are always workings. So, these all the 3 gates, they are always working, but we are interested about the output of this NAND gate only when this one of these values of A, B, C, D have changed and we are looking into this part.

So, that way so, that is the users convenience. So, user finds it ok, I am need to know the output at this point so that is the thing otherwise if you look from this gate's angle, then they are always being the computation. So, there is nothing that it is, there is nothing like sequential, it is not that this AND gate evaluates the output first, then this NAND gate it's not like that. So, all of them are working together, it is only the point of sampling at which the user is interested about the output value and that makes a difference.

So, that is why this software languages, they are not suitable for hardware because the these software languages, they will not be able to capture the parallelism that you have in hardware.

And similarly hardware languages, they are not suitable for software because hardware will assume that all the, whatever we have described in that language everything is in parallel. So, it will not be able to capture the inherent sequentialism that you need in case of software.

So, coming back to the point; so, we have got this hardware description language. So, there are 2 languages; one is one is known as VHDL which is which you this V stands for this word V is stands for VHSIC ok.

(Refer Slide Time: 05:48)

The slide has a yellow background. At the top center, the word 'Introduction' is written in a large, black, sans-serif font. Below it, there is a bulleted list of hardware description languages:

- Hardware description languages (HDL)
 - Language to describe hardware
 - Two popular languages
 - VHDL: Very High Speed Integrated Circuits Hardware Description Language
 - Developed by DOD from 1983
 - IEEE Standard 1076-1987/1993/200x
 - Based on the ADA language
 - Verilog
 - IEEE Standard 1364-1995/2001/2005
 - Based on the C language

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there is a logo for IIT Kharagpur and the text 'IIT KHARAGPUR'. In the center, there is a logo for NPTEL and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the bar, there is a small video window showing a man in a yellow shirt speaking.

So, VHSIC so, which is very high speed; so, this is VHSIC; very high speed integrated circuits, then hardware description language.

So, this VHSIC Hardware Description Language or VHDL; so, it was developed by department of defense from 1983, IEEE standards have come up ok. So, it is based on the ADA language. So, the basic language on which this VHDL has been designed that is ADA, on the other hand this Verilog is another standard language. So, that is again an IEEE standard has been defined and it is based on C language.

So, you can learn either of them. So, as I said that for this course will be we have taken up VHDL.

(Refer Slide Time: 06:42)

- Model and document digital systems
 - Different levels of abstraction
 - Behavioral, structural, etc.
- Verify design
- Synthesize circuits
 - Convert from higher abstraction levels to lower abstraction levels

So, why do we need this HDL? So, model and document digital systems, actually, the point at which it started is like this. So, there are large amount of designs that were there large number of circuits that were available, but the original designers are missing. So, original designers are not there.

Now, how to understand? So, like if I draw a complex circuit and I am not available to explain it. So, it is very difficult to understand the how what is the motivation how is this thing designed and how to preserve this design, how to, how to keep a note of the design content.

So, that is the modeling and documentation of the digital system. So, it originally started with the documentation of the digital system, we have got a circuit. So, you want to make a document to describe the circuit. So, naturally you need to say that these are the gates in my circuit I am connecting it like this. So, this is this has to be done in some formal way.

So, for documenting this digital system; so, we have to have this HDL was necessary, sometimes, we need to model the behavior of a circuit. So, for that also this HDL is necessary and there are different level of abstraction. So, we have got this behavioral level, structural level, etcetera. So, you can describe a circuit in terms of its behavior for example, if there is an adder so you can simply say $A = B + C$ that describes the behavior of the adder or you can say that my circuit contains so many AND gate, OR gate, XOR gate, etcetera

and they are connected in this fashion. So, that also defines an adder only, but in a different way that is the structural way of defining it.

So, the same circuit we can have a behavioral description, we can have a structural description. Second objective is to verify the design. So, once you have got a description of the design so you can try to verify the design by means of some sort of simulation or maybe by means of some formal verification technique you can try to extract the meaning of the description.

So, when you do that so you can get the meaning of it. So, that was another objective. So, this documentation was the first objective, modeling was the second objective, the third one is the verification of the design and of course, verification may be of 2 types. So, when I say I want to verify, then there can be two approaches one approach is via simulation, one approach is via modeling. So, simulation means we have got a description of a system, once I have described my circuit in terms of some language and we also so this is the description and we have got some expected behavior of the circuit and for that we design something called a stimulus.

So, this is these are define some certain stimulus and we have got some expected behavior for the, for each of this stimulus, we have got some expected behavior. Now what you can do you can try to apply the first stimulus onto the circuit and see whether it gives me the desired response and then you apply the second stimulus and see whether it gives the desired response only. For a combinational design so, you can apply this stimulus in any order. For a sequential design so, you have to apply them in the, in a proper order only. So, whatever it is. So, if I apply all these stimulus then I am expecting that the response from the circuit will be like this only.

So, if that thing happens, then I get the confidence that possibly my design is ok. Now I say possibly because it is not it is not possible to enumerate all different situation that that the circuit inputs can take because that way the number of inputs that will be needed to be fed will be very high, for example, if there is a 10 input circuit, then if you are to feed a combinational circuit only then if you are trying to apply all possible pattern. So, it will be 2^{10} , 1024. So, that is manageable, but as the number of inputs becomes higher than say 30 or 40. So, it becomes practically infeasible to apply all those patterns to the circuit and see the output.

And that the situation is more complex when you have got sequential circuits because then you have got even larger number of states that will come into picture. So, one type of verification is via this simulation. So, you have apply some stimulus to the description and try to see whether that stimulus is sufficient for, response is that you are getting from the circuit is as per your desired one. The other way of verification is by means of some formal method.

(Refer Slide Time: 11:41)

The slide has a yellow background. At the top, the title 'Applications of HDL' is written in a large, bold, black font. Below the title, there is a bulleted list of four applications:

- Model and document digital systems
 - Different levels of abstraction
 - Behavioral, structural, etc.
- Verify design
- Synthesize circuits
 - Convert from higher abstraction levels to lower abstraction levels

On the right side of the slide, there is a hand-drawn diagram. It shows a document icon pointing to a brain icon, which then points to a logic formula icon. Above this diagram, the words 'Formal Verification' are written in a cursive font. At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video camera icon showing a person's face.

So, this verification can be done by means of some formal methods which come under the broad heading of formal verification. So, there you try to describe the behavior. So, if this is the description of the behavior you try to describe this behavior in terms of some logic formula, you try to describe the behavior in some logic formula.

And you also have the circuit in your hand. So, if this is the circuit that you have. So, from the circuit also you can try to extract the logic formula and then try to establish an equivalency between these two logic formula ok, if we, if these two logic formula, they are equivalent to each other; that means, this circuit realizes this description properly, if there is a gap, then it is not so.

So, we can do this verification by means of this formal method also, but whatever we do so we need a description of the system first. So, that is another application area of these hardware description languages.

The third one that has become more popular now, but it was initially not that way is to synthesize the circuit. So, we start with a high level description of the circuit and then from there, we convert it into lower level of abstraction. So, high level of abstraction may be a, some language description of the circuit, that language description may be converted to say module level description like consisting of say adder, subtracter, registers multipliers ok. So, like that. So, that can be converted into a next lower level and that from that adder subtracter level description. So, we can again convert into gate level description consisting of gates and flip flops. So, that gives the next level of abstraction.

From there it can be further defined to transistor level description so that way, we can go to lower and lower abstraction levels and as we go there. So, we can and this process can be made automated. So, if we can automate that process; that means, we have got a methodology to design a circuit from its specification. So, that, so initially it was not the major goal of these languages, but now this has become one of the major goals. So, the simulation and this synthesis; so, these are the two major goals that we have from this HDL that we have; for this HDL is being used.

(Refer Slide Time: 14:15)

Input-Output specification of circuit

- Example: my_ckt
- Inputs: A, B, C
- Outputs: X, Y
- VHDL description:

```

entity my_ckt is
port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit);
end my_ckt ;

```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, next will see like; how can I specify a circuit. So, suppose this is a circuit. So, name of the circuit is my circuit. So, it has got 3 inputs, A, B and S and 2 outputs X and Y. So, this example my circuit it has got inputs A, B and C and it has got outputs X and Y. So, when

you are giving it a VHDL description. So, we write it like this so this entire circuit so this will be called an entity.

So, in VHDL every module that you have will be called an entity. An entity will have some ports by which it can interact with the outside world and accordingly, we can, it will be the behavior of this entity has to be described in either structurally or in a behavioral description, but the, this is the interface part. So, when you are defining the entity from the outside how does it look like. So, one important part here is the port. So, it has got a few ports this A, B, S, X and Y. So, these are the ports and out of that A, B and S; they are input port and they are of type bit and this X and Y, they are output port and they are also of type bit.

So, you see a number of keywords in this description. So, for entity is a keyword port is a keyword, then this in out. So, these are keywords and this bit they are actually the data type or the signal type. So, A, B, C can say they are called A, B, S, X. So, as they are signals. So, they are either input, output or it can be bidirectional also. So, you can have this type as in out or mode as in out and their type is bit. So, all of them are of type bit. So, you have got this. So, in this way you can define entities.

(Refer Slide Time: 16:09)

The screenshot shows a software window titled "VHDL entity". On the left, there is a code editor containing the following VHDL code:

```
entity my_ckt is
  port (
    A: in  bit;
    B: in  bit;
    S: in  bit;
    X: out bit;
    Y: out bit
  );
end my_ckt;
```

On the right, there is a block diagram of a circuit. It consists of a yellow rectangular box labeled "my_ckt". Three wires enter the left side of the box: "A" (top), "B" (middle), and "S" (bottom). Three wires exit the right side of the box: "X" (top), "Y" (middle), and another wire (bottom) which is not explicitly labeled but corresponds to the "S" input.

The bottom of the screen features a footer with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES".

So, this is the VHDL entity; my circuit is port A, B, S and these are input port and type B, X and Y, they are output port of type bit ok.

(Refer Slide Time: 16:24)

The slide is titled "VHDL entity". It contains VHDL code for an entity named "my_ckt". The code includes port declarations for inputs A, B, S and outputs X, Y, along with a generic declaration "generic S;". A callout bubble provides information about the entity name:

- Name of the circuit
- User-defined
- Filename same as circuit name recommended
- Example:
 - Circuit name: my_ckt
 - Filename: my_ckt.vhd

The code ends with "end my_ckt;". The entire code block is circled in green.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window of a speaker.

So, the things to be noted my circuit, this is the name of the circuit. So, every circuit should have entity should have a name. So, in a one description, I can have only 1 name for entity so and the names are unique. So, if there are multiple entities so they should have different names, but these names are unique. So, this is user defined. So, this names user can give. So, file name same as circuit name. So, this is A; this only recommended. So, this description in whatever file name your file, you are writing. So, normally we write them in files having extension .v or .vhdl like that.

Many tools; they will acts a different types of extensions. So, for this case it may be file name may be my_ckt.vhd circuit name is my circuit and file name is my_ckt.vhd, but this is just a just a suggestion or recommendation, but you may or may not follow it ok. So, that may or may or may not follow. Now next will be this.

(Refer Slide Time: 17:33)

VHDL entity

```
entity my_ckt is
  port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit
  );
end my_ckt;
```

Port names or Signal names

my_ckt

A → my_ckt → X
B → my_ckt → Y
S → my_ckt

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

These are the port names A, B, S, X, Y. So, they are port names or signal names. So, they are the interfacing signals that we have again the names here can be arbitrary. So, they can be chosen by the user, but again the same thing that the same name cannot repeat ok. So, that restriction will be there.

(Refer Slide Time: 17:54)

VHDL entity

```
entity my_ckt is
  port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit
  );
end my_ckt;
```

Direction of port
3 main types:

- in: input
- out: Output
- inout: Bidirectional

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Then the next part is the direction part. So, this in and out; so, they are directions. So, there can be three main direction types one is in for the input, out for the output and in out for the bidirectional; so, in out and in out. So, these are the bits ok.

(Refer Slide Time: 18:16)

VHDL entity

```
entity my_ckt is
  port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit
  );
end my_ckt;
```

A green oval highlights the port declarations (A, B, S) in the code. A callout bubble labeled "Datatypes:" lists "In-built" and "User-defined". To the right is a block diagram of a circuit labeled "my_ckt" with inputs A, B, and S, and outputs X and Y.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then comes the data type. So, these are the data types. So, they all of them are bit. So, there are some built in data type just like any programming language you have got data types like say integer, character, Boolean, real like that. So, here also this HDL language, it has got built in data types like bit is A built in data type, the other built in data types, they are like integer byte etcetera they are they are, but they can be that I can have some user defined data type also that is also possible.

(Refer Slide Time: 18:50)

VHDL entity

```
entity my_ckt is
  port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit
  );
end my_ckt;
```

A blue callout bubble points to the closing bracket at the end of the code, stating: "Note the absence of semicolon ';' at the end of the last signal and the presence at the end of the closing bracket".

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

As far the syntax is concerned there is a small observation that the on the last port we don't have the semicolon, last signal, we don't have the semicolon and the semicolon is put at the end of the circuit description the presence at the end of the closing bracket. So, this one so you see that this the there has to be a semicolon here. So, both the as if this semicolon is taken from here and put at the end of this closing bracket; so, this way we can define a VHDL entity. Next we will be looking into the built in data types.

(Refer Slide Time: 19:30)

Built-in Datatypes

- Scalar (single valued) signal types:
 - **bit**
 - **boolean**
 - **integer**
 - Examples:
 - A: **in bit;**
 - G: **out boolean;**
 - K: **out integer range -2**4 to 2**4-1;**
- Aggregate (collection) signal types:
 - **bit vector:** array of bits representing binary numbers
 - **signed:** array of bits representing signed binary numbers
 - Examples:
 - D: **in bit_vector(0 to 7);**
 - E: **in bit_vector(7 downto 0);**
 - M: **in signed(4 downto 0);**
--signed 5 bit vector binary number

[Hand-drawn diagram: Two rectangular boxes, one labeled 'CPU' and the other 'mem', are connected by a line.]

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, first the scalar or single valued, single valued signal types bit, boolean integer ok. So, these are the different single valued signal types bit, they can be getting the value 0 or 1, boolean can be true or false, integer can be there will be some range ok. So, you can also tell the range of the integer. So, normally there can be this unbounded integer.

So, in that case the system will assume some bound otherwise you can also tell the range of the integer so which is not possible in the software programming languages in most of the cases. So, we do not have this range available, but here we have got the range available. So, here we can say that A is an input port of type bit, G as a output of type Boolean and K is an output of type integer with the range -2^4 to 2^{4-1} . So, it is basically 4-bit signed representation like that ok. So, these are the single valued things.

For the aggregate one or the collection, so, we have bit vector because most of the times in digital design so we will be having connections. So, which are arrays nature for example,

if we are if we are having this say CPU and memory; CPU and memory, then we have got this address bus, data bus type of connections.

(Refer Slide Time: 21:16)

The slide is titled "Built-in Datatypes". It contains two main sections:

- Scalar (single valued) signal types:**
 - bit
 - boolean
 - integer
 - Examples:
 - A: in bit;
 - G: out boolean;
 - K: out integer range -2**4 to 2**4-1;
- Aggregate (collection) signal types:**
 - bit vector: array of bits representing binary numbers
 - signed: array of bits representing signed binary numbers
 - Examples:
 - D: in bit_vector(0 to 7);
 - E: in bit_vector(7 downto 0);
 - M: in signed (4 downto 0);
--signed 5 bit vector binary number

At the bottom of the slide, there is a hand-drawn diagram showing a bus connection. It consists of two rectangular boxes labeled 'C' (CPU) and 'M' (Memory). Eight lines, labeled D₀ through D₇, connect the CPU to the Memory. The lines are grouped into two sets of four, representing an 8-bit bus.

The slide footer includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the date "10.08.2018".

So, we say this CPU and memory. So, we have got this address bus and data bus type of connections. So, these are maybe 16 bit ok, some other data bus may be eight bit like that. So, this way there are a number of bits which are constituting each signal line. So, they are, instead of treating them as individual bits, so, it is better to treat them as a vector. So, that is the bit vector.

So, we have got this bit vector as a, we have got this bit vector as a collection. So, array of bits representing binary numbers. So, that is bit vector it can be signed. So, can array of bits representing signed binary numbers like say we have got D in bit vector 0 to 7. So, this is D is an input port where there are eight lines and the lines are named as 0 to 7. So, D₀ to D₇, we have got 8 bits here, similarly, E is input, it is again of type bit vector and 7 down to 0. So, if this is so we have got E₇ to 0. So, if you are assigning say if you also; so, the one is to, another is down to. So, if you are writing like say E assuming that E is out port, output port.

(Refer Slide Time: 22:39)

The slide has a yellow background with a title 'Built-in Datatypes' at the top. Below the title is a list of built-in signal types:

- Scalar (single valued) signal types:
 - bit
 - boolean
 - integer
 - Examples:
 - A: in bit;
 - G: out boolean;
 - K: out integer range -2**4 to 2**4-1;
- Aggregate (collection) signal types:
 - bit vector: array of bits representing binary numbers
 - signed: array of bits representing signed binary numbers
 - Examples:
 - D: in bit vector(0 to 7); *MSB*
 - E: in bit vector(7 downto 0); *MSB*
 - M: in signed (4 downto 0);
--signed 5 bit vector binary number

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

So, if you are right like say E should get the value D, then this D_0 will come to E_7 like that because this is the MSB here is the. So, this is the MSB here, but in case of E down to 0. So, this is going to be the MSB.

So, that way there, there values will be assigned properly then this M in sign 4 down to 0. So, a sign 5 bit, 5 bit vector that's a binary number ok. So, we can have different types of representations.

(Refer Slide Time: 23:19)

The slide has a yellow background with a title 'User-defined datatype' at the top. Below the title is a list of examples:

- Construct datatypes arbitrarily or using built-in datatypes
- Examples:
 - type temperature is (high, medium, low);
 - type byte is array(0 to 7) of bit;

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

Also we can have the user defined data type. So, this construct we can construct data types arbitrarily or using built in data or by using built in data types for example, we can say type temperature is high, medium, low or type byte is array 0 to 7 of bit. So, this so we are defining two different types here the first one is defining the type temperature. So, and its possible values are high, medium and low. So, the second one is defining a type called bite and it says that this is an array 0 to 7 of bit ok.

So, next if you say that I have got a signal X of type byte. So, this will mean that this is an array of bits 0 to 7 or if you have got the say T of type temperature. So, it can take up the value high, medium and low. So, apparently it seems that this the user defined type. So, why are we introducing this say other types like set temperature as a variable temperature as a type where these values are like this high, medium, low like that.

So in fact, if you look into this VHDL manual you will find that it will allow you to define any user define types, for even you can define the physical quantities like the size of a size of a system, then its weight and all. So, which are not at all related, related to its electrical quantities like say temperature of a body. So, that is not related to the electrical signals within that for that system.

So, but still they can be described. So, while describing a physical system. So, you may take help of those, those special types special data types and use them for your description so that actually makes that these are the spaces the description complete for the completeness of the description, so, you can use those things, though, they are not synthesizable for example, if you have got a type temperature high, low, medium then there is no way by which I can do a synthesis of this temperature in my circuit. So, that is not possible.

But still this strategy has been the given the; so that we can define some physical quantities by means of this VHDL language that makes VHDL really an exceptional one compared to other programming languages where we do not have this type of features.

(Refer Slide Time: 25:51)

Functional specification

- Example:
 - Behavior for output X:
 - When $S = 0$
 $X \leq A$
 - When $S = 1$
 $X \leq B$
 - Behavior for output Y:
 - When $X = 0$ and $S = 0$
 $Y \leq '1'$
 - Else
 $Y \leq '0'$

The diagram shows a yellow rectangular block labeled "my_ckt". Three input lines enter from the left: "A" at the top, "B" in the middle, and "S" at the bottom. Two output lines exit from the right: "X" at the top and "Y" at the bottom. The block represents a functional specification for a circuit.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next will be looking into the functional specification; so, functional speciation like we can say that behavior for output X. So, we can say that when S equal to 0, X is assigned A and when S equal to 1, X is assigned B. So, suppose this is the behavior of X and behavior of Y is when X equal to 0 and S equal to 0, then Y is equal to 1 else Y equal to 0.

So, this may be the description of my circuit. So, behavior of X and Y from the English language description of this my circuit may be we know that this will be the situation then the when S is 0, then X will get A when S is 1. So, there is a multiplexer here. So, between A and B; so, when and the select line is S and then it is X and if X is 0 and S equal to 0, then Y is 1 otherwise Y is 0. So, that way that is the description.

(Refer Slide Time: 26:52)

```
• VHDL description (sequential behavior):
  architecture arch_name of my_ckt is
    begin
      p1: process (A,B,S)
        begin
          if (S='0') then
            X <= A;
          else
            X <= B;
          end if;

          if ((X = '0') and (S = '0')) then
            Y <= '1';
          else
            Y <= '0';
          end if;

        end process p1;
      end;
```

So, how are you going to describe it in the VHDL? So, first one will be looking into a sequential behavior. So, the any description of this VHDL is put into the VHDL architecture.

So, for my circuit the first thing that I have seen is the entity. So, entity has told over the interface with the outside world and then I am trying to describe what is there inside. So, for that I have to use this architecture part.

So, this architecture keyword it starts with the architecture keyword again, I can have several architectures of the same entity. So, for this entity my circuit I am defining an architecture and these are particular architecture name is say arch name. So, I can have another architecture also defined for my circuit. So, there is no restriction about how many architectures I am I am going to define for this entity because this was more from the documentation point of view while initially designed that is why it is like this.

So, you can define one architecture from the behavioral side another architecture from the structural side like that. So, that is why I can have several architectures for the same entity.

So, here we have got this architecture arch name of my circuit is and then begin. So, these are all keywords architecture, of, is, begin, then this as; whenever you are trying to define something sequential so you have to take help of this process block ok. So, this process, it

so p1 is a process. Now if S equal to 0, then X= A, else X = B; X = 0 and S =0, then Y = 1, else Y = 0 end process p1.

So, this is the process description and it depends this process output depends on the signals A, B and S. So, they actually define the values on which they are this process values are going to change. So, if there is A change in the values of A, B and S then this description has to be followed to determine what are the values of X and Y. So, that is so what is done by this VHDL architecture block.

(Refer Slide Time: 29:12)

The screenshot shows a VHDL code editor window titled "VHDL Architecture". The code is as follows:

```
• VHDL description (sequential behavior):
  architecture arch_name of my_ckt is
  begin
    p1: process (A,B,S)
    begin
      if (S='0') then
        X <= A;
      else
        X <= B;
      end if;

      if ((X = '0') and (S = '0')) then
        Y <= '1';
      else
        Y <= '0';
      end if;

    end process p1;
  end;
```

An error message box is displayed, pointing to the line "Y <= '1';":

Error: Signals defined as output ports can only be driven and not read

So, there are some restrictions like this signals defined as output ports can only be driven and not read for example. So, this X is defined as an output port and I am trying to read the value of X here.

So, that is not possible because that is an output from the from my entity so I cannot read it. So, in this way from the, it can catch many of these errors in from that description itself. So, you see that these are the, these are the some of, these are some of the advantages that we have with VHDL ok. So, it can catch this type of design errors very easily, otherwise, it physically means that from the output, you are again drawing a line inside the circuit to check whether X equal to 0 or not. So, that is not a very good design practice. So, that has to be avoided.

Digital Circuits
Prof. Shantanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 47
VHDL
(Contd.)

(Refer Slide Time: 00:20)

The slide title is "VHDL Architecture". It contains VHDL code for an architecture:

```
• VHDL description (sequential behavior):
architecture arch_name of my_ckt is
begin
    p1: process (A,B,S)
    begin
        if (S='0') then
            X <= A;
        else
            X <= B;
        end if;

        if ((X = '0') and (S = '0')) then
            Y <= '1';
        else
            Y <= '0';
        end if;

    end process p1;
end;
```

A hand-drawn diagram shows a vertical bus line with multiple signal lines branching off it. One line is labeled "X" and another is labeled "Y". A callout box points to the "Y" line with the text: "Error: Signals defined as output ports can only be driven and not read".

The footer includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a professor.

So, this type of situations are to be avoided that I cannot take the output signals, output ports, I cannot read it, so we cannot, it cannot be read, they can only be driven. So, outputs can only be driven, so that actually is a good thing, because what happens is that if you are having circuit like this ok, and this is an output port, now you need to. So, suppose this point it goes to 10 different places in my circuit in my in my whole design that goes to 10 different places.

So, what will be the delay of this line or what whether you need a driver, here to be put or not whether you need a driver here to be put or not, so that depends about the number of fan ins. But, if I say that this line is also coming inside ok, then that calculation is becoming very difficult, so I cannot find out this driver strength.

(Refer Slide Time: 01:12)

VHDL Architecture

- VHDL description (sequential behavior):
architecture arch_name of my_ckt is
begin
p1: process (A,B,S)
begin
if (S='0') then
X <= A;
else
X <= B;
end if;
if ((X = '0') and (S = '0')) then
Y <= '1';
else
Y <= '0';
end if;
end process p1;
end;

Error: Signals defined as output ports can only be driven and not read

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if this is the block and this is the line going out, so depending upon the load that you have here, so you can define determine whether I need to put a driver here or not. But, if you are taking this line back inside the circuit, then I cannot do that calculation. So, that is why this VHDL designers, so they have said that this is not a good design practice, and it should be it should not be allowed, it should be avoided.

(Refer Slide Time: 01:42)

VHDL Architecture

```
architecture behav_seq of my_ckt is
signal Xtmp: bit;
begin
p1: process (A,B,S,Xtmp)
begin
if (S='0') then
Xtmp <= A;
else
Xtmp <= B;
end if;

if ((Xtmp = '0') and (S = '0')) then
Y <= '1';
else
Y <= '0';
end if;

X <= Xtmp;
end process p1;
end;
```

Signals can only be defined in this place before the begin keyword

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, if we follow that paradigm, then we can modify the design like this. So, we can say that instead of X, so we take another signal Xtmp. And then we say that the process

description becomes like this that if S equal to 0, then Xtmp equal to A, else X tmp equal to B. If Xtmp equal to 0 and S equal to 0, then we make Y equal to 1 else Y equal to 0. And finally, we transfer this Xtmp value to X, so that is a legal description. So, you can do this thing.

So, signals can only be defined in this place before the begin keyword. So, this is another restriction. So, it is defining like this that I am I am defining a block. So, this is my architecture ok. And within this architecture, at the very beginning, I have to tell: what are the signal lines I have. So, this Xtmp is a new signal line that I am defining. So, this is the Xtmp line. And whatever circuitry I have here after this, so they may take input from Xtmp or they may drive this Xtmp. For example, here, so this Xtmp line is driven by A, here Xtmp line is driven by B or sometime later this X is copied this Xtmp is going to be connected to X here so, that that may be possible.

But, I cannot define this Xtmp line inside the description of this process. So, you cannot take this signal line inside, so that is not allowed that is not allowed. So, signals can be defined can only be defined in this place before the begin keyword. So, before begin keyword I have to define all the signals.

(Refer Slide Time: 03:36)

The slide title is "VHDL Architecture". The code shown is:

```

architecture behav_seq of my_ckt is
  signal Xtmp: bit;
begin
  p1: process (A,B,S,Xtmp)
    begin
      if (S='0') then
        Xtmp <= A;
      else
        Xtmp <= B;
      end if;

      if ((Xtmp = '0') and (S = '0')) then
        Y <= '1';
      else
        Y <= '0';
      end if;

      X <= Xtmp;
    end process p1;
  end;

```

Annotations on the slide:

- A yellow callout box points to the line "signal Xtmp: bit;" with the text: "Signals can only be defined in this place before the begin keyword".
- A larger yellow callout box points to the sensitivity list "(A,B,S,Xtmp)" in the process declaration with the text: "General rule: Include all signals in the sensitivity list of the process which either appear in relational comparisons or on the right side of the assignment operator inside the process construct. In our example: Xtmp and S occur in relational comparisons A, B and Xtmp occur on the right side of the assignment operators".

The slide footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a photo of a speaker.

So, this is the general, so now if you as I said that this process A, B, S, Xtmp. So, what I mean is that if you look into this particular description, you will see that whenever any of these values are changing like you see if A value is changed, then this Xtmp has to be

recalculated. If B value is changed, then the X tmp has to be recalculated. Y value has to be recalculated, and X value has to be recalculated; so, any of this signal values changing I need to recalculate these things recalculate the values of X and Y.

So, this is done by means of this sensitivity list. So, these signals that we are putting within bracket after the process keyword. So, they are called the sensitivity list for the process. So, they will include all signals in the sensitivity list of the process, which either appear in relational comparisons or on the right side of the assignment operator, inside the process construct. So, you see this is the general rule, so which appear in the relational comparison. So, whenever I am doing these comparisons, so this will be done ok; or whenever it appears on the right side of this assignment A, B etcetera, so then also this is done.

So, if you do not do this, then what will happen? So, the for example, if I do not include A, include S in this at this point that means, my design is not dependent on the value of S. So, it is not sensitive to the value of S. So, even if S changes; my design is expected not to change, but that is wrong. Because in my description I see that I need to have this value when this value of S changes, there is a chance that is, so instead it was pre previously outputting A and now it should output B. So, this value of Xttmp has to change as a result X has to change, so it is not correct.

So, any signal that on which this description is going to depend, so they are those names must be appearing in this sensitivity list. Otherwise, there will be there is problem. So, we cannot the description is not correct ok, so that is the point of sensitivity list. So, in our example, this Xttmp, and S occur in relational comparison; A, B and Xttmp occur on the side of the assignment operators. So, they are all part of the, they are all part of this symbol, so that was the sequential description.

(Refer Slide Time: 06:15)

The screenshot shows a presentation slide titled "VHDL Architecture". The slide contains VHDL code for an architecture named "behav_conc" of an entity "my_ckt". The code defines a signal "Xtmp" and contains three concurrent statements using the WHEN-ELSE construct. To the right of the code is a logic circuit diagram consisting of an AND gate followed by an inverter. The input to the AND gate is labeled "A", and its output is labeled "Xtmp". The input to the inverter is labeled "S", and its output is labeled "B". Below the slide is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

```
• VHDL description (concurrent behavior):
architecture behav_conc of my_ckt is
    signal Xtmp: bit;
begin
    process
        begin
            if Xtmp <= A when (S='0') else
                B;
            Y <= '1' when ((Xtmp = '0') and (S = '0')) else
                '0';
            X <= Xtmp;
        end if;
    end process;
end;
```

So, I was defining the behavior of the process in terms of my circuit in terms of a sequential statement. So, first it will execute this statement; then it will execute this if statement; then it will execute this statement. There are three main statements that we have. So, they will be executed one after the other.

However, there can be another style of description which is known as concurrent description. So, and this is more close to the hardware, because we know that that in hardware everything is in parallel in nature. So, this in, as far as the hardware description is concerned, so it is written like this. So, again this has to be architecture, this is another architecture of this my circuit. So, you see the entity name remains same. However, name of the architecture changes, so behave concurrent. So, this is the name given.

And, here also the signal is defined, so signal is defined as this one X_{tmp} . And then I am writing a set of concurrent statements, so this is so this is one first concurrent statement; this is the second concurrent statement; this is the third concurrent statement. So, these three concurrent statements, so they are put one after the other. So, and they are as if they are three parallel piece of hardware, so all of them are executing in parallel. So, this X_{tmp} getting A when S equal to 0 else B . So, it is, it will be taken like this. So, this S is there, and this is 0 and 1. So, if S equal to 0, so this is A . And otherwise this is B , so this is the line X_{tmp} .

Now, second line say the Y is 1, when Xtmp equal to 0 and S equal to 0, so this, so this is my Y. So, this is another multiplexer, Xtmp equal to 0 and S equal to 0. So, both of them being 0, so that is your NOR gate. So, X tmp and S they will be coming there. And, so this is both of them being 0, if both of them are 0, then it should be 1. So, this I need to be this should be an OR gate; this is not an NOR gate, this is an OR gate. Both the inputs are 0, then this is equal to 1, this is wrong sorry.

(Refer Slide Time: 09:06)

```

  architecture behav_conc of my_ckt is
    signal Xtmp: bit;
  begin
    Xtmp <= A when (S='0') else
      B;
    Y <= '1' when ((Xtmp = '0') and (S = '0')) else
      '0';
    X <= Xtmp;
  end ;
  
```

So, this is the first part. This is my S is the select line. And this is A and B. When S equal to 0, then A so this is my Xtmp and this X tmp and S, so they are to be connected to form the input to the, select input of the marks. So, this is 0 and 1; and 0 will be selected when this is Y is equal to 1, when Xtmp equal to 0, and this is Y is S equal to also is equal to 0. So, this is a NOR gate.

So, you put a NOR gate, Xtmp is connected here and S is connected here. So, both of them being 0, the output is 1, as a result this will be selected; otherwise this will be 0. So, this is your Y. And, then X equal to Xtmp. So, from this point, so it is going to X also. So, you see it is some sort of a circuit, you can you can at most see look at there is a buffer here, and it goes like this. So, you see that this is a type of circuit that you get.

Now, all these three all these modules, so they are parallel they are working in parallel. So, there are three different statements, and they are working in parallel, so that is why it is a concurrent behavior of the same block. So, we can do this, so this is this is also valid.

So, somebody may describe the circuit from a sequential angle, somebody may describe the circuit from a concurrent angle, but, both of them are correct. So, this way I can have several architectures for the same behavior.

(Refer Slide Time: 10:46)

The slide has a yellow background and a blue header bar with various icons. The title 'Signals vs Variables' is at the top. Below it is a bulleted list:

- Signals
 - Signals follow the notion of 'event scheduling'
 - An event is characterized by a (time,value) pair
 - Signal assignment example:

Hand-drawn annotations include:

- A wavy line above the list with arrows pointing to the first two bullet points.
- A circle around 'X <= Xtmp' with an arrow pointing to the text 'Schedule the assignment of the value of signal Xtmp to signal X at (Current time + delta)'.
- A circle around 'y := x' with an arrow pointing to the text 'where delta: infinitesimal time unit used by simulator for processing the signals'.

At the bottom, there are logos for IIT Kharagpur and NPTEL Online Certification Courses, along with a video player showing a man in a yellow shirt.

So, far we have talked about signals. Sometimes we are talking using some variables also in our description. So, normally when you are writing a sequential description in our programming software programming languages, we are more familiar with variables not with signals. But, in case of VHDL, we have got two types of temporaries, **one some**, some of them are called signals; some of them are called variables.

So, signals are the temporaries for which there are dedicated signals line or wires that will be there in the system. So, as a result, there will be some delay associated with the values assigned to a signals like if I have got a single line like this. So, if this side you say X; and this side you say Y; and if I say Y gets the value of X, then there is a certain delay associated with this. So, this value of X cannot come to Y immediately, so it takes some time.

Whereas other type of temporary this X and Y, they happen to be variable, and if I say Y equal to X, Y assigned to X that this assignment is immediate, because it is just you can say it is another name of that variable. So, I even if even if I don't use even if I do not use a new variable Y, so I can take the value X directly and use it. But just for the sake of simplicity in my writing, so I have I am using a new variable name.

So, the signals are they follow the notion of event scheduling. So, once this and event is characterized by a time value pair. So, this X equal to Xt_{mp} means schedule the assignment of the value of signal Xt_{mp} to signal X at current time + delta. So, this X has got the, so this Xt_{mp} value will be coming to X, it is scheduled at current time + delta, some time will be, so will be spent. So, this may be infinitesimally small time amount used by the simulator for processing the signals, but it is not immediate.

Like if there are if there are if the two places, one place, so if there is a one place which is generating this Xt_{mp}, this block is executing the this statement X. And this X is used by another block, then you see that when this X has got the value of this Xt_{mp}, so it is not available immediately. So, as far as this block is concerned, so this will see the old value of X for some very small amount of time. So, that actually helps in the simulation process. Otherwise, if that was not there, then the simulation will not terminate. So, it will go on doing the thing again and again, so because it will be always getting a new value of Xt_{mp}. So, as a result it will be going into an infinite loop.

Just to avoid that the simulator designers they have said that this X signal assignment like X assigned the value Xt_{mp}, so this will schedule the assignment to take place after time delta, after the current time after some delta time unit, where this delta is infinitesimally small amount of time, so there is no hard and fast rule like how small it should be is determined by the simulator designer.

(Refer Slide Time: 14:11)

The screenshot shows a presentation slide with the title "Signals vs Variables". The slide content includes a bulleted list about variables, a code example, and a video player at the bottom right.

- Variables
 - Variables do not have notion of 'events'
 - Variables can be defined and used only inside the **process** block and some other special blocks.
 - Variable declaration and assignment example:

```
process (...)  
variable K : bit;  
begin  
...  
-- Assign the value of signal L to var. K immediately  
K := L;  
...  
end process;
```

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the NPTEL logo. On the right side of the footer, there is a video player showing a man in a yellow shirt.

On the other hand, variables they do not have the notion of events. And variables can be defined and used only within the process block. So, only within process you can define a variable. And variable declaration assignment, so it is done like this. So, this variable K bit, so the begin. So, assigned values of K assigned as L. So, this K assigned as L; so K is a variable, so it gets the value of L. So, it gets it immediately, so this L is a signal. So, K gets the value of L immediately.

But if it had been a signal assignment, then it will take some infinite single small amount of time for doing the assignment. So, we have got this signals and variables implemented. So, they have got there are two different types of notions that we have in VHDL. So, signals means they are say physical wires, which will be connecting as a result there will be delays and all; but for variable, so it is immediate.

(Refer Slide Time: 15:16)

Signals vs Variables

- Variables
 - Variables do not have notion of 'events'
 - Variables can be defined and used only inside the **process** block and some other special blocks.
 - Variable declaration and assignment example:

```
process (...)  
variable K : bit;  
begin  
...  
-- Assign the value of signal  
K := L;  
...  
end process;
```

Variables can only be defined and used inside the **process** construct and can be defined only in this place

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, it is just a short hand for some complex expression, so that is done here. So, variable can only be defined and used inside the process construct. So, signals can be defined in any architecture we before the begin statement. But, these variables they can be defined only within the process. So, they cannot be defined outside the process.

(Refer Slide Time: 15:34)

• Simulation is modeling the output response of a circuit to given input stimuli

• For our example circuit:

- Given the values of A, B and S
- Determine the values of X and Y

• Many types of simulators used

- Event driven simulator is used popularly
- Simulation tool we shall use: ModelSim

A → my_ckt → X
B → my_ckt → Y
S → my_ckt

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, how do we simulate a circuit? So, simulation is the modeling of the output response of a circuit to the given stimuli. So, we have got this values of A, B and S you need to determine the values of X and Y. There are many types of simulators that are available; one is called event driven simulator; so that is most popular. So, this event driven simulator is this is, a when event occurs, so, the event occurs on the simulator on the lines A, B and S, so this circuit will be simulated and we will get the outputs X and Y. So, Model Sim is a very popular simulation tool that is available. It is also available freely for download. So, you can check it, and you can use it for your simulation.

(Refer Slide Time: 16:19)

```
architecture behav seq of my_ckt is
begin
  process (A,B,S,Xtmp)
  variable XtmpVar: bit;
begin
  if (S='0') then
    Xtmp <= A;
  else
    Xtmp <= B;
  end if;
  if ((Xtmp = '0') and (S = '0')) then
    Y <= '1';
  else
    Y <= '0';
  end if;
  X <= Xtmp;
  XtmpVar := Xtmp;
end process p1;
end;
```

Time	A	B	S	Xtmp	Y	XtmpVar	X
0-	U	U	U	'X'	'X'	'X'	'X'
0	0	1	0	'X'	'X'	'X'	'X'
0+d	0	1	0	0	0	0	'X'
0+2d	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1+d	0	1	1	1	0	0	0
1+2d	0	1	1	1	0	0	1

Scheduled events list:
Xtmp = (0,0+d)
Y = (0,0+d)
X = ('X',0+d)

Assignments executed:
XtmpVar = 'X'

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

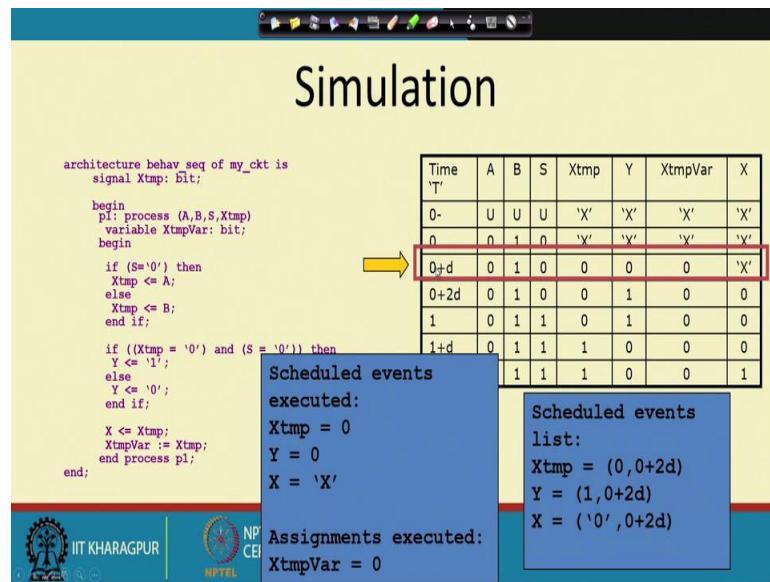
So, how does it simulate? So, if this is the description that I have. Now, at time t 0-, so 0- means just before starting the simulation; the values of A, B and S they are unknown. Then the value of Xtmp is undefined; value of Y is undefined; Xtmp variable is also undefined; then X the variable the signal X is also undefined. So, this is the condition just before starting the simulation. So, at time 0, if I say that I have applied a pattern 0, 1, 0 in A, B, S at time given, simulation starts at this time.

Then a time 0, so this is the situation, so 0, 1, 0. And this Xtmp is getting, so this evaluation is done. So, Xtmp will be assigned the value A, but after an infinite, after a small amount of delay. So, as a result this Xtmp, Y, XtmpVar and X they continue to hold the value X. Then, after some small delay d, this, this assignments will take place as a result is Xtmp gets the value since S equal to 0, Xtmp gets the value of A, so Xtmp becomes equal to 0. Similarly, Xtmp equal to 0; and S equal to 0. So, this Y also gets the value 1. So, Y is also getting the value, so Y sorry. So, at this time Xtmp is so it actually while evaluating, so it takes the previous value of this Xtmp, and that was equal to X. So, this condition was not satisfied. So, it got Y equal to 0. So, you get Y equal to 0. And this XtmpVar assigned to the value of Xtmp, so Xtmp value was 0, so that value is coming here.

Whereas, X the variable of the signal X, it continues to hold the previous value because this assignment has been scheduled, but this is the this is the this is with the value of X Xtmp value. So, this old value is taken, so it is using this. Whereas, this XtmpVar since it is the variable assignment, it takes the value of Xtmp immediately and becomes 0 here, whereas X continues to be a unknown. At 0 + 2 d, A, B, S remains unchanged. Now, this Y becomes equal to 1, because now Xtmp value has become 0, so Y becomes equal to 1, so that is changed. And, then this X is getting the value Xtmp, so X is getting the value 0.

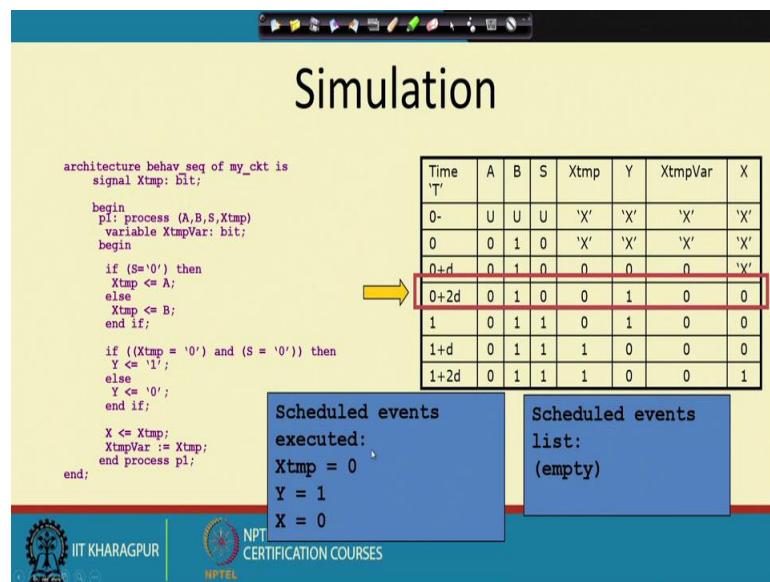
Now, at time 1, if we say that we change the input pattern say to 0, 1, 1, then this previous value of this this thing they continue to hold. And it in a similar fashion the rest of the simulation is done. So, you can the, whatever way we were describing here, so it is done here also. So, at time 0, so scheduled events lists, so it has got Xtmp will be assigned 0; at time 0 + d, Y will be assigned 0, at time 0 + d. X will be assigned X; at time 0 + d, so that is the scheduled event. So, of and the, but the variable assignment is executed. This XtmpVar equal to X, so that is executed.

(Refer Slide Time: 19:31)



Now, at this next time at $0 + d$, so these are the scheduled events executed Xtmp becomes equal to 0; Y equal to 0; and X equal to X. Assignment executed; XtmpVar equal to 0 and these are the scheduled events. So, Xtmp equal to $0 + 2 d$; Y equal to 1, at $0 + 2 d$. So, these are the scheduled events. So, it is at this time so this value should be assigned to the variables and signals.

(Refer Slide Time: 20:00)



Next at $0 + 2 d$, so this is the situation. So, these assignments are to be this is the scheduled list of events. And there is a scheduled events have been executed and there is no further

scheduled event, because this A, B, S, Xtmp, so they have not changed their values. As a result, no further simulation will be required. However, at the next time at time 1, when this schedule will be taking place this A, B, S value changes, then again the simulation will take place.

(Refer Slide Time: 20:28)

The slide has a yellow header and footer. The title 'Synthesis' is in the center of the yellow area. Below it is a bulleted list of points. At the bottom, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES'.

- Synthesis:
Conversion of behavioral level description to structural level netlist
 - Abstract behavioral description maps to concrete logic-level implementation
 - For ex. Integers at behavioral level mapped to bits at structural level
- Structural level netlist
 - Implementation of behavioral description
 - Describes interconnection of gates
- CAD tools are used to perform synthesis

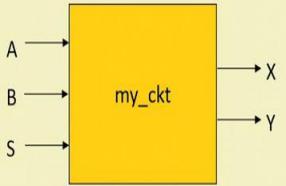
Now, as far as the synthesis is concerned, so synthesis is the process of converting a behavioral description to structural level netlist. The abstract behavioral description maps to concrete logic-level implementation. So, from the description, so it is converted into physical implementation. So, integers, behavioral level you might have defined something as integer, they will be mapped to bits at the structural level, so that gives rise to a structural level netlist.

So, it will be in terms of logic elements connection between them and all, so that will give us a structural level netlist. Implementation of so structural level netlist is nothing but an implementation of behavioral description. So, it describes interconnection of gates, how these gates are interconnected and all. And there are large number of CAD tools. So, there are CAD design companies that have come up with CAD tools for doing this conversion from behavioral level to the structural level, so that helps when you have got this complex design, so that is going to help you.

(Refer Slide Time: 21:31)

Structural level netlist

- Behavior of our example circuit:
 - Behavior for output X:
 - When $S = 0$
 $X \leq A$
 - When $S = 1$
 $X \leq B$
 - Behavior for output Y:
 - When $X = 0$ and $S = 0$
 $Y \leq 1$
 - Else
 $Y \leq 0$



my_ckt

A → my_ckt → X

B → my_ckt → Y

S → my_ckt

□ Logic functions

- $S_{bar} = \sim S$
- $X_{bar} = \sim X$
- $X = A * (S_{bar}) + B * S$
- $Y = (X_{bar}) * (S_{bar})$

IIT KHARAGPUR

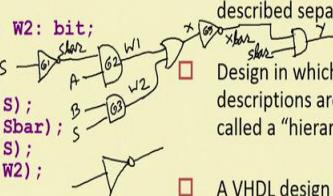
NPTEL ONLINE
CERTIFICATION COURSES

So, so, let us see how this structural level netlist is derived. So, coming back to our behavior, so it was like this when S equal to 0; X equal to A ; X, S equal to 1; X equal to B , and then this output was when X equal to 0, so this was the description. So, logic functions that are associated with it is we say that \bar{S} . So, this is calculated as $\sim S$, \bar{X} is $\sim X$. So, X is defined as A AND \bar{S} OR B AND S . And Y is defined as \bar{X} AND \bar{S} . So, these are the logic functions for this behavioral description, so that can be derived.

(Refer Slide Time: 22:20)

Structural level netlist

```
architecture behav_conc of my_ckt is
  -- component declarations
  signal Sbar, Xbar, W1, W2: bit;
begin
  G1: not port map(Sbar,S);
  G2: and port map(W1,A,Sbar);
  G3: and port map(W2,B,S);
  G4: or port map(X,W1,W2);
  G5: not port map(Xbar,X);
  G6: and port map(Y,Xbar,Sbar);
end;
```



Gate level VHDL descriptions (and, or, etc) are described separately

Design in which other design descriptions are included is called a "hierarchical design"

A VHDL design is included in current design using port map statement



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

And once we have derived that, so we can write down the corresponding structural level description. So, you see that this is, this is another description. And then we can have the, we can we can define some components and then we can we can use those components onto this. So, here this component in this case we do not need to define separately, because this NOT, AND, OR, so they are available as library modules in the in most of the VHDL synthesizers. So, we do not define these component declarations separately. But, we define the signal lines \bar{S} , \bar{X} , W1, W2, they are of type bit.

Now, next what we say is that G1, next, we take help of this statements like G1 port map it is a not gate port map (S bar, S). So, it means that I have got a got an inverter whose name is G1, in one side so, it has got S as input, and \bar{S} as output. So, this \bar{S} as output. So, it is the gate G1. G2 is an AND gate, so that has got a port map of, so AND gate has got so one output and two input. So, this is fed to an AND gate, this AND gate is named G2. And in G2, so I have got this \bar{S} line connected here, the A line comes here and this output is W1.

Similarly, I have got G3 as another AND gate. So, G3 is another AND gate, where I have got W2 as the output, and we have got this B and S as two inputs, B and S as two inputs. And then it says that G4 is another OR gate. So, G4 is another OR gate. And in this OR gate I have got X as the output and this W1 and W2 they are the two inputs. So, this W1 and W2, they happen to be two inputs. And then G5 is a NOT gate or inverter. So, this G5 is a NOT gate.

This is G5, where output is \bar{X} . And finally we have got G6 which is an AND gate. This G6 is an AND gate, where we have got this \bar{X} as 1 input; \bar{S} as another input. So, this is the \bar{S} . And we have got Y as the output. So, this is the final circuit that we get so ok. So, this is so this circuit realizes the same functionality that we have described previously. And you see that the same the description, so I am writing in terms of circuit components ok.

So, I have done the design myself. So, I have done the design in terms of some logic gates. And then I am describing the circuit behavior in terms of those logic gates. So, I have say it says that I have also these are the components that I need NOT gate, AND gate, OR gate. So, these are the three types of components we need. So, in this component declarations parts, so you can define those component, but here it is not necessary as I said that they are part of VHDL library.

So, then I have to say how the ports are mapped like say an inverter, so it has got this NOT, so this is this is the NOT component. And it has got inverter, so it has got this output, input. So, this output is mapped to \bar{S} , and input is mapped to S ok, so that way this component is instantiated. So, this is called these are called component instantiation, so that way can be done.

So, so, then, so this is similarly this AND gate, so this is another one G2 is one instance of AND gate and then the port mapping is like this. G3 is another instance of AND gate; and this is the port mapping. So, this way we have we can describe the whole circuit in terms of these components and their connections.

(Refer Slide Time: 27:00)

Other VHDL resources

- VHDL mini-reference by Prof. Nelson
 - <http://www.eng.auburn.edu/department/ee/mgc/vhdl.html>
- VHDL Tutorial: Learn by Example by Weijun Zhang
 - <http://esd.cs.ucr.edu/labs/tutorial/>

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Now, so these are the some of the further references for this VHDL. So, whatever we have done, so this is just a very preliminary description of this VHDL language, so that is just a start point. So, I wanted to give you a feel like how can we describe the circuits and systems using VHDL language. So, the very vast language and there are very good simulators available.

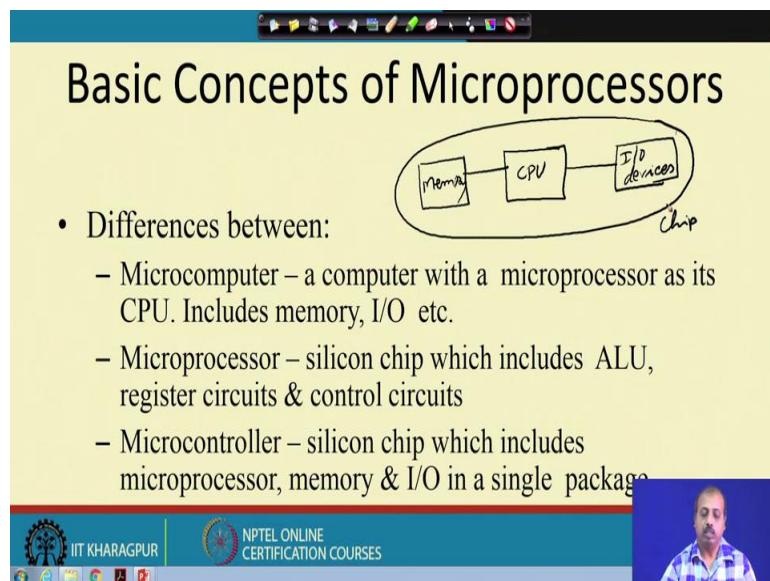
So, you can download those simulators. And you can refer to some of these mini reference and tutorial. So, they are very good, so you can look into them for getting further reference for that of and text books are also available. So, you can go through that for digital design.

Digital Circuits
Prof. Shantanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 48
8085 Microprocessor

We will start with the 8085 Microprocessor. So, this is one of the examples of digital circuits that realizes a complete processor. So, it is a complete central processing unit or CPU, we can say that does many operations and it can be so this is one of the basic unit that is there, when you are looking into any computer system. And this does all the computation as well as the control of the other chips, and circuitry that we have in the in the processor board or in the basic circuit board that you have.

(Refer Slide Time: 00:58)



Basic Concepts of Microprocessors

- Differences between:
 - Microcomputer – a computer with a microprocessor as its CPU. Includes memory, I/O etc.
 - Microprocessor – silicon chip which includes ALU, register circuits & control circuits
 - Microcontroller – silicon chip which includes microprocessor, memory & I/O in a single package

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so if you look into these microprocessors, so they are some microcomputer is a computer with a microprocessor as its CPU. So, to start with you can say that a from a computer system, so it is consisting of the components like the central processing unit or CPU that does all the computations. And then there is a memory unit from where so which actually holds the program to be executed by the CPU. And then it has got the CPU is also connected to some input output devices or IO devices.

So, of the way any computer system works is like this that the CPU it asks the memory for the next instruction to be executed. So, it has every CPU has got a certain fixed set of

instructions that it can understand and it can execute; so, this after getting the next instructions, so it will execute the instruction. And in the process, it may need some input from some I O device, or it may need to output some value to some output to some output device. So, this is the basic operation, though so CPU it accepts the instructions from memory. And produces output may be to some part of memory or to the input output devices.

Now, when we are talking about a microcomputer, so a computer that has got a microprocessor as its CPU. So, this so apart from the micro apart from the microprocessor, so it will also have memory, input-output modules etcetera. So, what is a microprocessor then, so microprocessor is a silicon chip that includes one unit known as arithmetic logic unit, it has got some registers, and it has got some control circuitry to control the operation of these registers. Then this ALU, and the instruction fetching and decoding instruction fetching decoding from memory understanding their types etcetera.

So, microprocessor is a basically a single chip, you can say it is a single chip CPU ok. So, there can be this discrete component CPUs, which was there in earlier systems; But, now almost all the computers that you see, so they host a microprocessor which actually takes care of the operations. And then we have got microcontrollers also, where this entire CPU memory and I O device, so they are put into a single chip. So, this whole thing if you put into a single chip ok, so then what you will get is a microcontroller ok.

So, it has got advantage, because this in that case this memory to CPU communication or say I O device to CPU communication, so it is not off chip. So, it is within the chip as a result, it is quite fast. On the other hand, so in normal system or normal computer system was a CPU to memory communication is a bit slow, because that is off chip communication. So, via some other signal lines particularly the metal lines, which is not on the silicon floor itself, so that makes this off chip communication slow; So, the microprocessor, it will be bit slow; some of in many cases as far as this communication is concerned compared to microcontroller.

(Refer Slide Time: 04:27)

What is a Microprocessor?

- The word comes from the combination micro and processor.
 - Processor means a device that processes whatever. In this context processor means a device that processes numbers, specifically binary numbers, 0's and 1's.
 - To process means to manipulate. It is a general term that describes all manipulation. Again in this context, it means to perform certain operations on the numbers that depend on the microprocessor's design.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so in this course, we will be looking into these microprocessors in more detail. So, what is a microprocessor? So, if we look into the term and the word comes from the combination micro and processor. So, what is a processor, so processor is a device that processes whatever. So, whatever is coming as input, it will process it.

Now, if whatever it comes so it may be meaningful to the processor, it may not be meaningful to the processor. If it is not meaningful to the processor, it will simply ignore it; or it will give some error output. And if it is meaningful, then it will execute it. So, processor is a device that processes numbers, specifically binary numbers, 0's and 1's. What do you mean by process, so processing means to manipulate. It is a general term that describes all manipulation. And again in this context, you can say that it means perform certain operations on the numbers that depend on the microprocessor's design.

So, microprocessor design that they have told like for a particular microprocessor, they decide like what are the operations that I will support in this processor. Accordingly what should be the format in which this instructions be given to me and all; And then this processor once it gets instructions in that format, it can execute it using its internal resources like this ALU the arithmetic logic unit, and the registers and all.

(Refer Slide Time: 05:53)

What about micro?

- Micro is a new addition.
 - In the late 1960's, processors were built using discrete elements.
 - These devices performed the required operation, but were too large and too slow.
 - In the early 1970's the microchip was invented. All of the components that made up the processor were now placed on a single piece of silicon. The size became several thousand times smaller and the speed became several hundred times faster. The "Micro" Processor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



What about micro? So, micro it is a, it is an addition. In fact, you can say the late 1960's, so processors were built using discrete elements. As I was telling I can have this separate registers clubbed together into one block, and then you can have this controller design separately. I can have this instruction fetch unit, which fetches instructions from memory separately that way I can have different, different processors; I can have ALU separate. So, this type of designs, so they were known as bit sliced design. But, the problem with there that type of design is that since they are all the components are discrete components, they are discrete chips, or then the communication between them takes lot of time.

So, from this microprocessor actually trying to reduce that time so, this is in 1970's, the microchip was invented. And all, the component that that made of processor when now placed into a single piece of silicon. And the size became several thousand times smaller, and the speed became several hundred times faster ok, so that is how the microprocessor was born. So, if we can take this off chip things into on chip as I already said, the speed will increase. And naturally the space requirement will also go down, because there is no off chip connection.

(Refer Slide Time: 07:17)

Was there ever a “mini”- processor?

- No.
 - It went directly from discrete elements to a single chip. However, comparing today's microprocessors to the ones built in the early 1970's you find an extreme increase in the amount of integration.
- So, What is a microprocessor?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Was there any mini processor that way? So, we are talking about general processor, we are talking about microprocessor. Was there any mini processor? The answer is no. It went directly from discrete elements to a single chip. However, comparing today's microprocessors to the ones built in early 1970's you can find an extreme increase in the amount of integration. So, this 8085 it is one of the very simple processors that we will look into.

But, if you look into today's processors that, are used in the computer systems today, so they are so complex that it is very difficult to cover in a single course. Several components are designed with high level of optimization. And with integration between the this processor designer, operating system designer, programming language compiler designers and all, so that way it makes it very complex. So, what is a microprocessor?

(Refer Slide Time: 08:16)

Definition of the Microprocessor

- The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logic logical operations according to the programs stored in memory, and then produces other numbers as a result. So, this is the very general definition of a microprocessor.

So, if you look into these terms one by one, so it is a programmable device. So, you can tell the microprocessor what to do ok. And that program has to be stored in some memory, so as it is said here. So, the program should be stored in some memory. And from there, it will be doing the operation; it will take the instructions from there.

Now, it will so if any particular instruction, so this talks about manipulating some numbers ok. And this numbers may be coming from the memory itself; or it may be coming from the input output devices the input devices. And then on those numbers that the processor has got, so it will do some operation. And those operations may be some arithmetic or logical operations.

So, those arithmetic logical operations are basically done by the ALU; or the arithmetic logic unit. And then some output is produced. And this output that is produced, so it may be stored in the memory or it may be stored in some it may be flashed onto some output device, it may be sent to some output device for some further understanding or processing.

(Refer Slide Time: 09:51)

Definition (Contd.)

- Lets expand each of the underlined words:
 - Programmable device:** The microprocessor can perform different sets of operations on the data it receives depending on the sequence of instructions supplied in the given program.
 - By changing the program, the microprocessor manipulates the data in different ways.
 - Instructions:** Each microprocessor is designed to execute a specific group of operations. This group of operations is called an instruction set. This instruction set defines what the microprocessor can and cannot do.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the definition of basic microprocessor. So, if we expand these terms one by one, so what a programmable device, a microprocessor can perform sets of operations on the data it receives depending on the sequence of instructions supplied in the given program. So, if the microprocessor works with a program and the program is stored in the memory. And this program will tell the instructions that the microprocessor should execute. By changing the program, the microprocessor manipulates the data in different ways definitely, so that is why the term programmable ok. So, this is a programmable device, so that is you can program it, but it is very complex device.

Then the instructions that a microprocessor can execute: So, each microprocessor is designed to execute a specific group of operations. And this group of operation is called the instruction set. So, today if you are designing a new microprocessor, then the first thing that we have to answer is: what is the instruction set that we are planning about. So, what is the instruction set of the microprocessor; or if you come cross a new microprocessor and try to use it, then the very first thing that we will get stuck with is: what is the instruction set for the microprocessor. So, what are the instructions that are supported by the microprocessor.

So, any processor study or processor design, it will start with the instruction set. So, this instruction set will define what the microprocessor can do, and what it cannot do. So, anything which is not documented in the instruction set, so that cannot be done by the

microprocessor. So, sometimes this appears to be a bit whimsical, because you may find that some operations are done, whereas some other operations are restricted. So, apparently it seems that why restriction, but that comes from the design side, may be the designers for their ease of design or some constraint in the design modified the operations like that.

(Refer Slide Time: 11:55)

The slide has a yellow background with a black header bar containing various icons. The title 'Definition (Contd.)' is centered in a large, bold, black font. Below the title, there is a bulleted list of points:

- **Takes in:** The data that the microprocessor manipulates must come from somewhere.
 - It comes from what is called “input devices”.
 - These are devices that bring data into the system from the outside world.
 - These represent devices such as a keyboard, a mouse, switches, and the like.

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo on the left and the NPTEL Online Certification Courses logo on the right. The NPTEL logo features a circular emblem with the text 'NPTEL' below it. The footer bar also contains a small number '8' on the right side.

What does it take in, so it takes in the data from somewhere ok. So, where from, so it can come from input devices like in a computer system I can have a keyboard, a mouse, or even may be some I can have a signal line, which may be made high or low. So, these are called these are the inputs for the microprocessor.

So, this input devices, they can give that data the data for the microprocessor to work on. So, these devices these are the, these input devices are the devices that bring data into the system from the output world. And these represent devices such as keyboard, mouse, switches like that, so that is that takes in.

(Refer Slide Time: 12:39)

Definition (Contd.)

- **Numbers:** The microprocessor has a very narrow view on life. It only understands binary numbers.
- A binary digit is called a bit (which comes from **binary digit**).
- The microprocessor recognizes and processes a group of bits together. This group of bits is called a “word”.
- The number of bits in a Microprocessor’s word, is a measure of its “abilities”.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And what do you mean by numbers, so numbers a microprocessor has a very narrow view of on life, you can say it only understands binary numbers. So, it is this statement is slightly you should say this is tricky, because as we know that any digital system it works with 1's and 0's only, so that is true for microprocessors also.

Though while understanding the operation of a microprocessor, we will see the, we will write in terms of some text some ASCII characters and all ok, with some mnemonics, and variable, register name and all. But, internally everything is nothing but some binary numbers. A binary digit is called a bit, it is binary digit. A microprocessor recognizes and processes a group of bits together. And this is called a word.

So, in our digital logic classes, so we know that we have got this bit representation, binary representation or individual bits are there. Now, the when this microprocessor is can they considered, so it may not work with individual bits that way. So, it may work with a group of bits together, which is called a word.

For example, when it is getting instructions from memory, so it may be each instruction is say 8-bit or 16-bit wide. So, as a result when it is getting the instruction, so it is getting a, it is fetching 8-bit from the memory; or if it is the doing some addition operation, so it is not bit by bit addition, but addition of two words, where each word is 8-bit ok. So, we have talked about full adder, half adder etcetera, while it was operating at bit level. But, here

this is we will be talking at the word level, where the word size is the size is the number of bits with which the processor works or the microprocessor works.

So, the number of bits in a microprocessor's word, is a measure of its ability. So, if a microprocessor is powerful microprocessor, then it will be able to work with larger sized word like. If we are if we say that a microprocessor is 8-bit microprocessor, so what we essentially mean, is that all the operations that it is doing is on 8-bit data. So, doing operating on 8-bit data is problematic, because then you can have positive numbers. If you are thinking about positive numbers, so you can go from 0 to 255 and both positive and numbers, so -128 to +127, so that is the sever restriction.

On the other hand, consider the situation where the microprocessor is a 16-bit microprocessor, it can do 16-bit operation. So, internal circuitry that it has can handle 16-bit of data at a time so, in that case, the number that we have. So, now the range of their ranges and all, so they are much larger, so that way it may be useful for us. Like if a microprocessor supports only 8-bit say addition, then definitely you can use that 8-bit addition to get 16-bit addition. But, that will be in terms of a number of instructions for the microprocessor, so that is basically a software solution I can say.

So, if I have got two 16 bit numbers to be added, so we can first add the least significant 8 bits. And if there is a carry, then that carry is transferred. And then in another instruction, we add the most significant 8 bits. And then another instruction, we may add the carry coming from the first stage to the second stage. This way the single 16-bit addition can be realized by say three 8, 8-bit addition. But, the time taken is more compared to the situation that if the microprocessor directly support supported 16-bit addition by a single instruction I could have done that 16-bit addition.

So, this way whatever is limited by the hardware in the microprocessor, it can be done by means of software, by means of program statements, but that will make the system slow. So, output will be the, it will take quite some time to get the output.

(Refer Slide Time: 16:49)

Definition (Contd.)

- Words, Bytes, etc.
 - The earliest microprocessor (the Intel 8088 and Motorola's 6800) recognized 8-bit words.
 - Later microprocessors (8086 and 68000) were designed with 16-bit words.
 - A group of 8-bits were referred to as a "half-word" or "byte".
 - A group of 4 bits is called a "nibble".
 - Also, 32 bit groups were given the name "long word".
 - Today, all processors manipulate at least 32 bits at a time and there exists microprocessors that can process 64, 80, 128 bits

So, the earliest microprocessors like say Intel 8008 and this 8088 and Motorola's 6800, so they recognized 8-bit words. In fact, 8085 also recognizes the 8-bit word. Later microprocessor like 8086 and 68000, so they are 16-bit words and then the group of 8-bits are referred to as half-word or byte. And a group of 4 bits is called a nibble. So, word size with a word size is 16-bit, then half of it is half-word or byte. So, word size can be say 32, 64 even 256 bit word size is also common ok, so that is there. So, it may be divided into smaller parts. So, a group of 8-bit will be called a byte, group of 4 bits will be called a nibble.

Similarly, if you take 32 bit group, so they are often called long word. So, all processors manipulate at least 32 bits, most of the processors that we have in today's time, so they manipulate at least 32 bits at a time and there exist microprocessors that can handle 64, 80 or 128 bits at a time, so that way it is much larger. So, it can handle more number of bits together, the word size is higher.

(Refer Slide Time: 18:06)

Definition (Contd.)

- Arithmetic and Logic Operations:
 - Every microprocessor has arithmetic operations such as add and subtract as part of its instruction set.
 - Most microprocessors will have operations such as multiply and divide.
 - Some of the newer ones will have complex operations such as square root.
 - In addition, microprocessors have logic operations as well. Such as AND, OR, XOR, shift left, shift right, etc.
 - Again, the number and types of operations define the microprocessor's instruction set and depends on the specific microprocessor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then looking into the arithmetic and logic operations so, every microprocessor has arithmetic operation such as add and subtract as part of its instructions set, because these are the basic operation. So, if addition and subtraction are not supported, then you cannot do any of this meaningful, any meaningful computation using that module, so that is why all microprocessors they have got this add and subtract as their in their instruction sets. And most microprocessors will have operations like multiply and divide. Of course, 8085 does not have multiply divide instructions, but there are processors, where you have got this multiplication and division type of operation.

Some of the newer one will have complex operation like square root. So, if you look into these digital signal processors, they are also microprocessors, but they can handle they can do even the FFT operation ok. So, they can do multiply, accumulate type of operation in one shot, so that way though these instructions are complex instructions. So, it, it may do some complex operation as well.

Apart from this arithmetic operation, microprocessors also do some logical operation such as AND, OR, XOR, shift left, shift right, etcetera. So, these are some logical operations, so that is why I say that the arithmetic and logic operations are done by the microprocessor. The numbers and types of operations define the microprocessor's instruction set and depends on the specific microprocessor.

So, what are the operations that you can do that will be determining the instruction set that the microprocessor should have. And it will have the, it is specific for the microprocessor, so it is not common to have the same set of instructions across all the microprocessors. But, more or less the basic operations will remain same like this addition, subtraction, then AND, OR, XOR, shifting, so these operations they remain same. Of course, the format may vary slightly from one processor to another processor.

(Refer Slide Time: 20:09)

Definition (Contd.)

- Stored in memory :**
 - First, what is memory?
 - Memory is the location where information is kept while not in current use.
 - Memory is usually measured by the number of bytes it can hold.
 - KB, MB, GB etc.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



This microprocessor, so they, they work with program and these programs are stored in memory that we have said. So, what is memory? So, memory is the location where information is kept while not in current use. So, here all the information are kept. And then, so memory is usually measured in terms of bytes it can hold; kilobyte, megabyte, gigabyte like that. And then that is connected to the processor. So, the processor will be getting those instructions and data from memory and operate from them, so that is the term stored in memory.

(Refer Slide Time: 20:44)

Definition (Contd.)

- Stored in memory:

- When a program is entered into a computer, it is stored in memory. Then as the microprocessor starts to execute the instructions, it brings the instructions from memory one at a time.
- Memory is also used to hold the data.
 - The microprocessor reads (brings in) the data from memory when it needs it and writes (stores) the results into memory when it is done.

So, when a program is entered into a computer, it is stored in a memory. And then the microprocessor starts to execute the instructions, it brings the instructions from memory one at a time ok. So, what we mean is that first time we write a program first time we write a program may be we write in some language say I have I am writing in say C language, say I am writing a program which is program.c, now this program it goes through a compilation phase. We have got a compiler, which generates the executable version or the binary file. This is the binary file executable file.

Now, this if this is the memory that I have in my system in this memory, so this binary file is loaded at some part ok. So, there may be this binary file is loaded here, and then the microprocessor when it is executing. So, it will start from the first instruction that we have, and proceed one after the other till it reaches the end of the program.

So, this program is stored in the when a program is stored in computer, it is stored in memory. And the processor starts to execute the instruction, it will bring one instruction at a time from a memory, and it will execute it. Memory is also used for holding the data like sometimes we need to hold some more data, some temporary data or some say data which is specific to some operation.

For example, if you are doing a filtering operation, then the filter coefficients they are to be stored ok. Filter signal value, so they come from the outside, but this coefficients they need to be stored. So, coefficients are stored in memory that way we can have some data

stored in the memory that is the microprocessor will read in those data from memory. And after doing the operations, so it will write the result onto some memory location ok, so that is also there. It will write it on some memory location as a temporary value; and then again when it is needed, so it will get it from memory and execute it.

(Refer Slide Time: 23:01)

Definition (Contd.)

- Produces: For the user to see the result of the execution of the program, the results must be presented in a human readable form.

- The results must be presented on an output device.
- This can be the monitor, a paper from the printer, a simple LED or many other forms.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

What do you mean by produces, so produces means for the user to see the result of the execution of the program, the result must be presented in a human readable form. So, what happens is that you see you have written a program that program is a binary file, so nothing much is understandable from there. Now, the program is executing, so output is again some binary data, so again nothing is understandable nothing much is understandable from there. So, this way it is progressing.

Now, you see that if you want that the value whatever the value the program has computed to be interpreted or to be understood, so we need to show it in some human readable form. So, result must be presented to some output device. And this can be monitor, a paper from the printer, a simple LED or many other forms. So, in but in some output format, it should be shown ok. So, this way it can produce the data.

(Refer Slide Time: 24:10)

A Microprocessor-based system

- From the above description, we can draw the following block diagram to represent a microprocessor-based system:

```
graph LR; Input[Input] --> Microprocessor[Microprocessor]; Microprocessor <--> Memory[Memory]; Microprocessor --> Output[Output]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

15

So, a microprocessor based system overall, it will look something like this. So, we have got this microprocessor and the memory. So, this microprocessor may get something from the memory or it may write something onto the memory. So, basically the microprocessor it gets the successive instructions from the memory, and it executes the instruction. The instruction execution may depict that it has to read some value from the input. So, it reads it from the input device.

And then it may say that it has to produce some output to some output device. So, it will produce it to some output device. May be as the input device we have got a simple keyboard, and at the output device we have got a set of LEDs. And the program memory that the program that is executing may be it will be reading the values from the keyboard, and then it will be producing some output pattern to the LED, depending on the input value that is read, so that is actually the program. So, where it is giving me the logic like how to produce the output from the input, so that is the program.

And it also includes the instructions to read from the keyboard, and then write onto the LED display that we have. So, ultimately when it comes to the LED display, so it is human understandable. Similarly, when we are giving the input from the keyboard, so that is also human understandable; so, this human understandability part is taken care of by the input output devices. Rest of the thing, they are taken by the microprocessor directly in term in binary form.

(Refer Slide Time: 25:46)

Inside The Microprocessor

- Internally, the microprocessor is made up of 3 main units.
 - The Arithmetic/Logic Unit (ALU)
 - The Control Unit.
 - An array of registers for holding data while it is being manipulated.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, what do we have inside the microprocessor, so inside the microprocessor, so it has got 3 main units; one is known as the arithmetic logic unit, one is known as the control unit, and another is the array of registers for holding data while it is being manipulated; So, this basically as we have seen by this time that this we have to do some operations so some computation.

For that we need some logic circuitry and that is done by the that is the arithmetic logic unit, which will do that all the arithmetic instructions like addition, subtraction, if multiplication is supported ok, and multiplication, division, shifting ok, then this logic operations like say logic operations like the AND, OR, XOR, so all those. So, they are done by the arithmetic logic unit. So, this is one of the very important components that we have inside the microprocessor. And then we have got an array of registers that can that hold the data while it is being manipulated like if I am doing some addition operation, then the operands of the addition they are to be held at some place at some time at some register.

So, otherwise the values will change, and the addition circuitry will not behave properly. So, those additions are done, so these temporary values are stored in those registers. So, we can have a number of registers, and normally any processor has got a set of registers, where it will be doing the operation and they, so, apart from that there is a control unit, which will be controlling the operation of all these other units in that is a registers, and ALU, then accessing memory and all, so that will be done by the control unit.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 49
8085 Microprocessor
(Contd.)

(Refer Slide Time: 00:21)

Organization of a microprocessor- based system

- Let's expand the picture a bit.

The diagram illustrates the organization of a microprocessor-based system. At the center is a blue rectangular box representing the microprocessor itself, divided into three main functional blocks: ALU (Arithmetic Logic Unit), Register Array, and Control. This central unit is connected to three external components via a horizontal double-headed arrow labeled "System Bus": an "I/O Input / Output" unit at the top, a "Memory ROM RAM" unit at the bottom, and another unlabeled yellow box to the right. The entire slide has a dark blue header bar with various icons and a red footer bar containing the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man speaking.

So, if we expand this microprocessor structure or system structure, then it consists of the modules. Like, if this is the microprocessor that we have consisting of this ALU, the register array and the control.

Then we have got this memory and IO. So, they are that are the other parts that we will have in the system and they are connected by means of some bus. So, which is known as the system bus. So, through this system bus this processor, the microprocessor will talk to the memory will also talk to the input output devices.

(Refer Slide Time: 00:49)

Memory

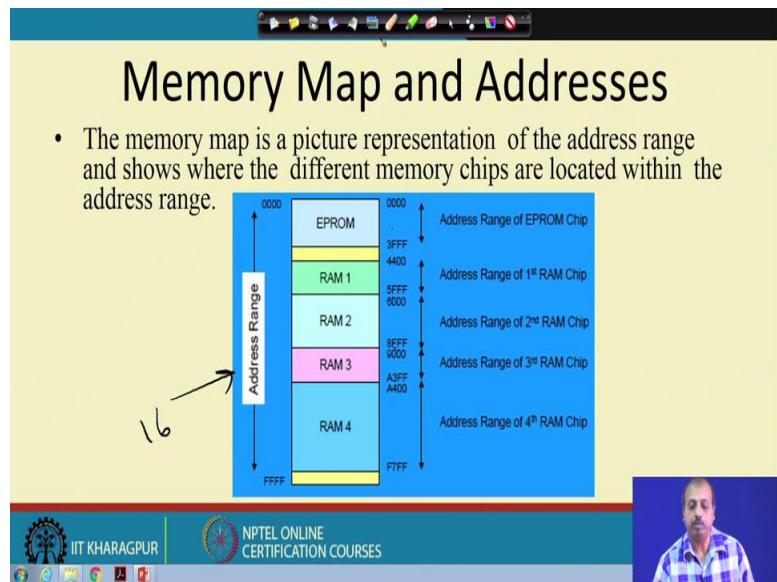
- Memory stores information such as instructions and data in binary format (0 and 1). It provides this information to the microprocessor whenever it is needed.
- Usually, there is a memory “sub-system” in a microprocessor-based system. This sub-system includes: Registers, ROM, RAM

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, under this, we have this memory will first look into this memory the part that we have and memory is nothing but a storage. So, it stores information such as instructions and data in binary format and it provides information to the microprocessor whenever it is needed. So, this is basically some storage that we have.

And as I said that when we compile a program and then load it into the memory. So, this memory is holding that program. So, like that it is a storage part. So, memory is a, we talk about memory as a subsystem and in this subsystem; we have got registers ROM and RAM. So, registers are part of the CPU so that is inside the microprocessor, but this ROM and RAM. So, they are outside the microprocessor and they are as a separate chips ok. So, they are microprocessor actually talks to this ROM and RAM by means of that system bus that we have seen but from the angle of storage, so, all these modules registers ROM and RAM. So, they are storing some information. So, we will put them under a broad heading of memory subsystem. So, in the memory subsystem all these information are stored, and then it can get the information the processor can get the information from that part.

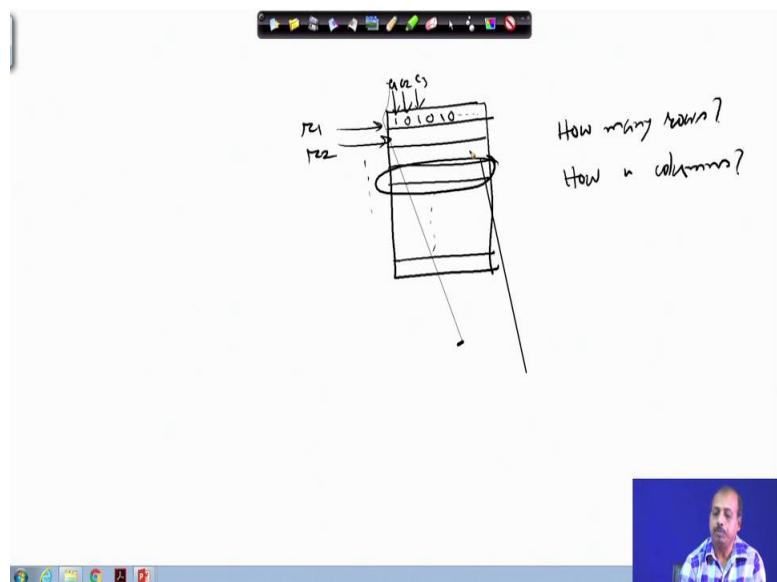
(Refer Slide Time: 02:08)



The slide title is "Memory Map and Addresses". A list bullet point states: "The memory map is a picture representation of the address range and shows where the different memory chips are located within the address range." Below the text is a diagram of a memory map. On the left, a vertical bar labeled "Address Range" has arrows pointing up and down. To its right is a stack of memory components: EPROM (top, yellow), RAM 1 (green), RAM 2 (light blue), RAM 3 (pink), and RAM 4 (yellow, bottom). Above the stack, address values are listed: 0000, 3FFF, 4400, 55FF, 6000, 8FFF, 9000, A3FF, A400, and F7FF. To the right of the stack, vertical dashed lines connect each component to its corresponding address range. The ranges are labeled: "Address Range of EPROM Chip", "Address Range of 1st RAM Chip", "Address Range of 2nd RAM Chip", "Address Range of 3rd RAM Chip", and "Address Range of 4th RAM Chip". At the bottom of the slide, there is a footer with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a man speaking.

So, next we will be talking about an important concept which is known as memory map and address. So, as I said that a memory is nothing but, a memory is nothing but some storage.

(Refer Slide Time: 02:24)



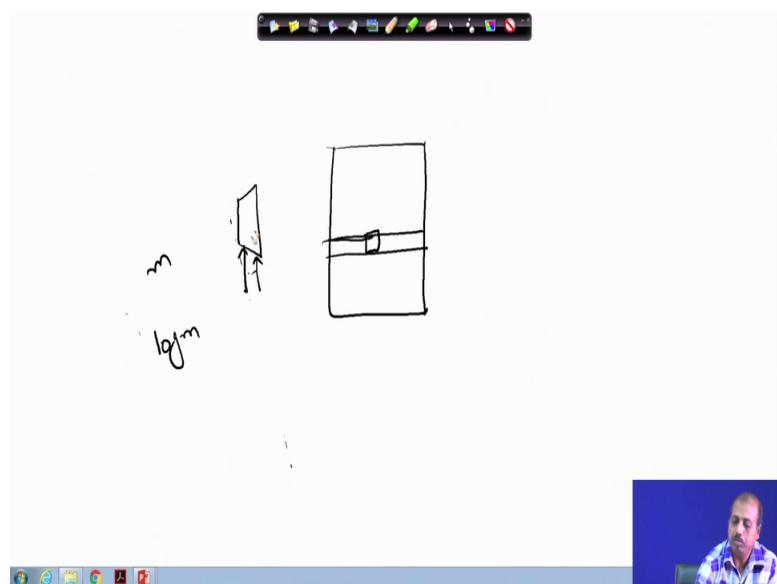
The slide contains a hand-drawn diagram of a storage matrix. It consists of a grid of vertical columns and horizontal rows. The top row of the grid is labeled with binary digits: 1, 0, 1, 0, Two arrows point to specific columns in the grid, labeled "r1" and "r2". To the right of the grid, two questions are written in handwriting: "How many rows?" and "How n columns?". At the bottom of the slide, there is a footer with various icons and a video player showing a man speaking.

So, this is the storage then there are two questions to be answered like how are you storing information there. So, we have said that this is in terms of 0s and 1s. So, these are the individual locations that I have individual rows that I have of 0s and 1s.

So, how many rows are there and how many columns? So, I can say my question is how many rows and how many columns. So, in individual column; so, this this may be storing the parallel bit pattern 1 0 1 0 1 0 like that ok. So, these are the column. So, this is the first column, this is the second column this is the third column like that.

Similarly, so, these are the columns C1, C2, C3, etcetera and these are the rows. So, this is the row 1 this is row 2 ok. So, like that. So, whenever I am trying to access this memory. So, it is always done in terms of rows ok. So, normally we do not access memory in terms of these individual bits or in terms of column. So, we will be always accessing a particular row together. So, we will always be accessing a particular row together ok. So, it is never that.

(Refer Slide Time: 03:49)

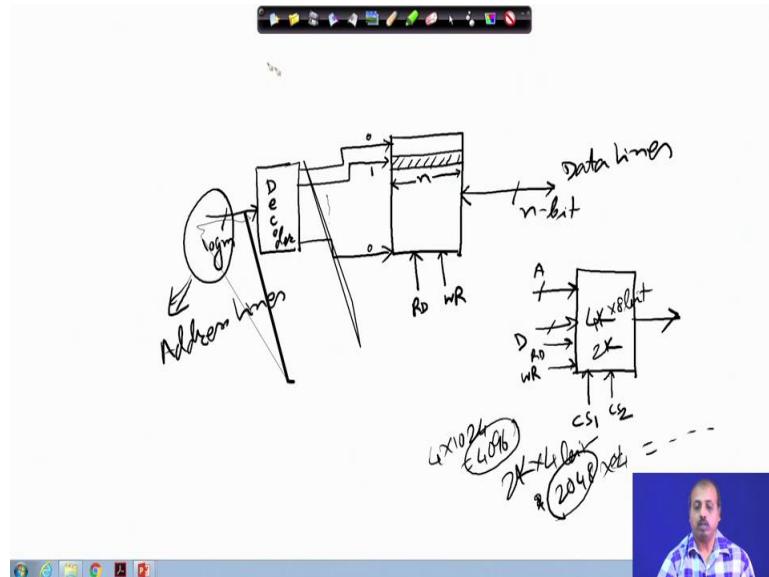


We access a particular bit within a memory say, it is not that we access a single bit separately ok. So, it is not done. So, this entire row is entire, this entire row is accessed by the processor.

Now, if I if this if I have got say m rows then somehow the processor need to tell like which row it wants to access fine; so to tell that it needs to; so, if there are m such rows. So, to identify any row here; so, I will need $\log m$, I will need $\log m$ number of bits to be specified. So, what I can do. So, I can have a decoder where I can say. So, I can put a decoder and the decoder will be the decoder can find out like where to which location I

am talking about. So, it is like this that if I put a decoder here and then here I tell what is the here I put that log m bits better, I can I should make a different notation.

(Refer Slide Time: 05:10)



So, this is my memory block that I have and this is the decoder.

So, here I am feeding log m number of bits as a result this decoder generates m such enable signals, it generates m such enable signals and depending upon at the value that I have given here. So, one of these lines will be enabled and rest are all disabled. So, may it may be that it has enabled it, it has put this line as 1 and all these lines are at 0 as a result. So, this particulars row of the memory will be accessed.

So, if we look into a memory chip, then we have got this log m lines coming as input one input and this is commonly known as the address lines. So, in the address lines, you need to tell which row, you want to access and as I said that all these location, all these bits. So, they are available at the output of this memory chip. So, this is output. So, if this is if the number of columns here is n the number of columns here is n, then at this point, I get an n bit output and this n bit output. So, they are known as the data lines, they are called data lines, fine.

Now, if it as we know that if it is a ROM, then I can put the; I can give it address and I can tell it I can give it the signal read. Accordingly, this content will be available here on the other hand, if it is a RAM, then there are 2 controls read and write. Now if I am trying to

read the content, then I will give the read signal put the address here, if I am ask to trying to write something onto to the location, then this line is bidirectional in case of RAM, then I put the value onto the these data line. So, put the address here and then you give a write signal. So, that whatever value is available on the data line gets written onto the location.

So, this way, we this is the generic structure of a memory of a memory block. So, we can say that it is like this it has got the address lines or A it has got the data line or D and it has got some control line. So, it has got this read and write controls and also many of this memory chips. So, they will have some chip select signals CS and they may there may be multiple such CS chip select signal CS1 CS2 like that.

So, and each this memory chip has got some capacity like it may be say 4K, it may be say 2K. So, this is that is 4K that identifies it is 4 kilobyte 4 kilo such locations are there. So, if it is 4K into 8 bit; that means, individual locations are 8 bit wide and there are 4 K, 4 K such that is 4×1024 that is 4096 locations are there in this chip.

Similarly, if it is 2 K by 4 bit. So, this is 2×1024 by 4 bit. So, this is 2048 rows and each of 4 bits. So, total whatever be the number of bits coming. So, that is the total capacity of this, but this is this has this has got 2048 locations compared to 4096 location here. So, this 2 K chip will have 2048 words in it.

Anyways so, that we have seen in the while discussing about this memory chips and all. So, in the context of microprocessors what happens is that every microprocessor will have certain number of address lines and a certain number of data line. So, it will be coming from the microprocessor itself and they are they those lines are fixed ok. So, if a microprocessor has got m address lines, then it can access 2^m memory locations from that and it has got, the n data lines will tell that the memory chip that we gone to connect. So, to provide me n bit data at a time.

Now, depending upon the structure depending upon the depending upon the chips that we are connecting and then we can have different type of connection. For example, you may say that, this is the total address range of the microprocessor like here it is said that ok; I have got this is running from 0000 to FFFF. So, this is through if the microprocessor has got 16 bits address line because it is FFFF. That means, it is now all the 16 bit value. So, this has got 64 K alternative alternate values in the address range.

So, my microprocessor address bus the address lines that the microprocessor generates must be 16 bit and then using that. So, it will be able to distinguish between any of this 64K locations, but the memory chips that we have. So, they are that memory chips may not be having 64K capacity in a single chip ok. So, they are so, I will need multiple such chips also I need to distinguish between some part of the, some part of the system. So, if the some part of the program may be fixed whereas, some part are varying.

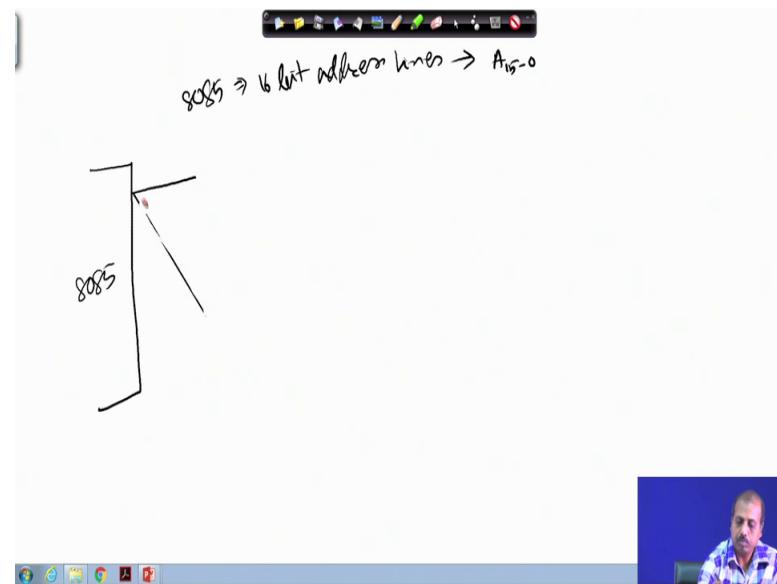
So, for example, the operating system part of the program of the system. So, that is fixed so that we need to have in some part of the memory. So, normally what is done is that at the beginning of this address range. So, we have got this we try to put one EPROM and in this EPROM, so, we put the operating system part and then in the later part. So, we try to put the RAM chips ok. So, where for that is for it to be used for different purposes for storage purpose.

So, the first RAM chip that is shown here so that will be ranging over the addresses 4400 to 5FFF in the second one, so that will be ranging from 6000 to 8FFF; so their capacities are not same, but in a general design, so, their capacities may be same also depending upon the RAM chip that we put and between this EPROM and this RAM 1. So, there is some space which his not utilized. So, this address range that is the 4000 to 43FF so that part of the address. So, that is not used by the, by the processor ok. So, there so, it is assumed that the processor will not generate any address in that range. So, if it generates then the system behavior is unpredictable.

However similarly towards the end so, it generates address up to F7FF and it does it from F800 to FFFF, so, this address range is never generated by the processor. So, it is assumed like that whatever program we will be loading into the memory execute, the processor will be executing, it will never generate addresses in the range 4000 to 44FF and similarly from F800 to FFFF. So, it will not generate any address in that range.

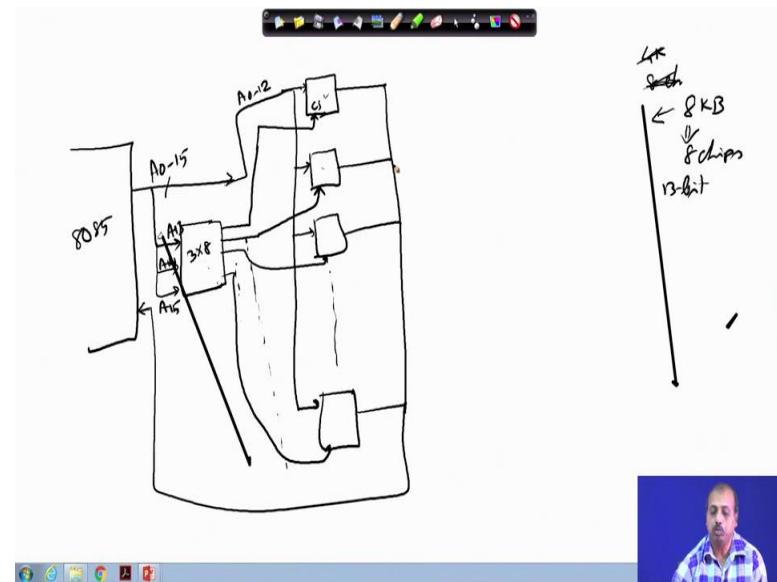
Anyway otherwise, we can connect it like this now how do you make this connection. So, you are as said that if, if this is this is the address lines that we have. So, here I have got 16 address the 16 address lines coming here and how this address lines are going to select this individual chips ok. So, that has to be seen. So, we will take an example and try to explain that part.

(Refer Slide Time: 13:39)



So, suppose I have, so, in the 8085 based system that we have, so, I have said that it has got 16 bit address lines suppose 16 bit address lines are named as say A0 to A15. So, they are named as A15 to 0 where A15 is the highest the most significant bit and A0 is the least significant bit.

(Refer Slide Time: 14:22)



So, from my 8085 chip; so, if this is the 8085 chip. So, these 16 lines are coming out sorry. So, from my 8085, chip these 16 lines are coming out. So, like this now this is my A0 to 15 that is coming. Now in my system total 64 K. So, maybe I have got say chips memory

chips of size 4 kilobyte only. So, how do I connect them? So, I will need 8 such chips to sorry I will need say suppose let us take a slightly sized chip suppose, I have got 8 kilobyte chips if I have got 8 kilobyte chips, then for 64 kilobyte I will require a 8 chips in that case ok.

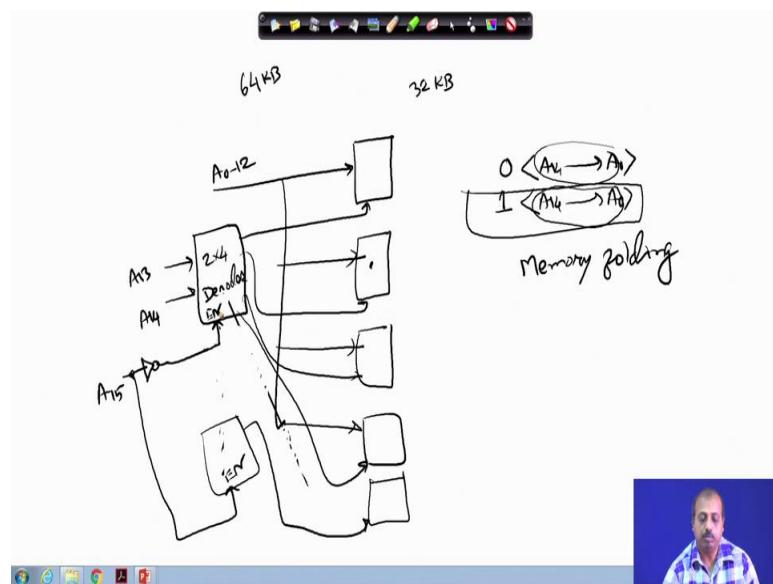
So, what I can do? So, I can, I can put 8 I can put 8 such chips in this way 8 such chips in this fashion and for 8 kilobyte. So, for 8 kilobyte the address lines needed is your $10+3=13$ bit. So, 13 bits are the address lines for this individual chips. So, I take this A0 to A12 to each of this chips to all this chips we put A0 to A12. So, those lines are going there. So, so, so this part is A0 to 12 and the bits 13, 14 and 15. So, they are fed to a 3 to 8 decoder they are fed to a 3 to 8 decoder and this output line. So, they will be selecting those lines the chips.

So, what I mean is that I have got a 3 to 8 decoder here, 3 to 8 decoder. So, the lines A13 lines A13, A14 and A15, they come here and this is the. So, this first line goes to the first line goes to the chip select of the first chip, then the next line will go to the chip select of the next chip. So, like that. So, what I mean is this first line it goes to the chip select of the first chip, the second line goes to the chip select of the second chip, third line goes to the chip select of the third chip. So, this way the last line will go to the chip select of the last chip.

Then what will happen; any address that is produced; so, if it is in the range of 0 to 8K first 8 K, then this chip will be selected and this data will be coming out of this chips. Now if I shot all these data lines together and feed it to the data line of the microprocessor feed, it to the data line of the microprocessor. Then this the location, content of the corresponding location will be available to the 8085 one to the data line.

Similarly, if the address generated is between 8K and 16 K. So, then this chip will get selected and this chip will provide the data ok. So, via this chip select line, so, it is ensuring that now what can happen is that many a times I do not need so many chips ok. So, that is I don't; so, in this case previous example.

(Refer Slide Time: 18:07)



The memory that we connected was also 64 kilobyte in size now it may so happen that I in my system, I need only 32 kilobyte, I do not need others, then if I got say 8 kilobyte chips. So, I will take this 4 such chips I will take 4 such chips and they will be getting the lines at this lines A0 to A12 as we have seen previously. So, they will be getting the line say 0 to A12. Now for differentiating between 4 chips; so, I will need a decoder which is a 2 to 4 decoder this is a 2 to 4 decoder. So, this decoder it will get the lines A13 and A14 from the microprocessor at this lines and then it will be generating the chip select signals for them, they will be generating the chip select signals for the individual chips memory chips.

Now, so, what happens for the address range for the addresses in the range which is which is more than this 32K; so, here it is implicitly assume that we have got this bit A15 is taken as 0 and this is the size A14 to A0. So, that will determine what we are going to which row which chip we are going to access.

Similarly, this A15 if this A15 is equal to 1, then also if you give some value. So, A14 to A0. Now suppose these 2 contents are same. So, this content and this content are same. So, you can understand that since this whole design that we have done, it is independent of the bit A15. So, in both the cases, it will access the same location. So, it will so, if the address range A14 to A0 is such that it falls into this chip, then in both the cases A15 equal to 0 and A15 equal to 1, it will be accessing the same chip.

So, this phenomena is known as memory folding. So, this is known as memory folding that is for different locations, so, you see different addresses it is accessing the same memory location.

So, this may or may not be desirable; so, in some system; so, if the memory folding is there. So, I cannot expand the system because now I cannot put further chips here because the pin A15 has not been used at all and if you can say that. So, how to stop this folding; so, folding can be stopped many a times what are consist that this 2 to 4 decoder. So, they have got some enable line as we know. So, it has got an enable line. So, if it has got some enable, then we can use that enable line to stop this phenomena ok. So, we can put an inverter and connect the A15 line here; So; that means, when this A15 equal to 0, then only this enable line will be equal to 1; as a result this decoder output is enabled other otherwise this decoder will be in a tri-stated state. So, it will not be selecting any of these outputs. All the outputs will be in 0 state.

Now, so, none of the chips will get enabled. Now if A15 equal to 0, then only this enable signal is high as a result appropriate memory location will be selected. So, this way; so, this is this address does not have any meaning in this case. So, if A15 equal to 1, then none of the memory chips will get selected.

So, that location is not within the addressable range generated by the processor in that case. So, that way memory folding can be resolved, but of course, if you are not bothered much, then you can definitely allow memory folding, because if you allow memory folding, then this extra logic gate this inverter and all are not required. Similarly, it may so happen that you can work with a simpler decoder that does not have any enable line in it.

So, you can directly use a chip which is a decoder, but does not use the enable line whereas, in this case you need to use a decoder where it has got a enable line if we are avoiding memory folding, then the advantage is that later on if you want to expand the system by putting another chip here, then you can have another decoder and for that decoder you take the enable line and connect it directly here, fine.

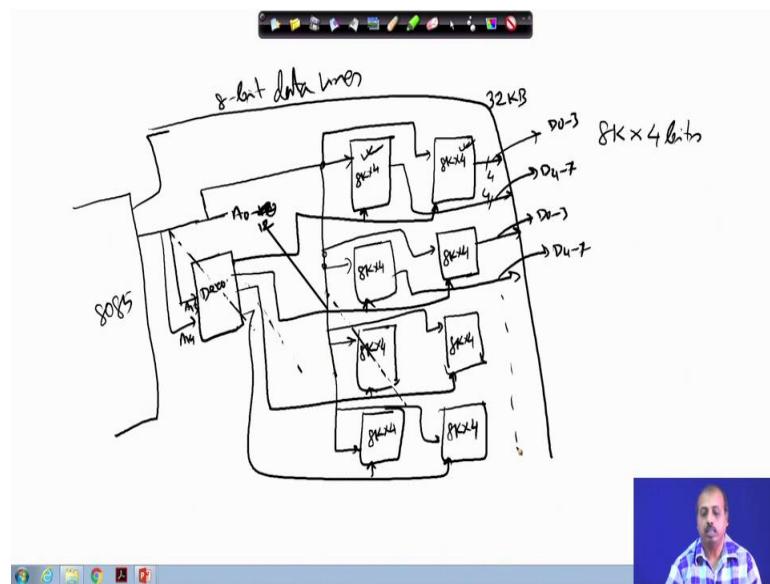
So and this chip, this decoder output say give chip select to the lower chip. So, that as a result what will happen is that this block will be selected only when A15 equal to one. So, you get the addition. So, you do not need to generate; you do not need to expand this decoding logic by putting, by putting a larger decoder, but you can just extend the design

you can just put another 2 to 4 decoder here and then by using this enable line. So, you can enable that decoder and the previous decoder. So, it will be valid for the upper 32 kilobyte and this decoder. So, this will be selecting the lower chip.

So, this way for memory expansion purpose it is better that we have got this memory folding resolved. So, that we can we can expand the system later that will not be possible, if we are not using memory folding. So, if we resolve it previously. So, if we have if we allow memory folding then of course, this cannot be done ok.

So, another possibility that we have is that this memory chips. So, it may not have the data line may not be sufficient for example, may be I will I am I require some memory of capacity say 32 kilobyte ok.

(Refer Slide Time: 24:36)



So, this individual chips that I have are say 8 kilobyte, it has got 8 kilobyte location, but 8 kilo locations, but individual locations are of 4 bits wide.

So, this we have seen previously that we have got 8 kilo and 4 bits. So, this is 8 kilo by 4 and this also. So, what you do in that case is you take 2 such chips in parallel ok. So, you have got this is another 8 kilo by 4. So, total 32 K. So, this is 8 kilo by 4 and this is another 8 kilo by 4. So, you take another such chips fine. And then from the processor side and so, this is say 8085, the address lines then the decoding and all. So, this address lines A0 to

so, 8 K; so, A0 to A12; so, they go to individual chips the; so, A0 to A12 is connected. So, you are making connections like that.

So, this address lines A0 to A12 are connected and then from the decoder if this is my decoder then we have got, So, this is A0 to A12. So, this is A13 and this is A14. So, they are. So, from this I get I generate the decoder signal and this chip select line goes to both the chips ok. So, these chips select line; it goes to both the chips which are in parallel. So, here this one this next chip select line, it goes to both the chips which are in parallel, third one goes to both the chips in parallel and the fourth one again goes to both the chips in parallel.

So, what happens is that if this decoder is selecting one chip. So, this is actually selecting a pair and then what we do if this is for the data bus, what we do is that this will give me 4 bits and this will give me 4 bits and they will be connected to the 8 bit output. So, this is suppose this is an 8 bit data bus date 8 bit data lines for the processor. So, first most significant 4 bits will be coming may be from come from this memory and the least significant 4 bits will be coming from this memory.

So, this lines; so, they will be connecting to the data line D0 to 3 and this lines, they will be connected to D4 to 7. So, everywhere it is like this. So, this is connecting D4 to 7 and this is connecting D0 to 3. So, this way we can make the other connection. So, this way you can make a memory system which is using the basic memory chips of different sizes as we as it is available and then we can connect them in some fashion by means of this decoding logic from the processor to get the full memory system implemented.

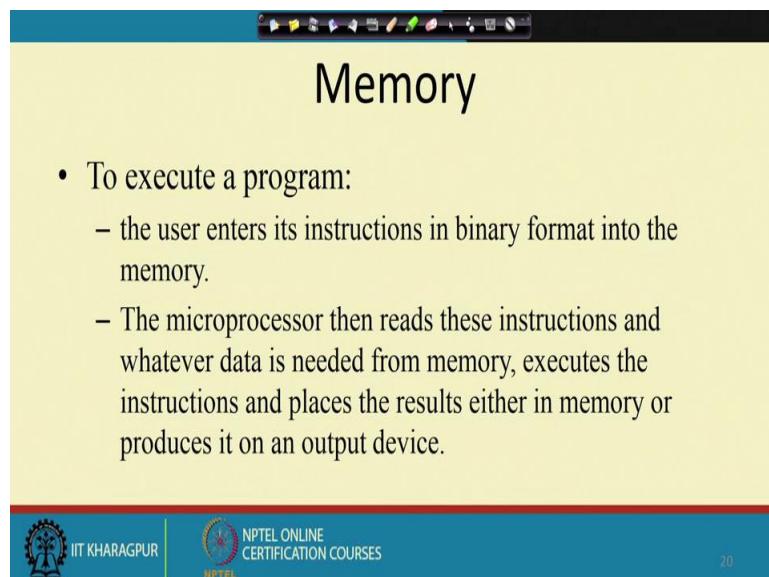
Digital Circuits
Prof. Shantanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 50
8085 Microprocessor (Contd.)

We have seen that in a computer system or in microprocessor based system, the microprocessor talks to the memory to get instructions to be executed, sometimes, it also gets the data from the memory or from the input output devices.

Now, as far as the user is concerned if the user wants to get some program executed.

(Refer Slide Time: 00:42)



The slide has a yellow background with a black header bar containing various icons. The title 'Memory' is centered in a large, bold, black font. Below the title is a bulleted list:

- To execute a program:
 - the user enters its instructions in binary format into the memory.
 - The microprocessor then reads these instructions and whatever data is needed from memory, executes the instructions and places the results either in memory or produces it on an output device.

At the bottom, there is a red footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, 'NPTEL ONLINE CERTIFICATION COURSES', and the number '20'.

So, the user has to enter the instructions that we have that the program has to in some binary format into the memory, the memory ultimately stores only binary information. So, the program that we have so that has to be coded into binary and this binary program has to be loaded into memory.

Now, when the microprocessor is reset; so, there is a set address from which it starts in case of 8085 that address happens to be 0. So, it accesses location 0 and reads the first instruction from there executes the first instruction; then it goes to the next instruction. So, that way it continues.

So, as you can understand that any computer system for starting at this 0 should have some monitor program or some operating system program, so, which will in turn transfer the control to the user program and user after the user program is over, then the control will be transferred back to the operating system. That is how the whole system works, but as far as the microprocessor is concerned, it reads instruction from the memory and if the data is needed from memory, so, it will read the data also from the memory; execute the instruction and then it will place the result back onto memory or onto the output device as the situation may be.

(Refer Slide Time: 02:03)

The slide has a title 'The three cycle instruction execution model'. Below the title is a bulleted list:

- To execute a program, the microprocessor “reads” each instruction from memory, “interprets” it, then “executes” it.
- To use the right names for the cycles:
 - The microprocessor fetches each instruction,
 - decodes it,
 - Then executes it.
- This sequence is continued until all instructions are performed.

A hand-drawn diagram is overlaid on the slide, showing three arrows pointing downwards from left to right, labeled 'Fetch', 'Decode', and 'Execute' respectively. A small arrow points from the text 'Then executes it.' to the 'Execute' arrow in the diagram.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a person speaking.

Now, in this execution process, you can understand that there are 3 stages, the first part of it when it reads the microprocessor reads the instruction from memory so, that is known as the fetch operation. So, it is fetching the instruction from memory. So, if the processor does not know whether it is an instruction or data or whether it is meaningful or not, but it will interpret that for the first access will give it the instruction. So, so that will be that that is called the fetch operation.

Now, after getting the first word from the memory; so, it may so happen that this processor will understand that this is the processor will understand that it needs some more part of the instruction or maybe more data from the memory. So, for that purpose it has to decode the instruction.

So, after decoding the processor understands what exactly has to be done whether it needs data from memory or not, whether it need some data from some input output port or not IO device, etcetera and then after getting those data it will execute it in the execution process may be some operation followed by storage of data; so, this entire thing comes as the execution.

So, you can say that the way the processor works is first it performs the fetch operation, first, it perform the fetch operation, then after fetching the instruction the processor enters into a phase which is known as the decode phase. So, from fetch, it comes to the decode phase and after decoding the instruction, it goes into execution of the instruction. So, this is the execute phase.

So, these are the 3 phases that we have in instruction execution, fetch, decode and execute. So, after the executive over; so, processor is done with the current instruction. So, you should get the next instruction from memory. So, what it does it now goes back to the, it now goes back to the previous next fetch operation to get the instruction from the next and to get the next instruction and execute it.

So, this way; this fetch, decode, execute it forms a cycle. So, if we look into any processor. So, it will have this fetch decode execute cycles ok. So, that the, that is the that they are actually in some sense we can say the that they are pipeline. So, fetch followed by decode followed by execute. So, in 8085; so, there is no overlapping of these stages, but if you look into later processors, you will find that there is an overlapping of these stages when the first instruction is fetched. So, no other instruction can be fetched at that time after that the first instruction goes to the decode stage and at that time the next instruction may be fetched.

(Refer Slide Time: 05:07)

The three cycle instruction execution model

- To execute a program, the microprocessor “reads” each instruction from memory, “interprets” it, then “executes” it.
- To use the right names for the cycles:
 - The microprocessor fetches each instruction,
 - decodes it,
 - Then executes it.
- This sequence is continued until all instructions are performed.

Fetch
Decode Fetch
Execute Decode Fetch
Execute Decode Execute

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then when the instruction the first instruction goes into the decode stage the first instruction goes into the decode stage the previously phased instruction. So, so, the when the first instruction the decode stage the next instruction can be in the fetch stage and when this first instruction goes to the execute phase; when it goes to the execute phase the next instruction, next instruction goes to the decode phase.

So, this goes to the decode phase then this instruction goes to the, second instruction goes to the execute phase and in the meantime the third instruction when the first you say second instruction was in the decode stage. So, the third instruction was being fetched, then the third instruction goes into the decode stage, then it goes to the execute stage.

So, that way there is an overlapping between this fetch, decode, execute. So, if you look into say this particular cycle, the first instruction will be executing, second instruction is in decode stage and third instruction is in the fetch stage. So, this is also known as fetch, decode, execute pipelining it is common in many of the processors.

Now, so, this sequence of this fetch, decode and execute it continues still all instructions are done. So, how do you know all instructions are done?

(Refer Slide Time: 06:35)

The three cycle instruction execution model

- To execute a program, the microprocessor “reads” each instruction from memory, “interprets” it, then “executes” it.
- To use the right names for the cycles:
 - The microprocessor fetches each instruction,
 - decodes it,
 - Then executes it.
- This sequence is continued until all instructions are performed.

HALT

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Like any processor it will have an instruction, so, which is, which tells the processor to halt; so, there is an instruction like halt. So, when this halt instruction is executed then the processor halts its execution. So, all the signals are in I can say inactive stage and so, to take the processor out of this halt stage special mechanism known as interrupt has to be used and when this interrupt is received, then the processor comes back to wake up mode and it will again start executing from some defined address ok.

So, this is the sequence of operation that happens in the execution of instructions.

(Refer Slide Time: 07:14)

Machine Language

- The number of bits that form the “word” of a microprocessor is fixed for that particular processor.
 - These bits define a maximum number of combinations.
- However, in most microprocessors, not all of these combinations are used.
 - Certain patterns are chosen and assigned specific meanings.
 - Each of these patterns forms an instruction for the microprocessor.
 - The complete set of patterns makes up the microprocessor’s machine language.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will see how this program can be written like binary program that we have talked about. So, how this can be developed; so, we will introduce the terminology called machine language. So, we are familiar with many programming languages like say C, C++, Java etcetera, but these languages they are for, to some extent human understanding. So, as a human being, so as a, if you read a C program you understand the meaning of that.

But a machine or a processor does not understand a program in that language. See it only understands the binary 0s and 1s ok. So, when you interpret the program that you have written in a high level language into a language it is understandable by the machine; so, this is called a machine language. So, in a, in case of a microprocessor the number of bits that form the word of a microprocessor is fixed for that particular processor. Word means in one access how many bits it gets or when it is doing some operations some arithmetic logic operation, so, what is the minimum number of bits on which it is operating.

In case of 8085; so, we will say that it is 8 bit processor because all the operations are 8 bit operations. So, if I say that my word size is say 8 bit; that means, when you get the first instruction when the processor is in fetch stage, it gets the first instruction and then the that 8 bit pattern it will determine what is the instruction. So naturally; so, this; so, with 8 bit, I can have at most 2^8 that is 256 combinations. So, these bits will define a maximum number of combinations that are possible for the machine instructions.

However, in most microprocessors all this not all these combinations are used. 256 is a very large number, but as far as this simple processors like 8085 is concerned; however, for complex processors that that number may not be that significant; however, for complex processors the word size is also large. So, it is 16 bit; so, that way your instructions of the number of possible combinations is also very high.

So, when the processor fetches a word from the memory, so, its type will determine the meaning of the state meaning of that word; so, certain pair. So, out of this all combinations that are possible all combinations are not used and it is up to the designer of the processor to choose certain pattern and assign themselves specific meaning.

Each of these patterns will form an instruction for the processor. So, so suppose I have got 256 patterns; now as a designer of a processor, I see that my processor will be executing 100 different instructions. So, I will be satisfied with 100 different codes. So, rest of the

codes are don't care for me because the processor will simply ignore those codes; so, this patterns they will. So, each of these 100 words; so, they are actually one instruction for the microprocessor. So, this complete set of patterns it will make the machine language of the microprocessor. So, this is called the machine language. So, the language it is understood by the machine.

(Refer Slide Time: 10:39)

The 8085 Machine Language

- The 8085 (from Intel) is an 8-bit microprocessor.
 - The 8085 uses a total of 246 bit patterns to form its instruction set.
 - These 246 patterns represent only 74 instructions.
 - Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.
 - For example, the combination 0011 1100 which translates into “increment the number in the register called the accumulator”, is usually entered as 3C.

8085 → mem $2^8 = 256$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Let us look into 8085 machine language. So, in, as we have said that 8085 is an 8 bit microprocessor; so, 8 bit microprocessor as I have said that it is the external data bus side is 8 bit and internally when it is doing the operations. So, all the operations are on 8 bit data; so, with 8 bit. So, we can have 2^8 that is 256 different combinations are possible ok. So, when this processor, if this is 8085 and this is your memory then as I said that if the at the very, very beginning, it does a fetch operation and gets 1 byte from the 1 word that is equal to 8 bit from the memory.

Now, this with is 8 bit I can have 256 different patterns possible, but in case of 8085 out of this 256 patterns only 246 patterns are used. So, remaining 10 patterns are not used by the processor. So, it uses a total of 246 different bit patterns to form the instruction set and these 246 patterns, they represent 74 instructions.

So, 74 types of instructions; so, we will see that within the same instruction they are may be variants. So, that is why for 74 instructions, we will need 246 different bit patterns. So, it is very difficult now as a user of the microprocessor systems. So, if I am asked to write

the program in this binary number system the binary pair bit pattern, then it is very cumbersome ok.

So, normally we can use some hexadecimal instead of binary, for example, if the bit pattern that I want to have that I have to enter is like this 0 0 1 1 1 1 0 0 which for in case of 8085 this corresponds to the instruction that implement the number of number in the register called accumulator. So, this 8085 has got a special register accumulator and this particular bit pattern to the processor, it we will mean that it asks for incrementing the accumulator register. So, we will see in detail after some time.

So, instead of entering the data as 0 0 1 1 1 1 0 0; so, we can break it into hexadecimal digits and this is 3 and this is C; so, 3 C. So, if we have got a hexadecimal keypad; so, we can enter this 2 digits very, very easily 3 C ok. So, this way it can be done.

(Refer Slide Time: 13:18)

Assembly Language

- Entering the instructions using hexadecimal is quite easier than entering the binary combinations.
 - However, it still is difficult to understand what a program written in hexadecimal does.
 - So, each company defines a symbolic code for the instructions.
 - These codes are called “mnenomics”.
 - The mnemonic for each instruction is usually a group of letters that suggest the operation performed.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now, this hexadecimal type of entry so that is also not very much comfortable for the users because then if the, so, the as so, if you if you look into a program that is written in this hexadecimal codes then naturally it looks very cryptic. So, as a reader of the program, **we** **it** we do not understand the meaning. Until and unless for each instruction we look into the corresponding meaning from the manual ok. So, the so, we need to bring some amount of readability of the program even in this machine language version ok.

So, that gives rise to another level of language which is known as the assembly language. So, entering instruction in hexadecimal is easier than entering in binary combination; however, it is still difficult to understand what a program written in hexadecimal does because I need to interpret each and every hexadecimal number that I have entered.

So, what has been done is that different companies have defined a symbolic code for the instruction.

(Refer Slide Time: 14:30)

The 8085 Machine Language

- The 8085 (from Intel) is an 8-bit microprocessor.
 - The 8085 uses a total of 246 bit patterns to form its instruction set.
 - These 246 patterns represent only 74 instructions.
 - Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.
 - For example, the combination 0011 1100 which translates into “increment the number in the register called the accumulator”, is usually entered as 3C.

INR A

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, as I said that previously we have seen this instruction that implements the number of a number in the register called accumulator. So, instead of writing this big statement it may be simply implemented as say INR. So, it may be simply INR and this accumulator is A. So, this is INR A; so, by this text, we understand that it is asking for implementing the register A.

So, if here you see. So, this is; so, it is easier to say instead of hexadecimal number. So, we can have some symbolic code for the instruction and these codes are called mnemonics. For example, that INR A the INR part is known as the mnemonic and A is the operand on which that mnemonic works. So, as a designer of the processor; so, we can think about different mnemonics that are possible that we will use with those mnemonics. So, we can use as a user of the system, we will know that those mnemonics and write programs using those mnemonics.

A mnemonic for each instruction is usually a group of letters that suggest the operation to be performed. So, this is as I said that this is INR. So, it is a combination of I, N and R. Almost all the processors will have some mnemonic called MOV, MOV. So, these are quite common. So, we will see as we proceed into the assembly language of 8085.

(Refer Slide Time: 16:00)

The slide has a yellow background with a blue header bar at the top containing various icons. The title 'Assembly Language' is written in a large serif font. To the right of the title, there is handwritten-style text 'LCS'. Below the title is a bulleted list:

- Using the same example from before,
 - 00111100 translates to 3C in hexadecimal (OPCODE)
 - Its mnemonic is: "INR A".
 - INR stands for "increment register" and A is short for accumulator.
- Another example is: 1000 0000,
 - Which translates to 80 in hexadecimal.
 - Its mnemonic is "ADD B". $\Rightarrow A \leftarrow A + B$
 - "Add register B to the accumulator and keep the result in the accumulator".

At the bottom of the slide is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it says 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a small video frame showing a man with a mustache speaking.

So, as I said that this 0 0 1 1 1 1 0 0. So, this is 3 C in hexadecimal. So, this hexadecimal code is called the OPCODE or the operation code. So, this actually tells which operation has to be done; so, that is why it is called OPCODE. So, this bit pattern or this hexadecimal code; so, this is called OPCODE and the mnemonic is INR A where INR is the actual mnemonic part and A is the operand ok. So, A is the short hand for accumulator. So, instead of INR A; so, it may be some other register also. So, that is the incrementing the, that particular register. So, INR part is the mnemonic and this A is the operand.

Take another example, suppose we have a bit pattern 1 0 0 0 0 0 0 0. So, this is a 8 0 in hexadecimal and if you look into the manual of 8085, you will find that this particular code corresponds to the mnemonic ADD B ok. So, the where ADD is actually the mnemonic and B is the operand. So, this whole thing is an instruction in the assembly language. So, what it does this is an instruction that tells the 8085 microprocessor that the user wants to add the register B to the accumulator and keep the result in the accumulator only.

So, this is what the user wants is something like this the accumulator A should get the content of register B added to it so A gets A+B. So, so, this is the there. So, this why the assembly language programs they actually help in understanding the meaning of the machine language program to the users not to the processors.

(Refer Slide Time: 17:40)

The slide has a title 'Assembly Language' at the top. Below the title is a bulleted list:

- It is important to remember that a machine language and its associated assembly language are completely machine dependent.
- For example, Motorola has an 8-bit microprocessor called the 6800.
 - The 8085 machine language is very different from that of the 6800. So is the assembly language.
 - A program written for the 8085 cannot be executed on the 6800 and vice versa.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. To the right of the footer, there is a small video window showing a man speaking.

So, processor is happy with the machine language code in some binary format, but for human understanding, so, this is in assembly language. So, it is important to remember that a machine language and its associated assembly language are completely machine dependent.

So, if you learn the machine language if you learn the assembly language of one processor it is not going to be same as the assembly language or another processor. So, because this is done by the processor designers and they do it at their, at their own effort you can say or at their own wish ok. So, it is very much unlikely that the two processors will have exactly one exactly same assembly language. There will definitely be some difference as, because the assembly language is very close to the actual hardware. So, since the actual hardware between 2 processors are varying. So, this assembly language is also going to vary.

So, that is why; so, normally we learn programming in high level language and then use some tool by which the program is translated onto say machine language or it can also be translated to assembly language. So, if you look into say any C compiler normally this C

compiler, they do have an option by which it can tail it to keep the assembly version of the translated program. So, it actually from your high level language program it generates the assembly language program and from the assembly language program, it generates the machine code.

So, you can tell the processor we can tell the compiler to keep the assembly language code available as a separate file. So, so, they; so, as a, as a if we are expert in the assembly language of that processor, then we can look into that code and possibly try to optimize that code further.

So, if you look into say Motorola's 8 bit processor 6800. So, this is also 8085 is from INTEL and this 6800 is from Motorola. So, this 8085s assembly language is machine language is very different from that of 6800 and. So, is the assembly language. So, they are different. So, program written for 8085 cannot be executed on 6800 or vice versa. So, that is true.

So, it is so, the target has to be specified like if you are installing any compiler, you must be knowing that it asks for the target processor for which the code has to be generated. So, this is because of this reason that the machine language of 2 different processors are totally different. So, something targeted to one processor will not work with the other processor.

(Refer Slide Time: 20:33)

“Assembling” The Program

- How does assembly language get translated into machine language?
 - There are two ways:
 - 1st there is “**hand assembly**”.
 - The programmer translates each assembly language instruction into its equivalent hexadecimal code (machine language). Then the hexadecimal code is entered into memory.
 - The other possibility is a program called an “**assembler**”, which does the translation automatically.

The diagram illustrates the process of translating C code into assembly code. It shows a box labeled 'C' with an arrow pointing down to a box labeled 'Compiler'. From the 'Compiler' box, an arrow points down to a box labeled 'asm'. From the 'asm' box, an arrow points down to a box labeled 'Assembler'. Finally, an arrow points down from the 'Assembler' box to a box labeled 'obj code'.

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

A small video window in the bottom right corner shows a man with glasses and a white shirt, likely the professor or speaker for the course.

Now, as I said that from the assembly language program is for user understanding. So, machine or the processor does not understand it. So, what we have to do is we need to translate this assembly language program into machine language program. So, how do we do it? First we can do an hand assembly. So, we look into the manual of the processor and then for every instruction that we have in the assembly language. So, we just see; what is the corresponding instruction we corresponding machine language code.

So, programmer will translate each assembly language instruction into its equivalent hexadecimal code. And then this hexadecimal code values will be entered by some keypad and ultimately the when it is entered that software which is getting the data from the keyboard will finally, store the corresponding binary pattern into the memory.

So, this is the hand assembly process for; so, for simple processors like 8085, it is possible that we do an hand assembly and get the program get the assembly language program translated into machine language. So other option is that we have an assembler; so, assembler is a tool that can translate assembly language program into machine language program. So, this assembler can be used that will translate the assembly language code into machine language.

Now, as I was telling; so, if I have got some high level language program say I have got a C program .C program. So, it passes through a compiler now this compiler it generate some assembly language program. So, dot let us say the extension is .asm and then this assembly language program it passes through an assembler, this passes through an assembler and it generates the machine code. So, this is the machine code.

So, now if you if you can write directly in the assembly language. So, you do not need this part. So, you can start straight way at this point and then you can generate the, you can run the assembler and get the machine code. So, normally when we are doing this translation this C compiler. So, it the, it integrates the assembly with it. So, we see this whole, whole thing as a as the compilation process see this whole thing as a compilation process that is why you do not see the assembler code.

However you can you can tell the compiler to keep the assembly file .asm files for your view viewing and doing some further optimizations. So, so that is the so, that is done by the automated tool called assembler.

(Refer Slide Time: 23:43)

8085 Microprocessor Architecture

- 8-bit general purpose μ p
- Capable of addressing 64 k of memory
- Has 40 pins
- Requires +5 v power supply
- Can operate with 3 MHz clock
- 8085 upward compatible

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next so, next we will be looking into this 8085 microprocessor architecture. So, this assemblers are definitely, it is targeted to different processors. So, if you are looking for 8085 will target it to 8085 otherwise you can tell that it is for other processor. So, it will do it accordingly.

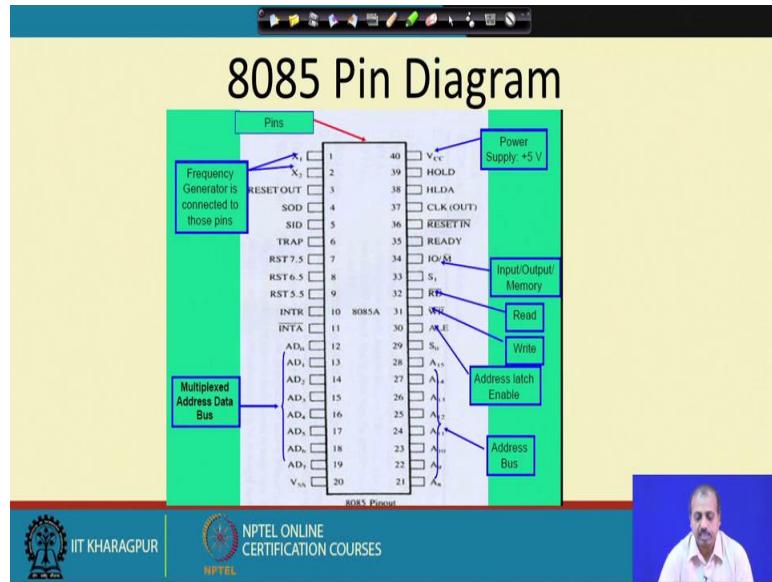
Now, in case of 8085 as I said that it is an 8 bit general purpose microprocessor. So, general purpose microprocessor means it is not doing some special operation, for example, if you are doing some signal processing tasks. So, there are digital signal processors that are used for that then for graphics processing we have got some special graphics processors. So, like that, but 8085 as a processor, so, it is a general purpose processor. So, general; so, it can do the basic arithmetic operation logic operations like that. So, it is not specific for a particular application.

It is capable of addressing 64 kilobyte of memory. So, this is. So, total memory space that it can address is 64 kilobyte. So, with 8085 system, so, you can connect memory up to 64 K and if you want to connect it is not that you cannot connect beyond this, but for that purpose. So, we have to take some extra hardware for the connection, but up to 64K. So, it can be directly interfaced with 8085.

So, as since 8085 comes as an IC chip. So, it has the chip has got forty pins in it the power supply requirement is +5 volt. So, the clock frequency it can operate is with 3 megahertz and it is upward compatible. So, upward compatible means that if you have got some older

versions of some processors then this is so, that that you can just take up that chip and put this chip into the, into the system. So, that is there. So, that is upward compatibility is maintained.

(Refer Slide Time: 25:44)



So, as far as the 8085 pin diagram is concerned. So, here it is like this. So, it is a forty pin chip as I have said. So, pin numbers are 1 to 40 out of that. So, pin numbers one and two. So, they are called X₁ and X₂. So, this is a frequency generator is connected to those pins. So, the so, normally what we do is that we connect a crystal between this 1 and 2 and there that crystal frequency will determine the frequency at with this 8085 will work. So, that the frequency generated is connected to this 1 and 2

Then another important client is V_{CC}. So, this is the power supply +5 volt and there will be there is a ground line there is a ground line. So, which will be giving as the ground. So, this V_{SS} this is the ground line.

So, other important pins; if you want to see, then it has got this address bus. So, this is A₈ to A₁₅. So, this is the higher, so, this is this is a higher order address bus and these pins AD₀ to 7. So, AD 0 to 7, so, that will constitute the lower order address bus. So, as I said that 8085 memory space that it can address is 64 K. So, for 64K, we need 16 bit bus ok. So, 16 because 2^{16} is 64 K. So, 16 bit is obtained like this. So, here I have got a bits A₁₅ to A₈ and here I have got pins A₇ to A₀ ok. So, A 7 is this one pin number 19 and A₀ is this one pin 12.

And we have got say 8 it is an 8 bit processor. So, whenever it accesses data from outside world. So, it is in terms of 8 bits. So, that 8 bit data bus is this D0 to D7. So, this is D0. So, this is D7. So, this address 0 and data 0. So, these 2 pins are same similarly address 1 and data 1. So, these 2 pins are same; so, this is called; so, we will see that to reduce the number of pin count. So, it has been done like this otherwise what would have happened is that the 16 bit address line. So, that is 16 pin and 8 bit data line that is 8 pin. So, $16 + 8 = 24$ pins would have already been done just to create this address and data line. So, we will not have much other pins left for other operations ok.

So, to just; so, the designers they have done some multiplexing of this address and data buses. So, that the number of pins required is less. Now there are some additional pins that we will see slowly as you proceed through this course, but some important things like this is the *read* signal. So, this is activated when the processor wants to read something from the memory.

Similarly, *write*. So, this is activated when the processor wants to write something onto the memory then there is S0, S1. So, this means. So, that gives me the status line. So, there are actually 3 pins the IO/ \bar{M} , S1 and S 0. So, they tells what the processor is doing at present. So, if you just want to know probe the processor and see; what is it doing. So, this IO/ \bar{M} S0 and S 1; so, this will tell you some and this will give you some indication about what the processor is doing now.

Apart from that; so, there are some interrupt lines INTR and this line TRAP, RST, this 5.5; 6.5; 7.5. So, they are actually some facility by which you can interrupt the normal operation of the processor and tell it to do something special ok. So, this is that, this is for that interrupt. So, we have got this another important pin that we have is through this SID and SOD. So, this is serial input data and serial output data. So, if you want to transmit some bits serially from this 8085. So, you can do it where this SID, SOD pins ok.

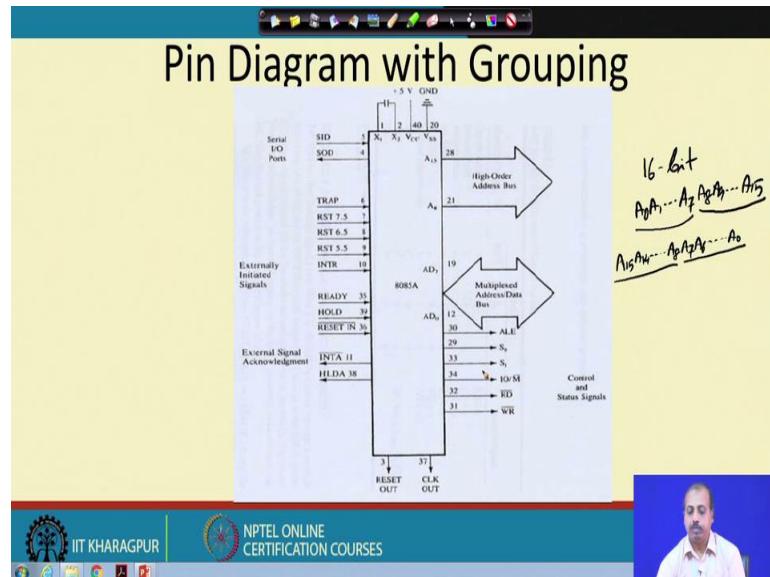
So, as we proceed further, we will be looking into more and more details of this individual pins. So, like say, we will see, we will we will be actually viewing these pins in terms of functional groups that is the functions that they are doing and we will try to group them together.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture -51
8085 Microprocessor
(Contd.)

So, 8085 pins, so if you look in terms of some grouping of them that is the related pins we keep them together; so, the pins which are doing a particular function.

(Refer Slide Time: 00:29)



So, if you see this in the first group of pins that we that will find corresponding to the address bus, and as I said that my address bus in case of 8085, so this is 16 bit. So, they are named, so this is A0 A1 A7 A8 A9 up to A15. So, out of that this A0 to A7 see this is lower order address bus consists of lower order 8 bits and A8 to A15, so, this is the higher order address bus or higher order address bit.

So, we should better, if I write this most significant bit on the left side, so I should write it like this ok. So, A7 A6 up to A0, so this is the higher order bus, higher order address bits this is the lower order address bits. So, this higher order address bits they are coming from the higher order address bus.

So, pin number 21 to 28 they form this higher order address bus and they are this A 15 to A8. So, A15 is pin number 28 and A8 is pin number 21 now this, the lower order address bus. So, A7 to A0, so that is multiplexed with the data bus so this they are so this is allocated pin number 19 to 12 ok, so this 8 bits. So, they are giving me address sometimes and also sometimes it will act as the data bus.

So, this pins 12 to 19 sometimes it will act as lower order address bus, sometimes it will act as the data bus. Now, the address bus is going out from the processor to the memory and the data bus maybe bidirectional. So, as a result this higher order address bus is going out of the memory and this lower order address bus which is multiplexed with the data bus, so this is bidirectional ok. So, apart from that we have got some other important pins like this ALE signals. So, this ALE signal will be used to tell the content of this multiplexed address data bus.

So, when this ALE signal is high then the processor has got the address put on to this multiplex bus, and when the ALE signal is low then this bus may be used as the data bus. So, when the address is put this ALE signal is active, so when address is not there ALE is de active. So, it may in that times this data this is treated as a data bus. S0 S1 IO/ \bar{M} I have already said that they are giving me the status of what the processor is doing now, then $\overline{\text{read}}$, $\overline{\text{write}}$. So, they are actually telling what operation the processor is doing is a memory read operation or a memory write operation. And IO/ \bar{M} so, this is actually is the pin that tells the operation that the processor is doing is it accessing a memory location or it is accessing some IO device.

So, if it is accessing an IO device, so in that case this IO/ \bar{M} line will be equal to 1 and if it is accessing a memory location then the IO/ \bar{M} line will be equal to 0. So, if IO/ \bar{M} is 0 and this $\overline{\text{read}}$ is also 0, that means the overall the processor is doing a memory read operation. Similarly, if IO/ \bar{M} is equal to 1 and this $\overline{\text{write}}$ is equal to 0 that means processor is doing some IO write operations. So, it is doing write operation on some output device it accessing some output device.

So, this is way this IO/ \bar{M} , $\overline{\text{read}}$, $\overline{\text{write}}$ and S0, S1 etcetera. So, this S0 S1we will see later. So, that gives us some in indication about what the processor is doing at present then we have got some additional pins like SID SOD lines I have already said so, they are for serial input data and serial output data. So, if you, so this 8085 it has got the feature by which it

can get data from outside world serially. So, this the SID pin and it can also output some data to the outside world by the SOD pin. So, that is serially the data will go, there are some ways by which you can interrupt the operation of this processor.

So, that is by means of this line TRAP, RST 7.5, 6.5, 5.5 and INTR. So, you look into this interrupt issues later, but they are actually to tell the processor to suspend the current operation whatever it is doing and do some special operation for the system. So, it may be that some something special has happened like user has pressed some key. So, that has to be acknowledged that also that value has to be read and all, or maybe there is some emergency situation has occurred some fire alarm has gone up. So, that way that has to be detected and that has to be taken, some special action has to be taken.

So, these are the interrupts that are coming then you what about the processor is doing. So, you can give it a reset signal by pin number 36 to activating this pin number 36 will reset the 8085 processor and so, whatever operation it was doing abort that operation the whole system will be reset. And when this RESET IN is activated, so this 8085 apart from resetting its internal registers and all, so, it will also send output high on the reset out line. So, if my system consists of the 8085 processor and number of other devices. So, those devices can also be reset by this reset out pin.

Now, for getting the clock to the system, so we have to connect some crystal between pins 1 and 2 so crystal is connected. So, accordingly the 8085 generates the clock and through this pin number 37 this clock is also outputted. So, if you, if you want to use this clock to some to some other circuitry. So, you can do that in using this pin number 37. So, most of the lines we have covered. Another important line that we have used hold and hold acknowledge. So, hold is actually telling the processor that, so that there is some other processor in the system like what happens is that if you connect says 2 such 8085 chips in a system and they are accessing the same memory ok.

So, at some time this address and data bus will be driven by the first 8085 chip, sometimes it will be done by the second 8085 chip. So, if there is no coordination between the two then they will be accessing try to access simultaneously on the data corruption will occur. So, what is done is that we have these hold pin. So, if whenever one processor wants to access the bus, so, it will say it will give a hold to the other processor and as a result is other processor will generate hold acknowledge. And after getting the hold acknowledge

only the first processor will understand that ok, the other processor has released this address bus and data bus for my usage and then it will proceed to use those address data bus lines. So, this way we can have multiple bus masters connected in a system and they will be, that will be controlled by these hold and hold acknowledge lines ok.

(Refer Slide Time: 07:54)

System Bus

- System Bus – wires connecting memory & I/O to microprocessor
 - Address Bus
 - Unidirectional
 - Identifying peripheral or memory location
 - Data Bus
 - Bidirectional
 - Transferring data
 - Control Bus
 - Synchronization signals
 - Timing signals
 - Control signal

IIT Kharagpur | NPTEL Online Certification Courses

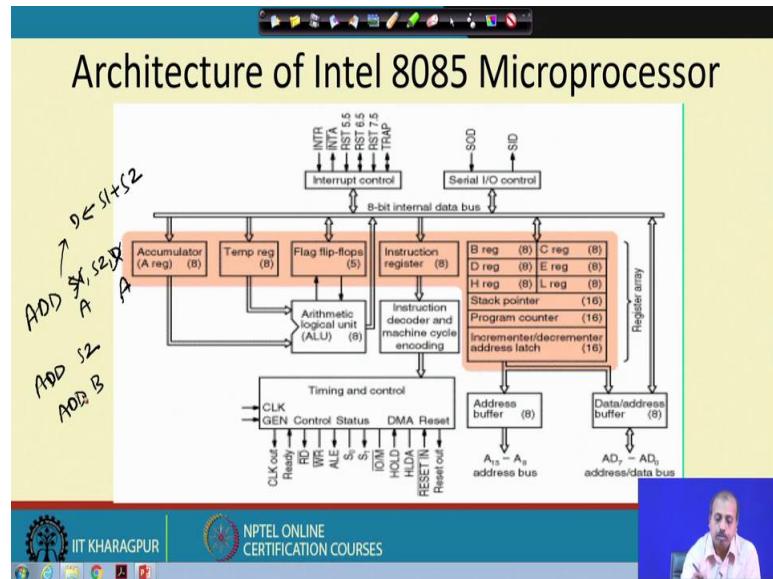
So, coming to the system bus, so this system bus, so these are the wires connecting memory and IO to the processor. So, you remember the diagram that we have shown previously, so if this is your CPU. So, we have got this memory and IO, so this is my system bus. So, CPU this memory and this IO they all hang from the system bus so, the wires connecting memory and IO to the microprocessor.

So, this is our system bus. The system bus so though I have shown it as a single bus, so, in reality it consists of three different buses address bus, data bus and control bus. So, again if I look into this memory and CPU connection in more detail so, this is my 8085 processor and this is the memory. Now the address lines, so they will go from 8085 to the memory chips, so this is these are the address line. So, address line is going to be unidirectional. So, this will also go to some IO devices this address lines also go to the IO devices to tell which IO devices which IO device it wants to access, so if it 8085 wants to access memory, so, it will be giving address for memory location if it wants to access some IO device it will give the address to the IO device.

Now address bus is unidirectional it goes out of 8085 and goes to the IO device for memory. And it will be used to identify the memory location or the IO device location now there is data bus. So, data bus is bidirectional because I can have transfer from 8085 to memory or from memory to 8085 to read and write both operations are possible that the, so, this is going to be a bidirectional one so it is also connected to the IO devices, so this is bidirectional connection.

So, this is data bus is bidirectional and for transferring data and we have got this the control bus consisting of other signal control signal, synchronisation signals like that. For example, in case of 8085, so, we have got a number of signals like say the read line, the write line ok. So, all these lines that we have so they are connected like this. So, there may be more such synchronisation lines as we look into more detail of this design, so will see. So, this part to this part is known as the control bus. So, this is called control bus. So, this the first upper one is address bus and this one is the data bus. So, we have got address bus, data bus and control bus which constitutes the system bus.

(Refer Slide Time: 11:05)



Okay, so, let us look something inside, somewhat inside the 8085 microprocessor, how does it look like? So, this 8085 microprocessor if you look into so first you will find the number of registers ok. So, there is a special register called accumulator. So, which is as the name suggests it accumulates the value. So, whenever you are doing some arithmetic

or logic operation in the processor, then one of the operands of that operation must be the A register and the result is also stored in the A register.

So, if suppose I am doing an addition operation, then this addition operation one of the operand must be in the A register and the other operand maybe in some other register or it may be some memory location, but the result will again be stored in the A register. So, the advantage that it gives is that if I am doing some addition operation, then you see there are three things to be mentioned the source 1, source 2 and destination fine. So, source 1 and source 2 are two registers, now our memory location and ultimate operation to be done is this D gets the value of $S_1 + S_2$; Now, if do this then I need to specify three different operands S_1 S_2 and D.

So, in case of 8085 the designers they have said that S_1 must be A and this destination must be A ok. So, this two are fixed so this two are accumulator. So, the instruction simplifies to ADD S_2 . So, it may be for example, say ADD B, so that will add the B register content with the accumulator to get the value, now here you see that we have got this accumulator is A. So, these are special register so its output goes to the arithmetic logic unit.

And there is a temporary register, so, where it holds the other operand for this arithmetic logic operations so whenever you are doing say ADD B. So what happens is that it is temporary register gets the content of B register and after that it is the operation is done by this arithmetic logic unit. So, this arithmetic logic unit as we know that is purely combinational unit, combinational circuit. So, if its inputs are fluctuating then the result will also fluctuate. So, if it is coming from some register then definitely the inputs will not fluctuate. So, whenever it is giving it is told that you do the operation of addition, so, it will this arithmetic logic unit will do that. So this is the temporary register.

Then there is a set of flip flop or called flag flip flop. So, they are actually telling the status of the last operation that last arithmetic logic operation that has been done. So, it may, so happen that after doing the operation. So, there was an overflow generated, so there is a particular bit in the flag register which will be set or it may so happen that the result becomes 0 ok.

So, in that case some again some flag bit flag flip flop is there which will be set and it will tell you like what is the content of , what, what was the status of the last operation, so that

way it is used ok. So, these are the, these are the special register plus there is set of general registers like this B, C, D, E, H and L. So, these are 6 registers that we have, so all these registers are 8 bit registers. So, they can be used as operands for these arithmetic logic operations. As I said that add B. So, when say add B, I am actually referring to this B register ok. So, that way I can have add C, add E like that ok.

So, they are these are the B, C, D, and H, L, so these are some registers. Now there are some special registers like there is something called an instruction register. So, this instruction register, so this is actually holding the instruction obtained from the memory. So, as I said that every, so when this 8085 starts, so, in the first in the first instance it gets the first instruction from the memory. So, after getting the first instructions, so it needs to decode it. So, before decoding the instruction is when it is fetched from the outside world. So, it is loaded into this instruction register.

So, in the successive phase this instruction register content it goes to this decoder, where it performs this instruction decoding and this instruction decoder output. So, that that actually identifies the instruction that it is going to do and then it is given to these timing and control module. So, which will generate all the control signals that will be needed at various time instants to do the operation. So, will see some timing diagrams latter which will tell you like how this is a control signals are to be generated and then.

So, this timing and control unit, so this generates the different control signal based upon this clock circuitry that we have. So, there was this clock signal that we are getting, now apart from that there are some special registers like there is a program counter. So, this program counter it holds the memory address from where the next instruction will be executed. So, as I said that when you reset the processor then this 8085 comes to a special address. And in case of this 8085 this address happens to be 0 the address location 0.

So, if you reset the processor then this program counter gets the value 0. So, that it will be accessing the next instruction from location 0. Now, as the first instruction is fetched this program counter value is automatically implemented and it points to the next instruction to be fetched. So, when the processor goes to the next fetch cycle, again the program counter output is put on to this address bus. This, this is higher order address bus and this is lower order address bus. So, this program counter content come to this address bus. And then it is, the instruction is the fetched from the memory and that comes again to the

instruction register that goes into this decoding and the decoder will be generating will give in information to this timing and control that will give control for the execution.

So, after this is over again this program counter output is put on to this address buffer and address data buffer. So, that way it will go, so this program counter is a special register who which tells like what is the next instruction the address what is the address of the next instruction to be executed. Now for this program counter, so, we have got incremental and decremental address latch. So, sometimes, so automatically with the address register in the program counter content has to be incremented or decremented.

So, that is done by this one this address latch. So, is doing the increment decrement and there is a stack pointer. So, will come to its usage latter that is there are some special operations that are needed to be done in the in any processor for storing the return address from subroutines and the stack pointer is used for that purpose, so will come to that later.

So, apart from that, so you will find a module which is known as the interrupt control. So, this interrupt control actually handles all this interrupt plans ok, accordingly it will tell the process have to suspend current operation do something else and that will be coming to this timing and control unit also to tell to tell it like what to do. And there is a serial IO control input control line SID and SOD. So, that module will be doing this serial communication. So, this is more or less the internal view of 8085 processor.

So as a user of the system, so, we know that okay there is a special register A and there are some general purpose register B, C, D, E and H, L. So, these are the 7, these are the 7 register that we have in our repository. So, I can use these registers for writing programs for the processor. And apart from that there are some flip flops which are the status flags and from the install from the manual of the processor, so, will know what are the arithmetic logic operations that the processer can do and accordingly we can that that will tell you, what is the functionality of the ALU.

(Refer Slide Time: 19:43)

Intel 8085 Microprocessor

- Microprocessor consists of:
 - **Control unit**: control microprocessor operations.
 - **ALU**: performs data processing function.
 - **Registers**: provide storage internal to CPU.
 - **Interrupts**
 - **Internal data bus**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, to summarise this 8085 microprocessor it consists of control unit ALU registers interrupts and internal data bus. So, control unit it will control the processor operation. ALU will perform a data processing function. Then this registers they provide storage internal to the CPU. So, this registers like if you have got some data.

So, normal way to have the data is that have it in the memory from there fetch it, but normally is in case of 8085 this memory happens to be a chip which is outside which is the memory is a location which is in a chip outside the processor. So, to access it, so it is much slower compared to on chip axis line. So, if you have if you are having communication within the chip. So, that is much, much faster than some communication which is off chip.

So, if I know that some of the variables are needed very often in my program. So, I may decide that I will put it in one of those registers B, C, D, E, H, L and use that. So, that for that particular variable I don't need to access the memory. So, that way the access will be faster. So, this registers they can provides storage internal to the CPU and there are interrupt facilities and there is internal data bus. So, these are the major component that we have in the 8085 processor.

(Refer Slide Time: 21:12)

The ALU

- In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.
- Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, we will start with the ALU so, this ALU as the name suggests arithmetic logic unit. So, this is doing the arithmetic and logic operation. So, apart, apart from this logic operation so, it also has the accumulator which is part of every arithmetic logic operation though in the diagram we have shown this the accumulator separately. So, for all practical purposes, so you can think that the accumulator is a part of the ALU because ALU always gets one of the operands from the accumulator and it produces result on to the accumulator and it also has a temporary register.

So, like in this diagram you see that we have got this ALU we have got the accumulator and the temporary register. So, they are very much integrated with the ALU. So, for all practical purposes I can say that these three units are together for and becoming an integrated part of the arithmetic logic unit now internally how is it implemented, so that is not known.

So, that is not divulged by 8085 designers, but conceptually it, we can say that they are all part of the accumulator ok. So, this ALU it holds a temporary register for holding the data temporarily for during execution of the program and this temporary registered is not accessible by the programmers. So, as per as the programmer is concerned there is no such temporary register available. So, it has got the register A, it has got the registers B, C, D, E, H, L.

(Refer Slide Time: 22:45)

Registers

- General Purpose Registers
 - B, C, D, E, H & L (8 bit registers)
 - Can be used singly
 - Or can be used as 16 bit register pairs
 - BC, DE, HL
 - H & L can be used as a data pointer (holds memory address)
- Special Purpose Registers
 - Accumulator (8 bit register)
 - Store 8 bit data
 - Store the result of an operation

The diagram illustrates the 8085 CPU registers. It shows a 4x2 grid of registers labeled A through F. The first column contains the Accumulator (A), B, D, and H. The second column contains the Flags (F), C, E, and L. Below this grid are the Program Counter (PC) and Stack Pointer (SP). An 'Address' bus is shown as a 16-bit width below the registers, and a 'Data' bus is shown as an 8-bit width to the right. Handwritten annotations include 'int *p' above the PC, 'int *p = 10' above the SP, and '[H] p = 10' with arrows indicating memory access.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, if you look into the registers. So, registers can broadly we classified into two categories one is called general purpose register and other is called special purpose register. So, general purpose registers, so we have got the registers B, C, D, E, H and L. So, these are the general purpose registers. So, they can be used singly or as a 16 bit pairs like you can say that also, so you can sometimes.

So, this for example, this BC pair, so BC pair so you can load some value want to the B register or you can load some value on to the C register. So, you can load some values there or you can say that I will load a value into this 16 bit pair ok. So, this is treated as a 16 bit registers B and C taken together considered as a 16 bit register, so that is possible.

So, for pairing, so the pairing is in terms of B, C, D, E and H, L. So, will see some instructions later that will show you how to access this B, C, D, E, H, L as 16 bit pair. And out of that pairs it is H and L, they are also used as data pointer. Now, pointer you know that that is the very interesting data structure that is there in many programming languages. So, where we do indirect addressing, so in case of say for example, the language like C. So, I can define an integer pointer int * p and sometime later I write *p = 10. So, whatever be the location at which p is pointing to so that location gets the value 10.

So, how this thing is going to happen internally, so for this pointer to be supported or this indirect addressing to be supported the underline processor must support this indirect addressing. So, in case of 8085, so this is done by this H and L pair. So, there are

instructions by which you can tell that I want to store the value 10 at the location pointed to by the H, L pair ok. So, all memory addresses are 16 bit, so this H, L pair is a 16 bit value. So, if you say that way so this memory location which is pointed to by H, L pair will be getting the value 10. So, they that actually, so if I my if the if the address of p address of this location is loaded onto the H, L pair, and after that I will do this M[H L] gets 10.

So, essentially it implements the pointer as a pointer operation. So, this H and L pair can be used as data pointer for holding the memory address. Now, apart from that there are some special purpose registers accumulators that we have already said. So, this is the source and destination of all arithmetic logic operation. Then we have got this store the result on is also store the result of this register. So, there are some flag registers, there are some flag registers which will tell you like the status of the last operation.

(Refer Slide Time: 25:48)

Flag Register

- 8 bit register – shows the status of the microprocessor before/after an operation
- S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)

Diagram of an 8-bit flag register:

D7	D6	D5	D4	D3	D2	D1	D0
(S)	Z	X	AC	X	P	X	(CY)

Handwritten notes on the left side of the diagram:

Sub B
A < B
A = 0

A handwritten diagram on the right side shows a 4x3 grid with arrows indicating connections between bits D7, D6, D5, and D4.

At the bottom of the slide, there is a logo for IIT Kharagpur and NPTEL Online Certification Courses, along with a video player showing a man speaking.

So, we have got this flag register which is also an 8 bit register and it shows the status of the microprocessor before or after an operation. So, out of these 8 bits all bits are not used, but the bits that are used are the sign, 0, auxiliary carry, parity and this carry. So, this is the may if you look into this register. So, then this bit D7, D7 is the sign. So, if you do the last operation and after doing the operation the result becomes negative for example, suppose I have got an instruction subtract B.

So, the essential it will do A gets A - B and after doing the operation if the value of A becomes negative, then the sign flag will be set to 1. Otherwise this sign flag is 0 then the next bit number D6 corresponds to the 0 flag, like if you do this operation if it happens that A and B they were same the values are same then after this A will become equal to 0 and whenever A becomes equal to 0 this 0 flag is set to 1.

So, many a times we need to compare between some two or more operands and depending upon the comparison result we want to take some action. So, this sign and 0 flags they will help that way bit number D5 is not used bit number D4 is for auxiliary carry. So, auxiliary carry is like this, so if you are doing some addition this is the first 8 bit this is the A register and this is the B register, okay this is the B register. So, so this is bit number 0 to 3 and 4 to 7, now if I the instruction is say add. And when doing this operation if some carry was generated, when I have added say this bit 3 is if some carry is generated and this carry is taking is going to the next one.

So, if these carry is generated, so this is known as auxiliary carry. So, it is at the middle this is sometimes useful in this your BCD addition and all, so, will see that. So this is the auxiliary carry bit, then this parity bit. So, we have already seen that whether the, you want even parity or odd parity. So, based on that this bit may be set to 0 or 1 to make it either even or odd and this bit number 0 is the carry bit. So, carry is after doing this whole addition if some carry is generated at this position.

So, after you have done this whole addition if some carry is generated then this that you carry is stored in the carry flag. So, this way this flag register it uses different bits to represent the status of the last operation and based on the value that we have. So, we can we can write program that will exploit the values and accordingly take some conditional decisions within the program to continue with some part of the program or to go to some other part of the program.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 52
8085 Microprocessor (Contd.)

(Refer Slide Time: 00:18)

The slide has a yellow background and a dark blue header bar. The title 'Sign Flag' is centered in a large, bold, black font. Below the title is a bulleted list of four points:

- Used for indicating the sign of the data in the accumulator
- The sign flag is set if negative (1 – negative)
- The sign flag is reset if positive (0 – positive)

At the bottom of the slide, there is a dark blue footer bar. On the left side of the footer, there are two logos: the IIT Kharagpur logo and the NPTEL logo. Next to the IIT Kharagpur logo is the text 'IIT KHARAGPUR'. Next to the NPTEL logo is the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, the number '37' is displayed.

So, as far as the sign flag is concerned, so as I said that it is used for indicating the sign of the data in the accumulator. So, after you have done some operation, if the accumulator content becomes negative, in that case the sign flag will be set to 1; and if the accumulator content is positive then it will be 0.

(Refer Slide Time: 00:36)

The screenshot shows a presentation slide with a yellow background. At the top, it says "Zero Flag". To the right, there are handwritten notes: "DCR B" above "INR B". Below "Zero Flag", there are two bulleted lists:

- Is set if result obtained after an operation is 0
- Is set following an increment or decrement operation of that register

To the right of the second list, there are handwritten notes: "ADD" above "SUB". Below "Carry Flag", there is one bullet point:

- Is set if there is a carry or borrow from arithmetic operation

At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL Online Certification Courses. On the bottom right, there is a small video window showing a man speaking.

So, then the 0 flag, so if the result is obtained is after the after doing the operation is 0, then this is carry 0 flag is set and following an increment or decrement operation of the of the register. So, if you are doing some increment or decrement or some registere then also this 0 flag is set.

So, for example, if I have got an instruction like say we have got some instruction like say decrement B ok. So, this B register value will be decremented and this decrement operation if it is either if the B register content becomes 0 due to that then also the 0 flag will be set or maybe increment operation. So, INR B, so these are not on the A register ok. So, with the other registers also this is possible.

Now, the carry flag, so carry flag is set if there is a carry or borrow from the arithmetic operation. So, if the carry is generated if you are doing an add operation ok. So, after doing the addition if a carry is generated from bit number 7 or if you are doing a subtract operation then a borrow may be generated from bit position 7. So, that way this carry or borrow if it is generated due to arithmetic operation then the carry flag will be set.

(Refer Slide Time: 02:02)

The slide has a yellow background with a black header bar containing icons. The title 'Auxillary Carry and Parity' is centered in a large, bold, black font. Below the title is a bulleted list of two items:

- Auxillary Carry Flag is set if there is a carry out of bit 3
- Parity Flag Is set if parity is even and is cleared if parity is odd

At the bottom left, there is a logo for IIT Kharagpur and text 'IIT KHARAGPUR'. Next to it is the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side, there is a small video window showing a man speaking.

Now, auxiliary carry, so auxiliary carry flag is set if there is a carry out of bit 3. So, this I have already explain, so bit position three onwards if there is a carry due to either carry or borrow ok. So, either addition or subtraction instruction. So, carry or borrow is generated from bit 3 then this auxiliary carry will be set, parity flag is set if the parity is even and is cleared if the parity is odd.

So, 8085 it follows an odd parity structure. So, if the number of bits in the accumulator is even then this parity bit will be set. So, that the total number of bits total number of bits become odd. And if the number of bits in accumulator is odd, then this parity bit will be set to 0 telling that this is a the number of ones there is already odd. So, this way this parity flag is going to be used and some programs. So, it can use this parity flag to know whether the number of ones in the accumulator is even or odd. So, we do not need to count we need to check separately. So, we can just do a check on the parity flag and take a decision.

(Refer Slide Time: 03:14)

The Internal Architecture

- We have already discussed the general purpose registers, the Accumulator, and the flags.
- The Program Counter (PC)
 - This is a register that is used to control the sequencing of the execution of instructions.
 - This register always holds the address of the next instruction.
 - Since it holds an address, it must be 16 bits wide.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

40

So, if you look into the internal architecture. So, we have already seen that general purpose registers the accumulator and the flags. Now next we look into the program counter, so there the program counter register PC. So, this is a 16 bit register and this register is used to control the sequencing of execution of instruction. So, as I said that at any point of time this program counter value it tells me, what is the address of the next instruction to be executed.

So, it always holds the address of the next instruction and it is auto incremented. So, as soon as we are accessing the next instruction this program counter value will be incremented, and it will point to the next address from where the fetch has to be done.

(Refer Slide Time: 04:09)

The Internal Architecture

- The Stack pointer
 - The stack pointer is also a 16-bit register that is used to point into memory.
 - The memory this register points to is a special area called the stack.
 - The stack is an area of memory used to hold data that will be retrieved soon.
 - The stack is usually accessed in a Last In First Out (LIFO) fashion.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, since it holds address, so this is a 16 bit register and there is a stack pointer register. So, stack pointer this is also a 16 bit register, and it points to some memory location and this the memory this register points to is a special area called stack. So, this stack is used for holding the returner.

Like in high level language program, so we are writing procedures and we know that from some main routine, so, we can call a procedure and from there the, from the procedure we come back to the main routine when the procedure is over.

Similarly, when you are writing program in assembly language in the assembly language then also you can write something called a subroutine. So, you can jump from the main program to a subroutine and after finishing that subroutine you can come back to the point at which you have called that subroutine, now this return address has to be saved somewhere. So, that at the end of the subroutine the processor will know where to go back ok. So, this return address is saved in the stack and the stack pointer is used for pointing to the memory location where the last address has been stored.

So, this stack is an area of memory that is used to hold the data that will be retrieved soon. So, this may be that return address or it may be some parameters that we have passed from this main program to the subroutine. So, like that so they are they can be retrieved from the stack. And it is usually accessed in a last in first out fashion that is quite obvious because, whenever you are this is your main program and from here.

So, you have this is the main program and from here, so you have called a subroutine. So, this is the subroutine S1 and within the subroutine somewhere here you have called another subroutine say S2. Now when S2 is over, so you have to go back to this point continue the execution and then from this end so you have to go back to this point.

So, this is in the so what so whenever you have jumping from M to S1. So, if this is the stack in the stack you are saving the return address. So, you are saving this particular return address, so let us call it *1. So, *1 is saved here then when you are going from S1 to S2 that return address S1 return address is saved here *2 ok.

Then from when S2 end so we take out this value and come back to * 2. So, that is a so the number *2 enter last into the stack, but it is taken out of the stack at the earliest. So, that is how it is called last in first out. So, whatever enter last will be taken out first. So, this way we can this stack will say it can be used for doing some operation.

(Refer Slide Time: 06:55)

The screenshot shows a presentation slide with a yellow header bar containing a toolbar icon set. The main title 'Non Programmable Registers' is centered in a large, bold, black font. Below the title is a bulleted list of components:

- Instruction Register & Decoder
 - Instruction is stored in IR after fetched by processor
 - Decoder decodes instruction in IR
- Internal Clock generator
 - 3.125 MHz internally
 - 6.25 MHz externally

At the bottom of the slide, there is a blue footer bar with three logos and text: IIT Kharagpur, NPTEL Online Certification Courses, and a video camera icon. To the right of the footer, there is a small video window showing a man speaking.

There are some non-programmable registers like say instruction register and decoder, this instruction register this is a special purpose register is also known as IR. So, whenever an instruction is fetched from the memory, so, we have got this we have got this the fetched pattern stored in the instruction register. And from the instruction register it goes to the decoder, the decoder will try to identify the meaning of the instruction.

So, this instruction is stored in the instruction register after fetched by processor and the decoder will decode the instruction into IR. So, this instruction register, so this is being read and written, but from the users point of view. So, this instruction register is not accessible because, user will not be able to write something on to the instruction register or read the content of the instruction register through some program.

Now, there is a internal clock generator circuitry. So, the externally, so you have to connects a clock, some crystal. So, if you connect clock with crystal 3.125 megahertz then outs externally. So, there is a externally we connect this a crystal of 6.25 megahertz, and internally it is divided by 2. So, it becomes 3.125 megahertz ok, so that is the clock frequency at which this 8085 will work.

(Refer Slide Time: 08:24)

The Address and Data Busses

- The address bus has 8 signal lines A8 – A15 which are unidirectional.
- The other 8 address bits are multiplexed (time shared) with the 8 data bits.
 - So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.
 - During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
 - In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Then there are address and data buses, so this address bus has got 8 signal lines A8 to a 15, which are unidirectional the other at 8 address lines are multiplexed with the 8 bit data bus. So, this AD0 through AD7 they are bidirectional and such they are both the purposes of the lower order address bus A0 to A7 and the data bus D0 to D7, simultaneously it is not at the same time precisely at the same time.

So, so when this address data bus this AD 0 to AD 7 is used as address bus, so, it is not used as date bus and similarly when it is used as a data bus, it is not used as address bus. So, during the execution of the instructions, so this the lines carry the address bits during the early part and then during the later part of execution they will carry the data bits.

So, basically if you are trying to access memory, so first you have to give the address and while giving the address. So, the address higher order part is an A8 to A 15 and the lower order part is in A0 to A7. After sometime when the address has been noted by the memory then you can, you can, you can withdraw the address from the address bus.

And now whatever value this memory put on to the data bus it comes as D0 to D7 to the processor. Similarly, if you are trying to write something on to the processor also to the memory also the same thing the first you put the address. So, memory understands the address then you withdraw the address and put the data bus content. So, to be written ok, so that is so now, the part acts as the data bus.

So, this in order to separate this address from data, so we can use a latch to save the value before the function of this bits changes. So, as I said that for initially for some time this A0 to A7 holds the address. So, in most of the memory design, so will find that this address bus content has to be held continually for that purpose we can have some external latch by which this address value is latched there.

(Refer Slide Time: 10:30)

Demultiplexing AD7-AD0

- The **high order bits** of the address remain on the bus for **three clock periods**. However, the **low order bits** remain for **only one clock period** and they would be lost if they are not saved externally.
- To make sure we have the entire address for the full three clock cycles, we will use an **external latch** to save the value of AD7–AD0 when it is carrying the address bits. We use the **ALE signal** to enable this latch.

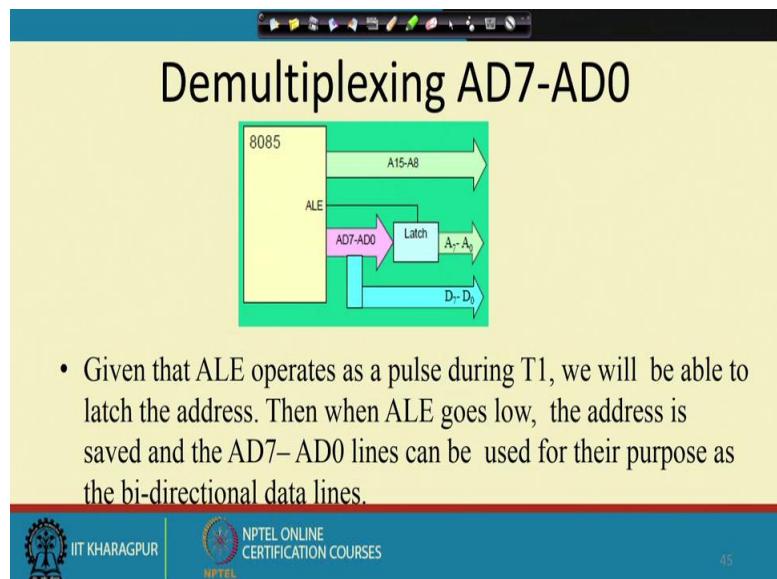
IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this higher order address bus, so this has got this higher order address bits they will remain on the bus for three clock periods for 8085 this memory, memory access takes three clock cycles.

So, for three clock periods the higher order address bus will hold the value. However, as far as the lower order address bus is concerned. So, it is only for one clock period this lower order address bus holds the address bits valid address bits. So, what we need to do is after this one clock cycle, so, this value is withdrawn, so somehow we need to save this address somewhere.

So, that the memory sees continually that it has got all those address lines, but this for that purpose we have to have some external circuitry. So, we, we use an external latch to save the value of AD 7 to 0, when it is carrying the address bits and we will use the ALE signal to enable this latch.

(Refer Slide Time: 11:30)



So, this is the situation, so you see that this A8 to A15. So, actually this side we have got the memory. So, this side we have got the memory it is this is the memory. So, these are the address lines, so this these are the address lines for the memory and these are the, this is the data line fine. So, this address lines, so this our this the higher order address bus it has got A8 to A 15 that is held continually for the entire read operation memory read or write memory axis operation. However, this AD7 to 0, so it has got this address values A0 to A7 only for the first clock cycle, and on for the first clock cycle is ALE signal is also activated.

So, externally what you do you use a latch. So, that this when this ALE is high this value of A7 to A0 gets the stored into this latch. So, after that after of the first cycle this ALE

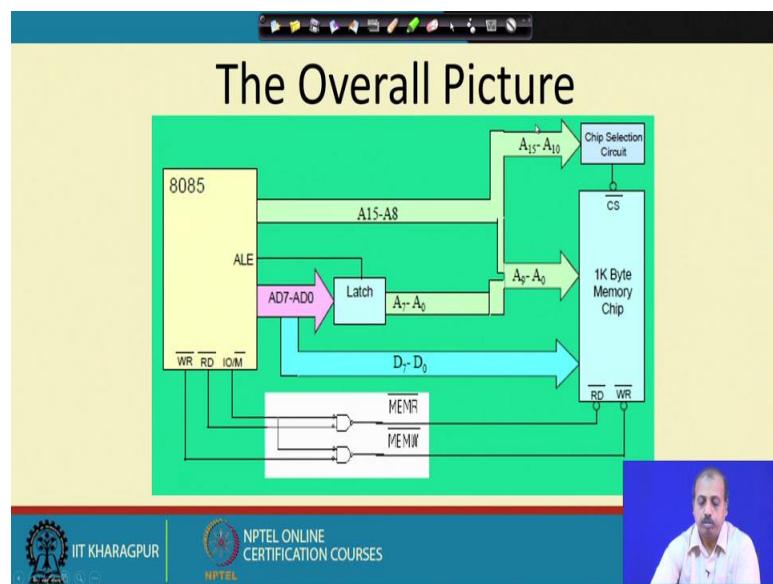
signal is deactivated as a result this latch becomes inactive and it does not change its value even if this lines input to the latch changes since this ALE signal is low. So, latch will not put that value on to it, so it will continually hold this A0 to A7 at the output.

Now, after that this D0 to D7, so these are the data bus. So, there either the memory will put the content on to this data bus, or the processor will if it wants to write to the memory, so it can put the data onto this data bus. Essentially, so, this process as far as the memory is concerned it sees this address 16 bit address line continually for four cycles and it also sees for the for four cycles and it also sees the data for the last part of the operations. So, as a result, so it can do this read write operation easily. So, this ALE operates as a pulse during the first clock cycle.

So, when this 8085 is reset, so it every instruction access for the first cycle, so, it will give a pulse on the ALE line, so this will able this will this will be able to latch the address, because A0 to A7 is also put on to this AD7 to 0 line. So, that the address will get latched and then ALE will go low.

So, address will be saved and the line AD7 to AD0 can be used for other purpose that is there for holding the data. So, this is known as the demultiplexing of the address data bus. So, in many processors wherever we try to reduce the number of pins, so we can use this demultiplexing technique. So, that pin requirement is low ok.

(Refer Slide Time: 14:22)



So next is overall picture is like this. So, if I have got this 8085 here and we are trying to connect a 1 kilobyte memory chip ok, then this 1 kilobyte means, so this is the address bus is n bits. So, this A0 to A9, so they are to be connected and lines A10 to A15, so they total, total address generated by 8085 is 16 bit. So, out of that A0 to A 15 they will goes through a chip selection circuitry. So, in the in some classes earlier we have seen how to use this decoders to generate the chip select signals for various chips.

So, some decoding logic will be put here, so that it will select this particular chip for some address range. Now this A8 to A15, so they are coming directly from the 8085 as higher order address bus out of that A10 to A15 is fetch the chip selection logic and A8 and A9 they will come to the lower side. Now for the lower order address bus, lower order address, so, AD7 to AD0, so they are passing through this latch and this ALE signal is selecting the enabling the latch. So, as a result A7 to A0 will be getting latched onto here.

So, ultimately, so this A0 to A7 and from this side you are getting A8 to A9. So, total A0 to A9. So, they are forming the address bus for the memory chip and this D0 to D7, so that is the data bus. So, it is there. Now I need to generate the read write signals for the memory. So, this IO/\bar{M} is low then this IO then this is a memory operation. So, this \bar{M} and $\overline{\text{read}}$, so if these two lines are inverted and then passed through a NAND gate then that means, so this is the memory read bar signal.

So, when the processor is doing a memory operation and it is doing a read operation then this memory read bar line will be low. So, as a result it is connected to this read bar pin of this chip. So, that it is doing the read operation similarly this $\overline{\text{write}}$ line and this IO/\bar{M} line. So, they are connected into this gate and it generates the memory write bar signal. And this memory write bar signal connects to the write bar signal of the chip.

So, this way this is the overall pictures, so if I have got more number of this is only 1 kilobyte memory chip interfacing that has been shown. So, if I have got more such chips then the chips selection circuit will generate appropriate chip selection logic. So, rest of the thing will remain unaltered.

(Refer Slide Time: 17:03)

The slide has a yellow header bar with a toolbar icon at the top. The main title 'The 8085 Instructions' is centered in a large, bold, black font. Below the title is a bulleted list of points about the 8085 processor's instruction set:

- Since the 8085 is an 8-bit device it can have up to 2^8 instructions.
- However, the 8085 only uses 246 combinations that represent a total of 74 instructions.
- Most of the instructions have more than one format.
- These instructions can be grouped into five different groups:
 - Data Transfer Operations
 - Arithmetic Operations
 - Logic Operations
 - Branch Operations
 - Machine Control Operations

At the bottom left, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side, there is a small video window showing a man speaking.

Next we will be looking into the most vital part of 8085 like as a user of the system like how can I what are the operations that I can do with this 8085 processor. So, that defines the instruction set of this 8085, so 8085 is an 8 bit device. So, it can have up to 2^8 that is 256 instructions out of that only 246 combinations are used and that represents a total of 74 instructions. And naturally most of the instructions have more than one format.

So, we will see that this instructions that we have, so they can be grouped into five different groups; data transfer, arithmetic, logic, branch and machine control. So, these are the different classes of instructions that we have in 8085, so you look into this individual classes one by one.

(Refer Slide Time: 17:58)

Instruction and Data Formats

- Each instruction has two parts.
 - The first part is the task or operation to be performed.
 - This part is called the “opcode” (operation code).
 - The second part is the data to be operated on
 - Called the “operand”.

Handwritten notes on the right side of the slide:

- Instruction
- + zero operand
- + single
- + Two
- ADD(B)
- LXI H, L16bit

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, each instruction has got two parts the first part is the task or operation to be performed.

So, this part is called opcode because it is operator it is telling the operation to be done.

And the second part is that data to be operated on and it is called operand.

So, we have got opcode and operands, so any instruction can be divided into this opcode and operand part. Now depending upon the instruction that we have, so number of operands maybe 0 it may be 1 or it may be 2. So, we will see some instructions like that. So, say, so based on the number of operands the instructions maybe classified as 0 operand, 1 operand, single operand, single operand or two operand. So, in case of 8085 we do not have more than two operands per instruction.

So, it is at most two operand, so 0 operand instruction like say the halt instruction. So, this is a zero operand instruction, so no operand is necessary single operand instruction I have already given a number of examples like say ADD B. So, this B is the operand other operands are other operand that is A is implicit for the instruction. So, you don't need to specify it separately. So, and this add is the memory, so that way this is a single operand instruction. So, two operand instruction like there are instruction like, which says that LXI H, some 16 bit value, some 16 bit value. So, this tells that this 16 bit value has to be loaded into the HL register pair.

So, we will see this instruction later, but here you see this is an example of two operand instruction where H is the first operand and the 16 bit value that we have that is the second

operand. So, this way I can have different number of operands in the instructions. Next we will see how this instructions will look like ok.

(Refer Slide Time: 20:13)

The slide has a yellow header bar with standard window controls. The main title 'Data Transfer Operations' is centered in a large, bold, black font. Below the title is a bulleted list of four points:

- These operations simply **COPY** the data from the source to the destination.
- MOV, MVI, LDA, and STA
- They transfer:
 - Data between registers.
 - Data Byte to a register or memory location.
 - Data between a memory location and a register.
 - Data between an I/O Device and the accumulator.
- The data in the source is **not changed**.

To the right of the list, there is a column of assembly language instructions with their meanings:

- MOV A,B $\Rightarrow A \leftarrow B$
- MOV D,E $\Rightarrow D \leftarrow E$
- MVI A,40 $\Rightarrow A \leftarrow 40$
- LDA <16-bit address>
↳ LDA 2000H $\Rightarrow A \leftarrow m[2000]$
- STA 2000H $\Rightarrow m[2000] \leftarrow A$

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

So, first category of instruction that we look into, so they are known as data transfer operation so, these operations they are many they are useful for copying the data from source to the destination address. So, the instructions that are common is MOV, MVI, LDA, STA etcetera.

So, it is for example, we can have instructions like say MOV A, B. So, this means the A register gets the content of B register or say MOV D, E. That means, the D register gets the content of E register then there is MVI instruction MOVE immediate. So, it is like this MVI A, 40, so that means the A register will get the value 40 ok.

Then LDA the format is LDA and a 16 bit address, like 16 bit address like LDA say 2000 Hex. So, this means the accumulator register will get the content of memory location 2000, fine then STA is for, it is just the reverse of LDA. So, STA 2000 hex that means, memory location 2000 will get the content of the accumulator. So, this way we have got this LDA, STA, MOV, MVI this type of instructions which is basically the copying of content of one, one register or memory location to another register or memory location.

So, data between they can transfer data between registers like MOV A, B, data byte to A register or memory location like say this one MVI A, 40 ok. Data between a memory

location and A register like say this one LDA we have seen, or data between IO device and the and the accumulator. So, there, so that can also be done, so this IO access will see separately later.

So, that is the IO device access can be done, but that data in the source is not changed. So, these are all copying, so the source content will always remain unaltered. So, source content is not tampered with, so these category of instructions of they are known as data transfer instructions.

(Refer Slide Time: 23:02)

The LXI instruction

- The 8085 provides an instruction to place the 16-bit data into the register pair in one step.
 - **LXI Rp, <16-bit address>** (Load eXtended Immediate)
 - The instruction **LXI B 4000H** will place the 16-bit number 4000 into the register pair B, C.
 - The upper two digits are placed in the 1st register of the pair and the lower two digits in the 2nd

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this LXI instruction, so this 808 so this LXI instruction it may this provides an instruction to place 16 bit data into the register pair in one step. So, this LXI the format is LXI register pair, 16 bit address. So, it is load extended immediate, so it tells that this LXI B 4000. So, this will tell that the 16 bit value 4000 will be put into the pair B C ok. So, this B will get 40 and C will get 00. The upper two digits are placed in the first register. So, this is a hexadecimal number, so if I talk in terms of 8 bit value. So, 40 is the first byte 0 0 is the next byte 40 is the most significant byte 00 is the least significant byte.

So, the most significant byte goes to the B register and the least significant byte goes to the C register, so this is the LXI instruction.

(Refer Slide Time: 24:02)

The Memory “Register”

- Most of the instructions of the 8085 can use a memory location in place of a register.
 - The memory location will become the “memory” register M.
 - ~~MOV M, B~~
 - copy the data from register B into a memory location.
 - Which memory location?
- The memory location is identified by the contents of the HL register pair.
 - The 16-bit contents of the HL register pair are treated as a 16-bit address and used to identify the memory location.

So, we can also talk about think about memory as a register. So, most of the instruction is 8085 can use a memory location in place of a register like say this instruction. So, this MOV M, B, so this memory location will become the memory register M, so this M , B. So this tells that the content of a registered B will be copied on to memory location M, but what is the address. So, I so this, so when I say B C D E, you are talking about a particular register, but when I say M so, that is it is a memory location, so this memory location address has to be told and this address is implied it is implicitly identified by the content of the HL register pair.

So, it is like this if I have got this HL pair has got the value say 1500 hex that is H is having 15, and this is in H and this 00 is in L. Say that is the situation then if this is my memory this is the my memory and this is the location 1500 and if the content, so, the now in this instruction what will happen is that it will first go to the HL register and it will find out what is the content. It finds that the content is 1500 and then it will be copying the content of B register, content of B register suppose B register was equal to say 20.

So, suppose B register was equal to 20, so this 20 will be copied onto this location 1500. So, that is the meaning of this MOV M, B instruction that we have. So, copy the data from register B into memory location and which memory location.

So, this is given by the 16 bit contents of the HL register pair and HL register pair treated as a 16 bit address and that is used for identifying memory locations. So, this is the way

by which we can implement pointers ok. So, this HL pair, so it can be loaded with the address of the pointer. So, if I have, as I was telling that if I have got a pointer P now.

(Refer Slide Time: 26:30)

The Memory “Register”

- Most of the instructions of the 8085 can use a memory location in place of a register.
 - The memory location will become the “memory” register M.
 - **MOV M B**
 - copy the data from register B into a memory location.
 - Which memory location?
- The memory location is identified by the contents of the HL register pair.
 - The 16-bit contents of the HL register pair are treated as a 16-bit address and used to identify the memory location.

So, you are writing like $*P = 10$, so that can be done in this way the first this B register can be loaded with the value 10 by instruction MVI B , 10. And then if this P happens to be the memory location the address of P is say 1000 then this LXI H, 1000 hex. So, that will be loading the HL pair with the value with this value, and then we will be doing this MOV M, B.

If we do MOV M, B then this content memory location thousand will get the value 10. So, this is equivalent to this statement $*P = 10$. So, this is the way by which pointers can be implemented. So, if we are using this indirect addressing in the HL register, using the HL register pair.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 53
8085 Microprocessor
(Contd.)

So, we can also use other register pairs.

(Refer Slide Time: 00:17)

The slide has a yellow background with black text. At the top right, there is a handwritten note: $x_8 = x_9$, $y_8 = y_9$, $\text{load } 2000H$. Below the note, there is a list of instructions:

- There is also an instruction for moving data from memory to the accumulator without disturbing the contents of the H and L register.
 - LDAX Rp (LoD Accumulator eXtended)
 - Copy the 8-bit contents of the memory location identified by the Rp register pair into the Accumulator.
 - This instruction only uses the BC or DE pair.
 - It does not accept the HL pair.

So, there is one instruction like LDAX. So, this LDAX instruction; so, it means that load accumulator extended. So, this one; so, this also accept say register pair as the operand, and then this register pair we can somehow this register pair if it is loaded with some value and then that will be acting as the address. So, copy the 8 bit contents of the memory location identified by the register pair Rp register pair into the accumulator.

So, this will take a value on to the accumulator. So, it does not put it on to other location, but it will put it on to this accumulator register and then this register pair, it can be B, C or D E. So, I cannot use H L, because H L is not necessary H L is used if I am using H, L then we have got this move instruction MOV A, M or MOV M, B is.

So, like that that H L is much more generic in nature. So, that is already it that facility has already been provided. So, what the designers did is apart from H L. So, if you want to use

some other address also other pointer like, so, in the previous example; so, we have seen that if I want to have p as a pointer, so, this p's addresses loaded into the H L pair now it may. So, happened that I want to do $*p = *q$, but p and q are 2 pointers. Now how to do this? So, this somehow this p p's address is loaded into H L suppose, this address is known to be 1000 and this q's address is known to be 2000. So, what you do you first load this H L pair with the value 1000 ok. So, this H L pair contents value of the 1000 and then you this for p, you do for this for this q part.

So, first of all this register pair has to be loaded with the value so far that we do say MVI or LXI D, 2000. So, LXI H, 1000 loads the address of p on to H register LXI D , 2000 loads the address of q on to the D register and then I can do LDAX D. So, this will give me the content of location 2000 that is $*q$ on to the A register and then we can do MOV M, A. So, that the content of that accumulator goes to the memory location.

So, this way, we can do this type of multiple pointers can be accommodated. So, this helps in the realizing the pointer arithmetic, this pointer assignments. So, this way we can, so, we can use this register pairs for this memory access, type, the pointer type of access. So, this LDAX is one instruction for doing it.

(Refer Slide Time: 03:59)

The slide has a yellow header bar with various icons. The main title 'Indirect Addressing Mode' is centered in a large, bold, black font. Below the title, there is a bulleted list of points:

- Using data in memory directly (without loading first into a Microprocessor's register) is called **Indirect Addressing**.
- Indirect addressing uses the data in a register pair as a 16-bit address to identify the memory location being accessed.
 - The HL register pair is always used in conjunction with the memory register "M".
 - The BC and DE register pairs can be used to load data into the Accumulator using indirect addressing.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '53'.

Now, indirect addressing mode; so, using data in memory directly without loading first into microprocessors registered is the indirect addressing.

So, we do not load the address into some register and then do the loading of the load the address or content of that memory location. So, this indirect addressing it uses the data in a register pair as a 16 bit address to identify the memory location being access. So, whatever we were discussing so far that LXI that the MOV A, M or that LDAX type of instruction.

So, they are actually using this indirect addressing. This H L register pair is always use in conjunction with the memory register M and this BC and DE register pairs are used to load data into the accumulator using indirect addressing. So, in the previous example, we have seen that way that H L pair can be used for the accessing, the memory location accessing memory and in that case, we just tell M and for BC and DE. So, we have to use this LDAX and the accumulator has to be involved in the process.

(Refer Slide Time: 05:12)

Arithmetic Operations

- Addition (ADD, ADI):
 - Any 8-bit number.
 - The contents of a register.
 - The contents of a memory location.
- Can be added to the contents of the accumulator and the result is stored in the accumulator.
- Subtraction (SUB, SUI):
 - Any 8-bit number
 - The contents of a register
 - The contents of a memory location
- Can be subtracted from the contents of the accumulator. The result is stored in the accumulator.

ADD B \Rightarrow A ← A + B
 ADD 50H \Rightarrow A ← A + 50H
 ADD M \Rightarrow A ← A + m[H]

Next we look into the arithmetic operations. So, arithmetic operations like say addition subtraction. So, this addition operation we have got two mnemonics available in 8085; one is called add, another is called ADI. So, add is any 8 bit number may be added the content of one register and are the contents of a memory location. So, you can use it, you can use an instruction like say ADD B as I said that this will do A gets A + B or you can say like an 8 bit numbers. So, we can say like ADI 50 hex. So that means, A will get A+50H or you can say like ADD M. So, where a will get the content of a plus memory location pointed 2 by the H L pair ok. So, that way we can have different types of addition

operation. So, it can be added to the contents of the accumulator and the result is stored in the accumulator; so, that destination and first operand, so, they are all accumulator.

Similarly, we have got subtract and subtract immediate. So, otherwise it is same. So, only it is instead of addition, it will do the subtraction operation. The result and the first operand are the accumulator.

(Refer Slide Time: 06:49)

Arithmetic Operations Related to Memory

- These instructions perform an arithmetic operation using the contents of a memory location while they are still in memory.
 - ADD M
 - Add the contents of M to the Accumulator
 - SUB M
 - Sub the contents of M from the Accumulator
 - INR M / DCR M
 - Increment/decrement the contents of the memory location in place.
 - All of these use the contents of the HL register pair to identify the memory location being used.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Next, so, if you are doing this arithmetic operations with respect to memory, so, we can do like ADD M. So, add the contents of M to the accumulator and the H L pair will give me the address. So, this I have already said. In fact, all these operations are add, subtract this increment, so, all these operations can be done directly on memory locations by having the proper address in the H L pair.

(Refer Slide Time: 07:18)

The screenshot shows a presentation slide titled "Arithmetic Operations". Below the title, there is a bullet-pointed list under the heading "- Increment (INR) and Decrement (DCR):". The list contains two items:

- The 8-bit contents of any memory location or any register can be directly incremented or decremented by 1.
- No need to disturb the contents of the accumulator.

On the right side of the slide, there is a handwritten note: "DCR B" with an arrow pointing to a circled "INR B". Another arrow points from this circled note to the text "MOV A,B", "ADD 1", and "MOV B,A".

The footer of the slide includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

Then this increment and decrement operations; so, this is another class of arithmetic operation. So, this is the content of 8 bit register or memory locations can be implemented or decremented by 1, it is, it does not need to disturb the contents of accumulator. So, I can simply have an operation like say DCR B or DCR B or INR B. So, like that.

So, if you would if you do not have this instruction and if you want to say increment this B by 1, then what you need to do is that MOV A , B, then ADD ADI 1 and then MOV B , A. So, this is the sequence that I have to do if I do not have the direct increment facility with the registers other than A. So, this that's why this INR instruction, so, they are taken directly so that they do not involve the accumulator at all. So, there is no need to disturb the content of the accumulator. So, it is directly from the instruction.

(Refer Slide Time: 08:35)

Manipulating Addresses

- Now that we have a 16-bit address in a register pair, how do we manipulate it?
 - It is possible to manipulate a 16-bit address stored in a register pair as one entity using some special instructions.
- INX Rp The register pair is incremented or decremented as one entity. No need to worry about a carry from the lower 8-bits to the upper. It is taken care of automatically.
- DCX Rp

INX D

(D) ↑ (E)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now how can we manipulate the address ok? So, we have got 16 bit address in some register pair. Now sometimes we need to increment or decrement it. For example, if I have got an array that array start address may be loaded, array is loaded into say memory location. So, the memory start address is say thousand.

Now, the initially we said the HL pair to the beginning of the array by loading thousand on to it, but after the first item in the array has been processed we would like to take the HL pair to the next address ok. So, for that purpose; so, we have got this INX operation. So, INX; so, it will increment the register pair by 1, by 1 ok. So, that way, it is possible to manipulate 16 bit address stored in a register pair as 1 entity and using some special instruction like INX Rp the register pair will be incremented as one entity and the carry is already taken care of. The good thing about this INX operation is that. So, the there; so, I have got this say if I have got say INX D, so, this D; so what it will do.

So, it will take the DE pair. So, this is D and this is E. So, it will do an increment, but if there is a carry generated. So, it will auto- automatically taken care, here. So, we don't need to, in a program, we don't need to take care of it separately, but if you **are** use INR E operation, then the carry that is generated again, you have to take care for doing that thing. So, this INX instruction has been provided for that purpose and similarly DCX instruction.

So, that is for doing the decrement operation ok. So, we have got this INX and **DC** and DCX operations.

(Refer Slide Time: 10:32)

The slide is titled "Logic Operations". It lists the following instructions:

- These instructions perform logic operations on the contents of the accumulator.
- ANA, ANI, ORA, ORI, XRA and XRI
- Source: Accumulator and
 - An 8-bit number
 - The contents of a register
 - The contents of a memory
- Destination: Accumulator

Handwritten notes at the bottom right:

- ANA R/M AND Accumulator With Reg/Mem
ANI # AND Accumulator With an 8-bit number
- ORA R/M OR Accumulator With Reg/Mem
ORI # OR Accumulator With an 8-bit number
- XRA R/M XOR Accumulator With Reg/Mem
XRI # XOR Accumulator With an 8-bit number

Handwritten examples:
ANA B \Rightarrow A \leftarrow A AND B
ANA M \Rightarrow A \leftarrow A AND m[HL]
ANI 20H \Rightarrow A \leftarrow A AND 00100000

At the bottom, there are logos for IIT Kharagpur and NPTEL Online Certification Courses, along with a toolbar and a status bar showing "SR" and "10:32 AM 11-Aug-2015".

Next we will see some logic operations. So, these instructions perform logic operations on the contents of the accumulator. So, you have got AND accumulator, then this AND immediate, then OR accumulator, OR immediate, then XOR accumulator and XOR immediate. So, these are the different operations that we have. So, AND accumulator with register or memory; so, you can say; you can write instructions like say AND accumulator B; so a will get A and B.

So, I can have this ANA M. So, in that case, accumulator will get A and memory location pointed to by H L pair. So, this way we can have the AND operation why we say or you can have that and immediate operation; so ANI some 8 bit value. So, it is say 20 hex. So, what it will do? A will get A AND with the 20 hex value that is 0 0 1 0 0 0 0 ok. So, this way we can have different types of operations- AND operations, OR operation, XOR operation and destination is always the accumulator and the source is also one of the source is also accumulator.

(Refer Slide Time: 12:13)

The slide has a yellow background. At the top center, it says "Logic Operations". Below that, under the heading "- Complement:", there is a bulleted list:

- 1's complement of the contents of the accumulator.
 - CMA No operand

At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL. On the right side, there is a video player showing a man speaking.

So, there is a complement operation. So, it gives the ones complement of the content of the accumulator. So, all the bits of the accumulator are complemented and the instruction is CMA. So, it is a 0 operand instruction.

(Refer Slide Time: 12:35)

The slide has a yellow background. At the top center, it says "Additional Logic Operations". Below that, under the heading "• Rotate", there is a bulleted list:

- Rotate the contents of the accumulator one position to the left or right.
 - RLC Rotate the accumulator left.
Bit 7 goes to bit 0 AND the Carry flag.
 - RAL Rotate the accumulator left through the carry.
Bit 7 goes to the carry and carry goes to bit 0.
 - RRC Rotate the accumulator right.
Bit 0 goes to bit 7 AND the Carry flag.
 - RAR Rotate the accumulator right through the carry.
Bit 0 goes to the carry and carry goes to bit 7.

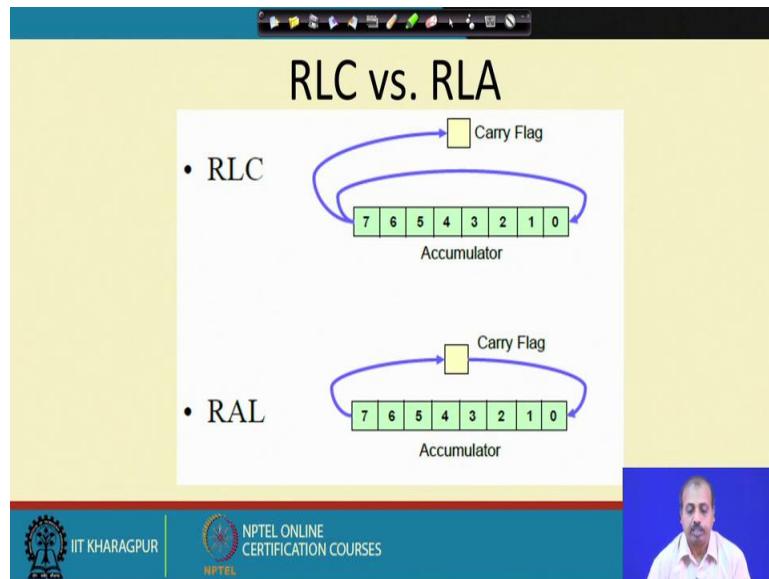
At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL. On the right side, there is a video player showing a man speaking.

Because it implicitly means the content of the accumulator will be a flipped ok. So, that way it is a 0 operand instruction. Other logical operations that we have is the rotate is one category of instruction. So, it rotates the contents of accumulator one position to right or

left. So, RLC; so, it will rotate the accumulator left ok. So, bit 7 goes to bit 0 and the carry flag. So, it is rotate left through carry ok.

So, bit 7 goes to the carry flag and this have a bits goes to the accumulator bit 7 goes to bit 0 and the carry flag.

(Refer Slide Time: 13:13)



I think there is some example after this RLC. So, this is what happens is that this bit 7, it goes to bit 0 and bit 7 also goes to the carry flag and this RAL instruction. So, this bit 7 goes to the carry flag and the carry flag comes to bit 0. So, this in here this carry flag also becomes a part of the accumulator as, and as if the rotation is done by taking this carry also carry bit also a part of the accumulator.

On the other hand this RLC, so, this is not taking this carry bit as a part of the accumulator though bit 7 is transfer to the carry flag, but this carry flag content is lost that way. So, this in the after the RLC instruction the carry flag content will be lost, but in case of RAL nothing is lost. So, the bit 7 goes is now stored in the carry flag and the previous carry bit is available in bit 0. So, all other bits are shifted ok. So, we have got the RLC instruction that way. So, this similarly we have got RRC and RAR. So, that is the rotate right. So, it will be rotated from the right end that bit 0 goes to bit 7 and carry flag and in case of RAR bit 0 goes to the carry flag and the carry goes to bit 7. So, they are there.

(Refer Slide Time: 14:32)

The slide has a yellow background with a title 'Logical Operations' at the top. Below the title is a bulleted list:

- Compare
 - Compare the contents of a register or memory location with the contents of the accumulator.
 - CMP R/M Compare the contents of the register or memory location to the contents of the accumulator.
 - CPI # Compare the 8-bit number to the contents of the accumulator.
 - The compare instruction sets the flags (Z, Cy, and S).
 - The compare is done using an internal subtraction that does not change the contents of the accumulator.

Handwritten notes are present on the right side of the slide:
 - A - B
 - CMP B
 - A ↔ B
 - CMP R/M
 - A ↔ M[+L]
 - CPI 20H
 - A ↔ 20H

At the bottom left, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. At the bottom right, there is a video player showing a man speaking.

So, we have got the logical operation. So, in case of logical operations we compare the contents of a register or memory location with the contents of the accumulator. So, CMP is the instruction. So, we can talk about CMP R register or memory. So, we can have instructions like say compare B. So, so, A and B register. So, they will be compared or I can say like compare M. So, the A and this memory location HL; so, they will be compared A and memory location H L they will be compared. So, this way we can have this compare instruction then we can have this compare immediate, so CPI so that compare immediate. So, we can have an instruction like CPI say, say twenty hex. So, here the A register will be compared with the immediate value 20 hex ok.

So, it compares the 8 bit value with the accumulator. So, the compare instruction it sets the flags like 0 flag carry flag and sign flag. So, they are set they are this compare instruction will set them. So, if the 2 contents are same, then this 0 flag will be set ok. So, if 1 is more than the other than the carry flag will be set or the sign flag will be set. So, comparison is done using an internal subtraction that does not change the contents of the accumulator.

So, internally we will be doing a whether processor does A - B. So, if it is compare B. So, it will be doing A - B and depending upon the result. So, if A and B are same then the result will be 0 in that case, the 0 flag will be set if A is greater than B, if says if A is greater than B, then this carry flag will be set and if A is less than B, then the sign flag will

become negative and the sign the sign operation result is negative and the sign flag will be set.

So, after this if we check this 0, carry and sign flag. So, we can understand; what was the outcome of the comparison. So, accordingly we can say it is greater than, less than or equal to ok. So, after doing this comparison we can put that type of condition checks and that is done by looking into the status flags.

(Refer Slide Time: 17:02)

Branch Operations

- Two types:
 - Unconditional branch.
 - Go to a new location no matter what.
 - Conditional branch.
 - Go to a new location if the condition is true.

Next will be looking into another category of instructions which are known as branch instructions; so, there are this branch categorization. So, you can broadly divided two categories the unconditional branch and conditional branch. So, in case of unconditional branch; so, it tell the when this instruction is executed. So, the processor is told that you go to the new address irrespective of what was the previous value, like if it is, so, suppose at present the processor was executing instructions in memory.

So, this is the this is the memory and say it is executing the instruction at location thousand and here the instruction is jump to 2000 in that case whatever be the content of the next instruction. So, the processor will simply ignore that and it will start executing the instruction from location 2000, it will go to the location 2000 and it will start executing from this point onwards ok.

So, that is why it is called an unconditional branch. So, there is no condition involved in the process ok. So, what the processor internally does is that after getting this instruction and after decoding this instruction the simply the program counter gets loaded with the value 2000. So, that is why the next instruction executed is from location 2000. So, there is a unconditional branch and there is a conditional branch also where some condition code is true, then only it will be going to that address. For example, if I have got an instruction like compare B and then if I say the jump on equal (JE 2000).

So, if this after in this comparison if A and B register values were same then it will be jump on equal condition will be true the 0 flag will be set. So, equality condition is true. So, in that case only; so, it will be jumping to the location 2000 otherwise, it will continue from here if the condition is valid, then it will start executing from this point if the if A and B are not same, in that case, it will execute the very next instruction and proceed like that. So, that is the conditional branch and previously what we saw that is an unconditional branch.

(Refer Slide Time: 19:39)

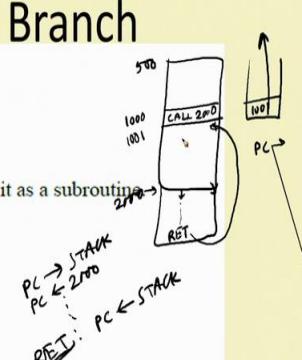
Unconditional Branch

- JMP Address
 - Jump to the address specified (Go to).

- CALL Address
 - Jump to the address specified but treat it as a subroutine.

- RET
 - Return from a subroutine.

- The addresses supplied to all branch operations must be 16-bits.





IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

So, next we will be so, this is an unconditional branch type of instruction like jump followed by address. So, this is the address is a 16 bit address. So, it jumped to the address specified to, something like a go to that we have in some high level programming language. So, there is another unconditional branch which is known as the call instruction. So, call

instruction is similar to the procedure call that we have in high level languages. So, here call address, call followed by a 16 bit address.

So, this is this new 16 bit address is a subroutine. So, so it is like this. So, suppose this is the main program that it was executing and then my, this is this is the loaded from say memory location 500 onwards. So, when it came to the memory location thousand ok. So, it is calling a subroutine. So, this is a subroutine call 2000. So, the subroutine is a small routine. So, which is loaded from location 2000; so, it starts at location 2000. So, what the processor will do when it gets this call 2000. So, it will jump to the location 2000 and it will start executing from this point onwards.

So, as it is executing when this subroutine is over, so, at the end of the subroutine there will be a return instruction and when this return instruction is encountered then the control will be going back to this point that is 1001. So, control will be going back to the location 1001 ok. So, this is the way the call and return instructions are executed. So, return from subroutine and going into subroutine. Now this is exactly the point where these stack come into picture like when it was going to this subroutine. So, in the stack it had set this return value 1001 and when it came to this return statement. So, it popped out this value from the stack 1001 and loaded it into the PC.

So, while going from while coming from this call 2000 to this; so, it set the content of the program counter which was equal to 1001 into the stack. So, the PC value was this PC value was copied on to the stack and then PC is loaded with the value 2000 and after sometime when the return instruction is encountered then at that time the PC value is loaded from the stack.

So, that it goes back to this 1001. This is how this return statement and this call statement they are executed. So, they are also some sort of branching. So, they are all unconditional branching that we have. Now other instructions that are possible.

(Refer Slide Time: 22:36)

The slide has a yellow background with a black header bar containing icons. The title 'Conditional Branch' is centered in a large, bold, black font. Below the title is a bulleted list of conditional jump instructions:

- Go to new location if a specified condition is met.
- JZ Address (Jump on Zero)
 - Go to address specified if the **Zero flag is set**.
- JNZ Address (Jump on NOT Zero)
 - Go to address specified if the **Zero flag is not set**.
- JC Address (Jump on Carry)
 - Go to the address specified if the **Carry flag is set**.
- JNC Address (Jump on No Carry)
 - Go to the address specified if the **Carry flag is not set**.
- JP Address (Jump on Plus)
 - Go to the address specified if the **Sign flag is not set**.
- JM Address (Jump on Minus)
 - Go to the address specified if the **Sign flag is set**.

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video thumbnail of a man speaking.

So, these are these are the conditional branch. So, go to new location if some specified condition is made. So, JZ jump on 0. So, go to address specified if the 0 flag is set similarly. So, 0 flag will be set as we know that if the value the previous comparison was equal. Similarly, JNZ jump on not 0. So, this jump on not 0, it will be true if the address is if the condition if the values were not same. So, that is non zero. Jump on carry, so, this is the value of comparison first operand was larger.

So, that is the carry flag set. So, jump on carry. So, similarly jump on no carry, jump on plus ok. So, if the sign flag is not set and jump on minus if the sign flag is set. So, these are the various jump condition, conditional jumps that we have in case of 8085 ok; so, using the carry, 0 and sign flags.

(Refer Slide Time: 23:38)

The slide has a yellow background with a title 'Machine Control' at the top. Below the title, there are two sections: '- HLT' and '- NOP'. The 'HLT' section contains one bullet point: '• Stop executing the program.' The 'NOP' section contains three bullet points: '• No operation', '• Exactly as it says, do nothing.', and '• Usually used for delay or to replace instructions during debugging.' To the right of the text, there is a small diagram of a rectangle with two arrows pointing from the bottom right towards the center, each labeled 'NOP'. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Some machine control instructions like halt. So, it will stop the execution of the program. So, this I have told some earlier. So, halt will halt the execution of this statements by the processor and it will be waiting till the processor is woken up by means of some interrupt or some external procedure.

Then another instruction which is there is known as the no operation or NOP instruction. So, it as the name suggests; so NOP is do nothing. So, it does nothing. So, apparently since why do we need this thing. So, as it is not doing anything so, but this instruction is sometimes useful because you want to delay you want to introduce some delay into a sequence of operation for example, it may so happen that you want them the next instruction to be executed after a small delay of some fractions. So, you can put if you NOP statement. So, that some small delay gets introduced into the system.

So, that is one utility another utility of this NOP instruction is for debugging. So, so, when you are developing a program. So, it may have some bugs. So, what we do we put some debugging statements in between in the, in the develop program, so, that we can halt we can we can stop the execution of the program in between do some checking of this register contents and all and then resume the instruction. So, what happens is that if this is the program then in between we put some special instruction that help us in the debugging operation. Now after the debugging is over now what is done is these instructions which are basically some sort of interrupt, so, they are modified to NOP. So, that in the final

version there is no debugging instruction included. So, they are all NOP instruction. So, this so, we can very easily modified this debugging instructions by NOP instruction. So, without hampering the code which is otherwise generated; so, all the addresses and all they remain unchanged and only this parts they are modified to NOP.

So, this way we can have some machine control instructions with the halt, NOP etcetera, so that are going to be helpful.

(Refer Slide Time: 25:59)

The slide has a yellow background with a black header bar containing various icons. The title 'Operand Types' is centered in a large, bold, black font. Below the title is a bulleted list of four types of operands:

- There are different ways for specifying the operand:
 - There may not be an operand (**implied operand**)
 - CMA
 - The operand may be an 8-bit number (**immediate data**)
 - ADI 4FH
 - The operand may be an internal register (**register**)
 - SUB B
 - The operand may be a 16-bit address (**memory address**)
 - LDA 4000H

At the bottom of the slide, there is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it features the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a small video window showing a man speaking.

Now, depending upon the operand so that there can be different types of operands therefore, example with the implied operand like CMA. So, this instruction is 0 operand instruction the operand the implied operand is the memory we have got this 8 bit number the immediate data. So, like ADI; so, ADI this is ADI 4FH. So, this is basically this addition operation with the immediate operand then we can have the some internal register or register operand. So, subtract B. So, the B is the register or it may maybe a memory address. So, this is LDA 4000 hex content of memory may be a four thousand hex will be loaded. So, so this way, I can have different types of operand; implied operand, immediate operand, register operand and memory address.

(Refer Slide Time: 26:50)

Instruction Size

- Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.
 - Typically, all instructions occupy **one byte** only.
 - The exception is any instruction that contains **immediate data** or a **memory address**.
 - Instructions that include immediate data use **two bytes**.
 - One for the opcode and the other for the 8-bit data.
 - Instructions that include a memory address occupy **three bytes**.
 - One for the opcode, and the other two for the 16-bit address.

CMA → 8 bit
MVI A → 40H
↓
8 bit 8 bit
LXI H → 16 bit
LXI D → 16 bit

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, depending upon the operand type; so, the instruction size will vary. Some of the instructions are only one byte long some are more number of bytes. So, for example, the instruction CMA that we have; so, CMA there is an 8 bit code for that as we know that all opcodes are 8 bit. So, it is only one byte long on the other hand. So, if I, if I take this instruction like MVI A, 40 hex, then this for this MVI A part.

So, I need to hold there is a code for this part. So, that this is an 8 bit code and then this 40 hex is another 8 bit number. So, total I will require 2 bytes. So, for holding this MVI A instruction. Or if I have this LXI H, A 16 bit value, then this LXI H parts, so, this will constitute the first byte and the next this 16 bit part.

So, they will be taking two bytes ok. So, total instruction is three byte long. So, in case of 8085 some of the instructions are one byte long, some of them are 2 bytes long, some of them are 3 bytes long so, the instructions which are which are not having any operand. So, they are 0 operand instruction, they occupy only 1 byte some instructions, they are some immediate data or a memory address. So, they will be having for immediate data that is the 2 byte instruction and if you have got a memory of a memory address so that is a 3 byte instruction.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 54
8085 Microprocessor
(Contd.)

The other important factor when we are considering a program stored in the memory in case of a Microprocessor Based System is the instruction size. That is how many bytes and instruction occupy when it is, are occupies when it is put into the memory. So, that will determine the overall size of the program.

(Refer Slide Time: 00:35)

Instruction Size

- Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.
 - Typically, all instructions occupy **one byte** only.
 - The exception is any instruction that contains **immediate data** or a **memory address**.
 - Instructions that include immediate data use **two bytes**.
 - One for the opcode and the other for the 8-bit data.
 - Instructions that include a memory address occupy **three bytes**.
 - One for the opcode, and the other two for the 16-bit address.

So, depending upon the operand type, the instruction may have different sizes. Now it will occupy different number of memory bytes. So, typically all instructions occupy one byte only. So, particularly that is for the opcode parts, so this every instruction has got an operator operation code or opcode. And that opcode is generally coded as an 8 bit data.

As you know that in a microprocessor, so, we have got 256 the possible opcodes that can be there of course, microprocessor 8085 is much less than that. But typically, so the opcode part will occupy 1 byte. The exception is because of this immediate data or memory address. Like you some instruction they have got an immediate data like MVI, MVI move immediate that has got an immediate data or if you, if you are mentioning memory address.

So, basically these two type of instructions like this MVI B, say 49 hex. So, this MVI B part, so that occupies 1 byte and the 49 hex this number occupy the another byte. So, that way it becomes a 2 byte instruction similarly this LXI instruction for example, LXI H, say 3523 hex. So, this will again this will occupy three byte because its LXI H part. So, this will take 1 byte and this 3523 so this is one 4 digit hexadecimal number. So, 2 hexadecimal digits can be put into 1 byte, so total 2 bytes are needed for the hexadecimal address part itself.

So, that way we can have three address 3 bytes. So, instructions that include immediate data use 2 bytes; one for opcode one for 8 bit data. And the instructions that include a memory address occupy 3 bytes one for opcode and other 2 for the 16 bit address that you want to put.

(Refer Slide Time: 02:28)

Instruction with Immediate Data

- Operation: Load an 8-bit number into the accumulator.
 - MVI A, 32
 - Operation: MVI A
 - Operand: The number 32
 - Binary Code: 0011 1110 3E 1st byte.
 0011 0010 32 2nd byte.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Quick Heal Total Security
Updated successfully.

So, like this as I have already said that MVI A, 32. So, that way, so this is the code of that is instruction like the manual of 8085 if you consult you will see that this is the code for the for the MVI instruction that is hexadecimal number. So, this the number 32 in decimal.

So, this is the, this is the sorry this is the opcode part, so 3E. So, 3E is the hexadecimal number corresponding to the opcode MVI A and then the next number should be 32. So, 32 this is the corresponding binary coding. So, this first byte is 3E in hexadecimal second byte is 32 in hexadecimal, so that way it is a 2 byte instruction.

(Refer Slide Time: 03:11)

Instruction with a Memory Address

- Operation: go to address 2085.
- Instruction: JMP 2085
 - Opcode: JMP
 - Operand: 2085
 - Binary code:

1100 0011	C3	1 st byte.
1000 0101	85	2 nd byte
0010 0000	20	3 rd byte

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, similarly this jump instruction is an example where we have got a memory address and the instruction turns out to be a three byte instruction, first part is the jump. So, that opcode is C3 and then 2085, so that is a two byte that will be required for the 16 bit address that you want to mention. And you notice here that this lower order byte that comes first and then the higher order byte. So, while coding it. So, it could C3 followed by 85 followed by 20. So, in first memory location will have C3, second location it will have 85, third location will have 20.

So, when it is doing the instruction fetch. So, in the first phase it will get C3 and in the successive phases it will get 85 and 20. And then it will then the processor will make a jump to that particular address.

(Refer Slide Time: 04:02)

- The microprocessor has different ways of specifying the data for the instruction. These are called “addressing modes”.
- The 8085 has four addressing modes:
 - Implied CMA
 - Immediate MVI B, 45
 - Direct LDA 4000
 - Indirect LDAX B

A ← m[4000]
A ← m[BC]

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

As far as, addressing modes are concerned, so addressing modes actually tell like, how many ways you can specify the data that that an instruction will process. So, there are 4 addressing modes like first one is the implied addressing mode, where the operand or the data is implied like CMA compliment accumulator. So, the instruction itself tells that the operation will be performed on the accumulator. So, that is the CMA implied type of operand, we have got immediate operand immediate data like MVI B, 45 hex.

So, this here in the instruction itself I am telling what is the immediate value that you want to move to B. So, this 45 is the immediate value then we have got direct addressing. So, in direct addressing, so we directly specify some memory address like the instruction LDA 4000. So, what this instruction does is that this A register the accumulator will get the content of memory location 4000.

So, this is LDA instruction by which you can load the accumulator or it can be an indirect of, indirect data like LDAXB. So, this says this means that the A register will get the content of memory location whose address is given by the pair BC. So, in the first case the LDA instruction, so we are directly specifying the address. In the second case LDA instructions. So, we are specifying the as specifying some sort of a pointer. So, BC registering pair is acting as a memory pointer in this case.

(Refer Slide Time: 05:46)

The screenshot shows a presentation slide titled "Data Formats". The slide content is as follows:

- In an 8-bit microprocessor, data can be represented in one of four formats:
 - ASCII
 - BCD
 - Signed Integer
 - Unsigned Integer
- It is important to recognize that the microprocessor deals with 0's and 1's.
 - It deals with values as strings of bits.
 - It is the job of the user to add a meaning to these strings.

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES". The slide number "72" is also visible.

So, as far as the data formats are concerned, so, in this is an 8 bit microprocessor, data can be represented in one of this 4 format which can be ASCII, so, where you are just storing the characters, it can be BCD Binary Coded Decimal, it can be signed integer. So, integer with sign and unsigned integer so it does not consider sign. So, since if the data is 8 bit. So, a signed integer we can go for from - 128 to + 127. Then your unsigned integer can go from 0 to 255 and your this ASCII and BCD, so they are ASCII is 7 bit, but it is extended to 8 bit and BCD is binary coded decimal.

So, since we need 4 bits for coding one decimal number decimal digit between 0 and 9 so, in one in 8 bits, so we can code to decimal digits. So, it is often represented as binary coded decimal format. It is important to recognize that the microprocessor deals with zeros and ones and it deals with values as string of bits. So, it always handles the data as string of bits. And it is up to the user to understand the meaning of what this string will correspond to.

Like when you are putting it on set of LED, so, it may be representing a pattern of glowing of those LED 's; if you are putting it on a some display, then maybe it is a character that comes out of that that may correspond to the ASCII number, ASCII character that you want to display. So, this way this data formats will vary.

(Refer Slide Time: 07:26)

The screenshot shows a presentation slide titled "Data Formats". The slide content is as follows:

- Assume the accumulator contains the following value: 0100 0001. There are four ways of reading this value:
 - It is an unsigned integer expressed in binary, the equivalent decimal number would be 65.
 - It is a number expressed in BCD (Binary Coded Decimal) format. That would make it, 41.
 - It is an ASCII representation of a letter. That would make it the letter A.
 - It is a string of 0's and 1's where the 0th and the 6th bits are set to 1 while all other bits are set to 0.

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES". The slide has a page number "73" in the bottom right corner.

Now, for example, this accumulator, suppose the accumulator has gone content 0 1 0 0 0 0 0 1. So, there are 4 ways by which you can interpret this result. So, if you are looking for an unsigned integer expressed in binary. So, this is the corresponding integer number 65. If you are looking it as BCD, in binary coded decimal format, the first 4 bits will correspond to 1 digit and the next 4 bits will correspond to the next digits. So, it is 41. If you are looking for an ASCII character, then this the 65 or this 41 hex you know that this correspond to the ASCII letter A uppercase A character.

So, let us see that is letter character A. If you are looking it is a string of 0's and 1's where 0th and the 6th bits are set to one and rest are all 0. So, this may be the interpretation when you are looking it as a bit string. So, this way we can have different interpretations of this of this bit strings that we have in the stored in the microprocessor.

(Refer Slide Time: 08:34)

Counters

- A loop counter is set up by loading a register with a certain value
- Then using the DCR (to decrement) and INR (to increment) the contents of the register are updated.
- A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

74

Next will be looking into something like a counter like many a times what happens is that you need to set some counter for all loading a register with some value. So, and then we can decrement that counter and put some generate some sort of repetitive action out of that. So, we put, we initialize some register to some value. And then we use the decrement and implement instructions to decrement or implement the register ok, that that has got the value.

So, DCR and INR, so these are the two instruction that we have we have in that purpose. A loop is set up with conditional jump instruction that loops back or not depending on whether the count as this termination count or not. For example, if you start with set all 0 and say that I will count up to 100. So, when they, after every increment the value increases by 1.

So, when it reaches 100, so you can do termination. So, you can do a compare whether you have reach the value 100 or not or other way maybe start with 100 and check for the condition that it becomes 0. So, by means of decrement instruction, so we decrement the value and then when it comes to 0 we stop the counter.

(Refer Slide Time: 09:50)

```
graph TD; Initialize[Initialize] --> Body[Body of loop]; Body --> Update[Update the count]; Update --> Decision{Is this Final Count?}; Decision -- No --> Body; Decision -- Yes --> Exit
```

The slide also features the IIT Kharagpur logo and the NPTEL Online Certification Courses logo at the bottom.

So, the overall operation is like this, first you have to initialize the counter, and then we have to have this body of the loop, we will have the body of the loop, whatever with action that you want to do. Then update the count, and then we check whether this is final count or not. If it is so then it comes out if it is not it goes back executes the loop body once more.

(Refer Slide Time: 10:16)

```
MVI C, 15H  
LOOP      DCR C  
        JNZ LOOP
```

The slide also features the IIT Kharagpur logo and the NPTEL Online Certification Courses logo at the bottom, and a small video window in the bottom right corner showing a speaker.

A sample of assembly language program for implementing this loop using decrement instruction DCR instruction is like this, we move immediate C, 15 hex. So, if you have C register gets the value 15 hex then we decrement C and jump on not 0 to loop.

So, we are decrementing the value of C, so then it will become 14 hex, 13 hex, 12 hex like that. So, till the values are not 0 the control will be coming back to this decrement instruction and when this value becomes 0 then it comes out of the loop. So, this way this program as such this assembly language program fragment is doing nothing more than waiting for some time ok. So, one way to wait for some time is by means of this N O P or NOP instruction and other way can we by means of putting this type of loops which are also known as delay loops. So, you put some delay into the operation of the system.

(Refer Slide Time: 11:15)

Register Pair as a Loop Counter

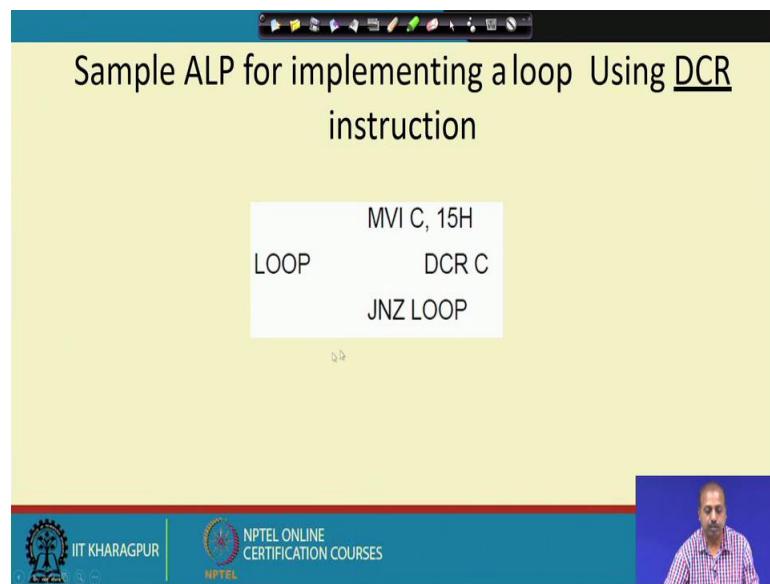
- Using a single register, one can repeat a loop for a maximum count of 255 times.
- It is possible to increase this count by using a register pair for the loop counter instead of the single register.
 - A minor problem arises in how to test for the final count since DCX and INX do not modify the flags.
 - However, if the loop is looking for when the count becomes zero, we can use a small trick by ORing the two registers in the pair and then checking the zero flag.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, sometimes what happens is that this delay that we produce is not sufficient because this, if you are using an 8 bit register for to hold the count then you can get a maximum delay of 255 units ok. So, you can calculate out of the 255 units the delay that is produced may not be sufficient. So, when you are when you are looking for some periodic task to be done. So, add some periodic intervals we will do the operation.

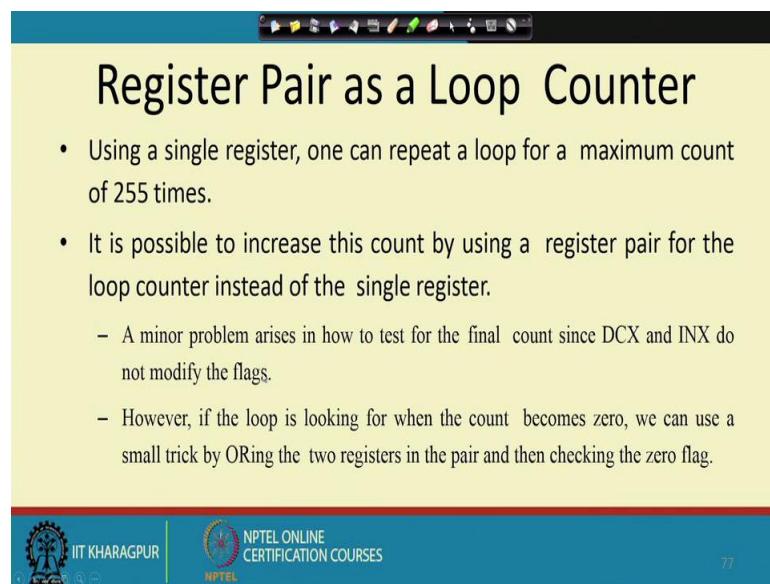
So, that small delay may not be sufficient, so for that purpose what we do we use some register pair. So, it is you use the register pair to increase the count value that is possible ok.

(Refer Slide Time: 12:04)



So, but there is a one small problem like in the previous instruction we have seen that after this decrement C we could do jump on not 0 because whenever this DCR instruction it affects the 0 flags. So, if the content becomes 0 then this the 0 flag will be equal to 1.

(Refer Slide Time: 12:17)



But in case of this loop pair, so what we have to do is that we have to use this DCX and INX instruction for register pair decrement and increment, and this register pair increment/decrement operations they do not affect the status flag.

So, this is by the design of the 8085 processor. So, we cannot answer why is it happening like this, but this by the design of the processor. So, if you are using this 8085 processor and you are using 16 bit count value decrementing or incrementing by DCX INX instructions, then we have to be a bit careful. So, if the loop is looking for when the count becomes 0, so we can use a trick like what you do we OR the content of 2 resistors in the pair and then check the 0 flag like this.

(Refer Slide Time: 13:09)

Register Pair as a Loop Counter

- The following is an example of a loop set up with a register pair as the loop counter.
- LXI B, 1000H
- LOOP DCX B →
- MOV A, C →
- ORA B
- JNZ LOOP

Handwritten note:

BC = 1000H
 $B=10$
 $C=00$
 $A=FF$
 $B \rightarrow 0FFF$
 $C=00$
 $A=FF$
 $O\bar{F}$
 \overline{FF}

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, suppose we want to have a delay of this 1000 hex ok. So, this 1000 hex we move it to this B register B register pair. So, as a result the BC pair gets the value the BC pair gets value 1000 hex or rather this B register gets the value 10 and C register gets the value 0 0. So, that is the situation now this DCX B instructions.

So, this will directly operate on this 1000 hex instruction value and after doing this decrement by one the value become 0FFF ok, then after. So, then I need to check whether the content is 0 or not, but the problem is when this DCX instruction does this. So, it does not affect the status flags ok. So, I need somehow some technique by which I can get the status flags affected and check whether the content has become 0 or not.

So, for that purpose the trick that is taken is MOV A, C. So, now, A register gets the value of this C register that is FF. So, this is the C register now and this is the B register now, content of B register and C register. So, C register A register gets the value FF and then

we do OR accumulator with B register, so with this FF and OF they are ORed, FF and OF are ORed.

So, when you do this OR operation, so you get the result FF, but this OR instructions, so this will affect the status flags; and you know that this FF it is not affecting the, it is not equal to 0, so the 0 flag is not set. So, this jump or not 0 condition will be true and then it will be jumping back to this DCX instruction. So, this way we can have register pair as a as the counter value, so that we can get larger delays.

(Refer Slide Time: 15:19)

The slide has a yellow background and a blue header bar with various icons. The title 'Delays' is centered in a large, bold, black font. Below the title, there are two bullet points:

- Knowing how many T-States an instruction requires, we can calculate the time using the following formula:
 - Delay = No. of T-States / Frequency
- For example a “MVI” instruction uses 7 T-States. Therefore, if the Microprocessor is running at 2 MHz, the instruction would require 3.5 μ Seconds to complete.

At the bottom of the slide, there is a red footer bar. On the left, it features the IIT Kharagpur logo and the text "IIT KHARAGPUR". In the center, it features the NPTEL logo and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right, there is a small video window showing a man speaking.

So, but, so far I have talked about delays, but I did not explicitly mention the time part of it I just we just had a notion that there is a delay introduced, that is coming in terms of this counter value that we have put into the loop. Now how to calculate the delay part? So, if for every instruction I can know how many clock cycles it takes. So, you know that 8085 it operates that some clock frequency for example, we can take it at to operate 2 megahertz ok.

So; that means, will it will for every clock cycle what is the duration, so 1 upon 2 megahertz. So, that is the duration for every instruction. So, that much is the for every clock cycle. So, every clock cycle is like that now if you for every, if you look into the manual of 8085, so, you will find that for every instruction its size and its number of clock cycles are mentioned.

So, from there you can calculate how many clock cycles are required and from there you can come to the, you come to the time needed for executing particular instruction. So, if for a particular program, so if you calculate the total number of clock cycles that is needed and you divide it by the frequency. So, you can get the delay that will be involved in executing the program fragment. So, if you are looking at a single instruction level for example, this MVI instruction it has 7T state. So, T state means each clock periods, so how much time each clock period is 1T state.

So, that is 7T states and if you assume that this microprocessor is running at 2 megahertz, then this instruction will required 3.5 micro second. So, this is 7 divided by 2 megahertz. So, that is 3.5 micro second to complete. So, MVI instruction requires about requires 3.5 micro second and that is exact ok. So, it is not dependent on any other factor, so that way it is exact calculation.

(Refer Slide Time: 17:24)

Delay loops

- We can use a loop to produce a certain amount of time delay in a program.
- The following is an example of a delay loop:

MVI C, FFH	7 T-States
LOOP DCR C	4 T-States
JNZ LOOP	10 T-States

- The first instruction initializes the loop counter and is executed only once requiring only 7 T-States.
- The following two instructions form a loop that requires 14 T-States to execute and is repeated 255 times until C becomes 0.

80

Now, if we are writing a loop like this MVI C , FFH, DRC and JNZ loop. So, much what is the delay that is produced so, first this loop initiation, so that is MVI instruction that required 7 clock cycles. Then we have got a decrement C again you refer to the manual of 8085 and you can find that decrement operation it requires 4 clock cycles. So, that is 7 + 4 and this JNZ loop, so this requires 10 clock 10 clock cycles ok. So, the, so you see it is quite obvious because this MVI instruction.

So, it offers the in the first byte it will get the opcode for MVI in the second cycle, so it will get the value FFX. So, then that is 4 + 3 fetch always takes 4 clock cycle or 4 T states. So, 4 + 3, 7 T states the similarly this decrement C of instructions it has got only the fetch part it does not have any other ones it has got the instruction you will decrement it.

So, that way number of clock cycles needed is 4 and this JNZ loop. So, loop is 16 bit value, so I need to 4 clock cycles to get the JNZ part and then 3 + 3 clock cycles for the getting the 16 bit address parts. So, total 10 T states will be needed for this JNZ instruction. So, the first instruction initializes the loop counter and is executed only ones requiring 7 T states, the following two instruction the decrement and JNZ. So, they form a loop that requires 14 T states taken this these two things together and they are executed 255 times until C becomes equal to 0.

(Refer Slide Time: 19:14)

Delay Loops (Contd.)

- We need to keep in mind though that in the last iteration of the loop, the JNZ instruction will fail and require only 7 T-States rather than the 10.
- Therefore, we must deduct 3 T-States from the total delay to get an accurate delay calculation.
- To calculate the delay, we use the following formula:

$$T_{\text{delay}} = T_0 + T_L$$
 - T_{delay} = total delay
 - T_0 = delay outside the loop
 - T_L = delay of the loop
- T_0 is the sum of all delays outside the loop.

\downarrow
 JNZ (4-bit)
 Success → 10
 fail → 7

IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES

 A video feed showing a person speaking.

So, from this you can calculate the time needed. So, so we need to keep in mind that in the last iteration of the loop the JNZ instruction will fail and you required 7 T states rather than 10 T states, because actually the way it executes is first it gets that JNZ part. So, JNZ and the 16 bit value, so this JNZ. So, this way for the when the it if you look into the manual it will find that there are two types or types of timing requirement, one is the loop is successful loop is successful another case the loop is fail it does, so JNZ condition is false.

So, if the loop is successful in that case it requires 10 clock cycles, if the loop is a failure then it requires 7 clock cycles or 7 T states. So, the way we should calculate is that the total delay is that, we should subtract 3 T states on the total delay to get an accurate delay calculation and to calculate the delay.

So, we compute the total delay is $T_0 + T_L$ where T_{delay} is the total delay, T_0 is the delay outside the loop that is initialization of the counter and all and T_L is the delay of the loop. So, this is the delay of the loop and T_0 is the sum of all these loops. So, once you do that, so you can get the actual delay part.

(Refer Slide Time: 20:44)

- Using these formulas, we can calculate the time delay for the previous example:
- $T_0 = 7 \text{ T-States}$
 - Delay of the MVI instruction
- $T_L = (14 \times 255) - 3 = 3567 \text{ T-States}$
 - 14 T-States for the 2 instructions repeated 255 times ($FF_{16} = 255_{10}$) reduced by the 3 T-States for the final JNZ.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, what we have, so this using this formulas. So, we can calculate the time delay of the previous example T_0 that is the MVI instructions. So, that took 7 T states that is the delay of the MVI and the T_L parts. So, that is 14 T states were necessary and that is repeated 255 times. So, $14 \times 255 - 3$ because for the last part, so this JNZ when it is failing. So, it takes only 7 clock cycles not 10 clock cycle or 3 clock cycles they are less.

So, that way this becomes 3567 T states, so 14 T states for the two instruction repeated 255 times reduced by 3 T states for the final JNZ. So, that gives us the total time needed and the total time needed is this is T loop parts.

(Refer Slide Time: 21:33)

Register Pair as a Loop Counter

- The following is an example of a delay loop set up with a register pair as the loop counter.

LXI B, 1000H	10 T-States
LOOP DCX B	6 T-States
MOV A, C	4 T-States
ORA B	4 T-States
JNZ LOOP	10 T-States

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this, if you say that total delay. So, T delay, so T delay is going to be equal to this 3567 + T that is 3547 T states and then you know depending upon the frequency of operation, so, you can calculate what is the overall time needed. So, this is I have got here $3567 + 7$ equal to 3574 clock cycles. So, this 3574 divided by 2 megahertz, so if I say that I will be operating at 2 megahertz. So, this is your equal to 1787 microsecond.

So, that will be the total delay produced by the previous program. So, if for the, so if the program is complex then it will take more time. So, if you are using register pair as a loop, loop counter then what will happen? Suppose, this is the program LXI B then DCX B, MOV A, C, ORA B or accumulator B and JNZ loop. So, this LXI takes 10 T states I have this part of the, it is part of information as I have already said is available from the manual. Then this DCX B instruction takes 6 clock cycles, MOV A, C takes 4 clock cycles, or accumulator B it takes 4 clock cycle, JNZ it takes 10 clock cycles.

(Refer Slide Time: 23:14)

Register Pair as a Loop Counter

- Using the same formula from before, we can calculate:
- $T_0 = 10$ T-States
 - The delay for the LXI instruction
- $T_L = (24 \times 4096) - 3 = 98301$ T-States
 - 24 T-States for the 4 instructions in the loop repeated 4096 times ($1000_{16} = 4096_{10}$) reduced by the 3 T-States for the JNZ in the last iteration.

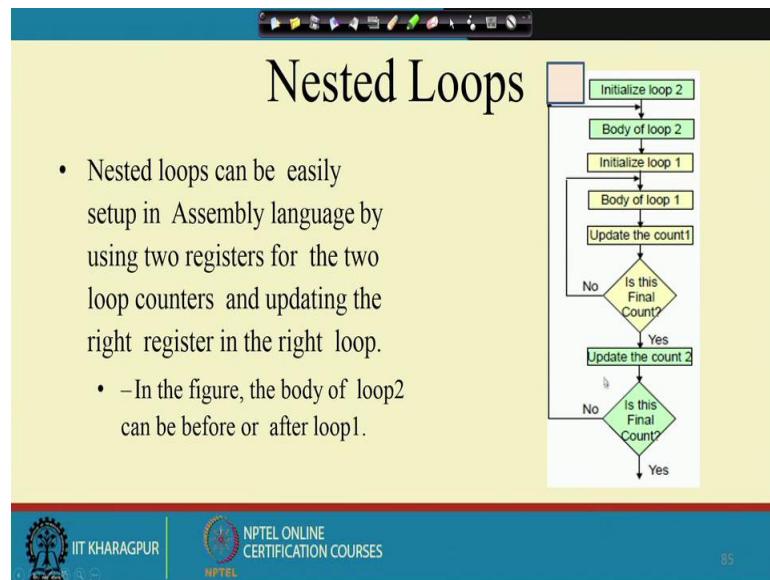
$\frac{98311}{2^{32}} = 49155.5\text{ }\mu\text{s}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, based on this we can calculate what is the total delay that is required. So, T outside is equal to 10 T states for the LXI instruction; T loop body, so, that is 24 because here the total size becomes $6 + 4 + 4 + 10$, so total is 24. So, 24×4096 that is the, that is the value that I have initialized, so 1000 hex that is 4096 in decimal, minus three that is for the last clock cycle. So, I do not do this JNZ instruction requires only 7 clock cycles or 3 clock cycles and less. So, total is 98301 T states. So, 24 T states of a four instructions in the loop repeated 4096 times reduced by 3 T states for the JNZ in the last iteration.

So, total time is $98301 + 10$, so 98311 so based on that. So, you can find out what is the total delay. So, this is your 98311 divided by 2 megahertz. So, so that way it is 49155.5 microsecond. So, this is the delay that will be produced by the previous instruction previous example. Now if you look into if you want to produce more delay ok.

(Refer Slide Time: 24:51)



85

Sometimes this is not sufficient, so you want to put some nested loop and get the delay value increased. So, how do you do this? We initialize so, we use two nested loop. So, there is a loop 1 which is this part, so initialize loop 1 body of loop 1 update count one. And if it is not this is the final count then you go back and again execute the body of loop 1.

So, this is the inner loop and then we have got another loop over and above the inner loop. So, that updates the counter 2 and then whether it is this the final count or not, if it is not it will again do the a body of loop to it will go like this, so this way we can have two nested loops. So, nested loops can be easily set an assembly language by using 2 registers for the 2 loop counters and updating the right registered in the right loop.

So, this body of loop 2 can be before or after loop 1, so that can be there. So, that does not matter. So, what I mean is this body of loop 2, so this part. So, it can be put before loop 1 or it can be after this after this also, so it does not matter ok.

(Refer Slide Time: 26:08)

Nested Loops for Delay

- Instead (or in conjunction with) Register Pairs, a nested loop structure can be used to increase the total delay produced.

MVI B, 10H	7 T-States
LOOP2 MVI C, FFH	7 T-States
LOOP1 DCR C	4 T-States
JNZ LOOP1	10 T-States
DCR B	4 T-States
JNZ LOOP2	10 T-States

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now we can calculate what is the total delay that is required that is generated by the process so, you see that we have got say this particular program. So, MVI B , 10 hex, then MVI C , so these are the 2 counters that we have B and C are the 2 register pairs are the register pair now the inner loop is consisting of the C register.

So, this decrement C JNZ loop 1, so this will be doing this inner loop and then we have got the outer loop in the outer loop we decrement the D and JNZ loop 2. So, that way it is doing this outer loop now if we to concert the manual and determine the number of T states needed for different instructions. So, this is like this, so 7 7 4 10 10 4 10 so like that. So, these are the delays now how to calculate the total delay that is produced.

(Refer Slide Time: 27:03)

Delay Calculation of Nested Loops

- The calculation remains the same except that it the formula must be applied recursively to each loop.
 - Start with the inner loop, then plug that delay in the calculation of the outer loop
- Delay of inner loop
 - $T_{OI} = 7$ T-States
 - MVI C, FFH instruction
 - $T_{L1} = (255 \times 14) - 3 = 3567$ T-States
 - 14 T-States for the DCR C and JNZ instructions repeated 255 times (FF=255) minus 3 for the final JNZ

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can do it like this, so calculation remains same except that the formula must be applied recursively to each loop. So, start with the inner loop and then plug that delay in the calculation of the outer loop. So, that is how because the inner loop becomes a part of the body of the outer loop you can say. So, we start with the inner loop calculate the time needed there and then put it into the body of the outer loop. So, for this first initialization that MVI C , FFX that instruction, so MVI C , FFX, so that required 7 T states so, these are delay for the inner loop.

So, this is the outer of one and then this is then you have got T_{L1} . So, the loop of this inner loop body of this the inner loop this two. So, this $10 + 4, 14$ and then for the last loop this JNZ will be 7 T state. So, that way it is done $255 \times 14 - 3$ that is 3567 T states; 14 T states for the decrement C instruction and JNZ instructions that is repeated for 255 times and then minus 3 for the final JNZ instruction.

(Refer Slide Time: 28:19)

Delay Calculation of Nested Loops

- Delay of outer loop
 - $T_{O2} = 7$ T-States
 - MVI B, 10H instruction
 - $T_{L1} = (16 \times (14 + 3574)) - 3 = 57405$ T-States
 - 14 T-States for the DCR B and JNZ instructions and 3574
 - T-States for loop1 repeated 16 times ($10_{16} = 16_{10}$) minus 3 for the final JNZ.
 - $T_{Delay} = 7 + 57405 = 57412$ T-States
- Total Delay, $T_{Delay} = 57412 \times 0.5 \mu\text{Sec} = 28.706 \text{ mSec}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then for the outer loop, so you have got this outer loop outside the loop. So, that is the 7 T states because of this MVI B instruction and when you are computing the loop outer loop body then it is 16×14 for the 14 for this parts. So, outer loop has got this decrement B and this JNZ. So, this is $14 +$ whatever be the delay you needed for this loop 1. So, that will come, so that is put here, so $14 + 3574$.

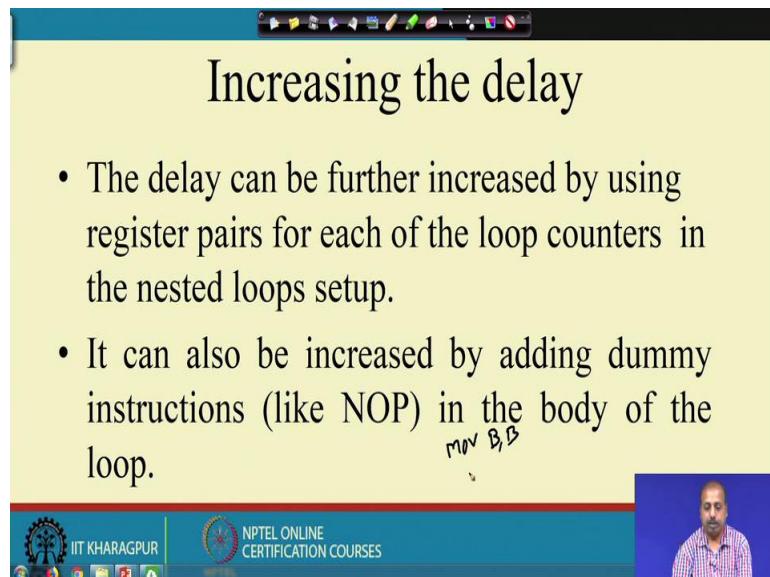
So, this outer loop, so that was 3567 that was $3567 + 7$ 3574. So, that 3574 is added and this whole thing minus 3. So, that makes it 57405 clock cycles, so total delay is this outer delay of 7 + 57405 T states. So, total 57412 T states. So, if we assume 2 megahertz operation then the total delay will be about 28.706 milliseconds. So, it is significantly high when you consider a single loop at a time. So, this way we can use nested loops for generating delays within the programs for microprocessors.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute Of Technology, Kharagpur

Lecture- 55
8085 Microprocessor
(Contd.)

So, to increase the delay, so delay can be increased further by putting some further instructions.

(Refer Slide Time: 00:21)



Increasing the delay

- The delay can be further increased by using register pairs for each of the loop counters in the nested loops setup.
- It can also be increased by adding dummy instructions (like NOP) in the body of the loop.

MOV B,B

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, one possibility is that we can increase the count value of this register pair that we have putting into. So, that way delay will increase significantly, sometimes what is required is that we can you do not need that much increase ok. So, it may be just very small amount of delay has to be inserted. So for that purpose we can put some instruction which does not have any effect, so like this NOP type of instruction. So, another so we can we can think about the different NOP type of instruction like you can say I put on instruction like MOV B , B. So, effectively does not have any operation that it is doing.

So, we does not do any do any complication are any movement as such. So, that way it is similar to the NOP, but anyway. So, this NOP is much more understandable, so that way many a many a time will put this NOP instructions inside the delay loops to increase the delay.

(Refer Slide Time: 01:24)

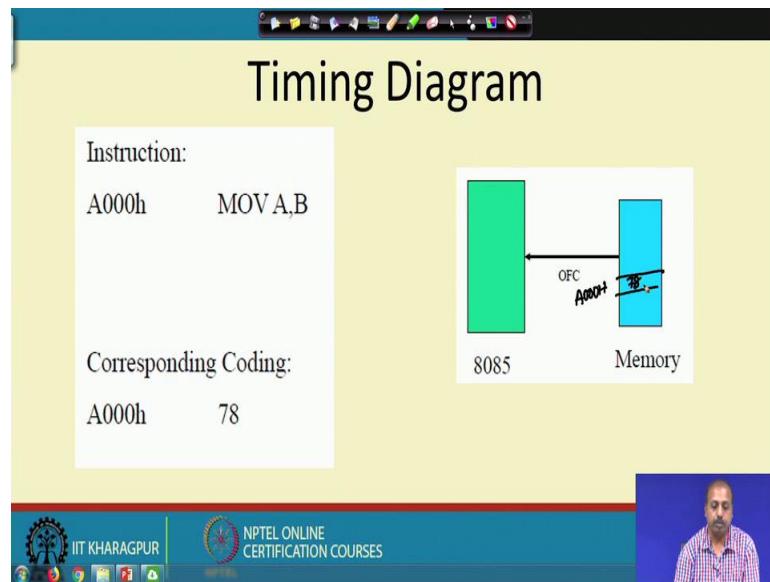
The screenshot shows a presentation slide titled "Timing Diagram". The slide content includes a bulleted list of signals that must be shown in a timing diagram for an 8085 processor. The signals listed are: Representation of Various Control signals generated during Execution of an Instruction; Following Buses and Control Signals must be shown in a Timing Diagram: Higher Order Address Bus, Lower Address/Data bus, ALE, RD, WR, IO/M.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

Next we will be looking into something, something called the timing diagram; So, this timing diagram this actually tells that when are the different control signals generated as we are proceeding through the operation of instructions. So, following buses and control signals must be shown in a timing diagram for if you the drawing the timing diagram of 8085. So, we must show these line higher order address bus, lower order address bus and data bus then the ALE signal, the read signal, write signal and $\text{IO}\bar{M}$ line.

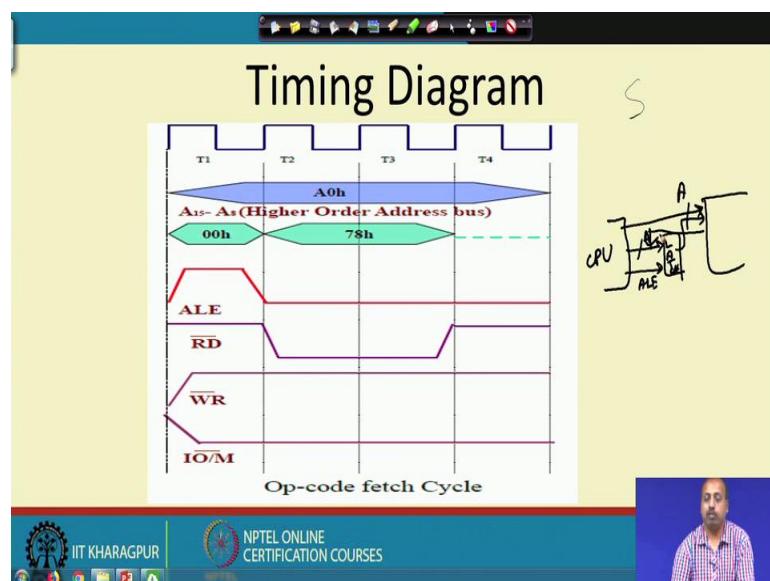
So, these are the minimum signals that you must show when you are trying to when you are asked to draw the timing diagram of any 8085 instruction, because that will tell us what a, how are the, how are the things going on inside the processor for executing the instruction.

(Refer Slide Time: 02:17)



So, let us consider this instruction `MOV A, B` ok, so here the opcode is 78 so, this opcode how, how this will be done. So, this is we assume that this instruction is that memory location A000 hex and that, so if you can say that if this is the location A000 hex, then here the 78 is available. Now, how is it going to be fetched to the processor and going to be executed, so that will see.

(Refer Slide Time: 03:06)



Now, so this is the timing diagram for this execution of this instruction. So, in the first clock as this requires only four clock cycles which is which forms a part of the opcode

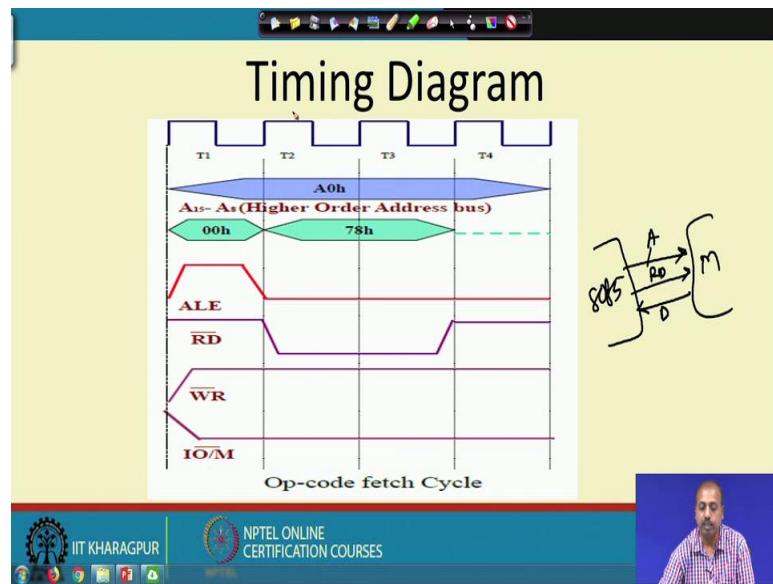
fetch cycle. So, in the, so this is the address bus content. So, this A15 to A 8 higher order address bus. It contents the higher order part of the address that is A0 and the next part the so this the lower order address part A0 to A7. So, they are contained in this the lower order address bus cum data bus.

So, that is 00 hex and when this lower order data is lower order address is available on the lower order address bus this ALE signal is activated ok. So, this ALE signal is activated as we know that there is a latch outside the processor and it is expect that if this is the processor and then there is a latch here. So, this lower order address bus comes here and the ALE signal goes there. So, ALE signal is used to latch this lower order address bus and then if this is the memory and this lower order address bus and the higher order address bus together.

So, they form the address for the memory and the data comes to the data bus directly like this. So, we have got this ALE signal activated here, and this ALE signal is activated when the address, lower order address data bus it contains the lower order address part. So, the process, so now to the memory the address has been given properly, now since this is a memory read operation.

So, fetch is a memory read operation, so this $\overline{\text{read}}$ signal is activated at the beginning of T₂ ok. So, during T₁, ALE was during T₂ this $\overline{\text{read}}$ signal is active. So, this $\overline{\text{read}}$ signal goes low telling that it is asking for the, for the instruction or the data from the memory and getting this $\overline{\text{read}}$ signal.

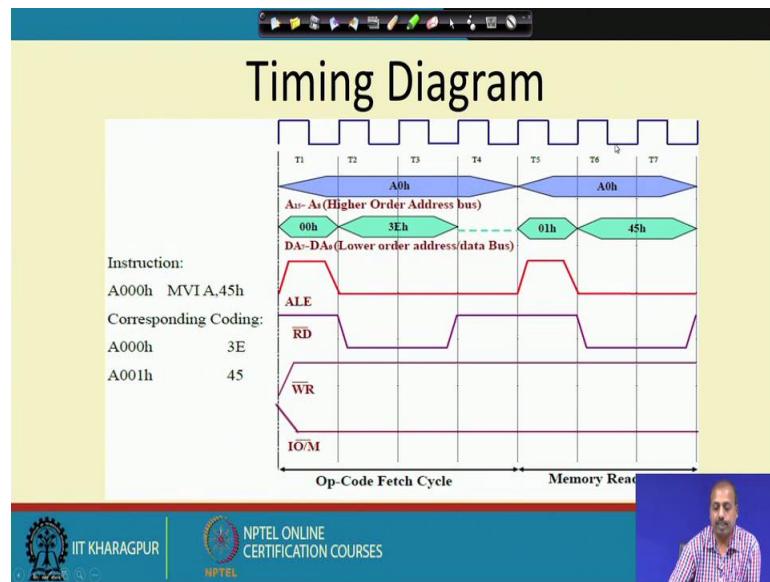
(Refer Slide Time: 05:30)



So, again if this is the CPU, this is the 8085 CPU and this is the memory so far we have been able to set the address bus and we have been able to give the *read* signal. So, when you give this read signal then the memory will put the data onto the data bus which happens to be the lower order address bus itself. So, this 78 value which is the content of the location A000 hex is put on to the data bus and then this processor will read it from the data bus.

Now, the *write* signal, this is not a memory write operation. So, this is kept continually high throughout the operation and is $\overline{IO/M}$ signal, so, this is made low because this is the memory operation and not an IO operation. So, that way this opcode fetch a part, opcode fetch cycle is executed for the MVI. So, if you draw this timing diagram this tells us explicitly how different control signals are being activated by the processor to get the operation done.

(Refer Slide Time: 06:34)



Let us take a slightly more complex example MVI A, 45 hex. So, here you can understand that in the first machine cycle is the fetch machine cycle and in the fetch machine cycle. So, it will fetch the instruction and since it can fetch only 8 bit of data at a time. So, if it will in the first cycle it will get the opcode part. This MVI A, 45 hex this instruction. So, this instruction has got a coding of 3E for the MVI A part and the next part is 45 hex.

So, if the instruction is again at location A000 hex. So, it will take two bytes to hold the instruction. So, location A000 hex we have 3E and A001 will have 45. Now the way the operation is done is similar like it is in the opcode fetch cycle it is similar to the previous case. So, this higher order address bus contains A0 hex, lower order address bus in the first clock cycle contains 00, the ALE signal is activated and the then so, ALE signal is activated then the 00 gets latched on to the memory on the address bus of the memory. So, we have got the address memory address bus contains A000 hex total 16 bit address. Now, read signal is given so the memory will put the content of that location 3E on to the data bus.

Now, the processor, write is high and IO \bar{M} bar signal is low that is as previously. Now, getting this 3E the processor understands that okay controller part to the instruction decoder and the control unit. So, it will understand that this is a MVI instruction MVI. So, MVI A instruction, so for that it needs to get the immediate value.

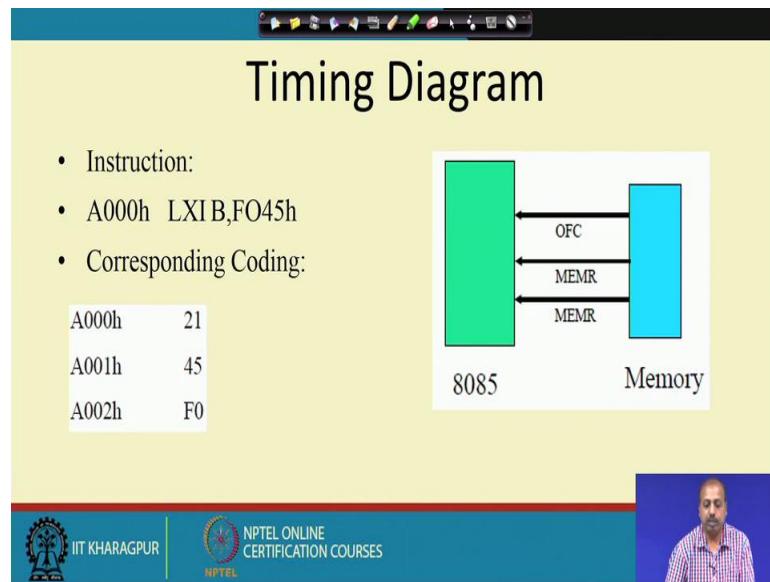
So, it has to read the memory once more. So, that is the, it has to go into a memory read cycle. In the memory read cycle it is similar to the fetch, but one T state less than the fetch. So, it is putting this higher order address bus, higher order address on the higher order address bus, lower order address on to the lower order address bus and it is giving the ALE signal.

Now, so this address part gets latched to the address bus of the memory and then it gives the read signal. So, in response to that the memory will put this 45 hex on to the data bus and this 45 hex comes to the processor ok. So, this way this 45 hex, so I get this value into the processor and the processor will now put this 45 hex into the A register that is done immediately after that 45 hex value has been obtained. So, it is put immediately into the A register, so it does not require any more clock cycle.

So, one interesting thing to note is that this fetch and at this memory read. So, both of, both this memory both these cycles are similar. So, both of them are doing some sort of memory read operation, but in case opcode fetch cycle it takes one 1 T state more than the memory read. So, this is most probably due to this decoding operation that the processor has to do after getting the opcode part, though it is not documented in the manual.

So, it is not very difficult to understand that the decoding takes some time and this extra clock cycle that is put in the opcode fetch cycle. So, that is most probably for doing that decode operation carefully. So, that is the timing diagram of this MVI A type of instruction.

(Refer Slide Time: 10:14)



signal is given, so this memory will get the content of this location that is 21 hex and putting on to the data bus.

So, here so put it on to the data bus when the $\overline{\text{read}}$ signal is low. So, this $\overline{\text{write}}$ and IOM both are $\overline{\text{write}}$ is high and IOM bar is low, because it is not a memory write operation and this is a memory operation, so that way. So, after that so this T4 is for the decoding part, so in the T5 clock cycle. So, it will understand the processor understand that okay now, I have to, these are, these are code of LXI this 21 hex is the code of LXI; it is the code of LXI B instruction; so, it as to get two more bytes and accordingly initialize the B and C registers.

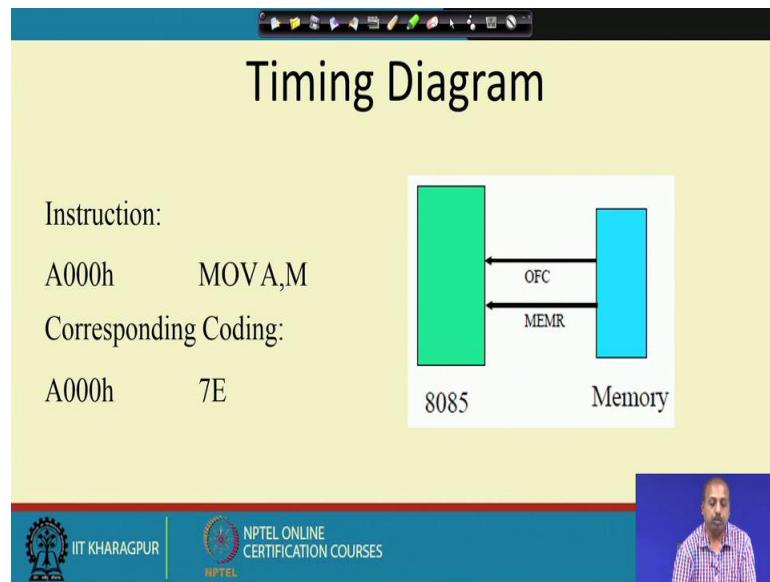
So, it will do that so it will put the higher order address part A0 on to the higher order address bus, it will put the lower order address part 01 hex on to the lower order address bus, give the ALE signal and then it will give the $\overline{\text{read}}$ signal. So, as a result memory will put this 45 hex on to the data bus and then this 45 hex comes to the processor and the processor put this 45 hex into the C register, because that is the lower order bytes, that goes the C register.

After that it comes to the next memory read cycle for again it put this A0 hex on to the higher order address bus, 02 hex this the next address is put on to the lower order address bus actually this constitutes the program counter. So, this higher order address and this lower order address it combines together into the program counter register and after every memory fetch or this memory read operation, so, this program counter is implemented.

So, that way it is doing this after every fetch. So, this is an instruction fetch operation that is going on. So, till the instruction complete, instruction has been fetch this program counter value goes on incrementing. So, that is the it becomes A002 and this is A0 is put on to the higher order address bus from the PC high and the 02 hex that is for the PC low that comes to the lower order address bus.

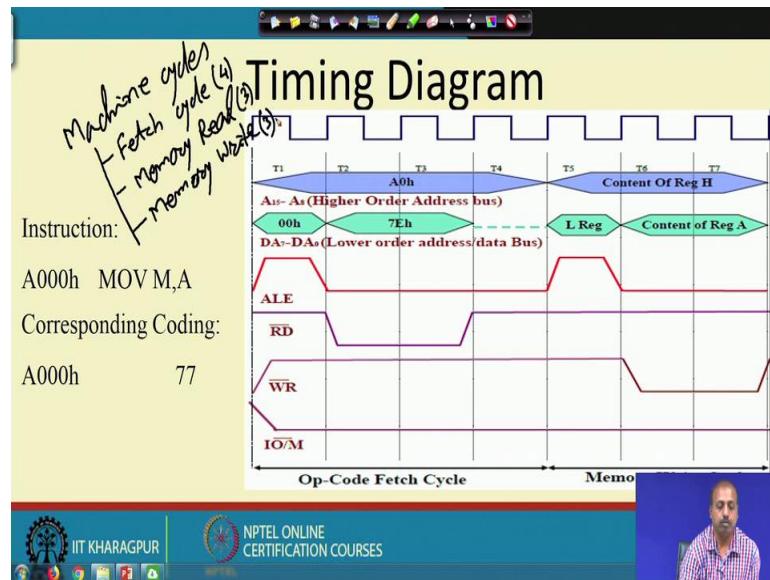
And similarly when this ALE signal is given so, address is latched and when this $\overline{\text{read}}$ signal is activated. So, this it is read from the memory. So, this F0 hex value that comes from memory and it is loaded into the B register. So, this way this way LXI B instruction is executed and we have got all entire timing, entire timing diagram so, that illustrates how this LXI B is done.

(Refer Slide Time: 13:44)



Next we look into another interesting instruction which is MOV A, M. So, this is slightly tricky because you see that the meaning of this instruction is that this content of memory location pointed to by the HL register pair will come into the accumulator, so the corresponding code for this is 7E ok.

(Refer Slide Time: 14:08)



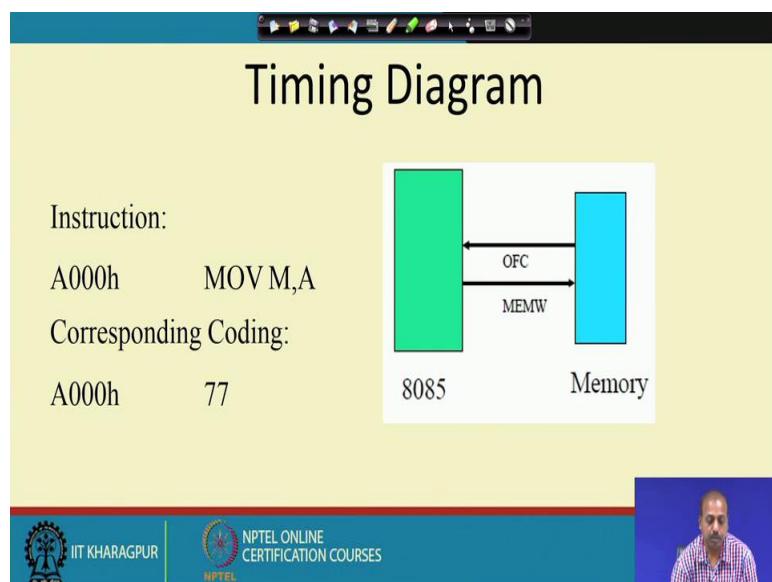
So, this is 7 E and how this instruction is going to be executed is like this. First it has to get, first part is same. So, that is the opcode fetch cycle and for the opcode fetch cycle. So, the A0 and 00 they are put on to the address bus, ALE signal is activated in T1, read is

activated in T2 and T3. So, this is we get the value 7 E into the instruction decoder. Now this instruction decoder finds that this is an MOV A, M instruction, now it has to start a memory write operation.

So, so this is the memory read operation because content of that memory location will come to the A register. There is a memory read operation, but what is the higher order address now? As I said that move in MOV A, M instruction the address is given by the HL pair. So, this H register pair value goes to the higher order address bus and L register value goes to the lower order address bus.

So, that way and the ALE signal is generated. So, it is, this value this address is latched and then the *read* signal is given. So, that the content of the memory location pointed to by HL comes to the data bus, and then when it comes to the data bus where the end when it is available, so, the processor will put the value into the A register, so this way it goes into an opcode fetch cycle followed by a memory read cycle.

(Refer Slide Time: 15:34)



The other way the MOV M , A, so that is that is another operation where we are trying to use this indirect addressing and here the content of the accumulator register will be saved on to the memory location pointed to by the HL pair and the coding for this is 77.

So, that is available from the manual of 8085 that the code for a MOV M , A is 77 hex. So, how is it done? So initially we put this higher order address bus contain the A0, lower

order address bus contain 00 and then ALE is given in the first clock cycle read bar is given in second part. So, as a result we get the instruction 77. So, this comes to the data bus then the processor will understand the, this is a MOV M , A instruction. So, then it will do a memory write operation because now the content of the A register should be put on to the memory, so this, so address of that memory location is given by the HL pair. So, this H register contains the higher order address part L register contains the lower order address part, ALE signal is given to, so latch this L register value to the lower address bus of the memory.

Then you see the *write* signal has been made low. So, *write* signal has been made low and the content of register A is put on to the data bus by the processor. So, the memory when it gets the *write* signal, so whatever is there in your in its data bus it will write it on to the memory location pointed to by its address bus. So, that is done, so content of that location comes to the content of A register goes to that particular location pointed to by the address bus of the memory.

So, to summarize you can see that, so far we have seen different types of cycle. So, these cycles they are known as they are known as machine cycles. So, there are different types of machine cycles that we have seen the first one that we have first machine any instruction to start with. So, it has got a fetch cycle or opcode fetch cycle.

So, that is common for all the instructions. So, after that, so there may be a number of memory read it can be a memory read, it can be a memory write. So, these type of cycles can be there. Now this fetch cycle. So, it takes 4 clocks clock state or T states whereas this memory read and memory write cycles, so they take 3 clock cycles. So, so far we have got introduced with three different types of machine cycles. We will see some more as we proceed ok.

So, this machine cycles are used for this, machine cycles are actually part of this is a combination of this basic clock cycles or T states and that way it constitutes different operations to be done by the processor.

(Refer Slide Time: 18:58)

The Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
 - Last In First Out.
- The stack normally grows backwards into memory.
 - In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.

A hand-drawn diagram illustrating the stack structure. It shows a vertical rectangle representing memory space. The top edge is labeled '0000' and the bottom edge is labeled 'FFFF'. Inside the rectangle, there is a horizontal line near the top labeled '8000H'. An arrow points upwards from the text 'The stack normally grows backwards into memory.' towards the '8000H' label, indicating the direction of growth. To the right of the rectangle, the text '64K' is written.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video frame showing a man with a beard and mustache, wearing a red and white checkered shirt, speaking to the camera.

Next we will be looking into another very important aspect that we have in any processor based program execution that is called the stack. So, if you think in terms of the architecture or organization of the system then the stack is an area of memory identified by the programmer for temporary storage of information. So, very often we keep, do some, we want to have some, some space where we do some book keeping job.

So, we keep some temporary data ok, so that we can put it, we can do it into the stack. Stack is a last in first out structure because, in case you may be familiar with stack as a data structure. Where there are two operation PUSH and POP and this PUSH operation it, it writes a, it puts the element on to the stack and the POP operation, so it takes the element out of the stack and this always operates in the last in first out formulation.

So, stack normally grows backwards into the memory. So, what we mean is like this so this is the memory this is the full memory. Now, so, this is the address say 0 say. So, if I have got a 16 bit, 16 bit address bus. So, that we goes from 0000 to FFFF, so total 64 K space. Now you can say that out of that my stack it will start at this location, so this is the location say 8000.

Now normally what is, what happens is that the stack starts growing from this point and it grows towards the upper side towards it grows towards the 0. Of course, that is a convention only, but the way this stack manipulation happen implemented in 8085

processor, it happens like this that it grows backwards up to toward the lower and lower addresses. So, programmer, as a programmer you can define the bottom of the stack and from that point onward it can grow up by reducing the address value.

(Refer Slide Time: 21:17)

The slide has a yellow background with a title 'The Stack' at the top center. Below the title is a bulleted list of four items. To the right of the list is a hand-drawn diagram of a stack structure. The diagram shows a vertical rectangle representing memory, with the bottom labeled 'FFFF'. An arrow points from the text 'SP' to the bottom edge of the rectangle. At the bottom right of the slide is a small video player showing a man speaking.

- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.
- LXI SP, FFFFH ✓
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

IT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this is the stack operation. Now we will see how this happens in the 8085 in the 8085 processor. So, given that the stack grows backwards into memory, so, it is customary to place the bottom of the stack at the end of memory to keep it far away from the user program. Like in the previous example I said that my memory ranges from 0000 to FFFF hex and I put the stack to start at 8000 hex.

Now, there is a possibility that some user program data will clash or program code will clash with that stack. Just to avoid it, so, this may be a safe practice to have the stack initialized at the very bottom of the memory. So, this stack may start at this FFFF hex and in 8085 the stack is defined by setting on special register which is known as the stack pointer register, this is this is a 16 bit register used for accessing the stack. So, what we want is that if this is my memory and then the last address is FFFF.

So, I want this FFFF to be the stack to be the beginning of the stack for that purpose we have to initialize the register SP to the value FFFF hex, and for that we have got this instruction LXI SP , FFFF hex. So, the stack pointer gets initialized to the end of the last memory location. So, from last point onwards it starts growing and it can grow towards the address 0.

(Refer Slide Time: 22:55)

Saving Information on the Stack

- Information is saved on the stack by PUSHing it on.
 - It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
 - Both PUSH and POP work with register pairs ONLY.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how can you save or get retrieve data information from stack? Information is saved on stack by pushing it on. So, you can PUSH data on to the stack. So, that will be saved on to the memory, memory locations pointed to by the stack pointer and then you can retrieved it by doing a POP operation. So, PUSH and POP, so these are the two operations for writing and writing to reading from the stack. 8085 provides two instructions similarly PUSH and POP. If you look into stack as a data structure so, will find that it defines to operation PUSH and POP in the stack.

And similarly here also we have got this PUSH and POP as two instructions in the 8085 processor. So, they PUSH and POP, they work with registered pairs only, so you cannot PUSH or POP a single register, an 8 bit register. So, you have to work with this register pair.

(Refer Slide Time: 23:58)

The PUSH Instruction

- PUSH B
 - Decrement SP
 - Copy the contents of register B to the memory location pointed to by SP
 - Decrement SP
 - Copy the contents of register C to the memory location pointed to by SP

*PUSH C X
PUSH L X S*

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

For example PUSH B, so when you do PUSH B then this PUSH this actually means we are trying to save the register pair B C onto the stack ok. So, it we cannot have an instruction like PUSH C because PUSH C is not a register pair. So, B is a register pair consisting of the registers B and C. So, we cannot have an instruction like say PUSH, you cannot have an instruction like say PUSH C, so this is not possible or say PUSH E. So, PUSH L so this is not possible, so this is are not ok. Now so how is it executed is like this see I have got a stack.

(Refer Slide Time: 24:48)

The PUSH Instruction

- PUSH B
 - Decrement SP
 - Copy the contents of register B to the memory location pointed to by SP
 - Decrement SP
 - Copy the contents of register C to the memory location pointed to by SP

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this is the memory and then suppose at present the stack pointer is somewhere here. So, stack pointer is pointing to this location and this is towards the address 0, lower address. Now if I am executing this PUSH B instruction, so, what I want is somehow this B and C register values will be saved on to the stack.

So, what is done? This stack pointer is decremented, so stack pointer now points to this. So, stack pointer now points to this location and then copy the content of register B to the memory location pointed to by SP. So, at this location with copy the content of B register into this and they need decrement the stack pointer further. So, stack pointer now points to this location, it points to this location and there we copy the content of the C register.

So, this way it operates, so when it is. So, PUSH B it first decrements the stack pointer then copies the content of B register on to that and then decrement stack pointer further, copies the C register content. So, you remember that in case of 8085 this lower order byte is always stored in the lower order address location. Now, out of this B C register pair, so C is the lower order register, so and B is the higher order register. So, this lower order register has gone to the lower order memory address C and this higher order address higher order register has gone to the higher order register higher order value.

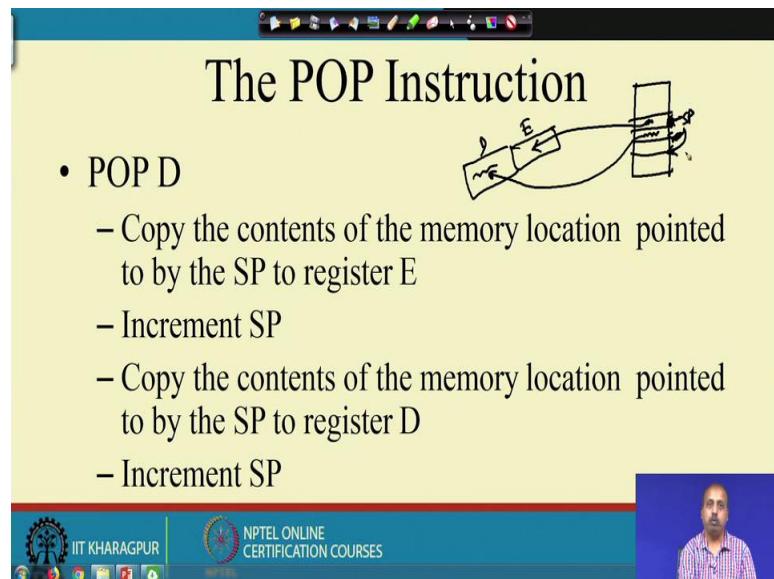
So, that is the higher order memory location ok, so this is the always true. So when you are doing a save, saving some, copying some 16 bit value into some memory locations. So, this is done the other way first the lower order is copied then the higher order is copied, but in case of stack in stack grows in the reverse direction it is going towards 0. So, this lower order register is copied later first the higher order is copied and then the lower order is copied.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 56
8085 Microprocessor (contd.)

Next, we will be looking into the POP Instruction. So, this is just a reverse of push.

(Refer Slide Time: 00:18)



The POP Instruction

- POP D
 - Copy the contents of the memory location pointed to by the SP to register E
 - Increment SP
 - Copy the contents of the memory location pointed to by the SP to register D
 - Increment SP

NPTEL ONLINE CERTIFICATION COURSES

So, in case of push instruction so, we are saving the content of this register pair on to the stack. So, this POP instruction is to retrieve the register pair value from the stack. Again the POP works with register pair not with a single register. So, the way it operates is that it copies the content of memory location pointed to by the stack pointer to the E register. Then implement the stack pointer and then it will be copying the content of this stack point memory location pointed to by stack pointer into the D register and then implement stack pointer again.

(Refer Slide Time: 01:01)

The screenshot shows a presentation slide titled "Operation of the Stack". The slide content is as follows:

- During pushing, the stack operates in a “decrement then store” style.
 - The stack pointer is decremented first, then the information is placed on the stack.
- During popping, the stack operates in a “use then increment” style.
 - The information is retrieved from the top of the stack and then the pointer is incremented.
- The SP pointer always points to “the top of the stack”.

The slide has a yellow background and a blue header bar at the bottom containing the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man speaking.

So, if we think if we if we this is the, if this is the sorry so, if we if we look into this one. So, if these suppose this is the stack, this is the portion of the stack, that we had this is the memory location and this is the stack. So, stack pointer is now pointing to this place. Now, when we do a POP D then so, these location's content so, these location's content in is first copied on to the E register.

So, this the D E pair so, this content will come to the E register and then the stack pointer will be implemented. So, stack pointer now points to this location, the next location and that locations content will be copied on to D register. So, whatever be the content here will be copied on to the D register and then the stack pointer will be implemented further. So, that it points to the next memory location ok.

So, that way this POP instruction is executed. So, it is just a reverse of push. So, these PUSH, POP instructions can be useful for saving some value temporarily into the memory. So, now, how this stack is used? So, during pushing the stack operates in a decrement then store style.

(Refer Slide Time: 02:28)

The PUSH Instruction

- PUSH B
 - Decrement SP
 - Copy the contents of register B to the memory location pointed to by SP
 - Decrement SP
 - Copy the contents of register C to the memory location pointed to by SP

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, first as if you look into the instructions then you see that here, it for the push instructions so, it is decrement SP and then do the copy then again decrement. So, it is first decrement and then copy, whereas for POP it is first copy then increment. So, we can say that during push operation the stack operates in a decrement then store style, the stack pointer is the decremented first and then the information is placed on to the stack and for popping, the stack operators as use then increment. So, as if it is all the copy then increment style.

The instruction is retrieved from the top of the stack and then the pointer is incremented. And stack pointer always points to the top of the stack. So, that is by the definition of this stack data structure, that it to always points to the top of the stack. So, that is taken care of in the 8085 stack operation also.

(Refer Slide Time: 03:20)

The slide has a yellow background with a title 'LIFO' in large bold letters. Below the title is a bulleted list:

- The order of PUSHes and POPs must be opposite of each other in order to retrieve information back into its original location

To the right of the list is a diagram of a stack. The stack is represented by a vertical rectangle divided into four horizontal sections. From top to bottom, the sections are labeled: 'PUSH B', 'PUSH D', '...', and 'POP D'. To the right of the stack, there is a small drawing of a hand holding a pen, pointing towards the stack. Below the stack, the labels 'POP B' and 'POP D' are written. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Stack works in a LIFO fashion last in first out fashion, because the basic 8085 designers, so, they have not put any restriction like the order in which you do a push POP operation, but as a user of the system. So, we have to be careful. So, the order of push and POP they must be opposite of each other in order to retrieve information back into it is original location. So, may be I have brought a piece of program and that in that. So, we don't want that wherever from what wherever place I have called this program. So, whatever register this program uses.

So, let us see that let us say that this uses the registers B C D and BC, DE, so, these register pairs it is using. Then what I want is that wherever be the program from where this one has been called, so, after this program has finished. So, that the B C D E register they should get their old values back. So, we will see later this type of situation occurs when we consider the interrupts in a system. So, what we want is before coming to this the routine, whatever be the values of B C, D E after finishing the routine the values should be restored to those.

So, for that purpose we have to we have to save this register. So, you should have 2 instructions here the push B and push D. So, these 2 instructions should be there for putting it or for saving them on to the stack. Now, at the end what is what is required is that we have to retrieve the content of this 2 this 2 this 4 register B C D E and how do we do that?

How do we do that is by means of this POP instructions and this popping has to be done in the reverse direction.

So, that is POP D and POP B. So, if we do not do that so, if we by mistake. So, if we write as first POP B and then POP D then POP D, then the content will be reversed isn't it? Because in the stack we have saved this B register first and then the D register. So, while popping out while popping out. So, I should get I should POP out D register first then the B register.

So, the order in which you have pushed into the stack. So, while popping out you should do it in the reverse order. So, that is why it is called a last in first out structure or LIFO structure.

(Refer Slide Time: 06:15)

The PSW Register Pair

- The 8085 recognizes one additional register pair called the PSW (Program Status Word).
 - This register pair is made up of the Accumulator and the Flags registers.
- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.
 - The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

Another, very interesting register that this 8085 has is PSW register pair. So, this is PSW physically the register consists of the accumulator and the status word register ok. So, this is called program status word or processor status word sometimes. So, this register pair is made up of the accumulator and the flags register. So, this accumulator and flag, they constitute this PSW register. It is possible to push the PSW onto the stack, do whatever operations are needed and then POP it off of the stack.

So, that way we can use this you can use this PSW register. The result is that the content of the accumulator and status flags status of the flags are returned to what they were before

operation were executed. Again the same thing many times we want that this A register and the status they should be available at the end of the program at the end of the execution of a piece of program. So, for that purpose at the beginning you can do a push PSW instruction. So, you can have an instruction like push PSW at the beginning and at the end. So, we can have POP PSW, we can have push PSW and POP PSW as the two instructions. So, that is so, that is the utility of this PSW register pair.

(Refer Slide Time: 07:43)

The slide has a yellow background. The title 'Subroutines' is centered at the top. Below it is a bulleted list:

- A subroutine is a group of instructions that will be used repeatedly in different locations of the program.
 - Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist anywhere in the code.
 - However, it is customary to place subroutines separately from the main program.

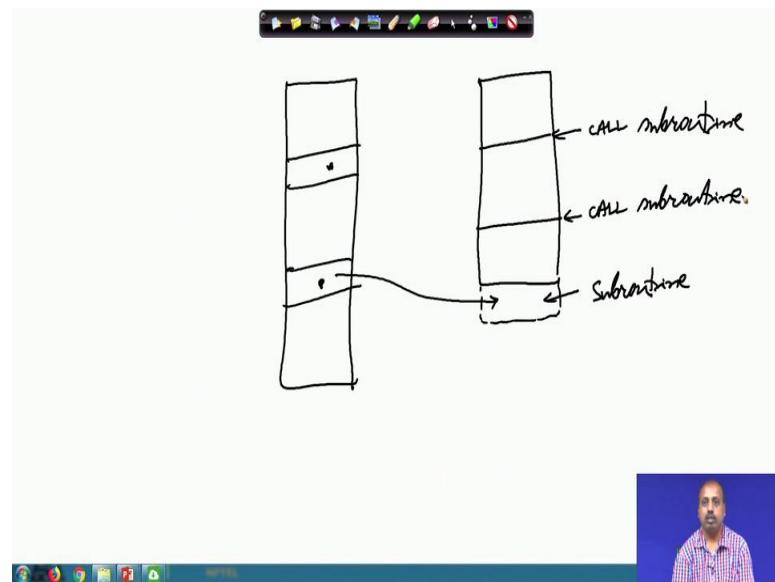
At the bottom of the slide is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it says 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a small video player window showing a man with a beard and mustache, wearing a checkered shirt, speaking.

Next, we will be looking into one very important concept, which is known as subroutine. So, a subroutine it is a group of instructions that will be used repeatedly in different locations of a program. So, rather than the same instructions putting same instructions several times they can be grouped into a subroutine and that is called the that is called from the different locations.

So, this is similar to the procedures that we have or the functions that we have in high level languages. So, here we write in case of assembly language program some part has to be repeated. So, you just put it separately and then we can you can just call this subroutine.

So, if I, if I have a piece of program.

(Refer Slide Time: 08:33)



And, if we find that in that program so; this some parts of the program are similar. Only some parameter some registers are changing and things like that, then what you can do? So, if this code and this code are very similar. So, you can reorganize this entire code like this. So, you put a, you put a small portion at the end. So, that is actually the code that we have here.

So, the since the codes are similar so, put the code only once. And so, this is this will be called a subroutine and this subroutine is called from different places, where this place you have got a call. So, this is a call to the subroutine. And, similarly sometime later again you give a call to the subroutine; so, this is a call to the subroutine.

So, this way what we are saving is we are saving the extras program length that we had. So, because of this is only twice that I have shown here. So, maybe 1 routine is needed very often. So, as a result so, it is called several time it is it is copied several times in a, in a straight line code.

So, if you are putting a subroutine, then that part can be saved and we can make the program size small particularly in microprocessor based system. So, since the space is a concerned. So, making them small saves the space. So, that is the utility of this subroutine. So rather than repeating the same instructions several times, so, they are grouped together into 1 subroutine and they are called from several locations.

In assembly language, a subroutine can exist anywhere in the code. So, though in my example I have shown the subroutine to be at the end, but it is, it is not necessary. So, you can put the subroutine at any place. So, it does not put any restriction in assembly language programs. So, there is no restriction. But for the readability purpose, so, they should be put separately from the main program otherwise readability becomes a problem.

(Refer Slide Time: 10:51)

The slide has a yellow background and a title 'Subroutines' centered at the top. Below the title is a bulleted list:

- The 8085 has two instructions for dealing with subroutines.
 - The CALL instruction is used to redirect program execution to the subroutine.
 - The RET instruction is used to return the execution to the calling routine.

To the right of the list is a diagram of a stack frame. The stack grows downwards. It shows memory addresses 1000, 2000, and 4000. At address 2000 is the instruction 'CALL 4000'. At address 4000 is the instruction 'RET'. A curved arrow points from the 'CALL' instruction to the 'RET' instruction, indicating the flow of control.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and 'NPTEL ONLINE CERTIFICATION COURSES'. To the right of the footer is a small video window showing a person speaking.

Next in case of 8085 there are 2 instructions that deal with subroutines, the call instructions that is used to redirect the program execution to the subroutine. And the return instruction or RET instruction used to return from the execution of the execution to the calling routine.

(Refer Slide Time: 11:08)

The CALL Instruction

- CALL 4000H
 - Push the address of the instruction immediately following the CALL onto the stack
 - Load 4000H to PC

Stack Diagram:

```
graph TD; SP[SP] --> 40[40]; 40 --> 03[03]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for example, these call 4000 hex. So, that is for calling the subroutine and we have got it return instruction. So, it is like this that if this is my main program and then so, suppose your main program starts at memory location 1000 and it goes like this. And, then this is the subroutine that you have starting at location 4000. Now, somewhere here at location say 2000 I want to give a call to this subroutine. So, for that purpose the instruction is call this is the call 4000 hex. So, the size of this instruction is 3 bytes as you can understand that call will take 1 byte and this 4000 H will take 2 bytes. So, total it is a 3 byte instruction.

So, location 2000, 2001 and 2002 so, they are holding the instruction. And, then when this instruction is executed what the system does, it jumps to the location 4000 and start executing from this point onwards. Now, any subroutine it should end with a return instruction or RET. So, when the processor it is your find this RET instruction, because RET is again 1 byte instruction because it has only the op code part. So, when this RET is found then the processor will try to come back to the location just after the call instruction, that is the 4002 ok. So, it will be 4003. So, 4000 4001 and 4002 they are the call instructions.

So, in this return is executed it will try to come back to 4003. Now, the question is how does it come back? So, how the processor will know in the return address I am not mentioning like where to go? So, how the processor will know where to go?

So, this is actually accomplished by the call mechanism. So, at the time of calling itself so, this is taken care of that we have got this return address saved on to the stack. So, in this call instructions push the address of the instruction immediately following the call into the stack. And, as I have said that call is a 3 byte instruction. So, that immediate next instruction is at address 4003.

So, this 4003 value will be pushed on to the stack and then. So, what happens is that? So, if this is the stack. So, value 4003 will be pushed on to the stack and as we know that it is a 40 is the higher order byte and 03 is the lower order byte. So, this is the. So, stack pointer will now point to this location ok.

Now, it will be it will be call, now the program counter value program counter register will be loaded with the 4000 hex. As a result from the next instruction the processor will execute from location 4000 hex. So, after some time so, it will be for finding the return instruction, and then it will be popping out the content from the stack to get the return address. So, we will see how is it done?

(Refer Slide Time: 14:23)

The RET Instruction

- RET
 - Retrieve the return address from the top of the stack
 - Load the program counter with the return address

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, return instruction so, it will retrieve the return address from the top of the stack. So, it will get the content from the top of the stack and it will load the program counter with the return address. And, as we know that the program counter register it actually determines the instruction, which will be executed next.

So, for the return what we need to do is to get the content from the stack and put it on to the program counter. So, that is exactly what is done in the return instruction.

(Refer Slide Time: 14:53)

*use SP with
CALL/RET*

- The CALL instruction places the return address at the two memory locations immediately before where the Stack Pointer is pointing.
 - You must set the SP correctly BEFORE using the CALL instruction.
- The RET instruction takes the contents of the two memory locations at the top of the stack and uses these as the return address.
 - Do not modify the stack pointer in a subroutine. You will lose the return address.

So, there can be some caution like the call instruction places return address at the 2 at the 2 memory locations immediately before where the stack pointer is pointing. So, we must set stack pointer correctly before using the call instruction. So, this is so, 8085 designers what they have done? So, they have made the call instruction execution like this that, this program counter value will be saved on to the stack.

Now, if so, but it does not do anything regarding the stack pointer initialization. So, if by mistake this stack pointer contains some garbage value, then the program then the program counter values will be saved onto that memory location. So, that makes it very difficult like. So, it is the user's responsibility to ensure that the stack pointer is pointing to a valid memory address, from where, which is not used for any other purpose and the return address can really be saved onto that address and can be retrieved later. So, before the call instruction is execute call instruction is put into a code. So, before the first call instruction, if this is my program somewhere I am doing say call to some address. So, before doing this call somewhere previously I must have executed this instruction LXI SP comma some valid address.

So, that the stack bottom is loaded on to the stack pointer stack pointer points to the bottom of the stack that should happen. And, after that only this call should be executed. So, this

is just a precaution. So, if you do not do this if you do not execute the LXI instruction then also this call will execute, but the effect may be disastrous.

So, this may be that it, it collapse some other values of your program other data of your program, because the stack pointer points to those arbitrary locations and then it tries to write this program counter value there. So, you must save the stack pointer correctly before using the call instruction. Similarly, the return instruction it takes the contents of 2 memory locations at that top of the stack and uses these as the return address.

So, you should not modify the stack pointer in a subroutine. So, if in a subroutine if we modify the stack pointer then this will be lost. Well I if this is my subroutine and there somewhere I change the stack pointer values. So, LXI SP to something else. And, then I put a return then what will happen? So, it will be it will be trying to load this stack it will load the stack pointer is something else. So, when the return is executed. So, it will try to get the return addresses from that other location not the location, where the stack this return address was saved while the call instruction was being executed.

So, you should not modify the stack pointer in a subroutine, otherwise we will lose the return address value. So, that we have to be careful.

(Refer Slide Time: 17:54)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. The main title 'Passing Data to a Subroutine' is centered in a large, bold, black font. Below the title, there is a bulleted list of points:

- In Assembly Language data is passed to a subroutine through registers.
 - The data is stored in one of the registers by the calling program and the subroutine uses the value from the register.
- The other possibility is to use agreed upon memory locations.
 - The calling program stores the data in the memory location and the subroutine retrieves the data from the location and uses it.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '113'.

So, you can also use this stack to pass data to a subroutine. So, in assembly language program. So, we can pass the data to the subroutine through registers that is a 1 possibility.

So, so this data is stored in one of the registers by calling the program and the subroutine uses the value from the register. So, by the calling the calling program for example, if it has to pass an 8 bit value, it may be that it saves the value in the B register and then calls the subroutine. And, in the subroutine we access the B register to get the value that has been passed. So, that is one possibility. So, you can use one register to hold some, hold some parameter that you want to pass to the subroutine.

But, since the number of registers are limited and they are also used for some other purposes, so, it is better that we have some option ok. So, and that option is via the stack so, if because memory is very large compared to the data, the space requirement of a program for the data space requirement of a program. So, we can use this stack for the purpose of this transfer of this parameters.

So, what the program does the calling program instead of storing the content of this data onto a register, it can store the content in a memory location and the subroutine will retrieve the data from the location and use it. So, that is possible. So, that way we can have some memory location and do that operation.

(Refer Slide Time: 19:40)

Call by Reference and Call by Value

- If the subroutine performs operations on the contents of the registers, then these modifications will be transferred back to the calling program upon returning from a subroutine.
 - Call by reference
- If this is not desired, the subroutine should PUSH all the registers it needs on the stack on entry and POP them on return.
 - The original values are restored before execution returns to the calling program.

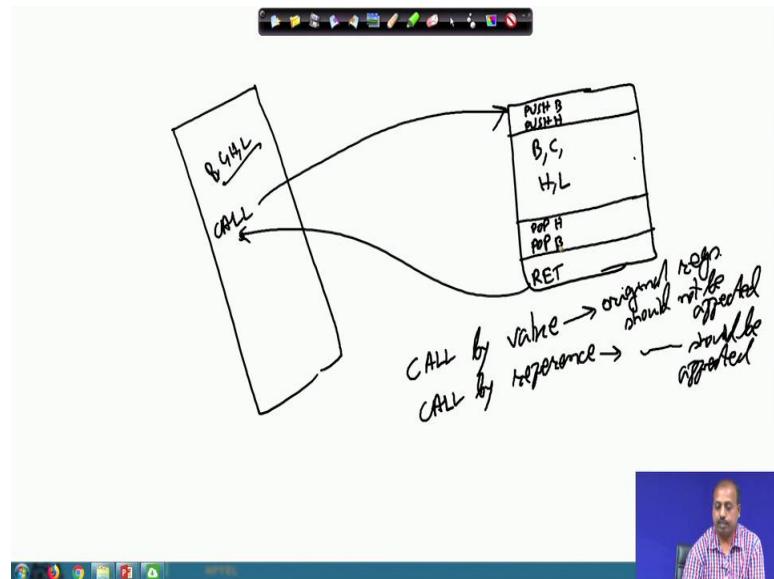
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

114

Now, if a subroutine performs operations on the contents of the register and then these modifications will be transferred back to the calling program upon returning from the subroutine. So, this is called call by reference and if this is not desired then subroutine should push all the registers it needs on the stack on entry and POP them on return. So,

this is the original values will not be lost. So, what we mean is like this that suppose I have a subroutine ok.

(Refer Slide Time: 20:17)



So, I have a subroutine where the so, this is my subroutine and there I am using this registers like it this part of the code I am using the registers B, C, H and L fine. Now what can happen? When these modifications are done? So, this is so, if this is the main program from where I am calling this subroutine. So, there is a call instruction here which actually calls this subroutine and somewhere here I have got the return. So, it will be coming back to this point.

Now, if this subroutine modifies this B C H L registers, then once you come back here you see that this B C H L are all modified. So, whatever be the values of this B C H L registers here, so, they are modified by the subroutine ok. So, if you want that so, there may be situations in which you want it. So, maybe it is doing some computation in the B C H L register values in the registers and then those values are needed by the calling program.

So, that is one possibility, other possibility is that this B C H L, so, they were used as some temporary storage by the calling by the by this subroutine, and the values of the, of those registers of the on the original program they should not be modified.

So, for that purpose so, we have got 2 type of situation one is call by value, another is call by reference. So, in case of call by value the original registers should not be affected, original registers should not be affected. So, this is the situation I was talking in second in the second case that is whatever be the values of B C H L here. So, that that should not be should not get disturbed and call by reference means original registers should be should be affected they should be affected.

Now, the second part the call by reference implementation is very simple. So, you do not have to take any precaution to restore the values of B C H L, but if you really want that this B C H L content should not be lost you have to have a call by value, then the first few instructions of this subroutine it should be push B and push H.

So, they will be saving those 2 registers and before doing this return instruction ok. So, I should POP them out. So, I should use instruction like POP H and POP B before the return. So, that the values are restored. So, the original content of B C H L registers. So, they are not lost. So, this way we can do this call by value and call by reference type of implementation by using the stack.

So, if the subroutine performs the contents of the registers if these modifications will be transferred back to the, they will be transferred back to the calling program upon returning from a subroutine. So, this is the call by reference and if this is not desired then the subroutine should push all the registers, it needs on to the stack on entry and POP them on return. So, this is the original values are restored before execution returns to the calling program. So, that has to be done.

(Refer Slide Time: 24:01)

The slide has a yellow background with a black border. At the top, there is a toolbar with various icons. The title 'Cautions with PUSH and POP' is centered in a large, bold, black font. Below the title is a bulleted list of cautions:

- PUSH and POP should be used in opposite order.
- There has to be as many POP's as there are PUSH's.
 - If not, the RET statement will pick up the wrong information from the top of the stack and the program will fail.
- It is not advisable to place PUSH or POP inside a loop.

To the right of the list is a diagram of a stack represented as a vertical rectangle divided into four horizontal sections. From top to bottom, the sections are labeled: 'B, C', 'D, E', 'B', 'PCL PCH'. To the left of the stack, there is some handwritten text: 'B, C' above the first section, 'D, E' above the second, and 'B' above the third. The bottom section is labeled 'PCL' and 'PCH'.

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there is a logo for IIT Kharagpur and the text 'IIT KHARAGPUR'. In the center, there is a logo for NPTEL and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the bar, there is a small video window showing a man speaking.

Now, push and POP as we have already say that the push and POP should be used in the opposite order the order in which you push the registers you should POP in the reverse order. And, there has to be as many POP's as there are push ok. So, if the number of suppose for example, when these see the number of push is say 4 and number of POP's is three; that means, the situation is like this is that when you are calling a subroutine you know that first if a first the return address is saved on to the stack. So, PC high and PC low. So, if you draw in terms of a stack so, it is like this first the PC high will be stored then the PC low will be stored.

And, then if you if you say that I will be I have pushed this registers B and C, then the C register B register will be saved here, then the C register is saved here then if I push D and E, then E register will be saved here and D register will be saved sorry D register will be saved here, and E register will be saved here. Now, while doing the POP so, naturally when you are returning so, it is desirable that when you are returning the stack everything has been erased from the stack and the stack has got this PCL and P C high at the 2 top most entries.

So, if there is a mismatch in the number of POP's, push and POP's like if may be I have just popped out these 2 by using a POP instruction and I have did not do any further POP's. So, the stack top actually contains the register values B and C the old values of B and C. Now, if you do a return now then this C register value will go to PC low and the B register

value will be go to PC high, and that is dangerous. So, it will take your program to somewhere else not to the point from where the subroutine was called. So, this RET statement will pick up the wrong information from the top of the stack and the program will fail and it is not advisable to push put to this push and POP inside a loop.

Because, it is often very difficult to judge like how many pushes or pops will be done and whether they are really balanced or not, because the last iteration of the push POP last iteration of loop normally it comes out without executing the body. So, in those cases it is difficult to ensure that this push POP they will be matching.

So, you try to push this push and POP instructions outside the loop and if it is very much necessary to put inside the loop body, then ensure that it is put at the beginning of the body and popped out at the end of the body and they are always executed. So, it is not there in some cases some the POP may not be executed it should not happen like that. So, that way we have to be careful with this push and POP instruction while writing the subroutines.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 57
8085 Microprocessor (Contd.)

So, conditional call so far whatever call and return we have seen; so that were unconditional. So, the call instruction will definitely take the control to the subroutine and return instruction will definitely take you back from the subroutine.

(Refer Slide Time: 00:13)

Conditional CALL and RET Instructions

- The 8085 supports conditional CALL and conditional RET instructions.
 - The same conditions used with conditional JUMP instructions can be used.
 - CC, call subroutine if Carry flag is set.
 - CNC, call subroutine if Carry flag is not set
 - RC, return from subroutine if Carry flag is set
 - RNC, return from subroutine if Carry flag is not set
 - Etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

116

Sometimes we need some conditional call and return instructions like the CC instruction. So, call subroutine if carry flag is set similarly, CNC is call subroutine if carry flag is not set so like that we can have some conditional calls. Similarly, RC is return from subroutine if carry flag is set and RNC return from subroutine if carry flag is not set then we have got this O flag so CZ call if O flag is set, so like that we can have. So, all these condition codes that you are that are allowed with this conditional branch instructions so, they are all allowed with this conditional, call instructions also.

But they are much less practiced compared to these branch instructions, because that makes the code complex and difficult to follow while reading. So, that is. So, normally what is done is that we do some jump some conditional branch checking, and then put the

call instruction on top of that ok. We try to make it as straight line as possible ok. So, that way it has to be done.

(Refer Slide Time: 01:32)

The slide has a title 'A Proper Subroutine' at the top. Below the title is a bulleted list of guidelines for writing proper subroutines. To the right of the list is a small diagram showing a vertical stack of memory locations labeled 2000, 2500, and 3000. An arrow points from the text 'According to Software Engineering practices, a proper subroutine:' to the diagram. The diagram also includes the assembly code 'CALL 2500' and 'RET'. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a person speaking.

- According to Software Engineering practices, a proper subroutine:
 - Is only entered with a CALL and exited with an RET
 - Has a single entry point
 - Do not use a CALL statement to jump into different points of the same subroutine.
 - Has a single exit point
 - There should be one return statement from any subroutine.
- Following these rules, there should not be any confusion with PUSH and POP usage.

Now, how to write a subroutine? Like what should be the, what are the general guidelines for writing a subroutine. So, proper subroutine is only entered with a call and exited with an RET instruction. So, that is it has got a single entry point, so do not use a call statement to jump into different points of the same subroutine.

So, what we mean is the guideline is like this see assembly language programs. So, since that control is entirely in the hand of the programmer. So, if this is my subroutine and supposes my subroutine starts at location say, say 2000, it starts at location 2000. Now we should issue a instruction like call 2000 so that is fine some. So, this subroutine supposes it spans from 2000 to 3000. Now it is highly possible that I put a call instruction for location 2500.

That is possible because 2500 is somewhere in between so the call this particular call will start from this point onwards. So, processor does not differentiate between whether this is a subroutine or not, but the processor will not differentiate between whether this is a subroutine or not; so it will always take you from this address onwards. So, 25 so if you whatever address you mention so it will take you to that address and it will go from there, but this so it is, but as you can understand that if you have got several such entry points

inside the subroutine then, the branching becomes, this understanding the program becomes difficult.

So, this is exactly what is done here the done by the programmer. So, programmer has to be careful, and has to take into consideration that I should not branch into there the location inside a program. So, do not use a call statement to jump into different points of the same subroutine. So, that you should not do and you should similarly you should have a single exit point. So, somewhere here in the program I put a statement in the subroutine I put a statement like jump to say 1500; that takes it outside the subroutine. So, we should not do that because then we have got some exit so which is not really very valid one. So, there should be only one return statement from any subroutine.

So, these are the guides from the software engineering angle, so as per as the processor instruction execution is concerned so these are the not at all any restriction. So, the programmer is absolutely free to do all these things, and assembly language programmers so they have got the highest level of freedom, so they can write programs in in any in any style that they feel like ok. So, just similarly this if for push and pop we have seen that there are some rules like push and pop should be done in the reverse order and all. So, those rules if they are followed properly, so that will avoid any confusion that can arise due to these push pop instructions so that that is there.

(Refer Slide Time: 04:47)

The screenshot shows a presentation slide with a yellow background. The title 'Interrupts' is centered at the top in a large, bold, black font. Below the title is a bulleted list of points about interrupts:

- Interrupt is a process where an external device can get the attention of the microprocessor.
 - The process **starts** from the I/O device
 - The process is **asynchronous**.
- Interrupts can be classified into two types:
 - **Maskable** (can be delayed)
 - **Non-Maskable** (can not be delayed)
- Interrupts can also be classified into:
 - **Vectored** (the address of the service routine is hard-wired)
 - **Non-vectored** (the address of the service routine needs to be supplied externally)

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is the logo of IIT Kharagpur. Next to it, the text 'IIT KHARAGPUR' is written. To the right of the logo, there is the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the far right of the footer, the number '118' is displayed.

Next will be looking into another very important concept which is known as Interrupt; interrupt is a process where an external device can get the attention of the microprocessor. So, in our day to day life also interrupt is very common, like say suppose we are sitting in a room and reading a book and then all of a sudden the door bell rings. So, we have to go and see like **what, so attend** give attention to why the door bell has ring somebody has come or not etcetera.

Now, the first thing that we should do is that; we should keep a note like the which page of the book I was reading. So, that I can come back and start reading from that. So that is the first point; then even before that maybe we were reading one line we were in the middle of a line. So, we first finish of that line and then a note down the page number maybe by means of a bookmark or so, we keep a note now the page that we are reading. And then go to attend the door call and then after doing whatever, after talking to the person who has arrived and then maybe we again come back and start reading the book.

So, that whole thing is an interrupt process. So, the similarly the processor when it is in it is normal execution what it is doing? So, it is just getting the next instruction from memory and executing it. After that it is again getting the next instruction and executing it. So, there is no way by which I can tell the processor see something emergency has happened. So, you need to look into this part and do the operation. So, so that until and unless the processor is asking for some input output type of operation. So, we are not getting any chance to tell the processor that; something extraordinary has occurred and you have to take care of this you have to do something for this.

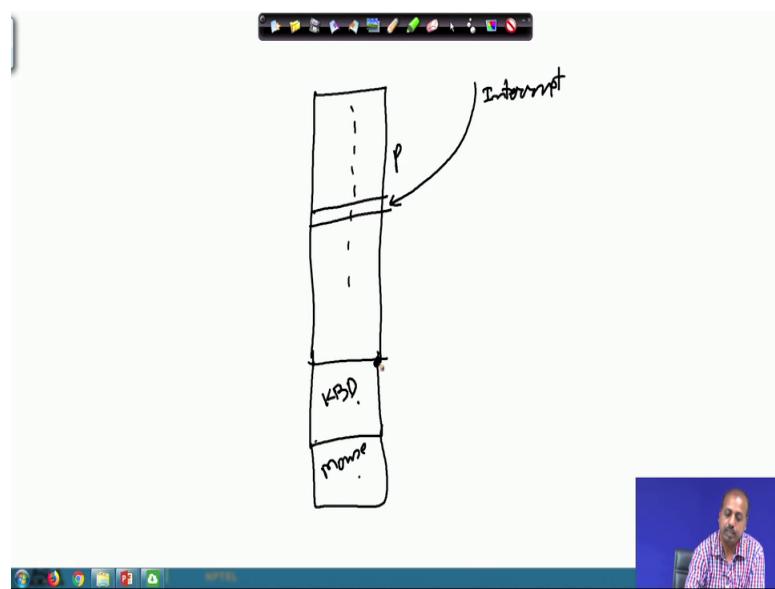
So, this is the process of interrupt. So, this is a interrupt is a process where an external device can get the attention of the microprocessor. The process starts from the IO device and the process is asynchronous. So, it starts from the IO device means the IO device will tell the microprocessor that; there is something emergency that has occurred and it has to it has to process that operation. And it is asynchronous because when this interrupt will come; so there is no such time there is no such fixed timing for that. So, so processor works in a synchronous fashion so it works at the, with the clocks and or the as some clock frequency and all, but this interrupt can come at any time.

Some of this interrupts are maskable interrupts because you may we can make the processor, you can say that it will wait for some time and some of them and non maskable

interrupt so they cannot be delayed. In terms of our day to day life, so suppose this micro oven whistle goes; that means, the say the food is ready. So, you know that even if you keep the food there for some time, nothing is going to happen ok. So, you can it is it is a maskable interrupt so we can do it later it can be delayed. And somebody the making a door pressing the doorbell maybe we need to attend it immediately. So, that has to be done immediately so that is non maskable.

Then interrupts can also be classified into vectored interrupt and non-vectored interrupts. So, vectored interrupt means the address of the service routine is hard wired. So, the processor knows the address of the subroutine to be executed, when this interrupt occurs. And the non-vectored interrupts so address of the subroutines need to be supplied externally. So, what we mean by this vectoring maybe we can explain further so it is like this.

(Refer Slide Time: 08:39)



See so also this is the, if this is suppose this is the program that the processor was executing. So, this is the program P that the processor was executing when the interrupt occurred. So, when the processor was at this particular line, the interrupt has occurred so then interrupt has come.

Now, what the processor will do? So, processor has to process this interrupt, and processing this interrupt means again executing some routine. So, what is done; depending upon this type of interrupt maybe this interrupt is that the user has pressed a key and that

has generated an interrupt or user has moved the mouse and that has generated interrupt. So, what the processor is supposed to do; is to read the character that has pressed or read the current mouse location, coordinate of the current mouse location for some purpose. So, that so the for those operations that is, reading the key where the content of the key or reading the mouse position etcetera.

So, there is there are routine. So, this may be the keyboard routine, which will be which will be stored which will be reading the keyboard. This may be the mouse movement detection routine.

So, if this interrupt is a keyboard interrupt so this these routines are already loaded into the memory. So, one possibility is that the processor already knows that for the keyboard interrupt so, what is the address; from which the, from where the program be executed. So, when this keyboard interrupt comes, the processor immediately branches to this address so that is one possibility.

And the other possibility is that the processor does not know where to go. So, in that case in that case the device that has generated the interrupt should tell the processor that; what is the address of the service routine the corresponding service routine, maybe for the mouse it does not the processor does not know like where to go, if the mouse interrupt occurs.

In that case this address of this mouse routine should be provided by the mouse device itself. So, it should tell the processor this is the address. So, once the mouse device tells the processor that this is the address, then the processor will go to that particular address. So, this is the difference between the vectored interrupt and non-vectored interrupt. So, in case of vectored interrupt, the processor knows the service address service routine address where to go, and in case of non-vectored interrupt so processor does not know the service routine where to go. So, in that in the second case the processor has to get the address from the device.

So, this is the difference between this vectored and non-vectored interrupt. So, in case of vectored interrupt the address of service routine is hard wired, so it is it is fixed by the designers, and in case of non-vectored interrupt; so this needs to be supplied externally by the device.

(Refer Slide Time: 11:31)

Interrupts

- An interrupt is considered to be an **emergency** signal.
 - The Microprocessor should respond to it **as soon as possible**.
- When the Microprocessor receives an interrupt signal, it **suspends the currently executing program** and **jumps to an Interrupt Service Routine (ISR)** to respond to the incoming interrupt.
 - Each interrupt will most probably have its own ISR.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

119

So, in terms of an interrupt, so it can be considered to be an emergency signal and microprocessor should respond to it as soon as possible, so as soon as possible it should be responded to. When the microprocessor receives the interrupt signal it suspends the currently executing program jumps to the interrupt service routine to or it is interrupt service routine so this is also known as ISR in short, to respond to the incoming interrupt and each interrupt will most probably have it is own ISR.

So, some of in some cases some of the interrupts we grouped together to have a common ISR, but in general each interrupt should is of different group, as a result it should have different types, so it should have a different service routine.

(Refer Slide Time: 12:21)

Responding to Interrupts

- Responding to an interrupt may be **immediate** or **delayed** depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
 - The vector is **already known** to the Microprocessor
 - The **device will have to supply** the vector to the Microprocessor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 120

So, if you try to look in to the spaces, the steps that it should take for responding to an interrupt. So, the steps that are needed for responding to an interrupt.

(Refer Slide Time: 12:32)

When int occurs

- ① Complete current instructions
- ② Save return address onto stack
- ③ Branch to the ISR
cont.

The first thing that the processor does is that; so when interrupt occurs the first thing that it should do is to complete the current instruction.

As you know that every instruction may be spanning over number of clock cycles depending upon the type of instructions so it may span over a number of clock cycles. So,

it should complete the current instruction. After completing the current instruction, the next thing that it should do is to save the return address, save the return address onto stack.

And then it will branch to the interrupt service routine. Now for in case of vectored interrupt the interrupt service routine address is known so that case; branch to external this interrupt service routine is easy or this it is basically a call basically a call instruction. So, this program counter values will be saved onto the stack and it will be going to the interrupt service routine, but in case of non-vectored interrupt, so this ISR address is not known.

So, in that case it will go into some other it will be the it will be doing something extra to get the interrupt service address and from where it will execute after getting that address it will go to the branch to that ISR. Now so, this is the 3 things that the processor does before going to the ISR. Now if we look into; how do we respond to an interrupt. So, this maybe this maybe immediate or delayed depending upon whether the interrupt is maskable or non maskable, and whether interrupts are being masked or not.

Some of the interrupts are maskable interrupts some of them are non maskable interrupt like. In case of processors, we can say the power failure is a non maskable interrupt so, so that if there is a power failure then that is very severe thing because entire system is going to come down. So, that way that is a non maskable interrupt. And some other interrupts may be maskable so that is the user may think that I should not I do this program should not be disturbed by all these interrupts ok. So, that way you can mask out the interrupts so that; goes interrupts will not come, will not disturb the program being executed.

So, this interrupt maybe maskable interrupt, maybe non maskable. So, if it is a maskable interrupt then user has a choice to mask out the interrupt. So, if the interrupts are masked then they will not be responded immediately till the user again unmask the interrupts. So, there are 2 ways of redirecting the execution to the ISR, depending on whether the interrupt is vectored or non-vectored. So, vectored interrupt means the, the microprocessor already knows the ISR address ok, as I already said. And in case of non-vectored interrupt the device will have to supply the vector to the microprocessors so that is the non-vectored interrupt.

(Refer Slide Time: 15:59)

The 8085 Interrupts

- The maskable interrupt process in the 8085 is controlled by a single flip flop inside the microprocessor. This Interrupt Enable flip flop is controlled using the two instructions “EI” and “DI”.
- The 8085 has a single **Non-Maskable** interrupt.
 - The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In case of 8085 processor, the maskable interrupt process in 8085 is controlled by a single flip flop inside the microprocessor, which is known as the interrupt enable flip flop. So, this is controlled by 2 instructions, enable interrupt and disable interrupt EI and DI. So, it has got 8085 as a single non maskable interrupt and this is the so this non maskable interrupt, so this these interrupt cannot be affected by this EI and DI instructions. So, the they are always enabled so whatever program is being executed, if you, if it, if it finds this interrupt has occurred non maskable interrupt has occurred, that will always be serviced by the microprocessor.

So, this is the, this it has got this maskable and non maskable interrupts by and, this maskable interrupts can be masked by DI, instruction they can be unmasked by EI instruction.

(Refer Slide Time: 16:59)

The 8085 Interrupts

- The 8085 has 5 interrupt inputs.
 - The INTR input.
 - The INTR input is the only **non-vectored** interrupt.
 - INTR is **maskable** using the EI/DI instruction pair.
 - RST 5.5, RST 6.5, RST 7.5 are all **automatically vectored**.
 - RST 5.5, RST 6.5, and RST 7.5 are all **maskable**.
 - TRAP is the only **non-maskable** interrupt in the 8085
 - TRAP is also **automatically vectored**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 122

So, 8085 has got 5 interrupt inputs one of the first one is the INTR. So, INTR is the, it is a non-vectored interrupt. In case of 8085, so there is only one non vectored interrupt, which is INTR. And the INTR is maskable using the EI and DI instruction pair. So, it can give this so if the processor thinks or the user thinks that I should not be destroyed by INTR for executing some part of my code. So, it can put a DI instruction at the beginning of that code and then EI at the bottom of the code.

(Refer Slide Time: 17:44)

DI

EI

So, it is like this; suppose this is a piece of program that I want to execute, and out of this code so in this part so I do not want to get disturbed by the maskable interrupts. So, what you do; you put a DI instruction here and then EI instruction here. So, therefore, during this part of execution only the non maskable interrupts, so they can the interrupt the program; maskable interrupts so they will not be able to mask the interrupt the system.

So, in case of this INTR, so this is a non-vectored interrupt. So, this read the ISR address is not known to the processor directly and then this is also a maskable because it is the by means of this EI, DI pair. There are some more instructions RST 5.5, RST 6.5 and RST 7.5 they are automatically vectored. So, they are vectored means their ISR addresses are known. So, the, it will be the processor will jump to a particular address.

And all these RSTs they are maskable so all of them are maskable. And they, they can the trap is the only non maskable interrupt in 8085 and trap is automatically also automatically vectored or trap is basically similar to RST 4.5.

(Refer Slide Time: 19:07)

Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



So, to summarize so INTR RST 5.5 6.5 and 7.5 they are maskable interrupts and trap is a non maskable interrupt. And this then this INTR as per as vectored non vectored classification is concerned; INTR is a non-vectored interrupt and 5.5 RST 5.5 6.5 7.5 and trap so they are all vectored interrupts.

(Refer Slide Time: 19:34)

Interrupt Vectors and the Vector Table

- An **interrupt vector** is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the **Interrupt Vector Table** (IVT).
 - The IVT is usually located in **memory page 00** (0000H-0FFFH).
 - The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.
 - The IVT is divided into several blocks. Each block is used by one of the interrupts to hold its “**vector**”

Now, how this vectoring takes place ok? So, for the vectored interrupt so, there is something called the interrupt vector table or IVT. So, any processor that is supporting vectored interrupt should define some part of memory, which is called the interrupt vector table or IVT. And the specific locations in the IVT are dedicated to different vectored interrupts, and what the processor is expecting is that; in that location so you should put that the interrupt service routine should start from that location.

So, this interrupt an interrupt vector is a pointer to where, the ISR is stored in the memory. All interrupts vectored or otherwise are mapped onto a memory area called interrupt vector table. This these interrupt vector table is usually located in the memory range 0 0 that is 0000 to FFF that is first 56 bytes

And the purpose of this interrupt vector table is to hold the vectors that; redirect the microprocessor to the right place when an interrupts arrives. And IVT is divided into several blocks east block is used by one of the interrupts to hold the vector, hold its vector. So, if you have got a n number of interrupts then this range of this 256 bytes. So, it is divided into n such blocks and each block will have the, it will hold the vector for the interrupt, one vector for the interrupt.

(Refer Slide Time: 21:04)

The 8085 Non-Vectored Interrupt Process

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
Diagram: A horizontal line representing time or states. It has several vertical tick marks labeled 'T'. An arrow points to the last 'T' state, which is labeled 'INT' at its end.
3. If there is an interrupt, the microprocessor will **complete the executing instruction**, and start a **RESTART** sequence.
Diagram: Shows a bracket under the 'T' states labeled 'INT' pointing to a 'RESTART' block. The 'RESTART' block has arrows pointing to 'INTA' and 'INTR'.
4. The RESTART sequence **resets the interrupt flip flop** and **activates the interrupt acknowledge signal (INTA)**.
5. Upon receiving the INTA signal, the **interrupting device** is expected to return the **op-code** of one of the 8 RST instructions.

So, in case of 8085 processor so, the for the non-vectored interrupt process that mean; when it is not non vectored, so, how does it operate? So, this is the vectored process should be enabled using the EI instruction, the EI instruction must be executed sometime earlier by the processor ok, for the interrupts to be enabled. And the 8085 checks an interrupt during the execution of every instruction. So, what happens is that, what happens is that; so, this 8085 instruction, so that is so it is taking a number of cycles, number of T states. So, if suppose this is these so these are the different T states that the processor, the processor is taking for executing.

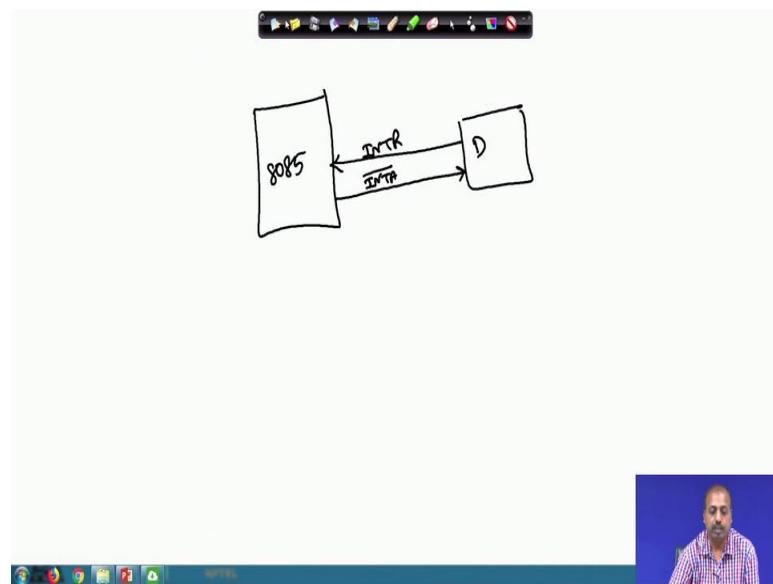
So, if this is the last T state, so in these T state so it will check whether any interrupt has occurred or not. And then if the interrupt has occurred then after finishing the current instruction, so it will go to that interrupt service routine ok. So, this is the point so this last, but one clock cycle the last, but one clock cycle of the total instruction execution ok. So, in every instruction execution the last, but one clock cycle it will check the interrupts. And if it finds that some interrupt is has occurred and it needs to be serviced; then it will be branching to that ISR **at** routine after finishing the current instruction.

So, if there is an interrupt the microprocessor will complete executing the instruction and start a restart sequence. So, restart sequence is the interrupt, somehow the interrupt service process. One thing you must note that it always completes the current instructions. So, it is not that it leave the first the original instruction halfway done, so, it is not like that. It

finishes the current instruction and then it attends to go to the interrupt service process. It starts the restart sequence of operations by which it will be doing the interrupt servicing.

What is this restart sequence? It first resets the interrupt flag that interrupt enable flag IE so this is disabled, so this becomes DI. So, that is, so this and then it will start the one cycle which is known as an interrupt acknowledge cycle. So, in case of 8085 if you remember that there were 2 pins, one is the 1 pin was the INTR which is an interrupt line and then there was another pin going out which is \overline{INTA} interrupt acknowledge bar. So, this actually tells that this is actually used for to that for that you to tell the device that, the interrupt has been received by the 8085 processor.

(Refer Slide Time: 24:05)



So, if this is your 8085 processor and this is the device, now this device has given an interrupt to the 8085. Now the device must be told that that is that its interrupt has been accepted ok, then the processor is going to do the associated operation somehow this has to be told ok. So, that is the role of this INTR and \overline{INTA} lines. And as you know that since INTR the address is the ISR address is not known, then the device should somehow supply this interrupt address the interrupt service address to the 8085 processor. So, we will see how it can be done.

Now, in case of so, upon receiving this INTA signal, the interrupting device is expected to return the op code of one of the RST instructions, one of the 8 RST instructions.

(Refer Slide Time: 25:12)

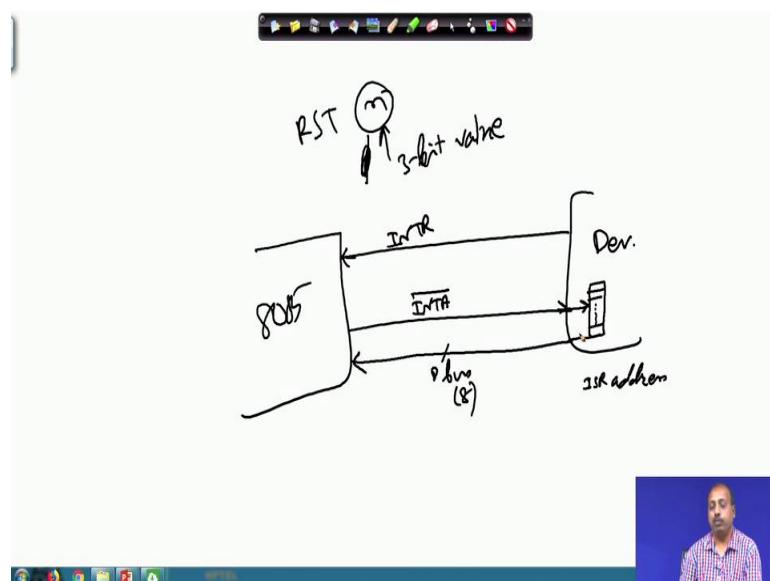
The 8085 Non-Vectored Interrupt Process

6. When the microprocessor executes the RST instruction received from the device, it saves the address of the next instruction on the stack and jumps to the appropriate entry in the IVT.
7. The IVT entry must redirect the microprocessor to the actual service routine.
8. The service routine must include the instruction EI to re-enable the interrupt process.
9. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | 126

So, so in general, this RST instructions are like this, so, this so this RST n.

(Refer Slide Time: 25:32)



So, this n can be a 3-bit value, this n can be a 3-bit value. And what the processor does is that; it multiplies this, what the processor does is that it is like this, so this is the 8085 processor and this is the device.

Now, it has given the interrupt and it has put the interrupt acknowledge line. Then the processor the 8085 expects that the device will put one 8-bit code here. So, this is the data bus line, this is the data bus line of the processor and it expects that one 8-bit value will be

coming here, which will correspond to one of this RST n instructions ok. So, upon getting that; it may be that device has got some special register here one 8-bit register here, which holds the ISR address, which holds the ISR address. And this interrupt acknowledge signal when it reaches this register. So, it puts the content of those content of the register on to the data bus.

In this way, the device may have some device address register; and that address register is nothing but the address of the ISR that it is executing that, that that should be executed to get the interrupt service. So, this data bus on the data bus so it will expect that one of the RST instruction code will be put here and this 8085 processor so it will execute the RST instruction then. So, so that this RST instruction so that will actually transfer the control to some location this whatever value be, whatever be the value of this n. So, depending upon that it will jump to a predefined pre specified location and from that point onward the processor will start executing.

So, it is expected that this interrupt service routine for this device is located from that particular address. So, this way this vectored interrupt can be non vectored interrupt. So, this non-vectored interrupt so this so it will be it will be doing it so, this will be yeah. So, this INTA receiving, the INTA signal this interrupting device is will put the output of one of the 8 RST instructions and upon getting this op code, this RST instruction. So, it will save the address of the next instruction in the stack and jump to the appropriate entry in the interrupt vector table.

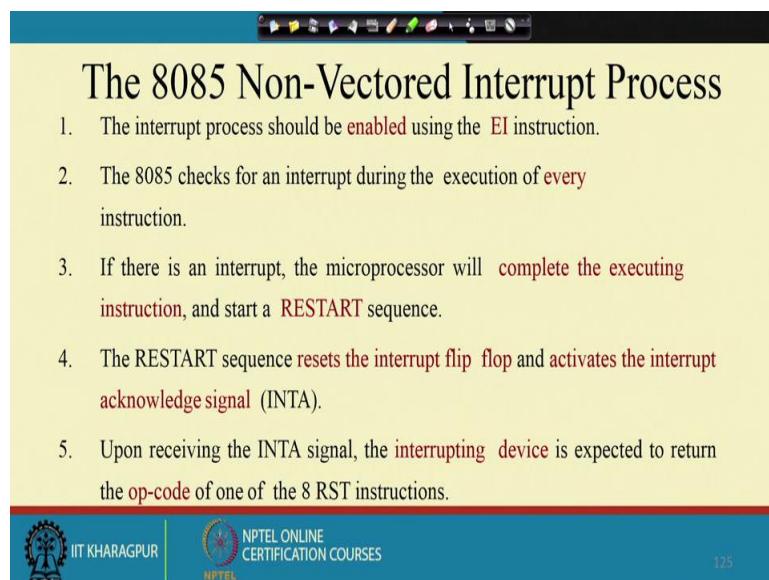
And this interrupt vector table entry it will it may redirect must redirect the microprocessor to the actual service routine, because the space may be small. So, in the interrupt vector table, so it may hold a branch instruction to a later part in the memory where it has it actually loads the entire service routine. Then the service routine will be executed and the at the end of the service routine there should be one return instruction, that will return the execution to the original program at where the interrupt was received.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 58
8085 Microprocessor (Contd.)

In our last class we started discussing on 8085 non vectored interrupt process. So, as we know that in case of non vectored interrupt, the interrupting device it should provide the ISR addresses, routine, address for the interrupt service routine.

(Refer Slide Time: 00:26)



The slide has a yellow background and a blue header bar. The title 'The 8085 Non-Vectored Interrupt Process' is centered at the top. Below the title is a numbered list of 5 points. At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES' and the number '125'.

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
3. If there is an interrupt, the microprocessor will **complete the executing instruction**, and start a **RESTART** sequence.
4. The RESTART sequence **resets the interrupt flip flop** and **activates the interrupt acknowledge signal (INTA)**.
5. Upon receiving the INTA signal, the **interrupting device** is expected to return the **op-code of one of the 8 RST instructions**.

Now, for the pre requisite for this Non-Vectored Interrupt Process to be initiated is that the interrupt process should be enabled using the EI instruction. So, somewhere in the beginning, so when the 8085 processor is reset, this all the interrupts are enabled.

However, due to, due to the course of some program execution maybe the interrupts have been disabled, so, through some DI instruction. So in that case, these interrupt will not be sensed. So, it is required that before this non vectored interrupt it can send, give an interrupt to the processor, the process should be enabled by means of an EI instruction. 8085 checks for an interrupt during the execution of every instruction.

So, as I said it is the last, but one clock cycle of an instruction at which the interrupt status is checked. So, that way, it is at the end of that instruction, so it will check whether an

interrupt has occurred or not. And if an interrupt has occurred, then instead of executing the next instruction in the sequence the processor goes to the interrupt service routine. If there is an interrupt, the microprocessor will complete executing the current instruction and start a restart sequence. So, restart sequence, it is basically a sequence of operations by which the service routine can be started.

So, first of all, it resets the interrupt flip flop. So, there we will see later that there is an interrupt enable flip flop. So, that flip flop is reset in the beginning of the restart sequence. And also, you remember that there was a pin of 8085 called interrupt acknowledge. So, INTA which is an active low signal and this is activated. So, interrupt acknowledge line is made low. So, from the device an interrupt signal has come and this interrupt acknowledge signal is generated by the processor and then it goes to the device.

So, device will now understand that my interrupt has been acknowledged. So, I have to provide the service routine address. So, upon receiving the interrupt acknowledge signal, the interrupting device is expected to return the op code of one of the 8 RST instructions. So, RST is the restart instruction. So, it takes along with it one value which is a 3 bit value. And so, 3 bit value means it can go from 0 to 7. So, 8 possible RST instructions are there starting with RST 0 going up to RST 7 as you will see subsequently.

(Refer Slide Time: 03:04)

The 8085 Non-Vectored Interrupt Process

6. When the microprocessor executes the RST instruction received from the device, it saves the address of the next instruction on the stack and jumps to the appropriate entry in the IVT.
7. The IVT entry must redirect the microprocessor to the actual service routine.
8. The service routine must include the instruction EI to re-enable the interrupt process.
9. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

8085 ZSR

8085

JMP 8085

IIT Kharagpur | NPTEL Online Certification Courses

126 08:00 AM 24-04-2018

So, the process the device somehow sends an RST instruction to the microprocessor. So, microprocessor now executes the RST instruction received from the device on the data bus

lines. It saves the address of the next instruction on the stack and jumps to the appropriate entry in the interrupt vector table. So, the appropriate entry is determined by the RST instruction that it is executing. So, based on the value of n in the RST n instruction, so it goes to a particular location in the IVT, Interrupt Vector Table and it starts executing from that point.

Now, the IVT entry, so it must redirect the microprocessor to the actual service routine. Actually what happens is that in that; so all the 8 RST services that we have. So, they are clubbed together. So, we do. So, if we start writing the interrupt service routine from that point onward in the IVT, then it will not have space for the for the interrupt, the interrupt service routines for other interrupts.

So, what is done? We normally put a jump instruction there, so that the processor executes the jump instruction and somewhere later the ISR maybe there. So, it is like this that suppose upon getting the interrupt the, so it comes to this particular location in the IVT. So, if your actual interrupt service routine is located from say location say 8000 onwards. So, this is the actual ISR. So, what it does is, it put puts a jump instruction here jump 8000.

So, upon getting the interrupt the processor comes to this point and execute the jump instruction. As a result it goes to the ISR and executes the Interrupt Service Routine. So, that is how this I that that is what is meant here the IVT entry must redirect the processor to the actual service routine. Now, the service routine must include the EI instruction because as soon as an interrupt occurs in the restart sequence the, the all the interrupts are disabled. And so, the very first thing that the interrupt service routine should do is to use the instruction EI to re-enable the interrupt process.

So, if it is not re enabled, then a subsequent interrupt will not be sensed by the processor. So, that may be a problem. Of course, if the user wants that I the my interrupt service routine should not be interrupted further then may not put the EI instruction at the beginning rather maybe towards the end of the interrupt service routine, this EI instruction is kept. So, at the end of the service routine, there must be one return instruction that returns the execution to where the program was interrupted.

So, every interrupt service routine should end with a return instruction and upon getting this return, the processor will take out the return address from the stack and it will return to that location.

(Refer Slide Time: 06:08)

The 8085 Non-Vectored Interrupt Process

- The 8085 recognizes 8 RESTART instructions: RST0 - RST7.
 - each of these would send the execution to a predetermined hard-wired memory location:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

Here is actually that IVT interrupt vector table. So, we have got this as I said that there are 8 possible non vectored interrupt RST instructions, RST 0 through RST 7 and each of these should send the execution to a predetermined hard wired location.

So, if the instruction is RST n, so what the processor does is that this n is multiplied by 8. And whatever be the result, so it jumps to that particular location. For example, this RST 0 it is transferring the control to the memory address 0, then RST 1, it will transfer the control to memory address 8, RST 2 will transfer to memory address 16 like that.

So, you can see that the RST 0 instruction is equivalent to this call 0000 hex. Similarly, RST 1 is equivalent to call 0008 hex. So, like that we have got this RST instructions they are equivalent to this call instructions. So, so you can understand that between RST 0 and RST 1, we have got only 8 bytes of location are free. So, if your ISR is very short, it can hold it can be held within 8 bytes.

So, you can start immediately you can start writing the interrupt service routine from the location from that location itself. So, but, normally we any meaningful ISR will have more than 8 bytes in length. So, it is not possible to hold that entire interrupt service routine. So, that is why, we normally put a jump instruction there and jump to the actual ISR service actual ISR which is located somewhere later in the memory. So, this is how this processor this RST instructions are sensed by the processor.

(Refer Slide Time: 08:11)

The restart sequence is made up of three machine cycles

- In the 1st machine cycle:
 - The microprocessor sends the INTA signal.
 - While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
- In the 2nd and 3rd machine cycles:
 - the 16-bit address of the next instruction is saved on the stack.
 - Then the microprocessor jumps to the address associated with the specified RST instruction.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, coming back to the restart sequence, so restart sequence it is made up of three machine cycles. So, that it is actually the execution of the RST instruction. So, how it does in the first machine cycle, the microprocessor sends the INTA signal and while this INTA is active the microprocessor reads the data lines expecting to receive from the interrupting device, the opcode for the specific RST instruction. So, RST n is a single byte instruction out of which the 3 bits are reserved for the value of n and rest of the bits identify the opcode RST.

So, what it says is that in the first machine cycle. So, maybe, may be the processor is executing some instruction and the end of it finds that there is one interrupt ok. So, RST n there is an interrupt on the INTA line. So, in the next machine cycle next instruction, after completing the current instruction; in the next machine cycle, it will activate the INTA signal. Getting the INTA signal, we said that the device is expected to put the address of the ISR onto the data bus which is in terms of the of the one RST instruction.

So, after putting INTA signal onto the to the device, the processor will read the data bus lines because it is expecting that the device has put one RST instruction onto the data bus lines. So, after getting that, in the second and third machine cycles, the next return address is saved onto the stack the 16 bit return address for the next instruction is saved onto the stack. And depending upon this value of n in the RST n instruction, the processor jumps to the address associated with the specified RST instruction. For example, if it is RST 2, it

will jump to the location 16. So, like that it will jump to the corresponding location and start executing program from there.

(Refer Slide Time: 10:10)

The slide has a yellow background with a black header bar containing icons. The title 'Restart Sequence' is centered in a large, bold, black font. Below the title is a bulleted list:

- The location in the IVT associated with the RST instruction can not hold the complete service routine.
 - The routine is written somewhere else in memory.
 - Only a JUMP instruction to the ISR's location is kept in the IVT block.

At the bottom of the slide, there is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it shows the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a small video window showing a man in a yellow shirt.

The location in the IVT associated with the RST instruction cannot hold the complete service routine as I said that service routine, we have got only 8 bytes free for holding this interrupt service routine at the location of the IVT. So, actual routine is written somewhere else in memory and only a jump instruction to the ISRs location is kept in the IVT block. So, normally this is the standard procedure for writing the interrupt service routine of any processor. So, in the IVT we keep a jump instruction and from there, it jumps to a particular some address and in that address the actual ISR is located.

(Refer Slide Time: 10:49)

Hardware Generation of RST Opcode

- How does the external device produce the opcode for the appropriate RST instruction?
 - The opcode is simply a collection of bits.
 - So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.



Now, the question is how and how a device can generate the opcode RST opcode ok. So, this is a very important thing because device is for some for some special purpose, but it has to do this additional job if it wants to be interfaced with 8085 using this INTR line. So, how it does the, as we know, that any opcode is nothing but one 8 bit collection of bits ok. So, device needs to set the bits of the data bus to the appropriate value in response to a to one INTA signal.

So, this is what has to be done.

(Refer Slide Time: 11:26)

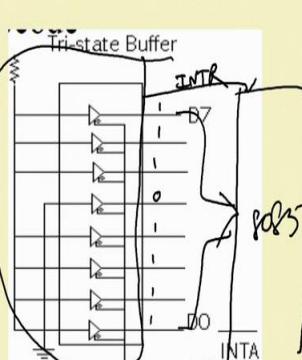
Hardware Generation of RST Opcode

RST 5's opcode is EF =

D D
7 6 5 4 3 2 1 0
1 1 1 0 1 1 1 1

INTA

Dev.



This is the strategy. Suppose, so, this is suppose this device this is a device and if this device has got a tri state buffer which is an 8 bit bus and the. So, this side I have got the, I have got the microprocessor. So, this side we have the microprocessor, this side we have got that 8085 and this D0 to D7, they are actually forming the data bus.

So, this is the data bus line for the 8085. Now, when this interrupt signal came to the processor the INTR signal came to the processor, the processor has activated this INTA line and this INTA line has been connected to a set of tri state buffers. So, when this line is 0 all this buffers are enabled. As a result, if you look into this connection pattern. So, these bits are all 1. So, and this bit is 0 and this bits are again all 1. So, if you look into the 8085 manual, you will find that this particular bit pattern 1 1 1 0 1 1 1 1. So, this corresponds to this RST 5 instruction, ok.

So, what happens is, when the device gives an interrupt and the processor gives interrupt acknowledgement, so, this tri state buffers within the device are enabled. So, that this is inside the device this whole thing is inside the device. So, when these interrupt acknowledgement comes, so, the device can very easily put the data bus on to the data bus this bit pattern 1 1 1 0 1 1 1 1 as a result, when 8085 will read it will understand that this is a RST 5 instruction and it will jump to the location $5 \times 8 = 40$.

So, it will go to the location 40 and start executing from there. And as I said, most probably there will be a jump instruction to the actual routine at the location 40.

(Refer Slide Time: 13:38)

The slide has a yellow header bar with a toolbar icon. The main title is 'Hardware Generation of RST Opcode'. Below the title is a bulleted list:

- During the interrupt acknowledge machine cycle, (the 1st machine cycle of the RST operation):
 - The Microprocessor activates the INTA signal.
 - This signal will enable the Tri-state buffers, which will place the value EFH on the data bus.
 - Therefore, sending the Microprocessor the RST 5 instruction.
- The RST 5 instruction is exactly equivalent to CALL 0028H

At the bottom, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES' and the number '132'.

So, during the interrupt acknowledge machine cycle, the microprocessor activates the INTA signal the signal will enable the tri state buffers which will place the value EFH onto the data bus. Therefore, the sending the microprocessor RST 5 instruction; so this is the standard procedure by which this interrupt INTR line and the device gets the corresponding, device produces the corresponding RST instruction for doing the for the processor to start the interrupt service routine.

Now, you see that this is one of the approach by which a device can generate this RST instruction. So, there can be many other ways by which it can be done. So, this is just an example ok. So, some device may be more intelligent and it can do it in a better fashion, but this is one of the very basic technique where you do not need anything more than a few tri state buffers to generate the RST instruction.

(Refer Slide Time: 14:40)

The slide has a title 'Issues in Implementing INTR Interrupts'. Below the title is a bulleted list:

- How long must INTR remain high?
 - The microprocessor checks the INTR line one clock cycle before the last T-state of each instruction.
 - The interrupt process is Asynchronous.
 - The INTR must remain active long enough to allow for the longest instruction.
 - The longest instruction for the 8085 is the conditional CALL instruction which requires 18 T-states.

Below the list is a note in red text: 'Therefore, the INTR must remain active for 17.5 T-states.'

The footer of the slide includes the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, 'NPTEL ONLINE CERTIFICATION COURSES', and the number '133'.

Next question is, suppose the INTR line has been activated, so, a device wants to interrupt the processor, microprocessor and it has raise the INTR signal.

Now, how long this INTR should remain high now as I said that the microprocessor checks the INTR line, 1 clock cycle before the last T state of each instruction. So, whenever a microprocessor whenever microprocessor executing some instruction, when it comes to the last but one T state. So, at that time it will try to see whether this interrupt has come or not. The interrupt process is asynchronous. So, it can come at any point of time. So,

naturally this interrupt must remain active long enough to allow the longest to allow for the longest instruction.

So, that to be on the safe side, if you want that your interrupt should always be sensed by the microprocessor, then it has to be done in such a fashion that even for the longest instruction, the interrupt line is active till the last but one clock cycle. In case of 8085, the longest execution time is for conditional call instruction which requires 18 T-states. So, to be sensed properly the INTR pin must remain active for 17.5 T-states ok

So, now you can understand that if I have got a 2 megahertz clock frequency than that is 17.5 divided by 2 megahertz. So, that, so much of time this interrupt line must be high. So, if the pulse is shorter than that if the interrupt line is not high for that much time it may. So, happen that the processor misses that point and it is not sensed by the processor.

So, this is the guideline for raising the INTR signal.

(Refer Slide Time: 16:32)

Issues in Implementing INTR Interrupts

- How long can the INTR remain high?
 - The INTR line must be deactivated before the EI is executed. Otherwise, the microprocessor will be interrupted again.
 - The worst case situation is when EI is the first instruction in the ISR.
 - Once the microprocessor starts to respond to an INTR interrupt, INTA becomes active (=0).

Therefore, INTR should be turned off as soon as the INTA signal is received.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

And how long can the INTR remain high? So, that is once it is made high how long can it remain high? The INTR line must be deactivated before the EI signal is executed. So, what, so as because as soon as this interrupt occurs, so, the processor automatically disables the interrupt and before enabling the interrupts again, so you should deactivate the INTR signal otherwise the microprocessor will find that again another interrupt has come. So, it

will take it the same interrupt. So, it will be sensed twice. So, it will be taken as two different interrupts.

Now, in case of worst, in the worst case situation the in the ISR the very first instruction may be the EI instruction. So, once the microprocessor starts to respond to INTR interrupt, INTA becomes active. So, that way because this EI is the first instruction, so it becomes active. So, INTR should be turned off as soon as the INTA signal is received as soon as the processor gets the as soon as the device gets the interrupt acknowledge signal. It should withdraw the INTR signal. Otherwise there is a chance that in the interrupt service routine. This enable interrupt will be the very fast operation as a result the processor will sense the interrupt once more ok.

So, this is the guideline. So, it should be, so when the INTR when the to interrupt the processor properly, device should be ready to keep the interrupt line high for 17.5 clock cycles and as soon as the interrupt acknowledge signal is received, it should deactivate the INTR signal.

(Refer Slide Time: 18:10)

The slide has a yellow header bar with a toolbar icon. The main title is "Issues in Implementing INTR Interrupts". Below the title is a bulleted list:

- Can the microprocessor be interrupted again before the completion of the ISR?
 - As soon as the 1st interrupt arrives, all maskable interrupts are disabled.
 - They will only be enabled after the execution of the EI instruction.

Below the list is a hand-drawn illustration of a vertical rectangle with a jagged top edge, resembling a piece of paper or a torn edge, with a small sketch of a hand holding a pen next to it.

Text below the illustration states: "Therefore, the answer is: ‘only if you allow it to’. If the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done."

The footer of the slide includes the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

Issues in implementing INTR interrupts, so like can the microprocessor be interrupted again before completion of the ISR. So now, the question is dependent on the style in which the ISR has been written. As soon as the first interrupt arrives, all maskable interrupts are disabled and INTR being a maskable interrupt, so, it also gets disabled and they are enabled only after execution of the EI instruction.

So, this is specified by the processor designer. So, 8085 designer, so they have told that this is the guide line. Now, the point is that if you are, if you allow the processor to be interrupted again like if your, if this is the ISR I have written and my EI is put here itself, now there is a very high chance that while I am still in the ISR another interrupt comes from the device.

So, that way it is again another interrupt will be sensed by the processor and it will again come to restart the ISR. So, that can happen or in the other case, you can put the EI at the bottom of this interrupt service routine as a result this will not be interrupted again. So, if it is put at the end, then this interrupts will not be coming again till the first one is over. That is why the answer is only if you allow it to if the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done. So, that that is the problem.

So, if you do not want that, you should put this EI instruction at the bottom. But again putting at the bottom has got some other consequence because you can that will also defer other more important instruction interrupts as well.

(Refer Slide Time: 20:04)

The slide has a yellow header bar with a toolbar icon. The main title is "Multiple Interrupts & Priorities". Below the title is a bulleted list:

- How do we allow multiple devices to interrupt using the INTR line?
 - The microprocessor can only respond to one signal on INTR at a time.
 - Therefore, we must allow the signal from only one of the devices to reach the microprocessor.
 - We must assign some priority to the different devices and allow their signals to reach the microprocessor according to the priority.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video frame showing a person in a yellow shirt.

So, if you have got multiple interrupts, then which interrupt should be having higher priority? So, that is one problem. Now, how do you allow multiple devices to interrupt using the INTR line? So, I have got a single interrupt line. Now if I have got a number of devices that can that can give interrupt to the, that can use this non vectored interrupt to the 8085 processor.

Now, how to do this? Now, microprocessor can respond to only one signal on the INTR at a time. So, it can since it is a uniprocessor system, so, it will be responding to only one device request. So, we must allow the signal from only one of the devices to reach the microprocessor. So, that is important. So, somehow we have to do that that processor does not get interrupt from multiple sources, because there is ultimately there is only one pin in the microprocessor.

So, you must assign some priority to different devices and allow their signals to reach the microprocessor according to the priority. So, this is the issue that is we have to somehow prioritize the devices and some device which has got higher priority should be able to interrupt the microprocessor before other low priority devices are interrupting it.

(Refer Slide Time: 21:24)

The Priority Encoder

The solution is to use a circuit called the priority encoder (74366).

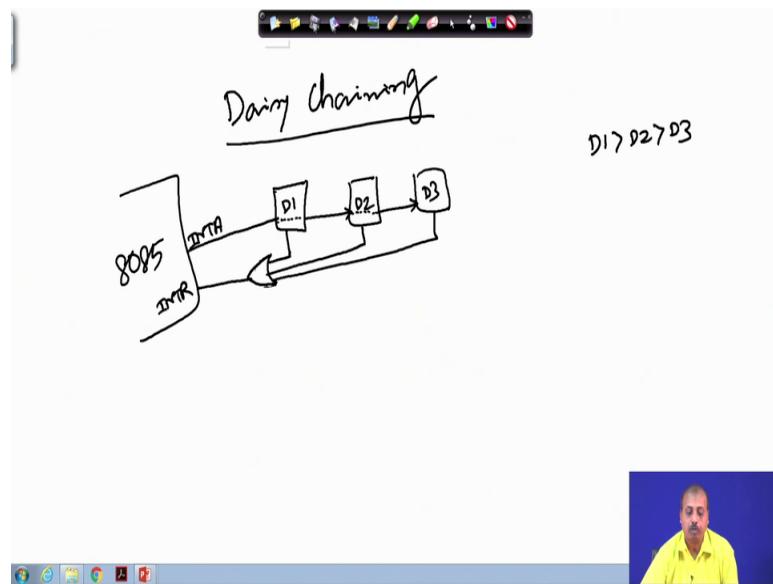
- This circuit has 8 inputs and 3 outputs.
- The inputs are assigned increasing priorities according to the increasing index of the input.
 - Input 7 has highest priority and input 0 has the lowest.
- The 3 outputs carry the index of the highest priority active input.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL



So, how to do this? So, there is a priority encoder 74366. So, I will come to that later. So, there is a simpler scheme by which you can do this which is known as daisy chaining.

(Refer Slide Time: 21:36)



So, daisy chaining policy is like that suppose I have got a number of devices. So, this is D1, D2, D3, like that and I have got the microprocessor 8085 here.

Now, what I want is that D1 will have the highest priority followed by D2 followed by D3 fine. Now how to do this thing? So, what we can do? So, we can take this interrupt line from this individual devices and pass them through an OR gate, pass them through an OR gate to get and connect it to the interrupt line of the processor. So, this is the OR gate we connect them and connect it to the interrupt line of the micro of the microprocessor.

Now, this interrupt acknowledge line, so this is connected to the interrupt acknowledge of the first D1 and then this is the INTA. And this INTA line, so it is passed from the device D1 to device D2 and from device D2 to device D3. It is passed like this. So, the idea is that if first device has raise the interrupt when the interrupt acknowledgement comes. So, it does not pass this interrupt acknowledge line to the output. So, it just consumes it.

So, it understands this is for me. So, it generates the corresponding RST instruction for the 8085. Now, if it happens the, if it happens like this that this D1 has not generated the interrupt, in that case this INTA line will be passed by D1 and it will reach D2. If D2 had raised the interrupt, then it will use this interrupt acknowledge line to generate the next to generate the next RST instruction. Otherwise it will pass this interrupt acknowledge line to the device D3. So, this way you see the devices which are closer to the processor, they

are having higher priorities compared to the devices that are further away from the processor.

So, this particular policy is known as Daisy Chaining policy. So, this is very simple to implement by means of some very simple logic. You can decide whether this devices can have some simple logic to decide whether the interrupt acknowledge line should be passed or not ok. So, this is one policy by which we can do this. The other policy that we have is by means of a Priority Encoder.

So, this 74366 chip, so, this is a priority encoder, this circuit has got 8 inputs and 3 outputs. The inputs are assigned increasing priorities according to the increasing index of the input. So, 0 has got the lowest priority going to 7. So input 7 has the highest priority and 0 has the lowest priority. And three outputs, they carry the index of the highest priority active input. So, this is an encoder. So, this encodes this 8 bit input into a 3 bit output.

So, this actually identifies the index of the interrupt line that is the highest priority interrupt line that we have.

(Refer Slide Time: 25:10)

The slide has a yellow header bar with the title "Multiple Interrupts & Priorities". Below the title is a white content area containing two bulleted lists:

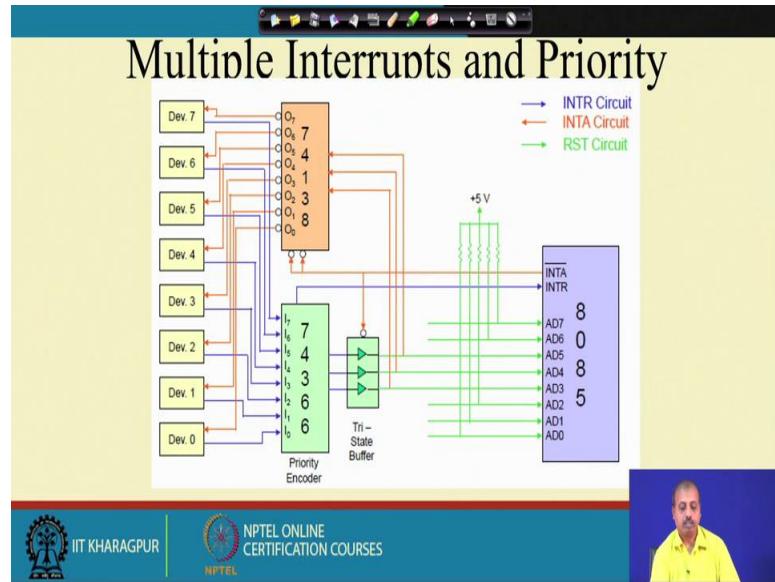
- Note that the opcodes for the different RST instructions follow a set pattern.
 - Bit D5, D4 and D3 of the opcodes change in a binary sequence from RST 7 down to RST 0.
 - The other bits are always 1^b.
 - This allows the code generated by the 74366 to be used directly to choose the appropriate RST instruction.
- The one draw back to this scheme is that the only way to change the priority of the devices connected to the 74366 is to reconnect the hardware.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video thumbnail of a man in a yellow shirt.

Opcodes for different RST instructions follow a particular pattern. So, if you look into this opcode for the RST instruction, you will find that there is an 8 bit instruction in which these bits D5, D4 and D3. So, they will determine the number n in the RST n, they will determine the value of n. And the rest of the bits are always 1, rest of the all other bits

are always 1. So, bits D7, D6 and then D2, D1, D0. So, they are all 1. So, this allows 74366 to generate this RST instruction directly. However, the problem that we have is that to change the priority, you have to change the connection pattern of the devices

(Refer Slide Time: 26:02)



So, we have got a connection like this. So, this is our 8085 processor. Now, this 74366, so, this is the priority encoder. So, when this device is getting this interrupts, so, this I₀, I₁, I₂ like that. So, accordingly, it passes through this tri state buffer when this interrupt acknowledge signal comes. So, this buffer is enabled as a result this 3 bit output. So, it is coming onto this line.

(Refer Slide Time: 26:37)

The 8085 Maskable/Vectored Interrupts

The 8085 has 4 Masked/Vectored interrupt inputs.

- RST 5.5, RST 6.5, RST 7.5
 - They are all **maskable**.
 - They are **automatically vectored** according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

- The vectors for these interrupt fall in between the vectors for the RST instructions. That's why they have names like RST 5.5 (RST 5 and a half).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

This 3 bit output is coming here and that connects to the data bus line D3, D4 and D5 and rest of the lines. So, they are raised high ok. So, by this they are all raised high.

Now, when this and this in the 74366 has got an interrupt line; so this interrupt line gives this INTR signal. So, that is given to a connected to the 8085 similarly \overline{INTA} line is connected to this 74138. So, this is another 3 to 8 decoder. So, this decoder is enabled based on whichever device has generated this, whichever device has been accepted here for the highest priority one. So, that value is again you see passed here. So, as a result, the corresponding device will get the interrupt acknowledgement, some sort of interrupt acknowledgement signal.

So, we can say. So, that line will be made 0 and the rest of the lines will be made 1. Suppose this device 2 had raised the interrupt, device 2 is the highest priority device that had raised the interrupt, then what will happen, here you will get the pattern 0 1 0, and upon getting 0 1 0 here this O2 line will be 0 and rest of the lines will be 1, so the two line being 0, so it will come to this device 2. So, device 2 sees it as an interrupt acknowledgement line as if the interrupt acknowledgement line has been passed to it. So it can accordingly reset the interrupt line and all to the rest of the things.

So, this way we can use this 74366 and 74138. So, these two encoder priority encoder and decoder in conjunction to interface multiple interrupts with different priority schemes.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 59
8085 Microprocessor (Contd.)

Next, we will look into the 8085 in the maskable or vectored interrupts.

(Refer Slide Time: 00:22)

The 8085 has 4 Masked/Vectored interrupt inputs.

- RST 5.5, RST 6.5, RST 7.5
 - They are all **maskable**.
 - They are automatically **vectored** according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

Handwritten notes on the left side of the table:
5.5 × 8 = 42
6.5 × 8 = 52
7.5 × 8 = 60

The vectors for these interrupt fall in between the vectors for the RST instructions. That's why they have names like RST 5.5 (RST 5 and a half).

INT# → 8085
RS# → 8085
RS# → 8085
RS# → 8085

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, 8085 has got 4 masked or vectored interrupt inputs – RST 5.5, 6.5 and 7.5 and there is one non maskable interrupt so, which is which is actually the trap instruction the trap interrupt. So, we will come to that later. Out of them this RST 5.5, 6.5, 7.5 so, they are all maskable interrupts and they are automatically vectored according to this particular table.

So, this 5.5 this is a pin; so unlike that RST instruction that we were discussing that INTR so, that we are having generating the RST instruction. So, if you look into this 8085 processor then you will find that apart from this INTR pin for generating for giving the interrupt so, we have got three more pins which are RST 5.5. They are named as RST 5.5, RST 6.5 and RST 7.5. So, these are the names of the pins and this name also tells the vector address.

So, like any RST instruction execution what happens is that the number is multiplied by 8. So, 5.5×8 , so, that gives me 42, ok. So, that is in hexadecimal so, that is equal to 2C

hex. Similarly, 6.5 is 6.5×8 so, that is 52 and 52 is again 34 hex. So, this way we have got this RST pins that are for they are also their addresses are also computed by multiplying this value of this 5.5, 6.5 etcetera by 8 and whatever value you get so, that becomes the interrupt address, ok, the ISR address that is why this is called vectored interrupts. So, unlike INTR where the device was providing the ISR address so, here it is not there. So, here it is hardcoded. So, if the interrupt comes in any of these lines then automatically the processor will branch to the corresponding address 2C, 34 or 3C.

Next so, this vector so, they are automatically vectored according to the this according to this table and the vectors for these interrupt fall in between the vectors for the RST instruction and that is why they are named like RST 5.5, so 5 and a half. So, therefore this 5.5 falls between RST 5 and RST 6 and if you look into the address; so, you can understand that this 2C is between the addresses for 5×8 and 6×8 , between those two so, this address comes, that is why the names are like that.

(Refer Slide Time: 03:17)

Masking RST 5.5, RST 6.5 and RST 7.5

These three interrupts are masked at two levels:

- Through the Interrupt Enable flip flop and the EI/DI instructions.
 - The Interrupt Enable flip flop controls the whole maskable interrupt process.
- Through individual mask flip flops that control the availability of the individual interrupts.
 - These flip flops control the interrupts individually

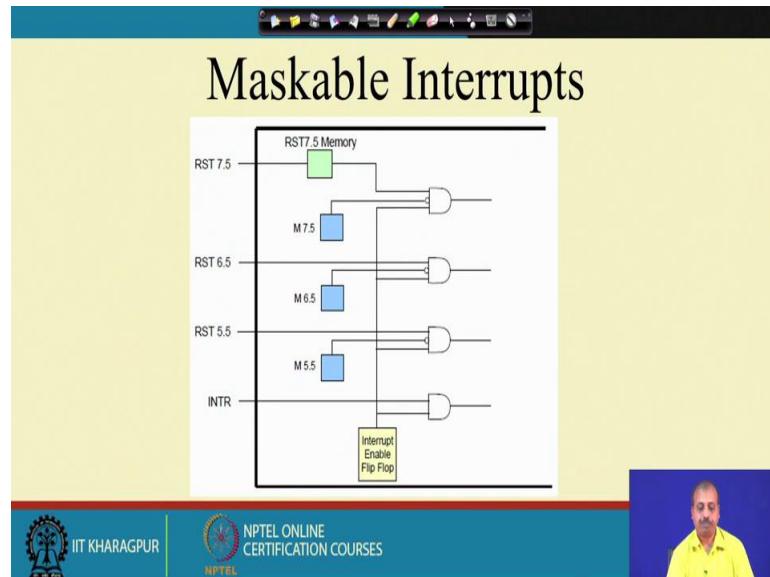
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

Now, these three interrupts are masked at two levels, so that this masking is a very important thing because sometimes we want that the processor should not be interrupted. So, we want to mask out this interrupts. So, we do not want the interrupts to reach the processor, then this masking can be done at two different levels; the first level of masking is via the interrupt enable flip flop and the EI DI instructions. So, in the 8085

microprocessor that is one interrupt enable flip flop called IE and this IE flip flop can be set or reset by using this EI and DI instructions.

So, this call so if the IE instruction is executed then this interrupt enable flip flop is equal to one and now, all the none of the interrupts are masked and then there if it is 0 then it is allowed to give the interrupts. So, this is one type of masking, then other type of masking that we have is the individual mask. There are individual mask flip flops that can control availability of individual interrupts at individual mask interrupt level also you can control the this interrupts, ok. So, if there are flip flops dedicated for individual interrupts.

(Refer Slide Time: 04:32)



So, this is the conceptually we can think of it like this as if this is finally, the interrupt lines going to the processor. So, if you say that these are going to the actual processor then this interrupt enable flip flop, so, that bit is ANDed with ANDed here. So, if this interrupt enable flip flop is 0, you see that none of the interrupts will be made equal to 1. So, all of them will be equal to 0. So, as if it is masked off and then this individually you can control then RST 7.5. So, you can, this is so, will come to this point later.

Now, this if you want to mask out this flip flop this interrupts individually instead of through this interrupt enable flip flop so, you can set this mask bit M 7.5, M 6.5, or M 5.5. So, if I set this bit to 1 and the rest of the bits to 0 then this set to 1 so, this will make this AND gate output 0. Similarly, if this bit is 0, then this is so, this inverted one comes as once it does not have any control over the output of the AND gate. So, it will be determined

by other inputs. So, you see that by putting a 1 here so, we can mask out the corresponding interrupt, ok. So, that is one type of masking.

And, this RST 7.5 it has got a special memory. So, it can remember whether one interrupt had occurred, maybe the interrupt is masked out. So, it does not reach the processor at present, but the mask with the memory bit of RST 7.5 will be set. So, later on the user program so, it can check that whether any interrupt had occurred when it was masked off. So, that for that purpose so, there is a memory bit for RST 7.5.

(Refer Slide Time: 06:27)

The 8085 Maskable/Vectored Interrupt Process

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
3. If there is an interrupt, **and if the interrupt is enabled using the interrupt mask**, the microprocessor will **complete the executing instruction**, and **reset the interrupt flip flop**.
4. The microprocessor then executes a call instruction that sends the execution to the **appropriate** location in the interrupt vector table.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

So, how it work? How it works? It is like this. The interrupt process should be enabled using the EI instruction so that is true for every interrupt to reach the processor and as I said that 8085 checks for an interrupt during the execution of every instruction. If there is an interrupt and if interrupt is enabled using the interrupt mask the microprocessor will complete the executing instruction and reset the interrupt enable flip flop. So, this is the extra thing that is added here. So, previously we did not note this particular part. So, if the interrupt enabled interrupt is enabled using the interrupt mask. So, it is either by setting those mask bits or the interrupt enable flip flop.

So, if the interrupt is allowed if the interrupt is not masked out then the after completing the current execute current instruction. The processor will reset the interrupt flip flop that interrupt and then it will branch to the appropriate location in the interrupt vector table. So, processor will then execute a call instruction that sends the execution to the appropriate

location in the interrupt vector table. So, that is how this maskable vectored interrupt process continues.

(Refer Slide Time: 07:43)

The 8085 Maskable/Vectored Interrupt Process

5. When the microprocessor executes the call instruction, it saves the address of the next instruction on the stack.
6. The microprocessor jumps to the specific service routine.
7. The service routine must include the instruction EI to re-enable the interrupt process.
8. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



And, similarly, when the microprocessor executes the call instruction it saves the address of for the next instruction onto the stack, the return address. The processor will jump to the specific service routine given by the RST instruction that it has got and then or the RST pin on which the interrupt had come. The service routine must include the instruction EI so that this interrupt is re enabled. So, otherwise the interrupt will remain disabled so, it will not be sensed further.

Then, at the end of the service routine the return instruction the RET it will return the execution to where the program was interrupted. So, this is the whole sequence that occurs in case of this mask interrupts.

(Refer Slide Time: 08:29)

Manipulating the Masks

- The Interrupt Enable flip flop is manipulated using the EI/DI instructions.
- The individual **masks** for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the **SIM** instruction.
 - This instruction takes the bit pattern in the Accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts.

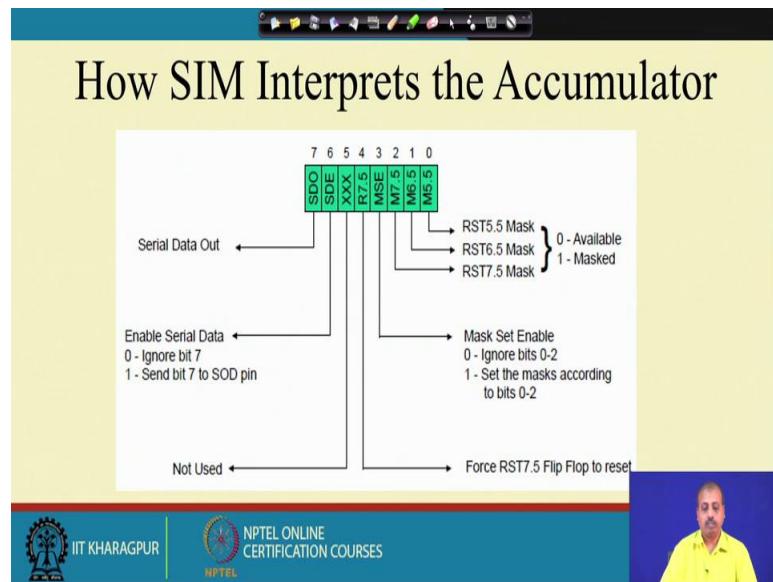
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the question is how can we manipulate the masks. So, this mask bits so, they are user programmable. So, as a so you can decide like which interrupts are useful for you and which interrupts you do not want to disturb the program. So, that way you can put this mask at different points. So, this interrupt enable flip flop is manipulated by this EI and DI instructions. So, if you set if you execute an EI instruction then the flip flop is set to 1, if you set or if you execute a DI instruction then that flip flop interrupt enable flip flop will be set to 0. So, that is one level.

Now, for this individual masking of RST 5.5, 6.5 and 7.5 so, there is a special instruction called SIM. So, it is known as Set Interrupt Mask or SIM instruction this instruction it will take a bit pattern in the accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts. So, this SIM instruction actually its a multipurpose instruction. So, it will be reading the accumulator and accordingly it will set some bits and the some of the bits are used for masking out this interrupt masks, the interrupt lines and some bits are used for serial data input output.

So, we will come to the serial data input output part later, but the part that is relevant for us now, is that some of the bits that they will be determining the masking pattern of the interrupts.

(Refer Slide Time: 10:07)



So, this is the situation. So, we have got this, this is how this SIM instruction will understand the meaning of the accumulator. So, accumulator is 8 bit, out of that this bit number 7 and 6 so, they are actually for serial transfer of data. So, we will not discuss about them now. Then this bit number 5 is not used, then rest of the bits so, this RST 5.5, 6.5 and 7.5 masks so, bit 0, 1 and 2 so, if you want to set the mask then you have to put a 1, if you want to make them available then you have to put a 0 there.

And, then this MSE is the mask set enable. So, whether you really want to set the mask or not. Actually the problem is that there are two things that are done by the same instruction one is the serial data input output and another is the setting of this masks. Now, in for serial data input output so, we will we may not be interested to change the interrupt setting for the current program. So, in those cases what we can do? We can just make this masks set enable bit 0. So, that whatever be the values of this M 5.5, 6.5, and 7.5 they will simply be ignored and the interrupt masking pattern will not change.

However, when you are using this SIM instruction for setting the interrupt mask so, you should set this MSE bit to 1, so that this pattern in set in M 5.5, 6.5 and 7.5 so, they are used for actually setting the masks for the interrupts. And, then we have got this R 7.5 bit that is bit number 4 it forces the RST 7.5 flip flop to reset. So, basically you may do this thing like without servicing the interrupts you just want to reset the interrupt flip flop. So, that can be done by setting this RST 7.5 R 7.5 bit to 1, ok.

So, this is you have to set a particular bit pattern in the accumulator and then execute the SIM instruction, so that the interrupt masking will be done.

(Refer Slide Time: 12:27)

SIM and the Interrupt Mask

- Bit 0 is the **mask** for RST 5.5, bit 1 is the **mask** for RST 6.5 and bit 2 is the **mask** for RST 7.5.
 - If the mask bit is 0, the interrupt is **available**.
 - If the mask bit is 1, the interrupt is **masked**.
- Bit 3 (Mask Set Enable - MSE) is an **enable for setting the mask**.
 - If it is set to 0 the mask is **ignored** and the old settings remain.
 - If it is set to 1, the new setting are **applied**.
 - The SIM instruction is used for multiple purposes and not only for setting interrupt masks.
 - It is also used to control functionality such as Serial Data Transmission.
 - Therefore, bit 3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



Now, so, as it is said that bit 0 is the mask for the RST 5.5, 1 is for the mask 6.5 and 2 is for the mask is the mask for RST 7.5. If the mask bit is 0 interrupt is available, the mask bit is 1 the interrupt is masked. Bit 3 mask set enable is an enable for setting the mask. If it is set to 0, the mask is ignored and the old settings remain that is so, if the particularly useful for this serial data transfer type of cases and if it is set to 1, the new setting will be applied.

The SIM instruction is used for multiple purposes and not only for setting interrupt mask, it is also used to control functionality for serial data transmission. That's why the bit 3 is necessary. So, as we have discussed previously that we want to do a serial transmission without affecting the interrupt flag. So, that is done by setting this bit 3.

(Refer Slide Time: 13:23)

SIM and the Interrupt Mask

- The RST 7.5 interrupt is the **only** 8085 interrupt that has **memory**.
 - If a signal on RST7.5 arrives while it is masked, a flip flop will remember the signal.
 - When RST7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal.
 - This flip flop will be **automatically reset** when the microprocessor responds to an RST 7.5 interrupt.
- Bit 4 of the accumulator in the SIM instruction allows **explicitly resetting** the RST 7.5 memory even if the microprocessor did not respond to it.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, this RST 7.5 interrupt is the only 8085 interrupt that has a memory. So, as we said that there was an R 7.5 flip flop that is available inside the processor. So, even if the some interrupt had occurred and you did not respond to it so this bit, will maybe the 7.5 was masked off, so that that is why it was not sensed, but this particular flip flop remain set. So, you may want to reset the flip flop. So, if a signal on RST 7.5 arrives while it is masked, a flip flop will remember thus there is a signal that something had occurred.

When RST 7.5 is unmasked the processor will be interrupted even if the device has removed the interrupt signal. So, actually this is a memory so it remembers that something some interrupt had occurred and maybe that the device that interrupted so, it has it has taken off the interrupt line, ok. So, it is not giving the interrupt anymore, but still since this flip flop is set. So, when the processor unmasks the 7.5 interrupt then this flip flop will be sending an interrupt to the processor, telling that something telling that in the previously something had happened on that particular line. So, this flip flop will automatically reset when the microprocessor responds to a RST 7.5 interrupt. So, on getting or servicing RST 7.5 this interrupt is cleared.

However, in some cases we may not be interested to service this RST 7.5 anymore because the device that had requested for the interrupt has already removed the request, maybe it is no more interested in the interrupt service routine. So, what happens is that in that case. So, we can use again the SIM instruction to reset this RST 7.5 memory. So, microprocessor

did not respond to the interrupt service routine, but we just revoke this interrupt pin or this internet flip flop by resetting this bit explicitly.

(Refer Slide Time: 15:32)

SIM and the Interrupt Mask

- The SIM instruction can also be used to perform serial data transmission out of the 8085's SOD pin.
 - One bit at a time can be sent out serially over the SOD pin.
- Bit 6 is used to tell the microprocessor whether or not to perform serial data transmission
 - If 0, then do not perform serial data transmission
 - If 1, then do.
- The value to be sent out on SOD has to be placed in bit 7 of the accumulator.
- Bit 5 is not used by the SIM instruction

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



The SIM instruction can also be used to perform serial data transmission over the 8085 serial output data pin. So, one bit at a time can be sent serially over the SOD pin and bit 6 is used to tell the microprocessor whether or to or not to perform the serial data transmission that is that SDE serial data enable so, that tells whether it should do the transmission or not.

So, if 0 then do not perform serial data transmission and if 1, then do the transmission. The value to be sent on the sent out on the serial output data has to be placed in bit 7 of the accumulator. So, whatever bit you want to transmit. So, that should be placed on the bit 7, so that when you execute the SIM instruction it comes to the serial data output of that of the register and then from there it is outputted to the SOD pin. And, bit 5 is not used by the SIM instruction. So, it is not a meaningful for the SIM.

(Refer Slide Time: 16:33)

Using SIM Instruction to Modify Interrupt Masks

Example: Set the interrupt masks so that RST5.5 is enabled, RST6.5 is masked, and RST7.5 is enabled.

- First, determine the contents of the accumulator

- Enable 5.5	bit 0 = 0	SDO
- Disable 6.5	bit 1 = 1	SDE
- Enable 7.5	bit 2 = 0	XXX
- Allow setting the masks	bit 3 = 1	R7.5
- Don't reset the flip flop	bit 4 = 0	MSE
- Bit 5 is not used	bit 5 = 0	M7.5
- Don't use serial data	bit 6 = 0	M6.5
- Serial data is ignored	bit 7 = 0	M5.5

Contents of accumulator are: 0AH
0 0 0 0 1 0 1 0

EI ; Enable interrupts including INTR
MVI A, 0A ; Prepare the mask to enable RST 7.5, and 5.5, disable 6.5
SIM ; Apply the settings RST masks

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us look into some example to set the interrupt masks so that RST 5.5 is enabled, 6.5 is masked and RST 7.5 is also enabled. So, if we want to do that first we have to determine what should be the content of the accumulator. So, first enable 5.5; enable 5.5 means I have to put this bit 0 to 0. So, it is not masked, then disable 6.5 so, it is masked, so I have to set this mask to 1. So, bit 1 should be equal to 1 and then RST 7.5 be enabled. So, this bit 2 is equal to 0.

Now, bit 3, so, since we are looking for setting this particular mask pattern for the interrupt, so this MSE bit should be equal to 1. So, bit 3 is equal to 1. Bit 4 was for R7.5, so, that is not reset. So, we do not use this RST 7.5 reset, so do not reset the flip flop. Bit 5 is not used. So, the bit 6 is for serial data output. So, we are not interested in serial data output now. So, this SDE serial data enable is made 0 and SDO can be any value so, this is just put a 0 there.

So, the instruction sequence for setting this mask and interrupting format is like this first we execute the EI instruction to enable the interrupt in interrupt enable flip flop and then MVI A, 0A hex. So, if you look into this pattern so, this is 0. And this is for next four bits constitute the hexadecimal number A, so, this is 0A. So, MVI A, 0A hex so, that will put this particular bit pattern onto the A register and then the SIM instruction is executed. So, this bits are copied and accordingly the corresponding the masking pattern is set for the

interrupts and this serial data that is disabled. So, this way we can use this SIM instruction for setting the interrupt masks.

(Refer Slide Time: 18:48)

Triggering Levels

RST 7.5 is positive edge sensitive.

- When a positive edge appears on the RST7.5 line, a logic 1 is stored in the flip-flop as a “pending” interrupt.
- Since the value has been stored in the flip flop, the line does not have to be high when the microprocessor checks for the interrupt to be recognized.
- The line must go to zero and back to one before a new interrupt is recognized.

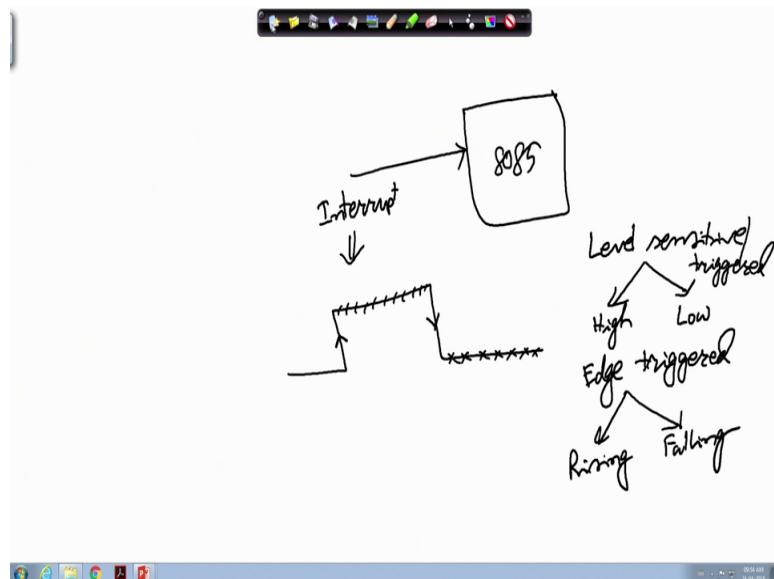
RST 6.5 and RST 5.5 are level sensitive.

- The interrupting signal must remain present until the microprocessor checks for interrupts.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another important thing that we have is about that triggering. So, this RST 7.5 is a positive edge sensitive.

(Refer Slide Time: 19:08)



So, triggering actually if we so, it is like this that so, this is the microprocessor 8085, now if this is some interrupt line that is coming it can be INTR or any of those interrupt. So, if an interrupt is coming so, that I can have several types of situation; like this interrupt signal

that is maybe it is coming as a pulse, ok. Now this interrupt may be sensed interrupt may be accepted by the processor if it is taken high. So, it is called level sensitive, it is level sensitive. So, level sensitive can also be of two types, level high and level low.

So, if we say that 8085 will sense this interrupt if the line is at high level. So, that is high level sensitive interrupt. On the other hand so, if we say that this will be sensed when the signal value is low, ok, so, that is the low level sensitive interrupt. So, basically I said that it will check for the interrupt on the last, but one clock cycle of every instruction execution; so, at that time if for a high level sensitive interrupt so, if it finds that the interrupt line is high then it will be sensed it will be taking that an interrupt has occurred. On the other hand a low level sensitive interrupt means that if it, if it finds that the signal value is low, then it will be getting that low level sensitive it will be getting the interrupt.

Now, apart from that another type of classification of interrupt can be the edge sensitive interrupts or edge triggered interrupt. So, this is called level sensitive or level triggered and we can also have this edge triggering, just like in flip flops we have got level triggered and edge triggered so, here also the similar concept is there. So, it will expect that on the interrupt line so, there is a Rising edge like this so, that will be taken as edge triggering that is that is Rising edge this is the Rising or Falling. So, this is Rising edge triggering and other possibility is it will get an interrupt on this falling edge. So, this is falling edge triggering.

So, using this technique using this or these are the possible alternatives that we can have. So, different processors will have different types of interrupting mechanisms. So, will see what happens in 8085 like; so, what are the interrupt mechanism that we have in 8085. So, out of the different, different interrupt mechanism things that we have so, this RST 7.5 is a positive edge sensitive. So, it will expect that when the positive edge occurs that is going that the line goes from low to high then a logic 1 is stored in the flip flop as a pending interrupt. So, out of this RST 7.5, 6.5 and 5.5, 7.5 has got a flip flop in it. So, that flip flop will be sensing the value. So, this is actually keeping the edge within it.

So, then this is the value has been stored in the flip flop the line does not have to be high when the microprocessor checks for the interrupt to be recognized. So, that is the beauty. So, if your device cannot keep the signal high for a long time. So, we should connect it

through RST 7.5 pin because we know that there a positive edge will be sufficient for getting the interrupt administered or sensed by the processor, because it will be setting that R 7.5 interrupt line interrupt flip flop and then it will be the sensed later by the processor when you complete the current instruction.

Now, the line must go to 0 and back to one before a new interrupt is recognized. So, this is another point. So, since this is all this is a positive edge sensitive. So, whenever the positive edge comes then only it is sensed. So, if it remains high for quite some time it will not be sensed as a new interrupt, because that that positive edge is required. On the other hand this 6.5 and 5.5 they are level sensitive interrupts and as their level sensitive the interrupting signal must remain present until the microprocessor checks for the interrupt. So, previously we have seen that the minimum duration for which the signal should be kept high is 17.5 clock cycles. So, this RST 6.5 and 5.5 these signals they should be kept high for that 17.5 clock cycles at least, for getting it sensed by the processor.

(Refer Slide Time: 24:02)

Determining the Current Mask Settings

- RIM instruction: Read Interrupt Mask
 - Load the **accumulator** with an 8-bit pattern showing the status of each interrupt pin and mask.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

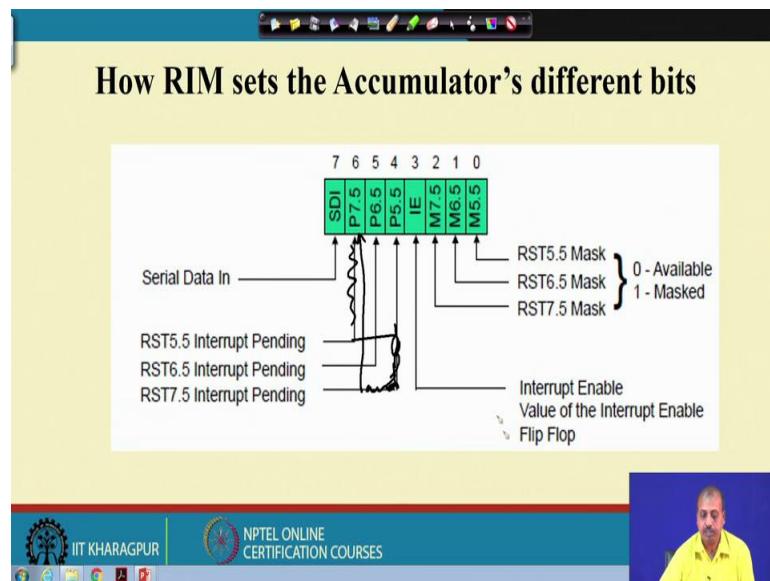
152

So, you can also determine the current mask settings like a using the RIM or RIM instruction which is Read Interrupt Mask, ok. So, what happens is this in this case is that the accumulator will be loaded with the 8-bit pattern for different positions. So, this RST 7 from the RST 7.5 memory so, it will come to bit 6 which we call present P7.5 that is the present value of 7.5. Then, this M7.5 mask bit so, this is copied into P6.5 bit number 5 of the accumulator, then this P M 6.5 will be copied into M5.5 sorry M6.5 is copied onto this

M6.5 bit, then this M5.5 is copied into this M5.5 bit. And, then we have got this P6.5. So, this is the pin value, this is not this is not present this is the pin value. So, pin value is copied into this P bits and the mask settings are copied onto the M bits, ok. So, this pin values are copied onto P bits and mask values are copied onto M bits. Similarly, the interrupt enable flip flop is copied onto this IE bit, ok.

So, in this way if you execute a RIM instruction you get the status of this mask bits this R 7.5 memory flip flop also the pin bits also the pins, because there is no point getting the pin of RST 7.5 because it is controlled by this flip flop. So, that is why it is not kept separately, but for 6.5 and 5.5 the pin values are copied here and stored here. So, this RIM or RIM instruction so, this will be loading the 8-bit pattern showing the status of each interrupt pin and this mask.

(Refer Slide Time: 26:08)



So, we can use this instruction for different purpose so, RIM instruction the other utility is to read the serial data. So, serial data inputs so, that is the other purpose and the other bits bit 6 is for RST 5.5 interrupt pending. So, if there is a pending interrupt from the pin so, it will be coming. Then, RST 6.5 is the this P6.5 is the RST 6.5 interrupt pending bit and RST this 5.5 actually this diagram it is slightly wrong. So, this is this is not this one. So, this is this and this is actually this one ok.

So, we have got this pending bits stored in P7.5, 6.5, 5.5 and we have got the mask bits M5.5, 6.5, 7.5 in this in this side and this interrupt enable value interrupt enable flip flop

value is kept in bit number 3, ok. So, this way this RIM instruction it will set the accumulator to different bit values. So, we can use this SIM and RIM instructions to set this interrupt patterns in a, to a particular value.

(Refer Slide Time: 27:38)

The RIM Instruction and the Masks

- Bits 0-2 show the current **setting of the mask** for each of RST 7.5, RST 6.5 and RST 5.5
 - They return the contents of the three mask flip flops.
 - They can be used by a program to read the mask settings in order to modify only the right mask.
- Bit 3 shows whether the maskable interrupt process is **enabled or not**.
 - It returns the contents of the Interrupt Enable Flip Flop.
 - It can be used by a program to determine whether or not interrupts are enabled.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



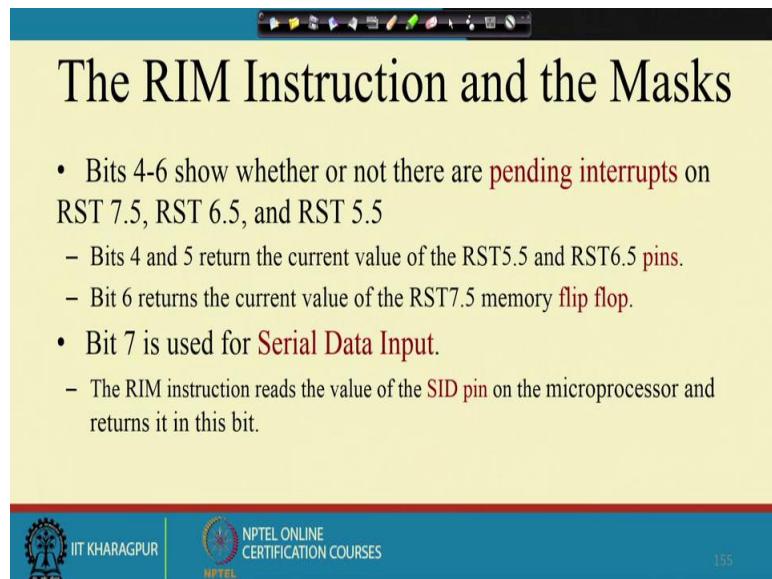
Now, so, this RIM instruction bits 0 to 0 to 2; 0, 1 and 2 they show the current setting of the mask for each RST 7.5, 6.5 and 5.5 pin. The return the contents of the three mask flip flops and they can be used by a program to read masks settings in order to modify only one only the right mask. So, if you want to modify it so, you can do that.

Bit 3 it shows whether the maskable interrupt process is enabled or not and it returns the contents of the interrupt enabled flip flop, ok. It can be used by program to determine whether or not the interrupts are enabled. So, to determine that so basically this RIM instruction is for getting the status of the interrupt occurrence in the processor, whether they are enabled, whether this masks are set and what are if something is pending and all any interrupt is pending. So, it is getting those information.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 60
8085 Microprocessor (Contd.)

(Refer Slide Time: 00:19)



The RIM Instruction and the Masks

- Bits 4-6 show whether or not there are pending interrupts on RST 7.5, RST 6.5, and RST 5.5
 - Bits 4 and 5 return the current value of the RST5.5 and RST6.5 pins.
 - Bit 6 returns the current value of the RST7.5 memory flip flop.
- Bit 7 is used for Serial Data Input.
 - The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES
NPTEL

155

This for the RIM instruction; so, other bits like bits 4 to 6 show whether or not there are pending interrupts on RST 7.5, 6.5 and 5.5 line. So, bits 4 and 5, they return the current value of RST 5.5 and 6.5 pins, bit 6 returns the current value of RST 7.5 memory flip flop. So, you note here that for bits RST 7.5, we do not read the pin value because RST 7.5 is an edge triggered flip flop. So, if it may be that that edge is already done; so, if you read the pin so it does not have any meaning and the pin that the edge is already registered in the R 7.5 memory bit RST 7.5 memory bit. So, that is already there. So, it is better to copy it from there.

So, this bits 4 to 6. So, they are holding the pending interrupt status and bit seven is for the serial data input. So, the RIM instruction it reads the value of the SID pin on the microprocessor and returns it in this particular bit. So, after executing the RIM instruction the SID pin whatever be the value, so, that will come to the most significant bit of the accumulator and then accumulator. So, based on that; so, we can we can read, we can read the value through the SI of the SID pin into the accumulator bit 7.

(Refer Slide Time: 01:40)

Pending Interrupts

- Since the 8085 has five interrupt lines, interrupts may occur during an ISR and remain pending.
 - Using the **RIM** instruction, the programmer can read the status of the interrupt lines and find if there are any pending interrupts.
- The advantage is being able to find about interrupts on RST 7.5, RST 6.5, and RST 5.5 without having to enable low level interrupts like INTR.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Now the pending interrupts that we are talking about see 8085 has got 5 interrupt lines and interrupts may occur during an ISR and remain pending. So, this can happen. So, using the RIM instruction the programmer can read the status of the interrupt lines and find if there are any pending interrupts. So, this is the utility of this of this RIM instruction. The advantage is being able to find about interrupts on RST 7.5, 6.5 and 5.5 without having to enable low level interrupt like INTR.

So, many a time, what happens is that maybe in my system RST 7.5 and 6.5, they are very important devices and 5.5 INTR, the devices connected to those lines are not that much important. So, maybe I was servicing RST 7.5 and in between some 6.5 had occurred, but 6.5 having lower priority than 7.5. So, it was not accepted by the processor.

So, this pending bit so at the end of the 7.5 ISR so if I check the pending bit, I will find that RST the 6.5 interrupt had come. So, I can directly branch to the 6.5 interrupt service routine from there and otherwise, what I have to do is I have to enable all the interrupts again and that may create some chaos because now many other interrupts may pending on 5.5 and INTR lines and they will also come into picture ok. So, that has to be avoided.

(Refer Slide Time: 03:15)

Using RIM and SIM to set Individual Masks

Example: Set the mask to enable RST6.5 without modifying the masks for RST5.5 and RST7.5.

- In order to do this correctly, we need to use the RIM instruction to find the current settings of the RST5.5 and RST7.5 masks.
- Then we can use the SIM instruction to set the masks using this information.
- Given that both RIM and SIM use the Accumulator, we can use some logical operations to mask the un-needed values returned by RIM and turn them into the values needed by SIM.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

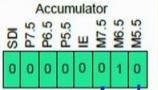
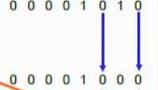
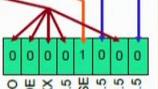
So, for that purpose, this RIM instruction is useful. So, here is an example set the mask to enable RST 6.5 without modifying the masks for RST 5.5 and 7.5. So, what you want is the current setting for 5.5 and 7.5, they should continue only the 6.5 setting should be 6.5 interrupts should be enabled.

So, first of all we have to do a RIM instruction to find the current setting of 5.5 and 7.5 masks, then we can use the SIM instruction to set the masks using this information. So, this RIM and SIM instructions, since they use the accumulator, we can use some logical operations to mask the un-needed values ok, the un-needed values returned by the RIM and SIM instruction and by this RIM instruction and then turn them into the values needed by the SIM instructions.

(Refer Slide Time: 04:10)

Using RIM and SIM to set Individual Masks

– Assume the RST5.5 and RST7.5 are enabled and the interrupt process is disabled.

RIM	; Read the current settings.	
ORI 08H	; 0 0 0 0 1 0 0 0 ; Set bit 4 for MSE.	
ANI 0DH	; 0 0 0 0 1 1 0 1 ; Turn off Serial Data, Don't reset ; RST7.5 flip flop, and set the mask ; for RST6.5 off. Don't cares are ; assumed to be 0.	
SIM	; Apply the settings.	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, here is the example. So, you see we assume that RST 5.5 and 7.5 are enabled and the interrupt process is disabled. So, as a result since that this IE is 0 and 7.5 and 5.5, they are not masked and this, this is 6.5 is masked. So, what we do is the first we execute a RIM instruction that gives me the current setting which is like this and then we want to we want to set the interrupt mask in some fashion. So, for that we have to set this we have to prepare for this SIM instruction. In the SIM instruction bit number 4 is the MSE Mask Set Enable. So, for whatever value I have got so with that; so, if I do an ORing with this particular pattern ok. So, then this bit becomes equal to 1.

So, the, with this, so, if you can do this ORing. So, you get the bit number 4 set whatever be the value that you read here. So, if you do or with this 0 8 hex. So, bit number 4 is set.

Now, what we want is that the serial data should be turned off. So, serial data turn off. So, so this SDE value should be equal to 0 SDO, it is, can be 0 or 1, it does not matter; so, it is made 0. Then RST 7.5 flip flop, so, that is do not we do not want to reset the RST 7.5 flip flop. So, this bit is set to 0 and this RST 6.5 is turned off. So, the mask and set the mask for RST 6.5 turn off. So, 6.5 turn off is this one M 6.5. So, that is made 0 and don't cares for the, and this and the mask for don't reset the mask for reset for RST 7.5 and set the mask for 6.5 off. So, mask for 6.5 has been set off and don't cares are assumed to be 0.

So, rest of the things they will taken as 0. So, this way we can get we can apply this particular setting. So, when I do an AND immediate with this. So, I will get this particular bit pattern and this bit pattern when it is put into a SIM instruction, so, it will do the desired thing that we want. So, this way, we can use this RIM and SIM instruction along with a few logical operations to set this interrupt line interrupt enable and for selecting some interrupt pins and getting them masked out or enabled so like that.

(Refer Slide Time: 06:50)

TRAP ⇒ RST 4.5 $49 \times 8 = 36$

- TRAP is the only **non-maskable** interrupt.
 - It does not need to be enabled because it **cannot be disabled**.
- It **has the highest priority** amongst interrupts.
- It is **edge and level sensitive**.
 - It needs to be high and stay high to be recognized.
 - Once it is recognized, it won't be recognized again until it goes low, then high again.
- TRAP is usually used for power failure and emergency shutoff.

Another very important interrupt that we have not discussed so far is the trap. So, this is the only non maskable interrupt that 8085 has; so, all other interrupts they are maskable, but trap is a non maskable interrupt. So, it does not have any effect the EI and DI instructions, they will not have any effect on the trap line. So, it does not need to be enabled because it cannot be disabled. It has the highest priority amongst all the interrupts and in fact, so, this trap is also known as RST 4.5. So, this trap is also the RST 4.5. So, when this trap occurs then the processor will jump to $4.5 \times 8 = 36$ in decimal; so, this will be jumping to location 36 so that is between this RST 4 and 5 so that is why it is called RST 4.5 and the corresponding address is 36 decimal.

So, it has the highest priority among all the interrupts, it is edge and level sensitive. So, it is a combination of this edge sensitive and level sensitive. So, it is edge and level sensitive it needs to be high and stay high to be recognized. So, since it is a very important interrupt. So, this is often used for power failure or some emergency condition now if there is a glitch

that is coming on the power supply line and it is sensed as an interrupt. So, that is that may be undesirable.

So, what is required in case of trap to be sensed is that it should be there should be an edge as well as it should there should be it should be high for quite some time for it to be sensed. So, it is edge and level sensitive, it needs to be high and stay high to be recognized and once it is recognized, it will not be recognized again until it goes low because it because of the nature that it is it is edge and level sensitive. So, that to bring that edge sensitivity into picture, so, once it is sensed it must go low and then only it can come high again.

(Refer Slide Time: 09:09)

Internal Interrupt Priority

- Internally, the 8085 implements an **interrupt priority scheme**.
 - The interrupts are ordered as follows:
 - TRAP
 - RST 7.5
 - RST 6.5
 - RST 5.5
 - INTR
 - However, TRAP has lower priority than the HLD signal used for DMA.

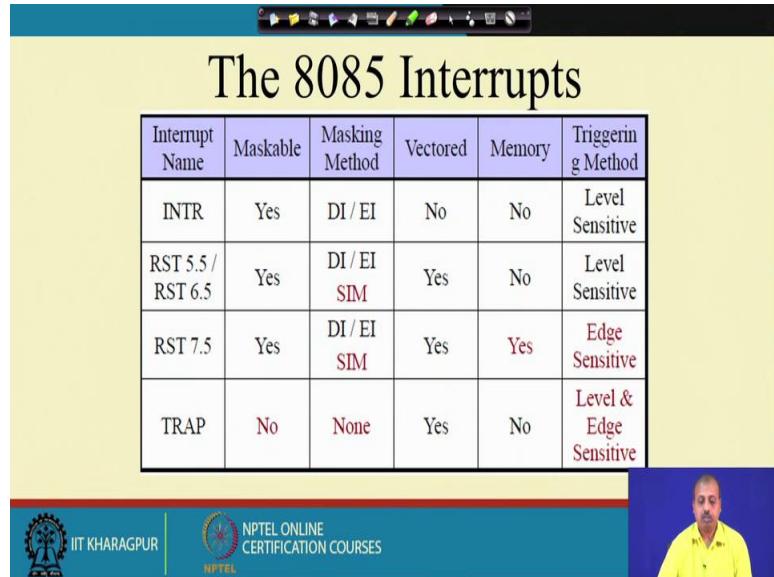
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, sorry; so, next we will be looking into the internal priority, internal priorities that we have. So internally 8085 implements an interrupt priority scheme and this interrupts are ordered like this the trap has the highest priority followed by 7.5, 6.5, 5.5 and INTR. However, trap has lower priority than the hold signal that is used for the DMA. So, the DMA will come later. So, this is for Direct Memory Access so for doing some operation data transfer directly between memory and the secondary storage without involving the microprocessor.

So, for that type of operation we need this direct memory access or DMA and this is this hold and hold acknowledge. So, these are the two pins associated with that operation. So, they are. So, when the hold signal comes then what the processor does is that it, it releases all the buses; so, it just as if it will not use any external bus. So, it will stop all the operations

and the bus is given to some other master. So, that these data transfer to memory can take place directly about this trap has got lower priority than the hold signal.

(Refer Slide Time: 10:21)



The slide title is "The 8085 Interrupts". Below the title is a table with the following data:

Interrupt Name	Maskable	Masking Method	Vectored	Memory	Triggering Method
INTR	Yes	DI / EI	No	No	Level Sensitive
RST 5.5 / RST 6.5	Yes	DI / EI SIM	Yes	No	Level Sensitive
RST 7.5	Yes	DI / EI SIM	Yes	Yes	Edge Sensitive
TRAP	No	None	Yes	No	Level & Edge Sensitive

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a person speaking.

So, to summarize; so, if you look into different interrupts the maskable; so, this interrupt INTR RST 5.5, 6.5, 7.5, they are all maskable. So, where trap is non maskable. Masking method INTR is DI EI RST 7.5, 5.5 and 6.5, 7.5. So, DI EI and SIM, trap cannot be masked so there is nothing. So, this INTR is not vectored, but remaining interrupts, they are vectored interrupts because for them, we know what is the address at which the execution of the interrupt service routine should be located.

And so, whether any of this interrupts have got memory yes RST 7.5 have got a memory others do not have. So, others they require that it should be high for that 17.5 T states for getting it sensed in the worst case. Triggering method so we have got INTR level sensitive 5.5, 6.5, they are also level sensitive 7.5 is edge sensitive and trap is level and edge sensitive. So, the edge should be there followed by it should be high for quite some time. So, this way this 8085 interrupts can summarize their different features.

(Refer Slide Time: 11:42)

Direct Memory Access

This is a process where data is transferred between two peripherals directly without the involvement of the microprocessor.

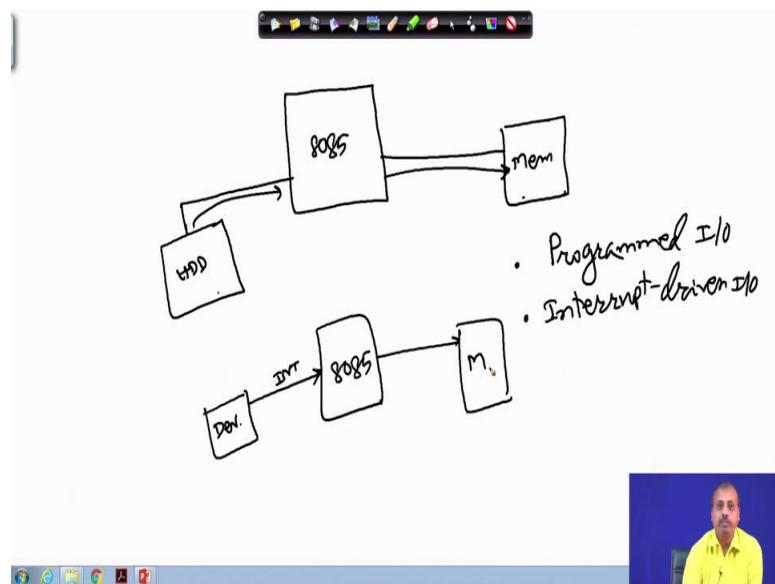
- This process employs the HOLD pin on the microprocessor
 - The external DMA controller sends a signal on the HOLD pin to the microprocessor.
 - The microprocessor completes the current operation and sends a signal on HLDA and stops using the buses.
 - Once the DMA controller is done, it turns off the HOLD signal and the microprocessor takes back control of the buses.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Next, we will be looking into another important concept which is known as direct memory access. So, this direct memory access is a process where data is transferred between 2 peripherals directly without involvement of the microprocessor.

(Refer Slide Time: 12:03)



So, it is like this that if we look into this data transfer process between any processor. So, suppose this is our 8085 processor and this is the memory ok. So, you have got your address data bus line connection and this 8085 is connected to some external disc external device which may be a hard disc for example.

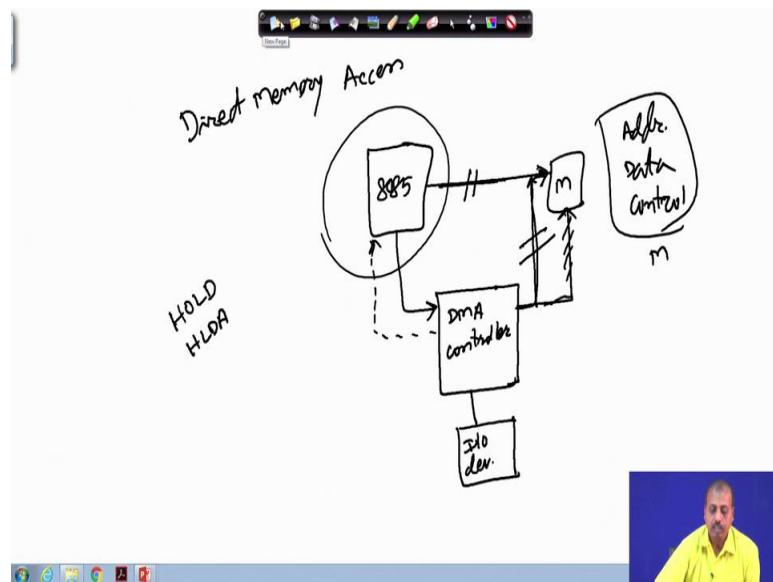
Now, if it is required that some data has to be transferred from this hard disc to memory then what is required is that 8085, if I do it through this 8085 processor, then what it will do it will read values from this hard disc and then write something on to the memory write the value on to the memory, then again, it will read the next byte from the hard disc and it will write the byte on to the memory so it will go like this. So, this is known as the programmed IO, programmed input output operation.

Difficulty with this programmed input output operation is that this the device the IO device that we are talking about. So, they are generally much slower compared to this processor like 8085. So, that a good amount of time 8085 just waits for the for the device to be ready to transfer the, to give the next byte next byte of data and so that 8085 can read it and transfer. So, that is the problem with the programmed IO.

There is another type of in data transfer which is known as interrupt driven IO. So, interrupt driven IO. So, it is like this that this processor. So, this processor does not tell the device whether you are ready with the data the device rather it gives an interrupt to the processor ok. So, it gives an interrupt to the processor telling that I am ready with the data. So, the 8085 goes into the ISR. So, in the ISR 1 byte of data or some amount of data whatever data this device is ready with are read and they are put into the memory. So, the point is that now this processor otherwise it is not waiting for the device to be ready. So, whenever the device is ready so it will tell the processor by an interrupt and that interrupt will be in the interrupt service routine will be actually the data transfer job that is done.

So, again; so, this is an improvement over the programmed IO, but it is still not that much fast.

(Refer Slide Time: 14:46)



So, what is done and the third possibility is known as the Direct Memory Access or DMA. So, direct memory access. So, we have got the 8085 processor. So, we have got the memory. So, this memory is connected to the 8085 processor plus there is another device which is known as the DMA controller or direct memory access controller.

So, what this direct memory access controller does is when 8085 tells this direct memory access controller and that I want to get some data transferred. So, this I O device; so, this is connected to the DMA controller. So, this I O device is connected to the DMA controller. So, this 8085 tells the DMA controller that I want some data to be transferred from the IO device to the memory or vice versa from memory to the IO device. So, this DMA controller will do that job. So, it will do transfer the content to the memory ok. So, after; so, this processor is free. So, processor can do anything else it wants to do. So, it is if it definitely it cannot use the memory, but it can do other computations that it within the processor and all.

So, those type of cases operations it can do and the DMA controller will do the transfer and when it is done with the transfer, then it will again tell the processor that I am done ok, I am done with the operation and then this 8; 8085 will restart will continue with the previous operation that.

Now, the point that is to be noted is that for reading or writing something to and from memory. So, what we need is that the memory's address bus and data bus and control bus. So, they are to be given proper values to the memory.

Now, in this situation what is happening is this is memory has got two drivers, one of this address data control buses are coming from this 8085 and in another situation it is coming from the DMA controller. So, actually though I have shown it like this, but in reality the connection is like this fine.

So, we have got two drivers for the same bus; so, to solve the problem. So, what is done is that when 8085 tells the DMA controller that it will lead, want some data transfer through DMA mode. So, it will release the control here ok, it will release the control of this bus and now this DMA controller will control this bus and after sometime when the transfer is over when it is informed to the 8085 that it is over then this is this control is taken off and this control is re established.

So, that way this whole DMA operation works and you remember that there were two pins hold and hold acknowledge, 2 pins of 8085. So, they were actually responsible for this DMA type of operation. So, we will see how this hold, hold acknowledge pins, they are going to be useful for this transfer.

(Refer Slide Time: 18:02)

The screenshot shows a presentation slide with a yellow background. At the top, the title 'Direct Memory Access' is displayed in a large, bold, black font. Below the title, a paragraph of text explains that DMA is a process where data is transferred between two peripherals directly without the involvement of the microprocessor. A bulleted list follows, detailing the process: it starts with the external DMA controller sending a signal on the HOLD pin to the microprocessor, which then completes its current operation and sends a signal on HLDA, stopping bus usage. Once the DMA controller is finished, it turns off the HOLD signal, and the microprocessor resumes control of the buses. At the bottom of the slide, there is a footer bar containing the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

So, this direct memory access is a process where data is transferred between 2 peripherals directly without the involvement of the microprocessor. So, processor is not involved in the in this transfer. The process employs the hold pin of the microprocessor. The external DMA controller sends a signal to on the hold pin, so, to the microprocessor that you please release the buses.

The microprocessor completes the current operation and sends a signal on the hold acknowledge and stops using the buses. So, it releases the buses that is what that is what I was saying and once the DMA controller is done so it turns off the hold signal and the microprocessor takes back the control of the buses. So, this is how this whole operation take place by means of this hold signal.

(Refer Slide Time: 18:55)

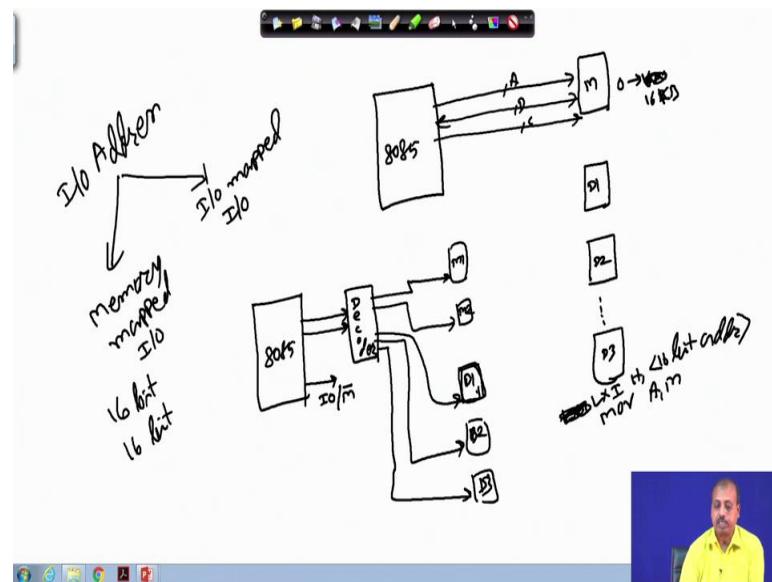
The slide has a yellow background and a blue header bar with various icons. The title 'Basic Concepts in Serial I/O' is centered at the top. Below the title is a bulleted list of requirements:

- Interfacing requirements:
 - Identify the device through a port number.
 - Memory-mapped.
 - Peripheral-mapped.
 - Enable the device using the Read and Write control signals.
 - Read for an input device.
 - Write for an output device.
 - Only one data line is used to transfer the information instead of the entire data bus.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '163'.

So, next we will be looking into the serial input output operation. Now this serial input output operation means that I want to transfer data serially, but before going to that. So, we want to discuss about the parallel IO transfer a bit. So, what happens is that ok. So, this as I said that the programmed IO or interrupt driven IO whatever you call it. So, ultimately the device have the processor has to read from the device or it has to write on to the device. Now, if I have got say if this is the 8085 processor.

(Refer Slide Time: 19:37)



And in this 8085 processor so I have got some memory and this memory it ranges over the address is say 0 to say 6K or say 16 kilobyte, 0 to 16 kilobyte. So, this range is this range by the memory by the memory. So, this address data buses are connected there by means of decoder and all. So, this is the address data and control bus connections. I am not showing the latches and latches and all demultiplexing the address data bus and all. So, I am not showing it separately assuming that all those are there.

Now, if there are a few devices connected to this processor. So, this is device 1, device 2, device 3 like that now these devices are to be also accessed now this for accessing a particular device. So, the processor needs to tell what is the corresponding address now while telling this address 8085 can do it in 2 different ways the IO address that it is talking about it can be classified into two categories one is called memory mapped address or memory mapped IO, another is called IO mapped IO. In case of memory mapped IO, so, what happens is that if this is the 8085 processor and these are the memory chips. So, suppose I have got 2 memory chips in my system; M1 and M2 and we have got a few devices D1, D2, D3, etcetera.

Now, this address decoder that I have the decoder the address decoder; so, it has got some of the bits of the address lines and it is generating the enable signals for the memory chips. So, some of these lines may be connected to the enable signal of the devices as well, fine. So, if I do this thing then what happens is that there is no difference between a memory

location and a device location. So, as far as the processor is concerned, so, it just needs to know what is the address of this device and that we what is; so, for a memory location the address is 16 bit. So, this device is also nothing, but a 16 bit address.

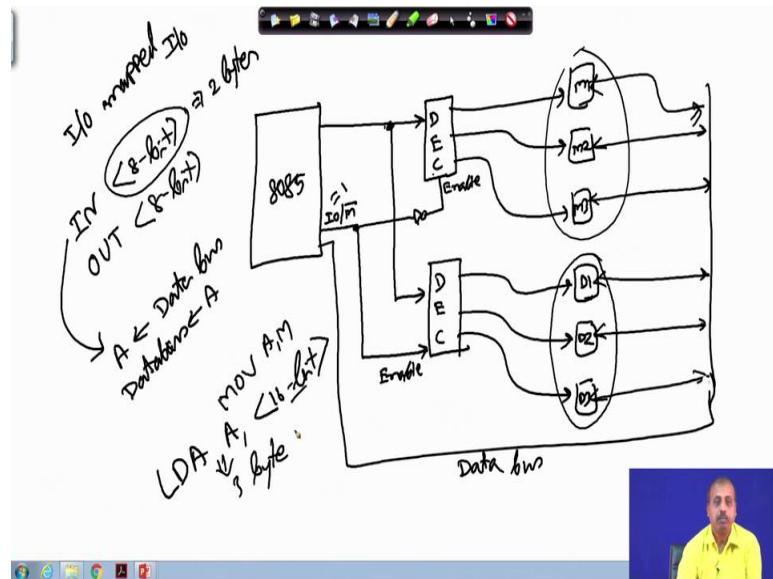
So, these are some special addresses that this memory should that that the processor should know, like if I have an instruction like LXI say if I have an instruction like MOV A, M and before that if I if I do 1 LXI H, some 16 bit address. Now, if this 16 bit address happens to be one location within this memory chips M1 and M2, then this MOV A, M instruction. So, it will read from the memory location and get it into the 8085.

On the other hand, if this 16 bit address is such that it is selecting this device then in the next MOV A, M instruction, so, it will read from the device and get it into the memory. Similarly, if this 16 bit address is actually such that this decoder enables this line D2, then this D2 will be selected as, that content will be selected and put into the 8085 accumulator.

So, this type of operations; so, they are known as memory mapped IO operation because I O devices, they are treated as nothing, but some extension of the memory locations ok. So, this type of IO operation so they are known as IO sorry memory mapped IO operation. So, the access is exactly like the memory addresses.

On the other hand, if you look into this 8085 processor; so, you know that there is another special pin which is known as IO/\bar{M} . So, whenever the processor is doing this IO or this memory operation, so, it is making this IO/\bar{M} line equal to 0 ok. So, that is so far we have ignored this line, but we can use it for a special purpose like we can make it like this.

(Refer Slide Time: 24:28)



So, say this is the 8085 processor and here I have got the decoder corresponding to the memory chips. So, this decoder is dedicated for this decoding of the memory like this enables this enables this. So, this enables this like that.

So, we can have another decoder which we can dedicate for the IO devices D1, D2, D3 like that. So, this D1 this devices so, they are enabled by this now the address line. So, they go here as well as the address line they come here, but the IO/ \bar{M} line that we have here. So, this IO/ \bar{M} line is connected to the enable line of this decoder and the inverted version of that is connected to the enable line of this decoder fine now the 8085 it so for IO operation; so, it has got another type of instruction which are known as in and out instruction fine. So, this in instruction is in and one 8 bit address and this out is out and 8 bit address.

So, in instruction what happens is that it reads from the data bus and the content comes to the accumulator register. So, whatever is available on the data bus comes to the accumulator register. Similarly the out instruction whatever be the content of the data accumulator. So, that is put on to the data bus line now this for this in instruction and out instruction this execution. So, this IO/ \bar{M} bit is equal to 1 where as for the instructions like MOV A, M where it is doing memory operation this IO/ \bar{M} line is made equal to 0.

So, this in out instruction; so, when they are executed this line is equal to 1 as a result these decoder is enabled and these decoder is disabled. So, it will not enable any of this memory

chips, but it will be depending upon the address that we have here. So, it will enable one of these devices and that device value, so, whatever be will be the value put on to the data bus. So, all of them are connecting to the data bus all of them are connecting to the data bus. So, like this and this is connected to the data bus of the processor. So, this is the data bus of the processor.

Now, if I do this if I do this then this is the data bus. So, for this in out type of operations; so, it will get the data from this devices onto this data bus and that will come to the 8085 chip where as for this memory operation so it will get it from there. So, the point is that this in out instructions. So, they are operated in the IO mode and they come under the heading called IO mapped IO, because the IO devices, so, they are having a separate decoder and they are having the IO addresses are mapped separately.

But only difference is that in case of IO mapped IO this in out instruction the address bus is 8 bit. So, it is not 16 bit. So, you can connect up to 256 devices only.

Now, but the advantage that we have is that this type of instructions. So, they have got only the length is only 2 bytes where as if you are having some LDA instruction LDA A, some 16 bit address which load some 16 bit value which load some memory content to the address is this 16 bit value into the a register. So, this instruction is a three byte instruction ok.

So, that way memory mapped IO the instruction sizes are larger for IO mapped IO the sizes are small, sizes are 1 byte less and as for the operation is concerned. So, it takes less number of clock cycles because it has to read only 8 bit. So, so it is taking only 2 machine cycles that is $4 + 3$, 7 cycles where as this memory operations. So, it is going to take some. So, this LDA instruction; so, this takes 3 plus 1; 4 machine cycles that is $12 + 9 + 4$ that is 13; 13 T states will be required for the LDA instruction. So, that will be there.

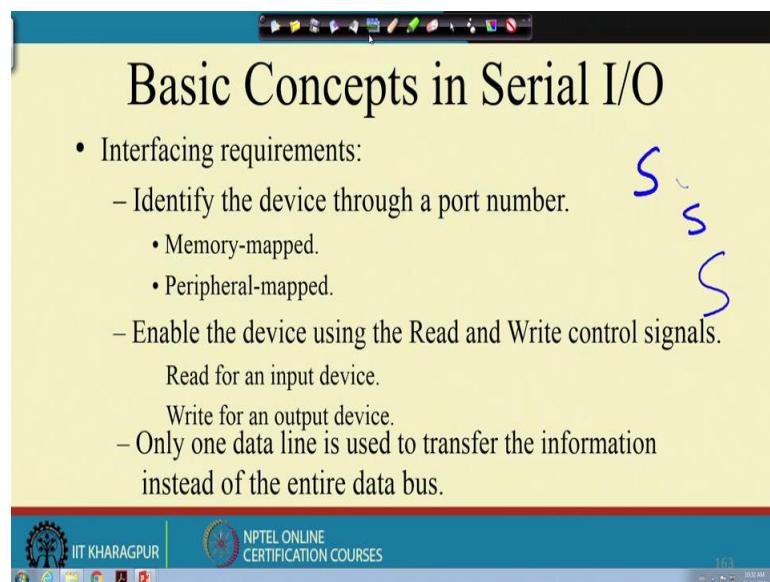
So, that is how this memory mapped IO and IO mapped, IO is there and in case of that it is the users choice of course, you need some extra hardware in case of IO mapped IO, but we can do that we can use it for doing the IO operations at a much lower cost as far as the execution time is concerned.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 61
8085 Microprocessor
(Contd.)

So, whenever we are trying to interface an IO device. So, these are the requirements.

(Refer Slide Time: 00:18)



Basic Concepts in Serial I/O

- Interfacing requirements:
 - Identify the device through a port number.
 - Memory-mapped.
 - Peripheral-mapped.
 - Enable the device using the Read and Write control signals.
 - Read for an input device.
 - Write for an output device.
 - Only one data line is used to transfer the information instead of the entire data bus.

So, we have to identify the device through a port number. So, device should have an address and that address has to be generated either in a memory mapped fashion or an IO mapped fashion. So, whatever be the strategy. So, we have to somehow select that particular device and enable the device using the read and write control signals.

So, if you are trying to make some input operation, then the read signal has to be given if you are trying to do some output operation then the write signal has to be given. So, read for input device and write for output device and only one data line is used to transfer the information instead of the entire data bus. So, this serial I O; so, if you are doing a serial IO, then instead of doing that is a 8 bit data transfer. So, we are doing only 1 bit data transfer and that we will see that the RIM and SIM instruction. So, they will be useful for this purpose.

(Refer Slide Time: 01:11)

Basic Concepts in Serial I/O

- Controlling the transfer of data:
 - Microprocessor control.
 - Unconditional, polling, status check, etc.
 - Device control.
 - Interrupt.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

164

So, next so, this data transfer that we are talking about. So, this is this is microprocessor control, it can be microprocessor control, the unconditional. So it may be that it is just microprocessor just transfer something to the I O device or read something from the IO device or it may be a polling that is at periodic intervals, so, it transfer some data or it may be some checking some status, etcetera or it may be controlled by the device. So, device may give an interrupts as I said that it is interrupt driven IO. So, device gives an interrupt to the processor telling that some data can be transferred now.

(Refer Slide Time: 01:51)

Synchronous Data Transmission

- The transmitter and receiver are synchronized.
 - A sequence of synchronization signals is sent before the communication begins.
- Usually used for high speed transmission.
 - More than 20 K bits/sec.
- Message based.
 - Synchronization occurs at the beginning of a long message.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

165

Now, in case of synchronous data transmission the transmitter and receiver; so they are synchronized with each other a sequence of synchronization signals is sent before the communication begins by which this the communication will be synchronized. So, they will be transferring at some clock rate. So, both side there is transmitter and receiver they should operate at the same clock rate for getting the values properly.

So, there, there is a synchronization signal usually used for high speed transmission. So, more than 20 kilo bits per second and it is a message based transmission synchronization occurs at the beginning of a long message. So, once it is synchronized, then only the actual transmission of data can start.

(Refer Slide Time: 02:29)

The slide has a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'Asynchronous Data Transmission' is centered in a large, bold, black font. To the right of the title, there is a video player showing a person in a yellow shirt. The main content area contains a bulleted list of characteristics:

- Transmission occurs at any time.
- Character based.
 - Each character is sent separately.
- Generally used for low speed transmission.
 - Less than 20 K bits/sec.

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there are two logos: the IIT Kharagpur logo and the NPTEL logo. Next to the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written. The right side of the footer bar is solid blue.

On the other hand, this asynchronous data transmission; so, a transmission can occur at any time. So, that way it is better. So, you do not have that synchronization sequence to be done and it is character based. So, every character is sent separately. So, there is a particular format for sending the characters and those characters will be having some those characters will be having some start bit, stop bit sort of thing. And then, based on that the individual characters will be sent and generally they are used for low speed data transmission; so, less than 20 kilo bits per second.

(Refer Slide Time: 03:04)

Asynchronous Data Transmission

- Follows agreed upon standards:
 - The line is normally at logic one (mark).
 - Logic 0 is known as space.
 - The transmission begins with a start bit (low).
 - Then the seven or eight bits representing the character are transmitted.
 - The transmission is concluded with one or two stop bits.

The diagram illustrates the structure of an asynchronous data character. It consists of a sequence of bits: Start, D₀, D₁, D₂, D₃, D₄, D₅, D₆, D₇, and Stop. A double-headed arrow below the bits indicates the duration of this sequence is 'One Character'. A horizontal arrow labeled 'Time' points to the right, indicating the progression of time from left to right. The Start bit is low (space), followed by the data bits D₀ through D₇, and then the Stop bit which is high (mark).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES
NPTEL

167

So, in case of asynchronous data transmission it follows agreed upon standard. So, there are some there is some standard for asynchronous data transmission. So, both the transmitter and receiver they must agree on the standard that it will follow. In terminology in case that the logic 0 is now called the space ok; that is the transmission and a logic one is called mark.

So, this is very old terminology. So, that is followed. So, logic one is called mark and logic 0 is called space and transmission begins with a start bit which is low. So, normally the line is high and when the transmission starts it sends a low start bit and then 7 or 8 bits representing the character is transmitted. So, that is for ASCII 7 bit you know and some other code like EBCDIC that is 8 bit there is extended ASCII which is again 8 bit.

So, anyway; so, that 7 bit or 8 bit will be transmitted. So, that is D₀ to D₇, those bits are transmitted and the transmission is concluded with 1 or 2 stop bits. So, after that 2, 1 or 2 stop bits are transmitted after 7 bits after 7 or 8 bits have been transmitted as per the protocol. So, 1 or 2 stop bits will be sent to tell that this is this particular character transmission is done.

(Refer Slide Time: 04:29)

Simplex and Duplex Transmission

- Simplex.
 - One-way transmission.
 - Only one wire is needed to connect the two devices
 - Like communication from computer to a printer.
- Half-Duplex.
 - Two-way transmission but one way at a time.
 - One wire is sufficient.
- Full-Duplex.
 - Data flows both ways at the same time.
 - Two wires are needed.
 - Like transmission between two computers.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now as per as the transmission policy is concerned it can be a simplex transmission. So, only one way transmission; so, the transmission is always from the master device to the slave device like if you are thinking of a system where microprocessor is connected to a number of devices and all these devices maybe output device. And then the processor always transfers to the, to the device. On the other hand if the input device, then the device processor always reads from the input. So, that way it is like a simplex transmission one way transmission.

So, only one wire is needed to connect the two devices because it is only one way transfer. So, communication from computer to a printer; so, in most of the cases this is a simplex excepting the situation that sometimes printer status has to be told and that comes as a message from the printer. So, for those types excepting those type of cases. So, it is the simplex communication.

Then we have got half duplex communication. So, 2 way transmission, but only one way at a time; so, it is a communication between transmitter and receiver, but at one the sender and the receiver about at one point of time the transmission is in only one direction. So, basically the telephone conversation. So, they are of half duplex nature. So, again one wire is sufficient because at one point of time the communication is going only in one direction, but the problem with simplex and half duplex is that the speed of transmission is low because we cannot communicate both way simultaneously.

So, the full duplex is a situation where the data can flow in both the ways at the same time. So, two wires are needed. So, like transmission between two computers. So, they transmit between themselves simultaneously. So, that is the full duplex type of communication.

(Refer Slide Time: 06:23)

The slide has a yellow background with a black header bar containing icons. The title 'Rate of Transmission' is centered in a large, bold, black font. Below the title is a bulleted list:

- For parallel transmission, all of the bits are sent at once.
- For serial transmission, the bits are sent one at a time.
 - Therefore, there needs to be agreement on how “long” each bit stays on the line.
- The rate of transmission is usually measured in bits/second or baud.

At the bottom left, there are two logos: the IIT Kharagpur logo and the NPTEL logo. The NPTEL logo includes the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the slide, there is a small video frame showing a man in a yellow shirt.

As per as the rate of transmission is concerned for parallel transmission all of the bits are sent at once, because this parallelly all the 8 or 10 bits can go, but for serial transmission the bits are sent one at a time. So, one bit at a time.

So therefore, there needs to be an agreement on how long each bits stays on the line. So, before starting the transmission the sender and receiver they must agree about the bit duration; the duration for which the bit should be lying on to on to the line. So, that is; so how long it should be there in the line the rate of transmission is usually measured in bits per second also known as baud. So, which is actually symbols per second baud is actually symbols per second, but many a times, we just use it for the sake of simplicity the bits per second also, but baud is actually symbols per second.

(Refer Slide Time: 07:18)

Length of Each Bit

- Given a certain baud rate, how long should each bit last?
 - Baud = bits / second.
 - Seconds / bits = 1 /baud.
 - At 1200 baud, a bit lasts $1/1200 = 0.83$ m Sec.

Now how can you determine length of, length of each bit given a certain baud rate how long should is bit lasts. So, that is the baud rate. So, if I take baud rate B as bits per second and second per. So, if I just divide other way seconds per bits equal to 1 upon baud. So, at 1200 baud ok; so, a bit will last $1/1200$ that is 0.83 millisecond; so, each bit from the start of communication from every bit will be 0.83 millisecond. So, that is how the bit durations a bit duration is determined.

(Refer Slide Time: 07:56)

Transmitting a Character

- To send the character A over a serial communication line at a baud rate of 56.6 K:
 - ASCII for A is 41H = 01000001.
 - Must add a start bit and two stop bits:
 - 11 01000001 0
 - Each bit should last $1/56.6K = 17.66 \mu$ Sec. Known as bit time.
 - Set up a delay loop for 17.66μ Sec and set the transmission line to the different bits for the duration of the loop.

Now, for transmitting a character; so, now, to send the character over a, over a serial communication line at a baud rate of 56.6 kilo bit per second as a baud rate of 56.6 kilo. So, the ASCII for A is 41 hex that is the; this is the bit pattern ok. So, we must add one start bit and two stop bits. So, this is the start bit ok. So, that is also first we start transmitting from the LSB side; so, first the start bit will go, then the 8 bits will go and then finally, 2 stop bits.

Now, each bit should be $1/56.6$ K that is 17.66 microsecond. So, this is also known as the bit time. So, after transmitting a character we should wait for 17.66 microsecond and then only we should transfer the next bit. So, we have to set up a delay loop of 17.66 microsecond and set the transmission line to the different bits for the duration of the loop. So, that has to be done.

(Refer Slide Time: 08:59)

The slide has a yellow background and a blue header bar. The title 'Error Checking' is centered in a large, bold, black font. Below the title is a bulleted list of points:

- Various types of errors may occur during transmission.
 - To allow checking for these errors, additional information is transmitted with the data.
- Error checking techniques:
 - Parity Checking.
 - Checksum.
- These techniques are for error checking not correction.
 - They only indicate that an error has occurred.
 - They do not indicate where or what the correct information is

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. To the right of the footer, there is a small video window showing a man in a yellow shirt speaking.

So, this program can be written very easily. So, we can do some serial transmission and then wait on a loop for 17.66 microsecond and then transmit the next character, next bit.

Now, in the process some error may creep in like say it is transmitted over a signal line. So, that signal line may get some error. So, that how to catch a, how to get rid off those errors; so, we have we have to at least check for the error and we need to, if you need to correct it some additional information has to be transmitted for the correction at least for checking also it has to be transmitted.

Now, for error checking techniques; so, there are well known techniques like parity checking and checksum. So, these techniques are for error checking and not for correction they just check whether the received bit pattern that it has got. So, whether it is possibly correct or not and then if there is an error then in many cases, so, it can detect that situation that an error has occurred and it can tell the transmitter that there was a problem with the last transmission so that the transmission can, may be repeated by the transmitter.

However any such error checking technique, it cannot guarantee that all possible errors will be caught at the same time, it cannot indicate whether the wire or what the correct information is, ok. So, what is the error or what is the correct information. So, that it cannot check, but it cannot tell, but many a times it can detect the situation that an error has occurred in the transmission.

(Refer Slide Time: 10:39)

Parity Checking

Make the number of 1's in the data Odd or Even.

- Given that ASCII is a 7-bit code, bit D the parity information.
- Even Parity
 - The transmitter counts the number of ones in the data. If there is an odd number of 1's, bit D is set to 1 to make the total number of 1's even.
 - The receiver calculates the parity of the received message, it should match bit D .
 - If it doesn't match, there was an error in the transmission.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, as far as the parity checking is concerned, so, maybe while transmitting the data bit. So, we make the number of ones in the data odd or even. So, if the ASCII is 7 bit code. So, bit D of the parity information, so, if the ASCII code is a 7 bit code, so, we can have a one another bit can be added for the parity information. For the even parity the transmitter will count the number of ones in the data. If there is an odd number, then this D bit will be set to 1 and make the total number of 1 is even and the receiver will now calculate the now the parity by of the received message and it should match the bit D ok. So, if it does not match; that means, there was some error in the transmission.

(Refer Slide Time: 11:29)

Checksum

Used when larger blocks of data are being transmitted.

The transmitter adds all of the bytes in the message without carries. It then calculates the 2's complement of the result and send that as the last byte.

The receiver adds all of the bytes in the message including the last byte. The result should be 0.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

174

So, that is how this parity checking works. Other possibility is using checksum. So, parity works at the individual character level because it otherwise what will happen is that it will not have that much capability to determine all the errors ok so that way because if there are so, more than one bit failure. So, this parity bits check may fail ok. So, that may say that the data is the data has been received correctly where as there are multiple failures for which it is not detected by that mechanism.

So, if you have got larger block of data then this checksum is useful. So, the transmitter adds all of the bytes in the message without carries and then calculates the two's complement of the result and send that as the last byte. So, suppose a transmission is a spanning over say 100 bytes what the transmitter will do it will just go and adding those 100 bytes and then at the end it will take A. So, that carries will be discarded and at the end it will calculate the two's complement of the result and the result will be sent as the last byte.

So, at the receiving end; so, it will again calculate all the sum out all the all the messages including the last byte and since, I have taken the two's complement there, so, if I add this the result should be 0 because that is a negative number. So now, the result should become 0. So, the receiver can simply check whether the sum of all these bytes received along with the last byte is 0 or not. So, that way this checksum can be used.

(Refer Slide Time: 13:05)

RS 232

A communication standard for connecting computers to printers, modems, etc.

- The most common communication standard.
- Defined in the 1950's.
- It uses voltages between +15 and -15 V.
- Restricted to speeds less than 20 K baud.
- Restricted to distances of less than 50 feet (15 m).

The original standard uses 25 wires to connect the two devices. However, in reality only three of these wires are needed.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 175

One standard for the communication, serial communication is RS 232 communication standard. So, it is a communication standard for connecting computers to printers modem, etcetera.

The most common communication standard is existing till 1950s, since 1950s, it uses voltage levels between +15 volt and -15 volt. So, -15 volt is taken as logic high, +15 volt is taken as logic low and it is restricted to speeds less than 20 kilo baud. So, the speed is low restricted distance is within 50 feet or 15 meter. So, distance is also restricted the original standard it uses 25 wires to connect the 2 devices. However, in reality only 3 of this wires are needed. So, original standard was 25 wires, but since the distance is not that much.

So, often three wires are sufficient.

(Refer Slide Time: 13:58)

The slide has a yellow header with the title 'Software-Controlled Serial Transmission'. Below the title is a bulleted list of steps:

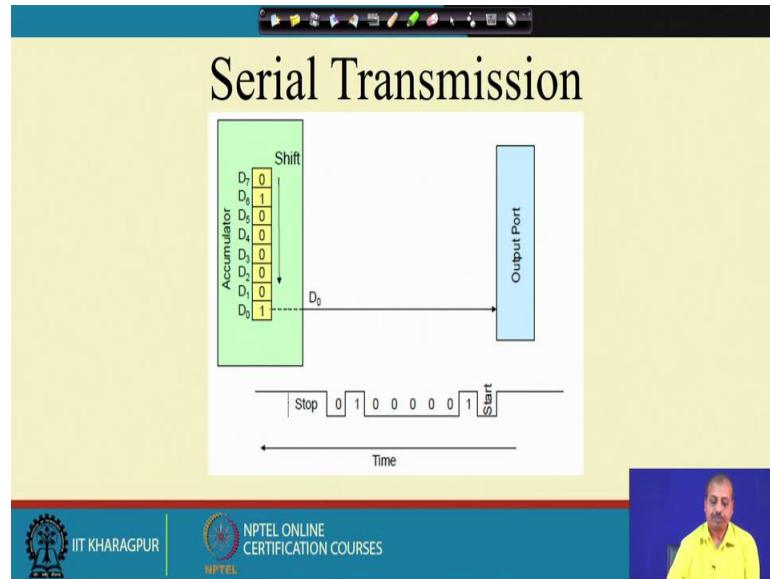
- The main steps involved in serially transmitting a character are:
 - Transmission line is at logic 1 by default.
 - Transmit a start bit for one complete bit length.
 - Transmit the character as a stream of bits with appropriate delay.
 - Calculate parity and transmit it if needed.
 - Transmit the appropriate number of stop bits.
 - Transmission line returns to logic 1.

At the bottom of the slide is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt.

Now, if we are using some microprocessor for transmitting this data serially, so, we can use some software controlled mechanism for this serial transmission. So, what are the main steps involved in this transmission of a character. So, transmission line is at logic one by default. So, that is high. So, first we have to transmit a start bit for one complete bit length. So, for 1 bit length; so, it should transmit a 0 which is the start bit.

Then it will transmit the character as a stream of bits with appropriate delay. So, it will start transmitting the character bits one by one. So, 7 bits will be transmitted. Then it will calculate the parity and transmit it if needed. So, it will calculate the parity and the parity if it is has to be 0 or 1. So, based on that it will be sent and **trans** then it will transmit appropriate number of stop bits and then the transmission line should return to logic 1. So, this is the flow of any software routine. So, that will be responsible for doing the serial transmission.

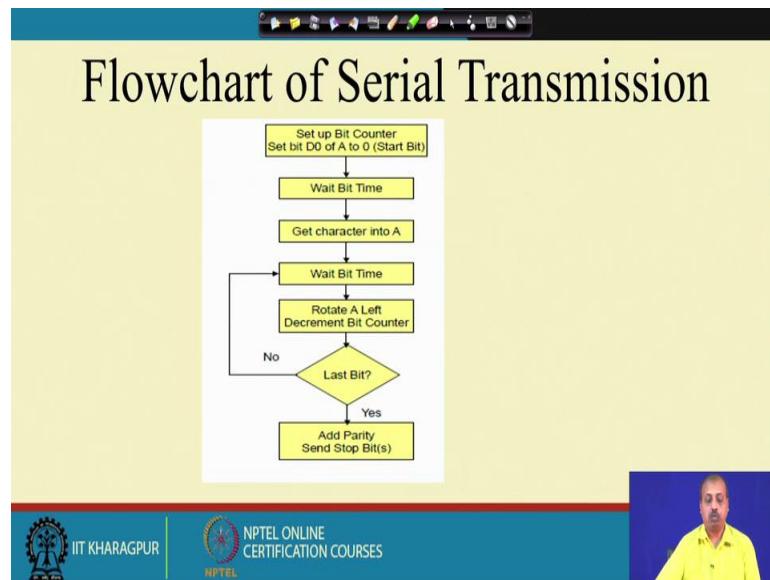
(Refer Slide Time: 15:08)



So, suppose I have got a microprocessor where in this bit D_0 is actually doing this is having the bit pattern and that is from D_0 . So, it is connected to the output port and that is going to the serial output.

Now if we are transmitting this pattern, then first it will be giving the start pulse which is low then this one will go then all these 0s will go, then 1, then 0; so all these bits. So, they will be shifted one by one and they will be transmitted over a from this bit D_0 on to the outside world serially. And finally, two stop bits will be transmitted ok. So, on this line D_0 ; so, over the time; so, this happens like this first start bit, then this 1 0 0 0 0, then 1 0, then 2 stop bits. So, that should be the mechanism.

(Refer Slide Time: 15:58)



So, how you should do the serial transmission? First we should setup bit counter. So, setup bit counter. So, that is actually setting the for the bit delay. So, how much delay should be there. So, that is determined by this bit counter, that depends on the baud rate that we have we are going to process we are going to have in our system. So, it will set up the bit counter and set bit D0 of A to 0 to generate a start bit. Then we wait for a bit time.

So, that way that 0 is transmitted over this from this D 0 of the accumulator for that much time, then we get the character into the accumulator wait for a bit time rotate a left ok. So, we rotate a left. So, that is basically a shifting and decrement the bit counter. So, so, we just; so, how many bits will be there. So, those bits will be transmitted; so, till it is last bit; so, it will be it will be going on doing this operation and then it will wait for the bit time. So, how much time it has to wait. So, it will wait for that. So, this way it will be going on. So, this bit counter actually is setting how many bits the data will have.

So, there is so many so many times, the bit will be the bits will be transmitted serially and between every 2 bit transmission, so, we are waiting for this bit time which is fixed by the baud rate, then after the last bit has been transmitted. So, here will send the parity bit and send will send the stop bits. So, they are not shown here very clearly because again, there will be bit times between this parity bit and stop bit two stop bits and all. So, they the bit timing information will be there. It is not shown here explicitly.

(Refer Slide Time: 17:46)

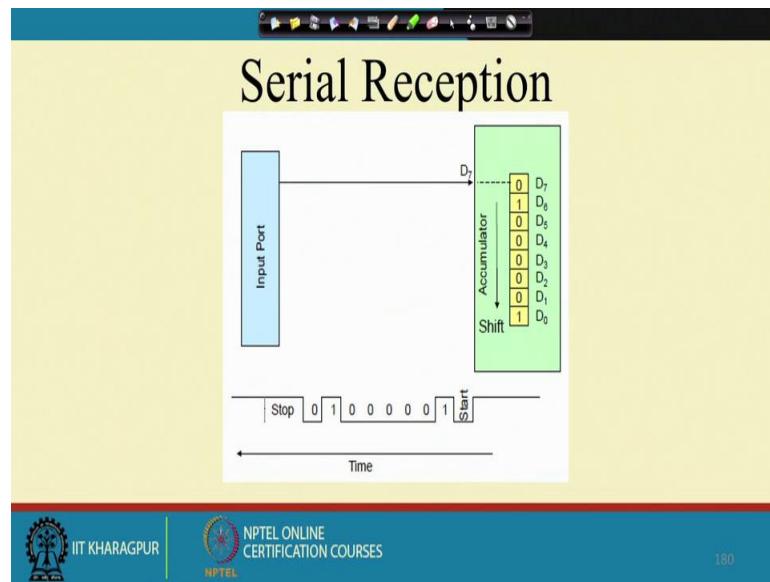
Software-Controlled Serial Reception

- The main steps involved in serial reception are:
 - Wait for a low to appear on the transmission line.
 - Start bit
 - Read the value of the line over the next 8 bit lengths.
 - The 8 bits of the character.
 - Calculate parity and compare it to bit 8 of the character.
 - Only if parity checking is being used.
 - Verify the reception of the appropriate number of stop bits

On the receiving side, the main steps involved in serial reception is that wait for a low to appear on the transmission line. So, transmission line is generally high; so the start, when the transmitter starts sending the signal. So, it will be making it low. So, that way it will be, it will be making the transmission line low. So, the receiver should wait for the start bit, it should read the value of the line over the next 8 bit length. So, that is the 8 bits characters will be there. So, it will read the 8 bit lines, 8 bit values.

Then it will calculate the parity and compare it to the bit 8 of the character to see whether the parity has been received correctly or not only if the parity bit checking is being used, then only this step will be done and verify reception of the appropriate number of stop bits that is 1 stop bit or 2 stop bit that depends on the standard that we have following. So, based on that it will be getting the appropriate number of stop bits it should get those bits.

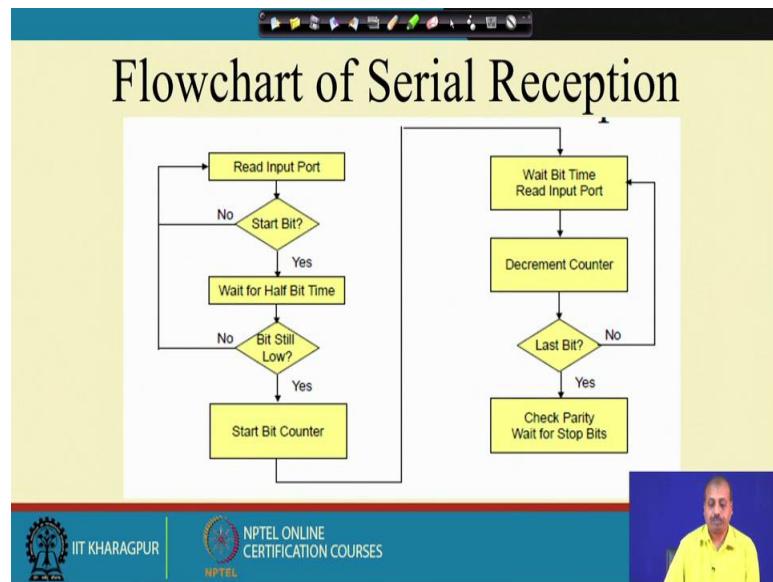
(Refer Slide Time: 18:49)



So, for transmission for the reception; so, it is from the input port. So, this D_7 the bit D_7 is actually getting the input the serial input in case of 8085 processor.

Now, it should get in this sequence first it should get a start signal. So, previously the line was high when it gets the start signal, it will know that now the serial transmission is going to start. Then this, so, this 0 1, then 0 0 0 0 1 then; so, this is the parity bit. So, since this is the following an even parity and this byte that we have sent already to ones are there. So, this parity bit is equal to 0 and then it will expect that 2 stop bits will be transmitted. So, all this should come on to the line D_7 . So, if there are this received in this sequence; that means, the character has been received correctly over the serial transmission.

(Refer Slide Time: 19:45)



So, the flowchart for the reception process is like this that it will read the input port. So, as long as there is no start bit detected, so, no action; so, it will just go on reading the input port. If a start bit has been detected, it will wait for half bit time and bit is still low then it will if bit is high; that means, there was a glitch ok. So, that part that has to be ignored.

So, if there is a glitch that bit is not low; that means, there is a glitch. So, it goes back to check the start bit again and if it is bit is still low; that means, it is not a glitch. So, it will start the bit counter it will wait for the bit time. So, how much time it has to wait for a bit; so, it will wait for that much time and then it will read the input port, will decrement the counter, so, that way it will go on. So, till whatever be the number of bits that it wants to that it expects to receive.

So, based on that this bit counter will as been set; so, it will be decrementing that counter and when all those bits have been received then it will be, it will come to the check parity and wait for the stop bit. So, it will again, this part is not written thoroughly. So, this is again waiting for two continuous 1 bit for the stop bit. So, that way this whole operation takes place. So, this is the flow chart for the serial reception part.

(Refer Slide Time: 21:15)

- The 8085 Microprocessor has two serial I/O pins:
 - SOD – Serial Output Data
 - SID – Serial Input Data
- Serial input and output is controlled using the RIM and SIM instructions respectively.

So, in case of 8085 processor, so, it has got 2 serial pins SO serial output data and SID serial input data. So, serial input and output is controlled using the RIM and SIM instructions that we have in 8085.

(Refer Slide Time: 21:33)

- The figure below shows how SIM uses the accumulator for Serial Output.

Diagram of the 8085 Accumulator Register Bits:

7	6	5	4	3	2	1	0
SDO	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

Serial Output Data ← → 0 – Disable SOD
1 – Enable SOD

So, just to recapitulate, the SIM is, so, it has got these bits 7. So, this accumulator bit 7 gets. So, whatever be the value there. So, serial data output and this SDE serial data enable; so, if the serial data enable is equal to 1, then whatever is the available in the SDO that is bit number 7 will be put on to the serial output data in the SIM instruction.

(Refer Slide Time: 22:03)

RIM and Serial Input

- Again, the RIM instruction has dual use
 - Reading the current settings of the Interrupt Masks
 - Serial Data Input
- The figure below shows how the RIM instruction uses the Accumulator for Serial Input

	7	6	5	4	3	2	1	0
SDI	P7.5	P6.5	P5.5	IE	M7.5	M6.5	M5.5	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Similarly, we have got for the serial input the RIM instruction. So, RIM instruction, so, here the serial data input that is coming that that pin value will be coming to SID pin. So, that pin value will be coming to bit number 7 of the accumulator which is the serial data input and here of course, there is no nothing like enable and all. So, this are, this is the RIM and SIM instruction that we have seen previously for doing this serial output and input.

(Refer Slide Time: 22:33)

Ports?

- Using the SOD and SID pins, the user would not need to bother with setting up input and output ports.
 - The two pins themselves can be considered as the ports.
 - The instructions SIM and RIM are similar to the OUT and IN instructions except that they only deal with the 1-bit SOD and SID ports.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, using this SOD and SID pins the user would not need to bother about setting up input and output ports, because there on the single line output is single line connection is there.

And these 2 pins themselves can be considered as port. So, you can connect it to some other device that support the serial transmission. And then it can go like this. Instruction SIM and RIM are similar to the out and in instruction except that they only deal with one bit serial output data and serial input data ports. So, we can think of this SIM and RIM as if they are equivalent to one bit operation.

(Refer Slide Time: 23:11)

```

    Example
    • Transmit an ASCII character stored in
      register B using the SOD line.

    SODDATA      MVI   C, 0BH ; Set up counter for 11 bits
    XRA   A       ; Clear the Carry flag
    NXTBIT      MVI   A, 80H ; Set D7 =1
                  RAR   ; Bring Carry into D7 and set D6 to 1
                  SIM   ; Output D7 (Start bit)
                  CALL  BITTIME
                  STC   ; Set Carry to 1
                  MOV   A, B  ; Place character in A
                  RAR   ; Shift D0 of the character to the carry
                         Shift 1 into bit D7
                  MOV   B, A  ; Save the interim result
                  DCR   C     ; decrement bit counter
                  JNZ   NXTBIT
  
```

Handwritten annotations:

- Register B starts with $B = 11010100$.
- After XRA A, the carry flag is cleared.
- After MVI A, 80H, D7 is set to 1.
- After RAR, the carry flag is set.
- After SIM, the start bit is output.
- The annotations also show the flow of control through the program.

So, this is an example to transmit an ASCII character stored in register B using the SOD line.

So, first we set up like how many bits we want to transfer. So, there is a 8 bit of ASCII data plus we will be transmitting one stop start bit and 2 stop bit. So, total 11 bits we want to transmit. So, in the C register we note down the counter for 11 bit. So, MVI C, 11; so that will have this 11 bits count the count 11 then XRA A. So, it will clear the carry flag and then MVIA, 80 hex. So, that is setting that D 7 equal to 1 and then this RAR. So, this will bring carry to D7 and set D6 to 1.

So, this is actually rotate accumulator right. So, they it will rotate right. So, the bit D 7 is since the carry was reset previously. So, by doing this rotate. So, we will get the bit for the carry coming to D7 and the bit numbers D7 was set to one previously, so, that goes to D6.

So, that now the SIM instruction. So, in the SIM instruction if you see in the SIM instruction; so, we have got this SDE pin, SDE bit has to be one for transmitting that. So, for sending the start bit, so, this SDO it should be 0 and the SDA should be 1.

So, that is exactly what has happened here. You see that after doing this, so, this bit D 6 has become equal to 1. So, SDE is equal to 1 and bit D7 is equal to 0. So, that is why it is transmitting the start bit. So, this SIM will transmit the start bit.

Then we have to wait for the BITTIME; so this call BITTIME. So, this makes this is a delay routine. So, it is not written here explicitly, but you can understand that it will be calling the delay routine for that half one bit time the now we want to transfer the characters ok. So, STC so that it will set carry to 1 and MOV A, B. So, that will place the character in a now we do a rotate right. So, as a result these shift D0. So, D0 will come to the carry bit and one will come to the D7, then we save this intermediary value intermediary character that that we have in B register then decrement C and JNZ next bit. So, next bit will be we are getting again this carry as D7 as 1 and then it will be doing a SIM.

So, let us take a pattern. So, I think we can we can take a pattern and try to understand that how it is working. Suppose if the I said that in the B register, I will have the bit pattern to be transmitted suppose I have the bit pattern 1 1 0 1 0 1 0 0. So, this is the bit pattern to be transmitted. So, up to this much from the beginning we have seen that we have transmitted the start bit. So, up to this much is okay and then what we are doing. So, we are setting the carry bit to 1.

So, this is the carry. So, carry is set to one and MOV A comma B. So, a register also gets 1 1 0 1 0 1 0 0 after that there is a shifting, rotate accumulator right. So, as a result; so, D 0 will be coming to the carry. So, this carry gets this D0. So, this 0 comes here and shift 1 bit to 1 into bit D7. So, this since carry was previously set to 1. So, this one will be coming to D 7. So, this all these things are shifted. So, this 0 goes here one comes here, then 0 1 0 1.

So, this, all these bits will be shifted. So, I will get 1 1 0 1 0 1 0 and then this one from the carry, so, it comes here by this rotate right instruction. So, it will be D0 bit D0 of this accumulator will be coming to the carry and this carry which was set to one. So, that will come to the bit D7, then this result this intermediary result is saved on to the B register. So, B has got this particular value and then decrement C. So, C have, 1 bit is decremented

and jump on not 0 to next bit. So, it will be coming here, then it will be moving A the 8 0 hex. So, this 8 0 is having this A register will have this 8 0 pattern, this is A register and then rotate right.

So, this one will be coming to this position and this carry. So, which was having the value 0 which is the LSB pattern that we have that LSB will be coming to this position now after that when I am doing a SIM actually this 0 will be transmitted which is nothing, but the LSB of the character. So, this character is now this bit is now transmitted.

In the next stage, since the B register has got this value. So, 1 0 has been already been transmitted next this 0 is there in the B register. So, this 0 will be transmitted which corresponds to the 0 that we have at the next position. So, this way serially the bits will be transmitted one by one through the SOD line using this SIM instruction and all.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture – 62
8085 Microprocessor
(Contd.)

(Refer Slide Time: 00:19)

The screenshot shows a Windows desktop environment with a Notepad window open. The Notepad contains the following assembly code:

```
MVI B, 00H
MVI C, 08H
MOV A, D
BACK: RAR      ; D0 to CY, CY to D7
       JNC SKIP
       INR B
SKIP:  DCR C
       JNZ BACK
       HLT
```

To the right of the code, there is a hand-drawn state diagram for the 8085 processor. It shows the累加器 (Accumulator) A and the数据寄存器 (Data Register) D. The D register is shown with its bits labeled D₇ through D₀. An arrow points from the D register to the CY (Carry) flag, which is highlighted with a circle. Another arrow points from the CY flag to the A register, indicating a rotation operation.

The Notepad window has a title bar "Count number of 1's in the contents of D register and store the count in the B register". At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player showing a person speaking.

So, we will look into a few examples of 8085 programming. The first example that we will look into is a program to count number of 1's in the content of D register and we want to store the value in the B register. So, our D register contains some numbers say. So, this is the D register then the content maybe say 10101111 and we want to count the number of 1, so in this case the answer should be 6. So, how do we do it? So, first this B register, since the B register will be holding the final value, so we are clearing the B register. So, B register made equal to 0, then in the C register we take it as a, some sort of counter like how many bits we have considered so far ok.

So, basic idea is that so we will move this D register content into A register and then to we will check what is the value of this least significant bit. And for the technique for checking there can be several ways by which you can check the content of this least significant bit one possible way is like this, so you have that carry flag. So, you do a rotation, so rotation

like this and this carry flag comes here to the most significant position. So, this is the bit D0 and this is the bit D7 of the A register.

So, if this, if this bit was equal to 1 then after this rotation the carry flag will be equal to 1. So, we can check the status of this carry and after 8 such rotations, so we will have the count we have we have already seen all the 1's in the B register. So, the program is like this that we take the we move the content of D register to A register then rotate accumulator right. So, this rotate accumulator right, so this does exactly the thing that we were looking into the D0 will be coming to the carry bit and the carry bit will come to the D7 register. Then jump on no carry, so if the carry is not saved; that means, the corresponding bit was reset the bit D 0 was reset, so jump on no carry to skip.

So, there we do not increment the value of the count B, but if the carry was set then it will come to this point INR B and then that B register value will be incremented by 1 it and it will be counting that that 1 it has seen that 1 bit that it has seen. And then it will decrement the C register because we have already seen we have already checked the value of 1 bit, so it will be decrementing the C register, the C register will becomes 7 and then till C register becomes 0. So, it will be jumping back to it will be jumping, jump on not 0 to this back this point and then again we are rotating the next bit.

So, by the, by the first rotation what has happened is, so the bit at position D1 has shifted to D0. So, in the next rotation this bit D1 will be coming to this carry bit. As a result we can again check the carry and then we can take a decision whether bit D1 was equal to 1 or not. So, this way we can count number of 1's in the D register and store the value in the C register in the B register.

(Refer Slide Time: 03:36)

Sort given 10 numbers from memory location 2200H in the ascending order

MVI B, 09 ;Initialize counter 1 START : . LXI H, 2200H ;Initialize memory pointer MVI C, 09H ;Initialize counter 2 BACK: MOV A, M ;Get the number INX H ;Increment memory pointer CMP M ;Compare number with next number JC SKIP ;If less, don't interchange JZ SKIP ;If equal, don't interchange MOV D, M ;Interchange two numbers MOV M, A	<p>2200 10 HL 2201 5 HL</p> <p>A = 10 D = 5</p>
---	--

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Next we will be looking into another program that will sort a given 10 numbers from memory location 2200Hex in the ascending order. So, from 2200Hex we will have 10 numbers and then those values will be sorted. The sorting algorithm that we will use is the basically whenever we find 2 numbers out of order we exchange them and this process we repeat for all the 9 entries and that way it is done.

So, how are you doing it? So first this B register is having the value 09, so this B register is holding the counter then so we actually need 2 counters. So, this B register any loop comparison, so you know that any sorting algorithm it uses nesting or nesting of 2 loops the bubble sort type of algorithms, so it uses nesting of 2 loops. So, here exactly what we have done here sp for the first loop the counter is in B register for the second loop the counter is in C register then this LXI H 2200Hex. So, this is the 2200Hex value is loaded into the HL pair and then we get the first number into the A register. So, from 2200 onwards we have this numbers stored.

So, if this is the location 2200 the very first thing that we need to do is to check the contents of 2200 and 2201 and if the numbers are out of order we need to exchange them. So, my HL pair is now pointing to this location 2200 by means of this instruction and then MOV A, M, so whatever be the value suppose the value was 10. So, the A register is having the value 10 now then INX H now the HL pair is moved to this point ok.

So, HL pair points to this the next entry and then compare memory, so this 10 is compared with the next 1 suppose the value here is 5, so 10 is compared with 5. So, if it is less then jump on carry, the jump on carry means the number memory the location the value at the next location is larger than the accumulator.

Otherwise the if the number is smaller than the accumulator which is the which is the situation here, then the carry flag is not set, or is not set, so jump on carry and jump on 0, so they will be checking whether it is less or equal; if it is not so then we need to interchange between these 2 memory locations. So, here actually that is that is the situation so for that purpose we MOV D, M. So, D register gets the value 5 and then MOV M, A, so memory register memory is pointed to by HL and A register value is 10, so at this location we get the value 10.

(Refer Slide Time: 06:43)

Sort given 10 numbers from memory location 2200H in the ascending order
(Contd.)

```

    DCX H
    MOV M, D
    INX H
    SKIP:   DCR C      ;Decrement counter 2
            JNZ BACK  ;If not zero, repeat
            DCR B      ;Decrement counter 1
            JNZ START
            HLT       ;Terminate program execution

```

A diagram showing memory locations 2200H and 2201H. At 2200H, the value is 105 (A). At 2201H, the value is 510 (D).

IIT Kharagpur | NPTEL Online Certification Courses

Then, then in the next, sorry then in the next point what we are doing we are decrementing H, so that this HL pair, this HL pair was pointing to this location and then this DCX H, so DCX H will take the HL back to this point the previous point and then we are doing MOV M, D will put this 5 value into this location.

So, as a result so these 2 values are interchanged. So, then we are doing INX H. So, H so the so HL pair points to this and then we are decrementing C, so counter, second counter value is decremented and we are checking whether the first loop should end or not. So,

JNZ jump on not 0 back. So, it will be going back to the location it will going back it will be going back to the location back here if the count value is non 0.

So, this C register value has become non 0, so that way this will be continuing the loop, so now it will compare. So, the next location so it will compare the next location with the with the next entry, so as a result these loop these 2 loops so in they are repeated 10 times, the each loop will be executed then all the numbers will get sorted. So, this is basically the bubble sort type of algorithm that we can implement in 8085 program.

(Refer Slide Time: 08:36)

Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H.

```
LDA 2200H ;Initialize counter
MOV C,A ;sum = 0
MVI B,00H ;Initialize pointer
LXI H,2201H ;Get the number
MOV A,M ;Mask Bit 1 to Bit7
ANI 01H ;Don't add if number is ODD
JNZ SKIP ;Store result in B register
MOV A,B ;SUM = SUM + data
ADD M ;Store result in B register
MOV B,A ;increment pointer
INX H ;Decrement counter
DCR C ;jif counter 0 repeat
JNZ BACK ;store sum
STA 2210H ;Terminate program execution
HLT
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will be looking into another program that will calculate the sum of series of even numbers from the list of numbers. So, we have got a list of numbers and then we are interested to sum only the even numbers from there, so odd numbers will not be added. So, the check for odd number is that the least significant bit is equal to 1 and for even number the least significant bit is equal to 0. So, the number of numbers is a stored in the location 2200 and then the actual numbers begin from the location 2201.

So, for the program we just we first load this accumulator with the content of the location to 2200, so the count value is available in the A register that is moved to C register. So, the counter is initialized to C and then B register so that is initialized to 0. So, B will contain the full sum so that is initialized to 0. So, the HL pair they should point to the successive numbers, so in the numbers start from 2201.

So, HL pair is initialized to 2201, MOV A, M, so this will get the first number into the accumulator and then we add immediate with 01 Hex. So, that way the bits 1 through 7 so they will be masked out only bit 0 will remain because, they will be ANDed with this 0 1, then if that bit is 1 if that bit was also 0 then this jump on not 0 will fail, that means the number was even. But if the number is odd then this JNZ will be successful and in that case it will be jump, it will not increment the sum value and it will come to this point.

But if the number is an even number then this check will be a failure, in that case this you will get the number into we will get the B register content into A register then we will be add M. So, this is basically with the previous sum we are stored we are adding the current even number and then storing back the result to the B register MOV B, A, so this will store the result of this addition into the B register. Then we increment H decrement the C register and then JNZ back, so if the C register value has not become 0 that is all the numbers are not checked, so it will be jumping back to this position and finally the number, the added values, so, they will be stored at the location 2201 Hex.

So, this way this program is doing the addition of only the even numbers from a series of numbers. So, we can find out we can write this type of programs in 8085, so whatever you have got whatever program you can think about in high level language, so, they can be written in assembly level or we can take help of some compiler and assembler to convert the high level program into assembly language program.

The advantage with the assembly language programming will be you know the exact size of the program, you also know the exact amount of time that will be taken for executing this program. So, if you consult the 8085 manual, then you can find out the individual number of bytes needed for individual instructions to compute the size of the program and you can also find out the total number of clock cycles needed by individual instructions. So, you can trace through the algorithm trace through the program and compute what will be the exact time needed for executing this program using 8085. So, that is how that is why we can take help of this assembly language programming.

(Refer Slide Time: 12:14)

Unpack the packed BCD number

```
LDA 3000H ;Get the packed BCD number from the memory, let it be 98H
MOV B,A
MVI C,04
ANI FO ;A = 90H
RRC ;Need to be rotated right for 4 times to get A = 09H
DCR C
JNZ L1
STA 3001
MOV A,B
ANI OF ;A = 08H
STA 3002
HLT
```

Handwritten notes:

- Memory location 3000 contains 98.
- The packed BCD value 98 is represented as 1001 1000 0100 1000.
- After rotating right 4 times, the value becomes 0100 1000, which is 4.
- After another rotation, the value becomes 0000 1000, which is 0.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will be looking into another program which will be converting a packed BCD number into an unpacked BCD number. So, packed BCD number is like this, say, suppose we want to we want to so we have the number say 98 ok. So, this 98 is stored in the in the location 3000, in the memory location 3000 this number 98 is stored. Now this is stored in a packed BCD format, so BCD stands for binary coded decimal. So, if you, if you take any decimal number say 45, then it consists of two decimal digits 4 and 5 ok

So, 1 possibility of storing this numbers is that you code this 4 into binary numbers, so 0100 and code this 5 into binary number so that is 0 1 0 1. So, this number, so this is a BCD number BCD coded form of the decimal number 45. So, we can instead of converting this into binary number, so we can, we can store them in the BCD format binary coded decimal format and this has got several utility while you are particularly displaying some decimal digits on to some display; so this format often useful.

So, many a times what happens is that we store 2 BCD digits per memory location. So, if you memory locations that generally byte organized so, per memory location we store 2 BCD digits. But, in the application so it may be required that we access them individually; each BCD digit we accept individually, so that is known as the unpacking of packed BCD number. So, when they are stored 2 BCD number 2 BCD digit per 8 bit pattern so that is called a packed BCD notation and when you are giving entire 8 bit for 1 number, that is when we are converting say for example this 98 when you are converting into 0 9 in 1 first

byte and 0 8 in the second byte, so that is the unpacked BCD numbers. So, how do we do this so you see first in the assuming that the number is 98 Hex, so we first move this and it is stored at the location 3000. So, LDA will get the number into A register, so A register is now holding 98 now that is moved to B register and in the C register we move the value 0 4 ok.

Now, we do an AND immediate F0. So, if you do and immediate F0, then F is 1 1 1 1 and 0 is this is 0000. So, this is the ANDed with 98. So, so 9 is 1 0 0 1 and 8 is 1 0 0 0 so if you do an AND immediate so you will get 1 0 0 1 0 0 0 0. So, you get the value, so you get the first digit of the BCD number. Now we want to get 0 9 from here for getting 09 so I have to rotate this right by 4 positions. So, this is exactly done in this part of the program, so RRC, so RRC will shift this thing by 1 bit position so this 0 will come.

So, this 0 will come to the most significant position then it will become 1 0 0 1 and then three zeros. So, it after 1 RRC so we will get the content like this , then we decrement C and jump on not 0 to L1 so that means, if the C value was been initialized to 4. So, it will do 4 such rotations, after 4 such rotations, so I will get the content as 0000 1001 so that is 0 9. Then we store the value at memory location 3001, at 3001 we store the value 09; so this is stored at location 3001 and then we have to do the similar thing now we have to get the value 8 ok. So, B register was already having the value 98, so we are again getting the value moved to A register and now AND immediate is 0F, so this 1 0 0 1 1 0 0 0.

So, if you AND immediate with 0F so you will get you will get so this 4 bits will becomes 0 and this will become 1 0 0 0 so you will get 0 8. So, now we do not have to rotate anything because, this is 8 is already in the least significant position so we just store the number in the location 3002. So, this way I can unpack a packed BCD number into individual digits. So this is another example. So, this way there can be innumerable number of programs that you can try out in 8085 assembly language.

(Refer Slide Time: 17:08)

The slide has a yellow background and a blue header bar. The title 'Other Important Instructions' is centered at the top. Below the title is a bulleted list of instructions. To the right of the list is a mathematical diagram for the ADC instruction. At the bottom of the slide is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person.

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L \leftarrow Mem[addr], H \leftarrow Mem[addr + 1]
- SHLD <addr>
- PCHL : PC \leftarrow HL
- SPHL : SP \leftarrow HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

$ADC \ B$
 $A \leftarrow A + B + C_f$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are some more important some other important instructions that we might not have covered in our course, but you can just have a look at them in any manual for 8085, some of the interesting instructions that we have is 1 is ADC. So, add register to accumulator with carry so because, we have got the add instruction which does not take at the carry into the number, but when you have got multi byte addition so then in that case this add with carry maybe essential ok.

So, this add with carry so this also adds the carry bit like if you say ADC B, so what we mean is we will get $A + B + \text{carry flag}$ and this carry flag will be the least significant bit. So, this A and B so they are 8 bit values and this carry will be converted into an 8 bit value like this and this is the carry. So, whatever be the carry so that will come at the least significant position and they that will form another 8 bit, so those 3 bit 3 will be added.

Then there is another important instruction which is CMC or complement carry. So, complement carry so this it just complements the carry, so if the carry was 0 it becomes 1 if the carry was 1 it becomes 0. Then there is another instruction which is known as DAA, which is decimal adjust accumulator. So, it is for converting 1 8 bit binary number into two 4 bit BCD digits. So, so it is for, you can say that converting 1 binary number into the BCD numbers, so DAA is an instruction for doing it. So, in our digital circuit combinational circuit classes, so we have seen that if for converting 1 number to BCD so

we need to add 6 if the number is greater than 9, because if the number is say for example if the number is say 10 ok, so in in binary notation if I use an 8 bit notation so this will be stored as 0 0 0 0 1 0 1 0.

(Refer Slide Time: 19:28)

Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L \leftarrow Mem[addr], H \leftarrow Mem[addr + 1]
- SHLD <addr>
- PCHL : PC \leftarrow HL
- SPHL : SP \leftarrow HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

DAD \rightarrow HL + DE
 \downarrow
 0000 1010
 0000 0001
 0000 1011

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now you see that when I am converting it into BCD notation, so what is required is that this number will be converted to 1 and 0 that is 0 0 0 1 in the first 4 bits and 0 0 0 0 in the next 4 bit. So, that is what we want and then this can be done, this can be done by doing the logic is that when this part of the number is becoming greater than 9, so we have to add 6 to it so this number is 10. So, 1 0 1 0 so with that with that if we add 6. So, that is 0 1 1 0 then the number will become 0 0 0 0 and then 1 and these 3 bits remain 0, so you get this 1 0. So, this way we can get the number as 10 ok, so when the number is greater than when this least significant 4 bits are greater than 9, so we can just add 6 and get the number.

So, this way this decimal adjust accumulator, so this is useful when you are converting a binary number into BCD number, then there is a double addition or DAD instruction, so that is for adding registered pair to HL so this register pair value. So, you can say like DAD D, so this HL pair we will get HL + DE ok. So, this is the DAD instruction. The IN instruction so for with this we have already discussed for getting some data from some input port, so you can use this in instruction. Then there is LHLD instruction, so LHLD with some address, so in that case the L register gets the content of the memory location which is pointed to which is given in this address.

(Refer Slide Time: 21:31)

The slide has a yellow background and a black header bar with various icons. The title 'Other Important Instructions' is centered at the top. Below the title is a bulleted list of instructions:

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L \leftarrow Mem[addr], H \leftarrow Mem[addr + 1]
- SHLD <addr>
- PCHL : PC \leftarrow HL
- SPHL : SP \leftarrow HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

Handwritten notes are present on the right side of the slide:

- Above the list: LHLD 1000
- Below LHLD 1000: L \leftarrow Mem[1000] -
- Below L: H \leftarrow Mem[1001]

At the bottom left is the IIT Kharagpur logo and text 'IIT KHARAGPUR'. At the bottom center is the NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. At the bottom right is a small video frame showing a man speaking.

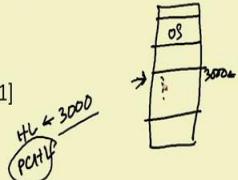
For example if you say like say LHLD 1000, then content of memory location 1000 will come to the L register and the content of memory location 1001 will come to the H register, so this way we can get this LHLD. So, this will load the content of 2 memory locations into HL pair. Then we have got SHLD instruction. So, which is just the opposite of LHLD, so instead of loading the HL pair, so it will be storing the value of HL pair into the corresponding addresses.

Then there is another very useful instruction which is known as PCHL, so this PCHL instruction the program counter is loaded with the value of HL register pair. And this is very much useful when you are trying to execute some other program, particularly from the operating systems point of view this PCHL is very, very much useful.

(Refer Slide Time: 22:33)

Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L \leftarrow Mem[addr], H \leftarrow Mem[addr + 1]
- SHLD <addr>
- PCHL : PC \leftarrow HL
- SPHL : SP \leftarrow HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



For example if I have got say in my memory, my program to the user program starts from location 3000 and the OS part is loaded up to the location this part is got OS now. How does OS transfer the control to the user program? So, this can be done by means of this PCHL instruction because for executing this users program, so somehow the value 3000 has to go to the PC register now how will it go. So, it is there is no direct instruction to load the PC register, so what the program will do it will what the operating system does is that it once it knows that the start address is 1000, somehow in the HL pair it loads the value 3000 and then it executes the PCHL instruction. So, that PC now gets the value of this HL pair so we get this PC value. So, this now, once the PC has got the value 3000, so it automatically starts executing from this point.

Similarly, we have got this SPHL instruction in which the HL pair is saved into the stack pointer, so this stack pointer is loaded into the HL pair. So, this is another this PC register and SP register they cannot be loaded directly so we can load it through the HL register pair. Then there is an exchange instruction so that exchanges the HL value with the DE value. So, this is also useful in some cases so this is exchanged, so this there is no the operand. So, it just exchanges the value of HL with the DE and we have got XTHL instruction. So, it exchanges the HL register with top of the stack. So, top of the stack content is exchanged with the HL pair.

So, this way these are some of the other instructions that we have for the 8085 programming. So, if you are so when you are you are some this instructions as we have seen. So, there more relevant when you are considering system programming into consideration; for example, when you are this instruction like PCHL, SPHL, then XTHL, so these instructions are useful when you are considering system programs.

On the other hand this ADC this ACMC this type of instructions are useful when I doing some computation and if you are having some decimal interface with your system then this DAA instruction is useful, that is for converting any number into its BCD format and for unpacking we have already seen another code which can unpack a BCD number packed BCD number into unpacked BCD number. So, you can refer to any book on 8085 that has got this whole discussion on this programming and all.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture – 63
8086 Microprocessor

So, next we will be looking into another microprocessor which is slightly more complex compared to 8085, which is the next family of microprocessor from INTEL known as 8086.

(Refer Slide Time: 00:28)

Overview

First 16-bit processor released by INTEL in the year 1978

Originally HMOS, now manufactured using HMOS III technique

Approximately 29, 000 transistors, 40 pin DIP, 5V supply

Does not have internal clock; external asymmetric clock source with 33% duty cycle

20-bit address to access memory \Rightarrow can address up to $2^{20} = 1$ megabytes of memory space.

Addressable memory space is organized in to two banks of 512 kb each; Even (or lower) bank and Odd (or higher) bank. Address line A_0 is used to select even bank and control signal \overline{BHE} is used to access odd bank

Uses a separate 16 bit address for I/O mapped devices \Rightarrow can generate $2^{16} = 64$ k addresses.

Operates in two modes: minimum mode and maximum mode, decided by the signal at MN and MX pins.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, first 16 bit processor, 8086 is the first 16 bit processor. So, internal processing is all in terms of 16 bit data unlike 8085 where it was all 8 bit operation. So, here all the operations are 16 bit operation, so it was originally release by INTEL in 1978, it was designed on HMOS technology, but later on it was upgrade it to HMOS III technology.

So, is a roughly 29000 transistors, it is a 40 pin dual in line package. So, like 8085 only it have 5 volt power supply, does not have internal clock. So, this external asymmetric clock source is 33 percent duty cycle, so you should have you have to connect the clock from the external source.

So, unlike 8085 where you can just connect a crystal and it acts as the, it can generate the clock internally in 8086 that is not possible. So, you have to give clock signal externally

with 33 percent duty cycle means 33 percent of the time the clock signal is high and 67 percent of the time the clock signal is low, 20 bit address.

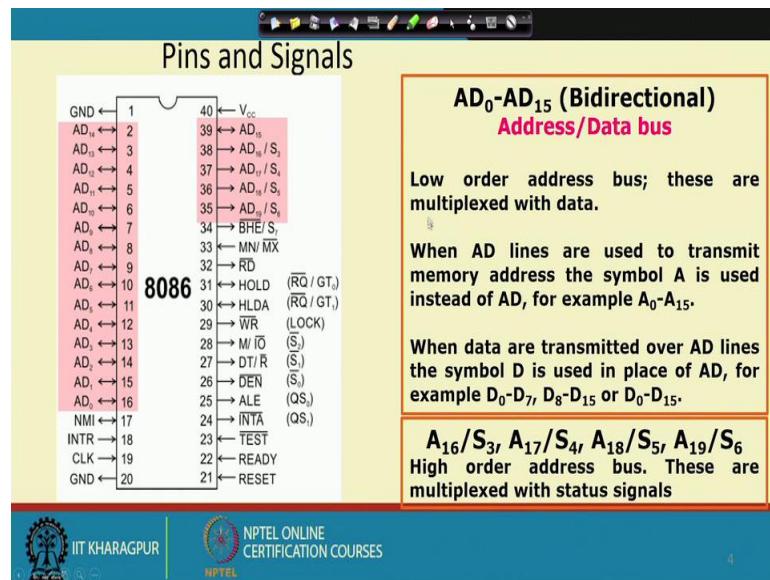
So, address bus in case of 8085 was 16 bit; in case of 8086 it is 20 bit address. So, naturally you can access address up to 2^{10} there is 1 mega byte memory space. So, addressable memory space is organized into two banks of 512 kilo byte each. So, one is the lower bank that which is the even on lower bank and odd or a higher bank.

So, address line A0 can be used to select even bank and control single \overline{BHE} is used to access the odd bank, so these are special signals that are available. So, that way we can we can divide this memory into two 512 kilobyte blocks. So, there is a 16 bit address space for the I O mapped devices unlike 8085 we had 8 bit IO address. So, here we have got 16 bit I O address. So, total number of IO devices that can be connected is 2^{16} that is 64 kilo. So, many addresses are available, so we can connect from any devices.

Another important thing that this 8086 has is that it operates in 2 modes one is known as minimum mode another is known as maximum mode. So, minimum mode operation is more or less similar to 8085 type of operation, where this process are happens to be the master and you can connect a number of you can connect memory chip and other peripheral devices to it.

However, in many systems what is required is that we have multiple masters, so the, they co-ordinate between themselves to get some operation done then gets some job done. So, that is the maximum mode of operation. So, there is a pin called MN/\overline{MX} so, if this pin is equal to 1, so the system will operated in minimum mode and if it is 0 then it will operate in the maximum mode.

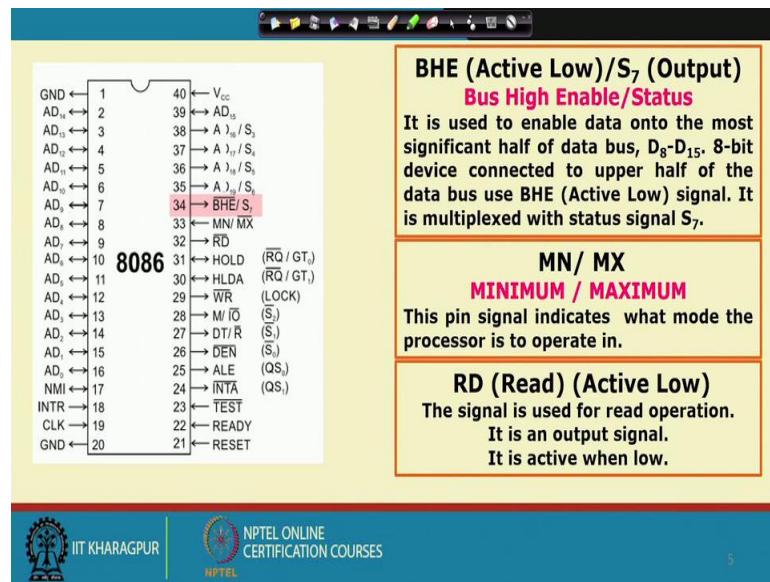
(Refer Slide Time: 03:28)



So, as usual we will be starting with the pins and signal. So, we have got this AD0 to AD15 which is bidirectional address data bus, so this AD0 to AD15 ok. So, these are the address data bus multiplexed address data bus, so just like in 8085 we had got this lower order address data bus was multiplexed, here entire address data bus is multiplexed. The AD lines are used to transmit memory address the symbol A is used instead of AD for example, A0 to A 5. So, when data is transmitted so we will be reading as D lines, so AD D0 to D7 D8 to D15 or D0 to D 15 like that.

So, the rest of the since this is a 20 bit address space as we said, so total 20 address lines are necessary; So, this AD16 to AD19 so though they are written as AD, but you can better read them as A16 to A19 because, there is no data bus line like D16, D17, D18, D 19. So, data bus is 16 bit only it ends at D 15, so you can be better read them A16 / S3. So, they are also multiplexed, but it is multiplexed with the status line S3, S4, S5 and S6.

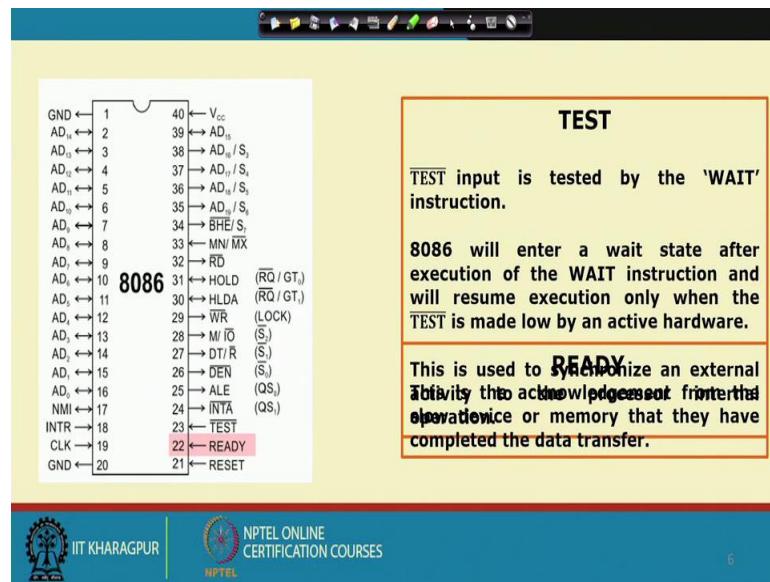
(Refer Slide Time: 04:46)



There is another signal which is **BHE** which is bus high enable or the S7 is the status BHE stands for bus high enable, it is used to enable data onto the most significant half of the data bus. So, D8 to D15 so when we want to say that we have got the data is available on the higher order data bus D 8 to D 15, in that case this **BHE** signal will be low telling the process device which is connected to D8 to D15 that data is available there. So, this that that is what is written here that 8 bit device connected to upper half of the data bus can use this BHE signal which is active low and it is multiplexed with the status signal S7, so **BHE** is multiplexed with S7.

Then there is a **MN/MX** pin so this 1 pin number 33 **MN/MX** pin. So, it will indicate whether the processor is in minimum mode or in maximum mode then there is a read bar line which is active low. So, it when it is doing a memory read or IO read type of operation then this **read** line will be low, then is another important line which is known as test.

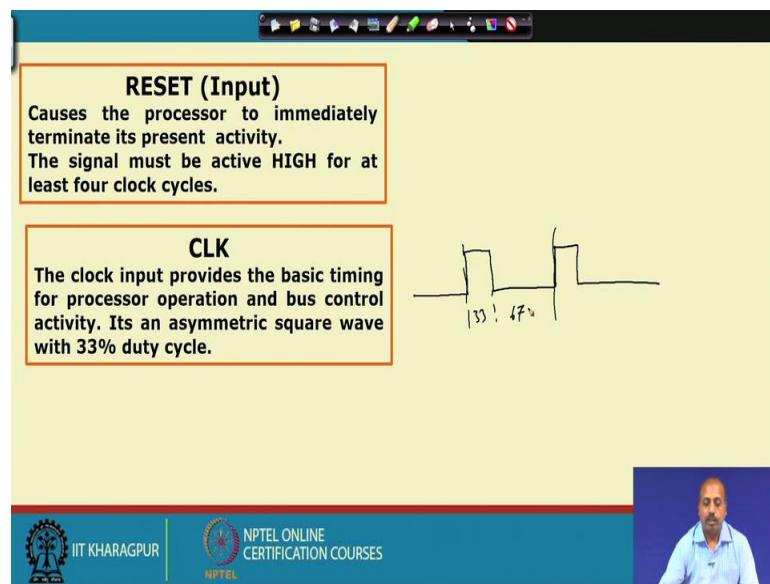
(Refer Slide Time: 05:56)



So, this test line so test bar pin is used tested by the wait instruction. So, sometime what happens is that while there are multiple there are many operations that are going on in the system, so we need to wait for some event or some particular operation to be over. So, in that case the processor 8086 has got a special instruction wait and what this wait instruction does is that it suspends the operation of this processor till this *test* line becomes low.

So, this 8086 will enter into a wait state after execution of this wait instruction and we will resume execution only when test bar is made low by some active hardware. So, that way it is there is *read* there is a ready signal, ready signal is used for external memory can tell whether the some device is slow or not then it will be the ready signal, so it will extend the memory operation.

(Refer Slide Time: 07:04)



Then there is a reset signal which causes the processor to immediately terminate its present activities, so just like 8085 reset signal. The signal must be active high for at least 4 clock cycles to be acknowledged because, if it is not so then maybe it is a very small amount of time and the reset was actually not intended, just a spike came. So, just we avoid that situation so it is required that the reset should be active for 4 clock cycles.

Then the clock we have already said, the clock will provide the basic timing for the processor operation and bus control activity, asymmetric square wave with 33 percent duty cycle. So, it is like this so this is the 33 percent of the time the clock signal is high and 67 percent of the time the clock signal is low. So, it is like this so if this is the total time period from here to here, so this is the 33 percent and this is 67 percent. So, 67 percent the signal should be low anyway so that is how this clock signal should be provided.

(Refer Slide Time: 08:15)

RESET (Input)
Causes the processor to immediately terminate its present activity.
The signal must be active HIGH for at least four clock cycles.

CLK
The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle.

INTR Interrupt Request
This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, there is one interrupt line unlike 8085 which where you have got a number of interrupt lines, so here we have got only one INTR pin which is interrupt request. So, this is the triggered input and sample during the last clock cycle of each instruction to determine the availability of the request. So, as true for 8085 also; so this interrupt line is sampled on the last clock cycle of an instruction execution. So, that way it decides whether an interrupt is interrupt has arrived or not and if an interrupt has arrived, so it will go into an interrupt acknowledge cycle.

(Refer Slide Time: 08:54)

Min/ Max Pin

GND	1	40	V _{cc}
AD ₁₅	2	39	AD ₁₅
AD ₁₄	3	38	AD ₁₄ /S ₁
AD ₁₃	4	37	AD ₁₃ /S ₁
AD ₁₂	5	36	AD ₁₂ /S ₁
AD ₁₁	6	35	AD ₁₁ /S ₁
AD ₁₀	7	34	BREQ/S ₁
AD ₉	8	33	MN/ MX
AD ₈	9	32	RD
AD ₇	10	8086	HOLD
AD ₆	11	31	HLDA
AD ₅	12	30	WR
AD ₄	13	29	M/I/O
AD ₃	14	28	D/T/R
AD ₂	15	27	DEN
AD ₁	16	26	ALE
NMI	17	25	INTA
NTR	18	24	TEST
CLK	19	23	READY
	20	22	RESET
		21	

The 8086 microprocessor can work in two modes of operations : **Minimum mode** and **Maximum mode**.

In the minimum mode of operation the microprocessor do not associate with any co-processors and can not be used for multiprocessor systems.

In the maximum mode the 8086 can work in multi-processor or co-processor configuration.

Minimum or maximum mode operations are decided by the pin MN/ MX(Active low).

When this pin is high 8086 operates in minimum mode otherwise it operates in Maximum mode.

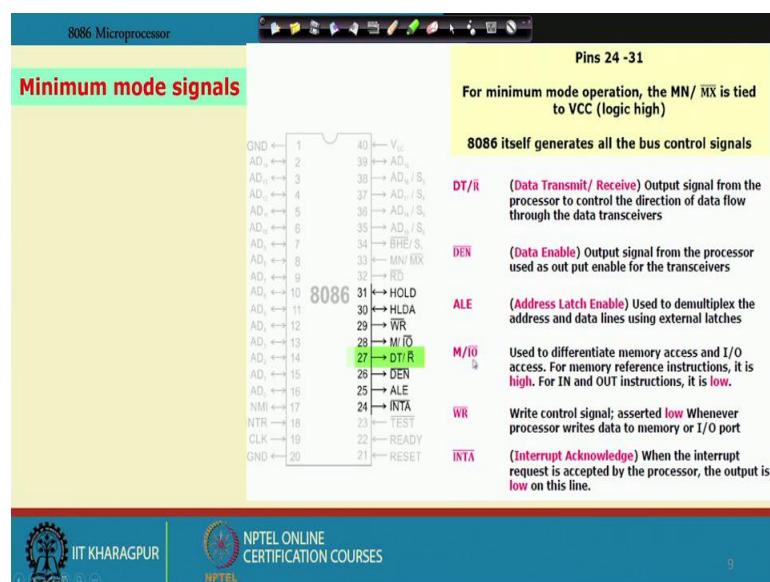
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, coming to this min max pin, so this min max pin there are 2 modes of operation for 8086, one is known as minimum mode another is known as maximum mode. So, minimum mode is just like a single processor system, so in minimum mode of operation the microprocessor do not associate with any coprocessors and cannot be used for multiprocessor system. So, if there are multiple processor, so we get a multi processor system many times this 8086 it can connect some coprocessor with it; for example, there is a very well known coprocessor 8087 which is a math coprocessor.

So, that whatever mathematical operations are required so 8086 will transfer those instructions to 8087 for execution that way operation can be made faster. So, while doing those operations the coprocessor also needs to access the memory and other devices, so we have to give the control of the bus through that device in that case or through that coprocessor in that case. So, in minimum mode of operation, so this is not possible here in the 8086 itself is the sole master of the whole system.

In a maximum mode of operation so it can 8086 can work in multi processor or coprocessor configuration and minimum maximum mode is configured by this MN/\overline{MX} pin. So, this MN/\overline{MX} line being equal to 1, so it will min it is the minimum mode and it is it is equal to 0, so it is a maximum mode of operation.

(Refer Slide Time: 10:25)



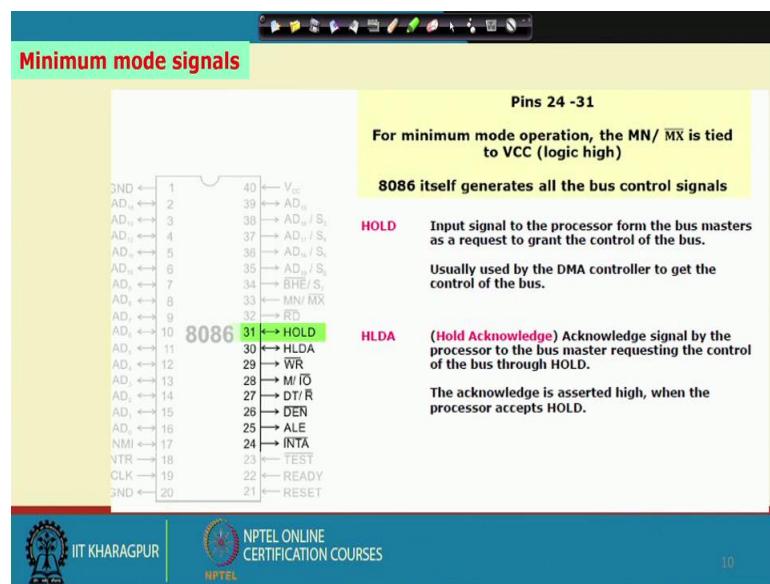
So, minimum mode signal, so these are the important signals that we have in minimum mode of operation 1 is \overline{DTR} . So, \overline{DTR} is the data transmit or receive. So, this is the output

signal process from the processor to control the direction of dataflow through the data transceiver. So, if it is using some if it is using some asynchronous transfer then this \overline{DTR} can be useful. Then \overline{DEN} data enable. So, this is output from the processor as output enable for the transceivers, so these are actually for serial transmission.

Then ALE so this is address latch enable for this is for demultiplexing the address and data bus for using external latches. Then M/\overline{IO} so in case of 8085 we have IO/\overline{M} . So, here it is the name is the convention is just reversed for memory access this M/\overline{IO} line is equal to 1 and for I O access this M/\overline{IO} line is equal to 0 and then we have got this write bar write control signal. So, it is asserted low whenever processor writes data to memory or I O code and there is \overline{INTA} line.

So, it is interrupt acknowledge when the interrupt request is accepted by the processor the output will be low. So, that way that acknowledge signal will go to the device and device will know that the processor has done into an interrupt acknowledge mode, so it will provide the interrupt service routine address.

(Refer Slide Time: 12:04)



Some more minimum mode signals. There is a hold signal, so, basically this is for relinquishing the control of this data and address bus. So, there is a mode of transfer which is known as DMA direct memory access, where this another bus controller it is used for transferring the content directly from memory one memory location to other memory block transfer or from IO device to memory. So, like that so this DMA controller it has got a

number of secondary devices has connected to it and then through this through this DMA controller. So, we can transfer data blocks from this IO devices to the memory without any intervention of the processor.

So, this for that purpose the processor 8086 needs to release the control of this address and data buses and for that we have got this hold and hold acknowledge lines. So, if a request comes on the hold signal hold line; that means, the processor will understand that some other, other master is asking for bus access. So, it will be giving this hold acknowledge, so it will release the control of the bus and it will be asserting this hold acknowledge line. So, that way these are the requesting processor will understand that this hold has been granted.

(Refer Slide Time: 13:20)

Maximum mode signals

During maximum mode operation, the MN/ $\overline{M_X}$ is grounded (logic low)

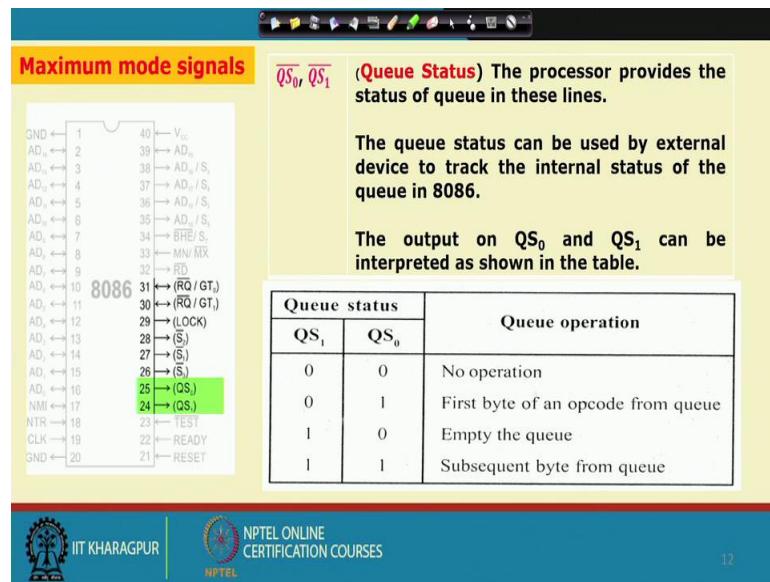
$\overline{S_0}, \overline{S_1}, \overline{S_2}$ Status signals; used by the 8086 bus controller to generate bus timing and control signals. These are decoded as shown.

Status Signal			Machine Cycle
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

On the other hand for the maximum mode of operation, so this min max bar line has to be grounded and this status signals S0, S1 and S2 they are used by 8086 bus controller to generate bus timing control signals. These, so the decoding is like this if it is 0 0 0 so, it is interrupt acknowledge, 0 0 1 so this read IO port. So, like that so that way this is actually giving some information about what is the operation that this 8086 is doing now, so this status line gives that indication.

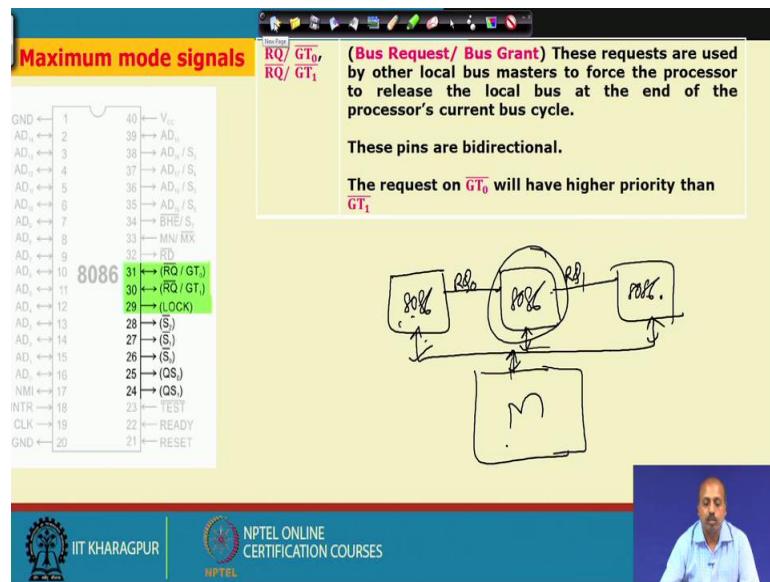
(Refer Slide Time: 13:56)



There are 2 mode signals $\overline{QS_0}$ and $\overline{QS_1}$ so they are queue status. So, we will come to this queues slightly later, so what happens is that internally 8086 process possesses one instruction queue and in that instruction queue, all the instructions when they are fetched from the second from the memory so they are kept ok. So, that speeds up the operation of this processor so we will see that.

So, what is the status of that queue, so that is actually indicated by these QS0 QS1 line. So, if it is 0 0 means queue is no operation is being done on the queue, if it is 0 1. So, it is a first byte of an opcode from queue has been taken 1 0 the queue is empty and 1 1 it is subsequent byte from the queue. So, these are the various queue operations that are taking place, so this is the processor gives this indication to the outside. So, the external device can track the internal status of this 8086 queue by means of these lines.

(Refer Slide Time: 15:04)



Then this RQ and GT lines, so bus request and grant bus grant lines. So, these are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. So, just like in minimum mode we had the hold and hold acknowledge line, in maximum mode we have got this request and grant lines.

So, any processor which is asking for this bus access so it will give a request on this request line and getting this requests if the processor will finish the current bus cycle. So, whatever it was doing so that cycle is finished and then the bus is released. So, when the bus is released then this corresponding grant signal is given.

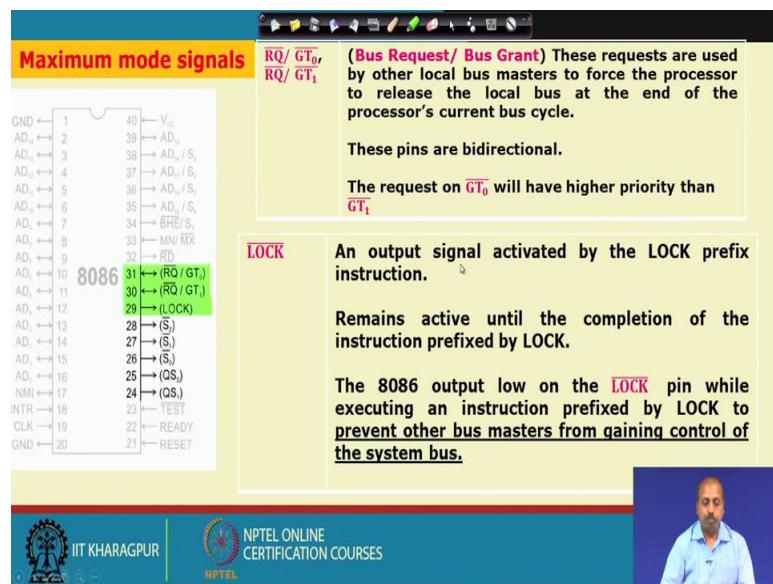
So, we have we have got 2 lines RQ GT 0 and RQ GT 1. Out of this two GT 0 has got higher priority than the GT 1. So, if we have got 1 device connected to GT 0 another device connected to GT 1, so GT 0 device will have more priority. So, in this way I can connect a more than 1 processor. Like it may so, happen that we design a system like this, so we have got three 8086 processors, three 8086 processors and then they are all accessing the they are all accessing the same memory, the memory part is same and they are feeding the they are controlling this memory by means of this address and data lines so it is like this.

Now, whenever any processor needs to access the memory, so it has to tell the other processors that I need to access it. So, it will it may be this is connected over this RQ 0 line and this is connected over RQ 1 line. So, if this is the main master then when this fellow needs to use the bus, so when RQ 0 line it will send a request to this accordingly if

it is granted then this processor will be allowed to access this memory this bus will be controlled by this first processor.

On the other hand if this device needs this processor needs to use the bus then it will send a request on this RQ 1 line and if it is granted then it will be using the bus by this line. So, this is how this request and grant lines will work just like this hold and hold acknowledge in case of minimum mode of operation.

(Refer Slide Time: 17:32)

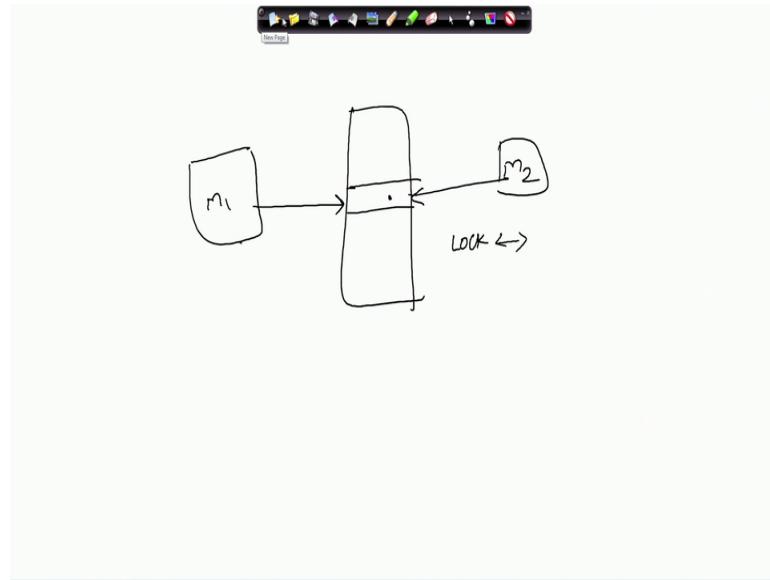


So, there is another pin which is known as *lock* pin, so this is an output signal activated by lock prefix instructions. So, lock prefix instruction means that we will see that there are some instructions in 8086 where you can put a lock prefix. So, you can, you can write lock in front of them. So, it means that it will remain active until the completion of the instruction prefix by lock. So, this 8086 output low on the *lock* pin while executing instructions prefix by lock to prevent other bus masters from gaining control of the system bus.

Like if suppose in the previous example that we took, so we had three 8086 processors. So, when one processor is trying to request for the bus from another processors, so first instead of sending the request directly it may check the *lock* pin of the other processor. So, if that *lock* is 0 that means, the processor is executing some lock type of instruction, where, lock prefixed instruction, where it will not release the bus because then

the memory content may become inconsistent. The reason is that if there is memory location like this say.

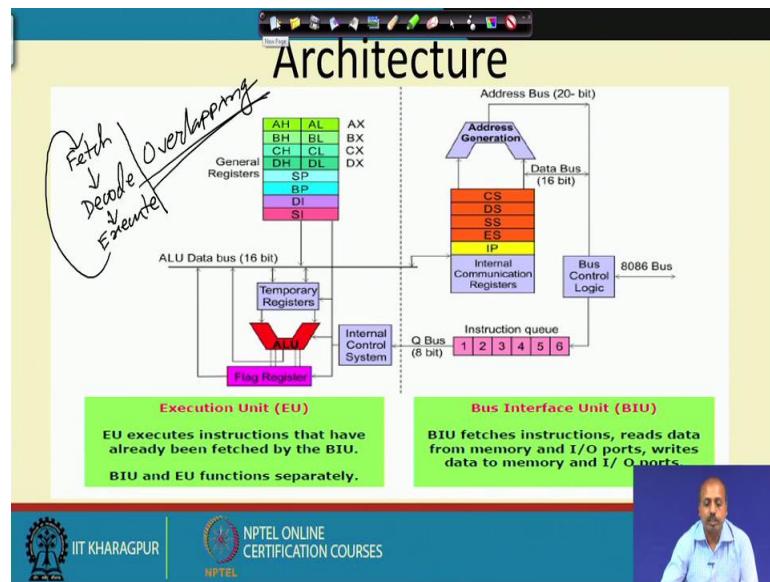
(Refer Slide Time: 18:54)



If this is say the memory and then there is 1 location and we have got 2 masters, master 1 and master 2 and both of them want to modify this location. Now, naturally so this is a serious condition because, if the modification is not done completely by 1 processor then other fellow may write something and the original content will be lost. So, that way lead that that may lead to some inconsistent step, so this is a very important problem when we consider the operating system design. So, what is done is that this updation, the writing on to this location.

So, that instruction is prefix by a lock part ok, so that as a result if M1 is executing this. So, this M2 while requesting for the memory; So, it will first see the lock line of this M1 and if this lock line is low that means, it will not be allowing to access the, to access the buses. So, it should not send the request lines. So, that is the purpose so we can use it for this lock type of instructions. So, next we will be looking into the internal architecture of 8086 and how does it differ from 8085.

(Refer Slide Time: 20:11)



So, internally this 8086 can be divided into 2 distinct modules, 1 is known as execution unit another is known as bus interface unit. So, in the execution unit is the portion where it actually does the execution of the instructions, so these actual operations are carried out in this part. So, we have got say in this part we have got a set of registers AX, BX, CX, DX, SP, BP, SI, DI, so these are few registers that we have here and out of this AX is A 16 bit register. So, all these registers are 16 bit registers so they are called general purpose registers, so, GPRS and this AX, BX, CX, DX, so they can be divided into 2 parts high and low like A high, A low, B high, B low like that.

This SI, DI, SP, BP, so, they are they are having some special purposes so we will come to that later. Then this is the ALU which will be doing the operation and there are temporary registers from which this ALU input is fed. So, this data bus is 16 bit data bus and this ALU is a 16 bit ALU. So, it can do the operation 16 bit operations so we can get the operands from these registers or you can get the operands from outside through this bus and that way it will be coming to this ALU.

So, there is a control system so that will be controlling the operation that this ALU does and the outcome of the ALU operation. So, that is by viewing in terms of some flag registers or flag bits may be set here, so that maybe so that maybe that may be checked by subsequent instructions and all.

So, this EU it will execute instructions that have already been fetched by the bus interface unit. So, if you remember 8085 so what happening is that one instruction is fetched then it is decoded then it is executed. So, it was going in a cycle like fetch, then decode and then execute and after the execution of the first instruction is over then only the second instruction is fetched, so it was going like this.

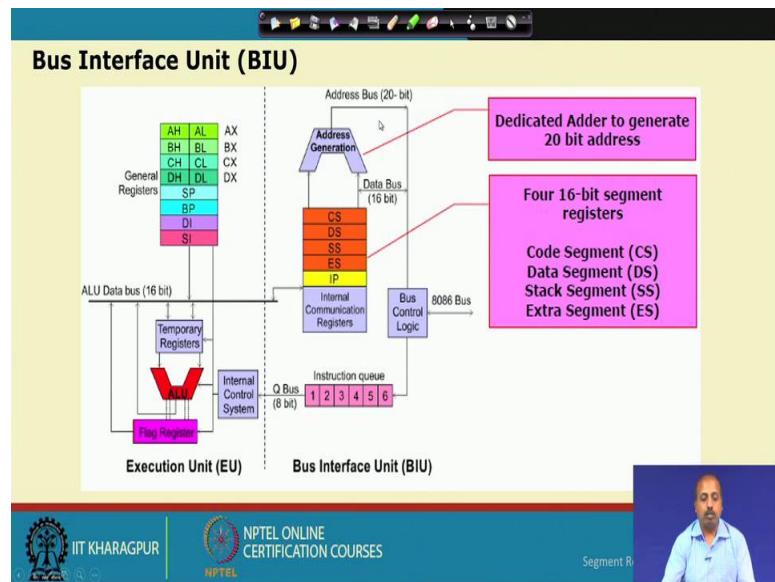
In case of 8086, so this structure is bit modified, so where there is this fetch part and decode parts. So, they are this fetch part particularly so that is separated into a different module. So, we have got a this bus interface unit, so which will fetch the instructions and data from the memory for I/O, I/O codes writing the data. So, they will be done by this bus interface unit and in this bus interface unit also. So, we have got a few registers CS, DS, ES, SS and the IP.

So, they are all 16 bit registers. There are some internal communication registers so that are that is internal not visible to the programmer there is an address generation unit. So, this so that takes two 16 bit values and performs a 20 bit addition. So, result of this ALU is a 20 bit ALU, so it will be producing a 20 bit output and the 20 bit address goes via this bus control logic to 8086 bus so and internally there is a queue there is in there is an instruction queue.

So, whenever we are doing some operation in this part, so these buses are this bus is free. So, at that time this bus interface unit it fetches subsequent instructions from the memory and once the instructions have arrived so they are kept on to this queue. So, this is a 6 byte queue. So, subsequent bytes are fetched from the memory and they are kept into this queue.

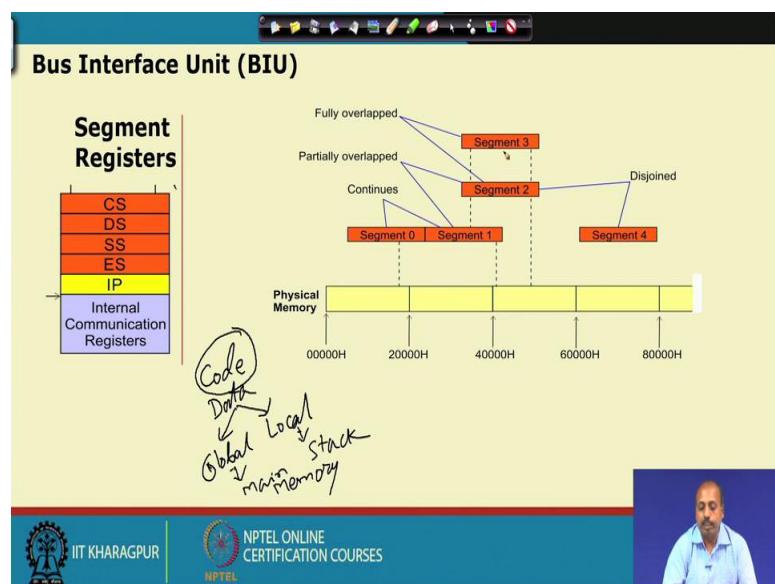
So, this helps in some amount of overlapping of this fetch, decode, execute phase, so we have got some overlapped execution of. So, it is it is a fetch, decode, execute overlapping. So, this is a very important concept that we will find in later processors also it started with 8086.

(Refer Slide Time: 24:18)



So, we next we will be looking into detail like what is going to happen. So, we will start with the bus interface unit. So, as I said that this address generation is done by a 20 bit address. So, this 20 bit ALU so this will generate 20 bit address and this CS, DS, ES, SS. So, they are all 16 bit registers so they are called segment registers CS is called the code segment register, DS is called data segment register, there SS is stack segment register and ES is extra segment register. So, we will be looking into detail of these segment registers and how are they generating the address.

(Refer Slide Time: 24:52)



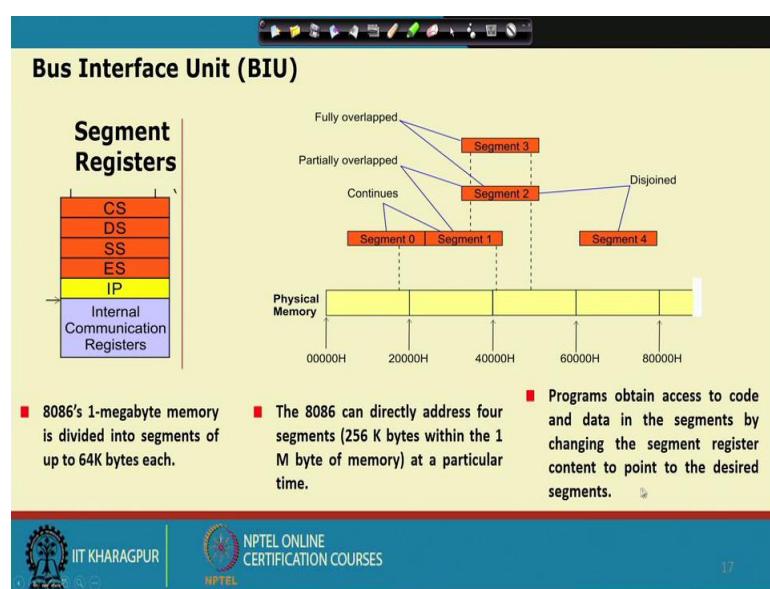
So, these are the segment registers CS, DS, SS and ES and then you can if you look into a code then any program. So, if you look into any program that program can be divided into the different parts, like we have got the code part, we have got the data part; which is this data can again be divided into global data and local data.

Now, for any program this code part is they are unchanged they remain same and this data the global data and local data. So, normally this local data is created onto the stack and this global data is put on to the main memory; all are part of main memory but they are so they are allocated at the time of this program loading itself this global data and local data is as you are making procedure calls this local data is filled up.

So, we have got this segment registers and you can divide this physical memory into a number of segments, like say this segments like say 0000 to 2000H, then 4 0. So, we are divided into a few segments now you can name this segments like say this is segment 0, this is segment 1 so that way, so this segment actually this is yeah. So, this is segment 0 and segment 1 so this is a continuous segment; then this segment 2 and 3.

So, that way we can think of the memory to be organized into a number of segments and once we have done that for this code may be located 16, in segment 0, data may be located in segment 3, stack may be located in segment 4. So, like that we can control in which portion of the memory which part of the program will be loaded.

(Refer Slide Time: 27:03)

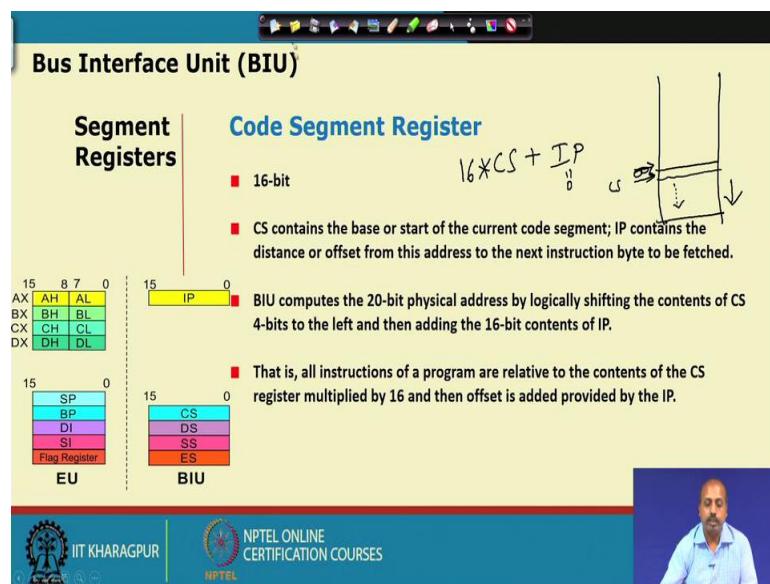


So, this is, this facility this segmenting facility is available in 8086 and so the total 1 megabyte of memory it is divided into segments of up to 64 bytes each and this it can directly access 4 segments of 64 kilobyte segment. So, these 256 kilobytes can be accessed by this any program 8086 at a particular time ok. So, how does it access?

So, program obtain access to code and data in the segments by changing the segment register content to point to the desired segments. So, if I say that this segment 0 will be corresponding to the code, then this CS register will be made to point segment 0. Similarly if I say that segment 4 will be the data segment, then this DS register is made to point to segment 4.

So, each segment is 64 kilobyte so I have got at most 4 segment. So, at any point of time so you can have total 256 kilobyte of space accessed by 8086 and of course you can, you change the content of this segment registers all the segment registers to have more number of segments. Like at after sometime if you find that DS should better point to segment 3, so you can change the content of DS so segment 3 becomes the data segment so that is possible.

(Refer Slide Time: 28:14)



So, this code segment register is 16 bit register, so this is it contains the base or start address of the current code segment and IP contains the distance or offset to from this address to the next instruction byte to be fetched. So, what happens is that this address when it is

formed, so this address is formed as, so this address is formed as $16 \times$ code segment value + instruction pointer value.

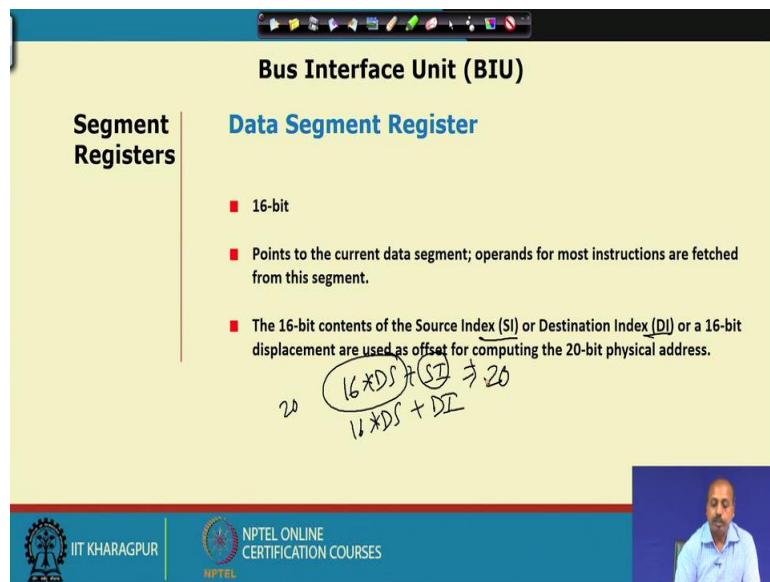
So, whatever, so if this is the memory and I have initialize the code segment to be equal to this and say the address is increasing in this form this is the code segment. So, CS register points to the first 1 so this value this location address is in CS register and the IP value is made equal to 0 say.

So, when I am accessing first location so the $16 \times$ CS + IP. So, it will be accessing this location then IP value will be incremented in subsequent accesses it will, so it will accesses these locations. So, this IP contains the offset and this CS contains the base value. So, that way this access will be done by the code segment register.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture – 64
8086 Microprocessor
(Contd.)

(Refer Slide Time: 00:17)



Bus Interface Unit (BIU)

Data Segment Register

- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

20 bits
(16 * DS + SI) > 20
16 * DS + DI

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For the data segment register, so we have got the DS register. So, that is also a 16 bit register and this data segment register, so this is used for accessing data and we have got this. For the offset part, so, these registers this SI and DI registers so they actually act as the offset part. So, here the calculation will be like $16 \times DS + SI$ or $16 \times DS + DI$. So, that way we can have this 20 bit address computed. So, when I say multi so this multiplied by 16 means it is left shifted by 4 bit. So, this result becomes 20 bit result and with that another 16 bit value is added, so overall a value is becoming 20 bit. So, this way we can access the data part.

(Refer Slide Time: 01:08)

Segment Registers

Stack Segment Register

- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as **PUSH** and **POP**.

$$16 \times SS + SP$$
$$16 \times SS + BP$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For stack part so it uses this stack segment register SS and the otherwise the calculation is same, so there are 2 registers which are dedicated for accessing the offset which is one is the stack pointer and another is the base pointer. So, both of them the calculation will be that $16 \times SS + SP$ or $16 \times SS + BP$, so that way it accesses the 20 bit address for the within the stack segment. So, and there is another segment register which is extra segment.

(Refer Slide Time: 01:52)

Segment Registers

Extra Segment Register

- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, many times what happens is that we have the 1 data segment is not sufficient, so we need to simultaneously access 2 data segments. So, for that purpose another data segment

has been provided by this ES register, extra segment register. So, here also operation can be done plus this ES DI pair, so that also helps in the string operation. So, we will see that 8086 has got a powerful string operation so, for this string operation this is going to be useful.

(Refer Slide Time: 02:23)

Bus Interface Unit (BIU)

Segment Registers

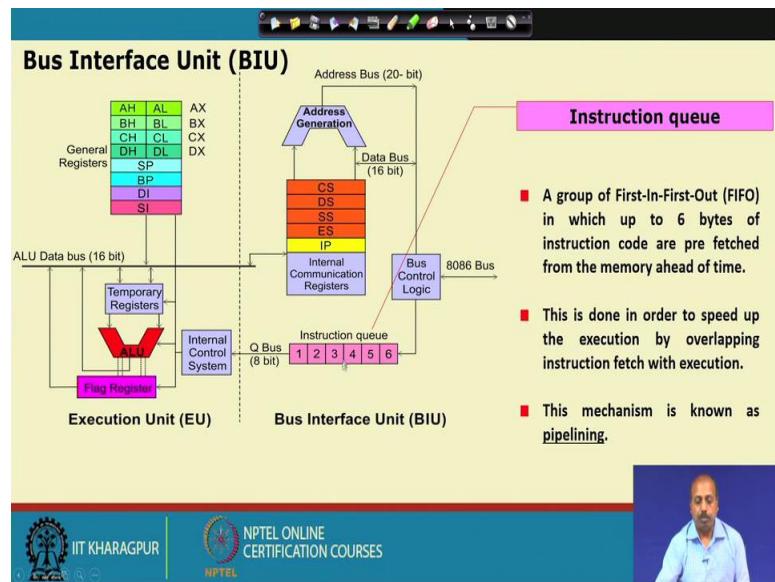
Instruction Pointer (IP)

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Then this instruction pointer so that is a 16 bit register and instruction pointer or IP. So, this is written as IP so this points to the next instruction to be executed within the current code segment. So, this IP value is a 16 bit value so, that will be incremented after each memory access and that way it will go. So, this $16 \times CS + IP$ so that way the next addresses will be calculated.

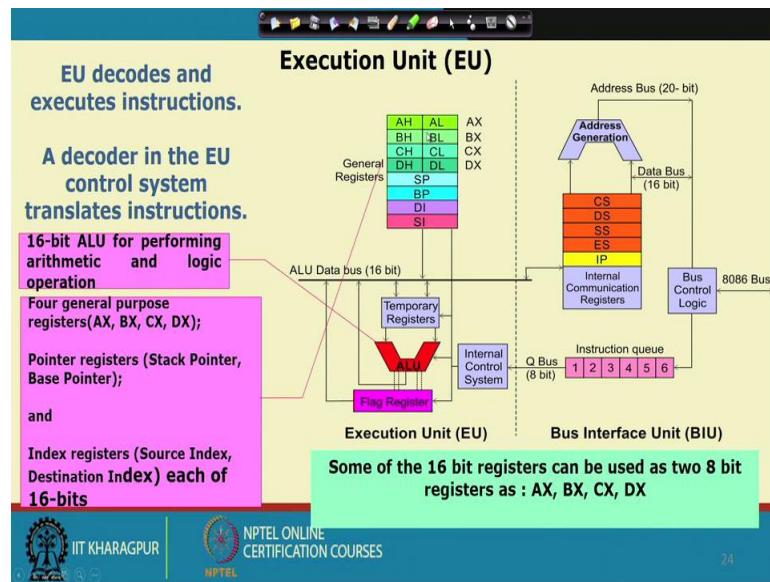
(Refer Slide Time: 02:53)



Next we will come to this instruction queue, so this is a group of first in first out in which we have got up to 6 bytes of instruction code are pre fetched from the memory ahead of time. So, as I said that whenever this bus is free, this bus interface unit so it will fetch the successive bytes from memory. So, we have got instructions in 8086 which varies from 1 byte to 6 bytes, so that way this 6 byte instruction queue has been provided, so that those instructions can be pre fetched.

So, this is done in order to speed up the execution of overlapping instruction fetch with execution. So, as I was telling over left phase execution and this mechanism is also known as pipelining. So, this execution fetch and execution they are pipelined.

(Refer Slide Time: 03:43)



Next we look into the execution unit part, so this execution unit it decodes and executes the instruction so that is the major responsibility. A decoder in the execution unit control the execution unit control system translates instruction, so this is this control system. So, this is translating the instructions into the control signals to see how this instructions can be executed.

There is the 16 bit ALU which perform the arithmetic and logic operations, 4 general purpose registers AX, BX, CX, DX, there are 2 pointer registers stack pointer and base pointer and there are 2 index registers SI and DI ok. So, these are the several registers that we have in the execution unit, some of the 16 bit registers can be used as two 8 bit registers like this AX can be taken as and AL two 8 bit registers, BX can be taken as BH BL.

(Refer Slide Time: 04:36)

EU Registers

Accumulator Register (AX)

- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | 25

So, like that so this AX is the accumulator, so just like 8085 we had A, so here we have got the AX register. It consists of two 8 bit registers AL and AH which can be combined together and used as a 16 bit register AX, AL in this case contains low order byte of the word and contains the higher order byte. The I/O instructions that use AX and AL for inputting or outputting 16 or 8 bit data to or from an I/O port. So, this is for I/O access and multiplication division instructions also use AX or AL.

(Refer Slide Time: 05:13)

EU Registers

Base Register (BX)

- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | 26

So, we have got multiplication division operation that use either AX or AL. Then there is another, the next register BX, so this is called base register so otherwise this is a general purpose register. So you can use it for other operations like so, holding some data operand performing addition subtraction operation, so for that you can hold it as some another operands there.

So, again BL is the lower order byte BH is the higher order byte, so this is the only general purpose register that can be used for addressing the 8086 memory. So, that is why it is called the base register, so particularly when you are accessing say array type of locations, so this BX register can be used as to hold the base address. So, we will see that later or memory references utilizing this register content for addressing use DS as the segment register. So, DS is the segment register and BS BX will be the offset, so like that it will be done.

(Refer Slide Time: 06:11)

EU Registers

CX < 10

START Loop START

Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as **SHIFT**, **ROTATE** and **LOOP** use the contents of CX as a counter.

Example:

The instruction **LOOP START** automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

If it is zero, 8086 executes the next instruction; otherwise the 8086 branches to the label START.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Then there is a, another register CX this is also general purpose register which is given the special name counter register. So, the name comes from the fact that for some instructions like shift, rotate, loop etc, so we have to tell like how many bits you want to shift or rotate or for a loop instructions. So, 8086 allows you to repeat a block of statements in number of times. So, how many times that block will be repeated that is determined by the CX register content.

So, this will be that's why the name counter comes here, so this CX is called the counter register the instruction loop starts. So, these automatically decrement CX by 1 without affecting flags and we will check if CX equal to 0. So, these so loop start when whenever this instruction comes. So, the control start is actually a level. So, it will be going back to that start point and it will again start the execution of the loop. So, normally what we do is that if we want to make a loop, so we keep this body and. So, this is the level start and at this point we say loop start and before that the somehow the CX register is initialized to number of times we want to repeat the loop.

Suppose we want to repeat it 10 times, so it will be when it comes here. So, CX value will be decremented if it is nonzero, so control is automatically branch to this. So, we do not have to spend any more instruction for that. So, if it is 0 so it will be the 8086 will execute the next instruction otherwise it will branch a to the label start.

(Refer Slide Time: 07:46)

The slide is titled "Data Register (DX)" and is part of a section on "EU Registers". It contains the following text:

- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16×16 multiplication or the high 16-bit dividend (data) before a $32 \div 16$ division and the 16-bit remainder after division.

At the bottom, there are logos for IIT Kharagpur and NPTEL Online Certification Courses, along with a small video thumbnail of a person speaking.

Then there is a data register DX so this is this is for holding data. So, this does not have other special functions. So, it is DL, it can be divided into DL and DH it is used to hold the high 16 bit result in 16×16 by 16 multiplication or the high 16 bit dividend before a 32 by 16 division and the 16 bit reminder after a division. So, this is particularly used for this data operation that is why the name data register has come.

(Refer Slide Time: 08:16)

EU Registers

Stack Pointer (SP) and Base Pointer (BP)

- SP and BP are used to access data in the stack segment.
- SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.
- SP contents are automatically updated (incremented/decremented) due to execution of a POP or PUSH instruction.
- BP contains an offset address in the current SS, which is used by instructions utilizing the based addressing mode.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 29

Other two registers are stack pointer and base pointer, so this stack pointer is they are used to access both SP and BP they are used to access the stack segment. SP is used as an offset from the current stack segment during execution of instructions that involved stack segment in the external memory. So, basically the local variables the parameters that you pass, so for the accessing those this stack pointer is used. And then this stack pointer this SP, BP content so, they will automatically be updated are implemented or decremented for a POP or PUSH type of operations. And BP is a special register sometimes using stack pointer alone is not sufficient because stack pointer may get corrupted.

So, we if we want to access within the stack segment then this BP pointer may be used, so what I say is like this that is if this is the memory and maybe so this party is the stack so this whole thing is the stack. So, your stack segment register points to this and now to access the individual locations within this. So, one option is you can use the stack pointer to do that. But the stack pointer if it gets corrupted then this return address will be a problem so return address may get maybe modified.

So, what we do instead of that stack pointer, the base pointer register is used to get offsets within this and stack pointer is not touched, so that this procedure call return can walk properly but for accessing parameters and all we can use this base pointer. So, both of them will use stack segment as the base register.

(Refer Slide Time: 10:05)

The slide is titled "Source Index (SI) and Destination Index (DI)". On the left, there is a vertical box labeled "EU Registers". To the right of this box, under the title, are two bullet points:

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man speaking.

So, another pair of registers which are common in 8086 is a source index SI and destination index DI. So, they are used for indexed addressing and also for the string processing. So, this SI and DI purposes so we will see later when we go for this string operations.

(Refer Slide Time: 10:26)

The slide is titled "Flag Register". It shows a diagram of the 8086 flag register, which consists of 16 bits numbered 15 down to 0. The bits are grouped into several flags:

- Sign Flag**: Set when the result is negative.
- Zero Flag**: Set if the result of a computation or comparison is zero.
- Parity Flag**: Set to 1 if the lower byte of the result contains an even number of 1's; set to 0 if odd.
- Auxiliary Carry Flag**: Set if there is a carry from the lowest nibble during addition or borrow for the lowest nibble.
- Carry Flag**: Set if there is a carry out of the most significant bit (MSB) during addition or a borrow in subtraction.
- Overflow Flag (OF)**: Set if an overflow occurs during a signed operation.
- Direction Flag (DF)**: Used for string manipulation; if '0', processes from low to high; if '1', processes from high to low.
- Trap Flag (TF)**: If set, the processor enters single-step mode after each instruction.
- Interrupt Flag (IF)**: Causes the 8086 to recognize external maskable interrupts.
- Carry Flag (CF)**: Set by the processor during addition or subtraction.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES".

Flag registers, so there are a number of flag bits like this carry bit, so carry flag is there this. So, this carry then we have got parity, then there is auxiliary carry, then there is 0, then we have got this sign flag, then this trap flag. So, this is this is it can generate some internal interrupt, so this is mostly used for debugging type of operation. Then there is a

interrupt flag so it will cause it will cause the 8086 to recognize the external mask interrupts.

Then if you if you clear this IF, then it will disable all the interrupts, so that is the same thing that we have we say RIM SIM type of instructions in 8085. Then there is a DF flag with the direction. So, direction flag it will it will be clear when you go to the string instructions and there is an overflow. So, if an over flow occurs then this flag will be safe, so these are various flag registers that we have in 8086.

(Refer Slide Time: 11:23)

Sl.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, this overall registers so they can be classified into a number of groups like general purpose registers, we have got a 16 bit or 8 bit 16 bit AX, BX, CX, DX 8 bit AL, BL etc. We have got pointer registers SP and BP we have got index registers SI, DI instruction pointer IP segment registers we have got this CS, DS, ES, SS as 16 bit segment registers and there is a flag which is flag register which is again 16 bit.

(Refer Slide Time: 11:55)

Registers and Special Functions		
Register	Name of the Register	Special Function
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
AL	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
BX	Base register	Used to hold base value in base addressing mode to access memory data
CX	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
DX	Data Register	Used to hold data for multiplication and division operations
SP	Stack Pointer	Used to hold the offset address of top stack memory
BP	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
SI	Source Index	Used to hold index value of source operand (data) for string instructions
DI	Data Index	Used to hold the index value of destination operand (data) for string operations

So, this summarizes the operations, so we have already discussed about them. Next we come to the addressing mode.

(Refer Slide Time: 12:00)

Addressing Modes	
<ul style="list-style-type: none">■ Every instruction of a program has to operate on a data.■ The different ways in which a source operand is denoted in an instruction are known as addressing modes.	
1. Register Addressing	Group I : Addressing modes for register and immediate data
2. Immediate Addressing	
3. Direct Addressing	
4. Register Indirect Addressing	
5. Based Addressing	
6. Indexed Addressing	Group II : Addressing modes for memory data
7. Based Index Addressing	
8. String Addressing	
9. Direct I/O port Addressing	Group III : Addressing modes for I/O ports
10. Indirect I/O port Addressing	
11. Relative Addressing	Group IV : Relative Addressing mode
12. Implied Addressing	Group V : Implied Addressing mode

(Refer Slide Time: 12:08)

The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

The content of 8-bit register DH is moved to another 8-bit register CL

$(CL) \leftarrow (DH)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 35

So, there are different addressing modes, which are, which are possible way in 8086 like the first addressing mode is the register addressing mode. So, the instruction will specify the name of the register which will hold the data to be operated by the instruction like MOV CL, DH. So, the operands of the values are available in the registers only CL will get the content of DH. So, this is the content of 8 bit register DH is moved to another 8 bit register CL.

(Refer Slide Time: 12:33)

In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction

Example:

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

$(DL) \leftarrow 08_H$

MOV AX, 0A9FH

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX register

$(AX) \leftarrow 0A9F_H$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a immediate addressing when the source is an immediate 1 immediate value, like MOV DL, 08H. So, this 0 8 H this immediate value is moved to the DL register in 8085 we had MVI type of instructions. So, here we do not have MVI, so here it is MOV only. So, operand itself will identify that this is a, is an immediate operand. Then MOV AX, 0A9FH. So, this will be telling this is telling that we want to move this number to AX 16 bit number to AX register out of which this will get 0 A and AL will get 9F.

(Refer Slide Time: 13:13)

Addressing Modes : Memory Access

- 20 Address lines \Rightarrow 8086 can address up to $2^{20} = 1\text{M}$ bytes of memory
- However, the largest register is only 16 bits
- Physical Address will have to be calculated
Physical Address : Actual address of a byte in memory, i.e. the value which goes out onto the address bus.
- Memory Address represented in the form –
Seg : Offset (Eg - 89AB:F012)
- Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20-bit address

89AB : F012 → 89AB → 89AB0 (Paragraph to byte $\rightarrow 89AB \times 10 = 89AB0$)
 F012 → 0F012 (Offset is already in byte unit)
 + -----
 98AC2 (The absolute address)

38

Then for memory access there are 20 address lines, so 8086 can address up to 2^{20} 1 mega bytes of memory. However, the largest register is only 16 bit. So, physical address that is formed so that is calculated as actual address of a byte in the memory, that is a values which will go to the address bus. So, this is in the form segment colon offset so this is the segment and this is the offset part.

So, that means, so, it will be starting the segment register will contain the value 8 9 AB and the offset register will contain F012. And the calculation will be done like this; that this segment register value will be left shifted by 4 bits. So, in hexadecimal notation it becomes 89AB0 and with that this F012 will be added. So, this is the absolute address that is put on to the bus. So, as I was telling the value calculated is $16 \times$ segment register + offset $16 \times$ segment + offset, so that calculation is being done here.

(Refer Slide Time: 14:15)

The slide has a yellow background with a black header bar containing icons. The title 'Direct Addressing' is centered in a large, bold, black font. Below the title, a text box contains the following content:

Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

The effective address is just a 16-bit number written directly in the instruction.

Example:

MOV BX, [1354H]
MOV BL, [0400H]

$$BX \leftarrow (16 \times DS + 1354H)$$

The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

So, next will be looking into direct addressing, so in direct addressing the address is mentioned directly, so this MOV BX, 1354 H. So, this 1354 denotes the content contents of the location to be accessed. So, this is when executed the instruction will copy the content of the memory location into BX register and the actual operation the memory contain memory the BX will get the content of memory location $16 \times DS + 1354 H$.

So, content of this particular memory location will come to the BX register. Similarly, this BL 0400 so $16 \times DS + 0400$ so that will come, so we can this addressing mode is called direct because the displacement part of the operand from the segment base is specified directly in the instruction so, it is given here directly.

(Refer Slide Time: 15:18)

Register Indirect Addressing

In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers: BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example: MOV CX, [BX]

Operations:

$$\begin{aligned} EA &= (BX) \\ BA &= (DS) \times 16_{10} \\ MA &= BA + EA \end{aligned}$$

Note : Register/ memory enclosed in brackets refer to content of register/ memory

$$\begin{aligned} (CX) &\leftarrow (MA) \text{ or,} \\ (CL) &\leftarrow (MA) \\ (CH) &\leftarrow (MA + 1) \end{aligned}$$


IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Then register indirect addressing mode, so here with the offset part is contained in some register like say MOV CX, [BX]. So, BX content will be used as the offset, so this effective address is the content of the BX register and this is the memory will be this base address that is DS register multiplied by 16 plus this effective address. So, this way so this way we will be calculating in that total actual address value. So, in this instruction so CX will get the content of memory location MA which is computed, the address is computed here memory address or it is equivalent to this, so CL getting MA and CH getting MA + 1.

(Refer Slide Time: 16:04)

Based Addressing

In Based Addressing, BX or BP is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.

When BP holds the base value of EA, BP and SS is used.

Example:

MOV AX, [BX + 08H]

Operations:

$$\begin{aligned} 0008_H &\leftarrow 08_H \text{ (Sign extended)} \\ EA &= (BX) + 0008_H \\ BA &= (DS) \times 16_{10} \\ MA &= BA + EA \end{aligned}$$
$$\begin{aligned} (AX) &\leftarrow (MA) \text{ or,} \\ (AL) &\leftarrow (MA), \quad (AH) \leftarrow (MA + 1) \end{aligned}$$


IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Then there is a based addressing mode so where these base register BX can be used to give the offset part. So, this BX will hold this effective address which is this 20 bit physical and 20 bit physical address will be calculated from BX and DS like here. So, BX with this BX this 0 8 H will be added ok. So, this BX + 0008 H is added that gives the effective address and base address is DS \times 16 and the actual memory address is MA + BA + EA, so that is giving the memory address. So, then this AX register gets the content of memory MA or it is equivalent to this, so this way this base register BX is used for based addressing.

(Refer Slide Time: 16:51)

Indexed Addressing

SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:
MOV CX, [SI + 0A2H]
Operations:
 $FFA2H \leftarrow A2H$ (Sign extended)

$EA = (SI) + FFA2H$
 $BA = (DS) \times 16^{10}$
 $MA = BA + EA$
 $(CX) \leftarrow (MA)$ or,
 $(CL) \leftarrow (MA)$
 $(CH) \leftarrow (MA + 1)$

Then we have got indexed addressing, so in this indexed addressing this index register SI or DI they can be used to hold the index value of the memory location. So, this MOV CX, [SI + 0A2H], so this will be containing this the effective address will be SI + this offset. So, this 0A2H so when it is sign extended so it will become FFA2H, so that will get added and this will be multiplied base address will be multiplied by 16 DS multiplied by 16 will give the base address and memory address is calculated like this. So, this is it is sign extended to 16 bit before adding to the base value in case of indexed addressing.

(Refer Slide Time: 17:34)

Based Indexed Addressing

In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example:
MOV DX, [BX + SI + 0AH]

Operations:
 $000A_H \leftarrow 0A_H$ (Sign extended)

$EA = (BX) + (SI) + 000A_H$
 $BA = (DS) \times 16_{10}$
 $MA = BA + EA$

$(DX) \leftarrow (MA)$ or,
 $(DL) \leftarrow (MA), (DH) \leftarrow (MA + 1)$



Then we have got this based indexed addressing where the base register and this index register. So, both are used like MOV DX, [BX + SI + 0AH]. So, this is sign extended first of all this 0AH is sign extended, so it remains it become 000A and effective address is BX + SI + 0A and this base address is DS × 16 and actual memory address is BA + EA. So, then the content of this memory address is transferred to the DX register, that is the base addressing.

(Refer Slide Time: 18:07)

String Addressing

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.

Segment register for calculating base address of source data is DS and that of the destination data is ES

Example: MOVS BYTE

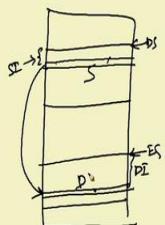
Operations:

Calculation of source memory location:
 $EA = (SI) \quad BA = (DS) \times 16_{10} \quad MA = BA + EA$

Calculation of destination memory location:
 $EA_E = (DI) \quad BA_E = (ES) \times 16_{10} \quad MA_E = BA_E + EA_E$

$(MA_E) \leftarrow (MA)$

If DF = 1, then $(SI) \leftarrow (SI) - 1$ and $(DI) = (DI) - 1$
If DF = 0, then $(SI) \leftarrow (SI) + 1$ and $(DI) = (DI) + 1$



Then there is one string addressing so this is employed in string operations to operate on string data. So, we have got in 8086 instructions called like MOVS type of instruction, so here we have got 1 example MOVS byte. So, here the operation that is done is when I say MOVS; that means, we are trying to move a byte from 1 memory location to another memory location, without involving the without involving a transfer to a processor register.

So, how is it done so this effective address, so for the source memory location; so this effective address is available in the source index register SI and DS contains the base of that. Actually the idea is like this, so, if this is your memory and then if this is so this maybe the source block S and this is the destination block D and we may be interested to transfer the content of this memory location from here to say here we want to transfer it to here.

Now, how to do that so DS register points to this, ES register points to this address and then this SI contains this offset. So, SI is SI contains this difference and DI contains this difference ok. So, when this MOVS instruction is executed first the content of this memory location is taken so $16 \times DS + SI$. So, that way gives the memory address source memory address and for the destination address, so this is $16 \times ES + DI$ so that is done. So, that becomes the destination address then this locations content is transferred to this location and then depending upon the direction flag setting.

So, DF is made equal to if DF is equal to 1 then this SI and DI are decremented by 1, if DF equal to 0 they are incremented by 1, SI and DI are incremented by 1. So, this is the string addressing so we will be looking into instructions later, but these instructions are useful like when you are trying to transfer a chunk of memory block from some memory locations to some other memory location this can be used.

(Refer Slide Time: 20:37)

The slide has a yellow header bar with the title 'I/O Port Addressing'. Below it, a text box contains information about addressing modes. It includes examples for direct and indirect port addressing, operations, and content moved. The footer features logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

I/O Port Addressing

These addressing modes are used to access data from standard I/O mapped devices or ports.

In **direct port addressing mode**, an 8-bit port address is directly specified in the instruction.

Example: IN AL, [09H]
Operations: PORT_{addr} = 09_H
(AL) ← (PORT) Content of port with address 09_H is moved to AL register

In **indirect port addressing mode**, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in the DX register.

Example: OUT [DX], AX
Operations: PORT_{addr} = (DX)
(PORT) ← (AX)

Content of AX is moved to port whose address is specified by DX register.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

For I O port addressing so there are 3 addressing these addressing modes can I access standard I O mapped devices or ports. So, you can have IN AL, 09H. So, this is a 8 bit port address, so this is the direct port addressing. So, you can mention one 8 bit port address like here 09 is the port address. So, this AL gets the content of the port whose address is 09 H.

And in indirect port addressing mode, so, this port address is 16 bit address and that 16 bit address should be available in some register. So, for example, it may be 16 bit port address so it is stored in the DX register. In that case, so we can say like OUT DX, AX so that means, then the content of this AX register will be outputted to the port whose address is available in the DX register ok. So, this way we can go for this I O port addressing.

(Refer Slide Time: 21:36)

Relative Addressing

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

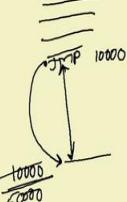
Operations:

$000A_H \leftarrow 0A_H$ (sign extend)

If ZF = 1, then

$EA = (IP) + 000A_H$
 $BA = (CS) \times 16_{10}$
 $MA = BA + EA$

If ZF = 1, then the program control jumps to new address calculated above.
If ZF = 0, then next instruction of the program is executed.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Then there is a relative addressing so in this addressing mode the effective address of a program instruction is specified relative to the instruction pointer by an 8 bit signed displacement. So, what is done suppose so this is in relative addressing is in relation to the jumping within the program, like if we want to jump from say, suppose we have written a program, where we have got we have got these statements and at this point I from this point I want to jump to this address ok.

So, then one possibility this address is say 10000, so I can directly say here jump 10000, but the problem with this type of a concept is that if it is next time the program is loaded from a different address. Then this 10000 may not the address may not be valid; maybe this should actually be say 50000 if the program is loaded from a different address maybe this address should be 50000.

So, just to avoid that situation so in case of relative jump what we do we tell that distance between these 2 points, like here so we say there is a jump on 0 if the 0 flag is set then it will jump to 0A H. But, what is actually done is with the current instruction pointer value this 0A H will be added and when this addition is done then when this addition is done.

So, we have got this new address calculated and then wherever the program is loaded this distance between these 2 points always remain like 0A H ok. So, that way this distance is unchanged so we do not have any problem there. So, it is irrespectively it is it is not dependent on the address at which the program is loading. So, there in case of 8086 we

have got this relative addressing and this relative addressing will help us in making the program relocated.

(Refer Slide Time: 23:39)

The slide has a yellow background. At the top, the title 'Implied Addressing' is centered in a large black font. Below the title, a text box contains the following information:
Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.
Example: CLC
This clears the carry flag to zero.

At the bottom, there is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it shows the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a light blue shirt speaking.

Then there are implied addressing where you do not need to tell the operand like this CLC. So, clear carry so clear carry means the carry flag will be cleared. So, the no other operand need to be specified because it is directly specified in the carry flag.

(Refer Slide Time: 23:58)

The slide has a yellow background. At the top, the title 'Instruction Set' is centered in a large black font. Below the title, a text box contains the following information:
8086 supports 6 types of instructions.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

At the bottom, there is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it shows the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a light blue shirt speaking.

Next we will looking into the instruction set. There are 6 types of instructions supported in 8086, data transfer instruction arithmetic instruction logical instruction string manipulation process control and control transfer.

(Refer Slide Time: 24:11)

8086 Microprocessor

Data Transfer Instructions

Instructions that are used to transfer data/ address in to registers, memory locations and I/O ports.

Generally involve two operands: Source operand and Destination operand of the same size.

Source: Register or a memory location or an immediate data

Destination : Register or a memory location.

The size should be a either a byte or a word.

A 8-bit data can only be moved to 8-bit register/ memory and a 16-bit data can be moved to 16-bit register/ memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, data transfer instructions, so the instructions that are used to transfer data or address into registers memory locations and I O ports, so they come under this data transfer instruction. So, we have so whenever we have got data transfer we need to tell the source and the destination; so, the source operand and destination operand so, they are of same size. So, source may be register or memory location or an immediate data destination has to be a register or a memory location, so naturally immediate cannot be the destination.

So, we have to have some memory location or a register and we cannot have both the operands as memory. So, it is not possible that both the operands of this data transfer instruction they are memory except in that string type of operation, so it is not possible. Size should be either a byte or a word and the 8 bit data can be moved only to 8 bit register or memory say and 16 bit data can be moved to only 16 bit register or memory.

(Refer Slide Time: 25:10)

Data Transfer Instructions	
Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT ...	
MOV reg2/ mem, reg1/ mem MOV reg2, reg1 MOV mem, reg1 MOV reg2, mem	$(reg2) \leftarrow (reg1)$ $(mem) \leftarrow (reg1)$ $(reg2) \leftarrow (mem)$
MOV reg/ mem, data MOV reg, data MOV mem, data	$(reg) \leftarrow data$ $(mem) \leftarrow data$
XCHG reg2/ mem, reg1 XCHG reg2, reg1 XCHG mem, reg1	$(reg2) \leftrightarrow (reg1)$ $(mem) \leftrightarrow (reg1)$



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



So, here we have got a few examples, like these are the common mnemonics that we have with the data transfer instructions MOV, XCHG, PUSH, POP, IN OUT etcetera; like we can have say MOV register 2, register 1, so register to MOV AX, BX like that MOV memory, register 1. So, we can specify some memory address and the register 1 content is moved there or this is from this memory location the value is loaded on to this register.

Then we have got this register this is for the immediate part. So, we have got some immediate data may be mentioned and that goes to register or that goes to memory location then we have got exchange. So, exchange means these two are these two contents are exchanged, so register 2 and register 1 are exchanged or this memory and register 1 are exchanged, so these are the MOV instructions.

(Refer Slide Time: 26:01)

The slide is titled "Data Transfer Instructions" and includes the mnemonic "MOV, XCHG, PUSH, POP, IN, OUT ...". It is divided into two main sections: "PUSH reg16/ mem" (pink background) and "POP reg16/ mem" (light green background).
PUSH reg16/ mem:
- **PUSH reg16:** $(SP) \leftarrow (SP) - 2$, $MA_s = (SS) \times 16_{10} + SP$, $(MA_s; MA_s + 1) \leftarrow (reg16)$.
- **PUSH mem:** $(SP) \leftarrow (SP) - 2$, $MA_s = (SS) \times 16_{10} + SP$, $(MA_s; MA_s + 1) \leftarrow (mem)$.
POP reg16/ mem:
- **POP reg16:** $MA_s = (SS) \times 16_{10} + SP$, $(reg16) \leftarrow (MA_s; MA_s + 1)$, $(SP) \leftarrow (SP) + 2$.
- **POP mem:** $MA_s = (SS) \times 16_{10} + SP$, $(mem) \leftarrow (MA_s; MA_s + 1)$, $(SP) \leftarrow (SP) + 2$.
The slide also features the IIT Kharagpur logo, NPTEL logo, and a photo of a speaker in the bottom right corner.

Then the PUSH instruction so, we can have this PUSH register 16 bit register or PUSH memory location. So, if it is PUSH 16 bit register first the stack pointer is decremented address is calculated, so this is $16 \times SS + SP$ and then the this the 2 locations this location and the next location, so there it get the content of the 16 bit register. Similarly PUSH memory, so here also similar thing the content of this memory location and this next location total 16 bit location so that they move to stack. Similarly, the reverse of PUSH we have got POP instruction, so that can also work in conjunction is 16 bit register or memory.

(Refer Slide Time: 26:45)

The slide is titled "Data Transfer Instructions" and includes the mnemonic "MOV, XCHG, PUSH, POP, IN, OUT ...". It is divided into four quadrants:
Top Left (Pink): IN A, [DX]
- **IN AL, [DX]**: $PORT_{addr} = (DX)$, $(AL) \leftarrow (PORT)$.
- **IN AX, [DX]**: $PORT_{addr} = (DX)$, $(AX) \leftarrow (PORT)$.
Top Right (Light Green): OUT [DX], A
- **OUT [DX], AL**: $PORT_{addr} = (DX)$, $(AL) \leftarrow (PORT)$.
- **OUT [DX], AX**: $PORT_{addr} = (DX)$, $(AX) \leftarrow (PORT)$.
Bottom Left (Pink): IN A, addr8
- **IN AL, addr8**: $(AL) \leftarrow (addr8)$.
- **IN AX, addr8**: $(AX) \leftarrow (addr8)$.
Bottom Right (Light Green): OUT addr8, A
- **OUT addr8, AL**: $(addr8) \leftarrow (AL)$.
- **OUT addr8, AX**: $(addr8) \leftarrow (AX)$.
The slide also features the IIT Kharagpur logo, NPTEL logo, and a photo of a speaker in the bottom right corner.

Then this IO operation like in operation in instruction, so IN AL, DX; so, this is the DX register contain the 16 bit port address and that value comes to AL register. So, or you can move by 16 bit data can be moved, so by means of this AX ok. So, you can say like in AX, [DX]. So, that way the port 16 bit port value comes to the AX register or we can similarly we have got this if you are doing immediate addressing then this is the 8 bit address we can mention ok. So, this we have already seen similarly for the out instruction also we have got either 16 bit port or 8 bit port.

(Refer Slide Time: 27:26)

Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

ADD reg2/ mem, reg1/mem	$(reg2) \leftarrow (reg1) + (reg2)$ $(reg2) \leftarrow (reg2) + (mem)$ $(mem) \leftarrow (mem)+(reg1)$
ADD reg/mem, data	$(reg) \leftarrow (reg) + data$ $(mem) \leftarrow (mem)+data$
ADD A, data	$(AL) \leftarrow (AL) + data8$ $(AX) \leftarrow (AX) +data16$

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Then under the arithmetic instruction category so we have got all this instruction add, add with carry, subtract, subtract with borrow, increment, decrement, multiply, division compare etcetera. So, add is register to register 1 so this is register 2 will get register 1 + register 2 add register 2 memory, so register 2 get register 2 plus memory. So, like that we can have a number or different types of add operation. Note one thing that there is nothing like add memory 1, memory 2, so both operands cannot be memory 1 of them must be a register.

Then these add register memory, data, so here this is the immediate addition the ADI type of instruction that we had in 8085. So, this is this data value will be added to the register or here this data value will be added to the memory location, then add A, data, so this is again the 8 bit addition. So, this way we can say AL AX so AL sorry this is yeah this A, data, so this 8 bit data can be added to AL or 16 bit data can be added to AX.

(Refer Slide Time: 28:33)

The slide is titled "Arithmetic Instructions" and lists mnemonics for ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, and CMP. The content is organized into three main sections:

- ADC reg2/mem, reg1/mem**:
 - ADC reg2, reg1: $(reg2) \leftarrow (reg1) + (reg2) + CF$
 - ADC reg2, mem: $(reg2) \leftarrow (reg2) + (mem) + CF$
 - ADC mem, reg1: $(mem) \leftarrow (mem) + (reg1) + CF$
- ADC reg/mem, data**:
 - ADC reg, data: $(reg) \leftarrow (reg) + data + CF$
 - ADC mem, data: $(mem) \leftarrow (mem) + data + CF$
- ADDC A, data**:
 - ADD AL, data8: $(AL) \leftarrow (AL) + data8 + CF$
 - ADD AX, data16: $(AX) \leftarrow (AX) + data16 + CF$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a photo of a man speaking.

Then this ADC add with carry so we have got a different types of add with carry instructions here so we can do that. So, otherwise it is same as the add instruction.

Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 65
8086 Microprocessor (Contd.)

(Refer Slide Time: 00:16)

Arithmetic Instructions	
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP ...	
SUB reg2/ mem, reg1/mem	
SUB reg2, reg1 SUB reg2, mem SUB mem, reg1	$(reg2) \leftarrow (reg1) - (reg2)$ $(reg2) \leftarrow (reg2) - (mem)$ $(mem) \leftarrow (mem) - (reg1)$
SUB reg/mem, data	
SUB reg, data SUB mem, data	$(reg) \leftarrow (reg) - data$ $(mem) \leftarrow (mem) - data$
SUB A, data	
SUB AL, data8 SUB AX, data16	$(AL) \leftarrow (AL) - data8$ $(AX) \leftarrow (AX) - data16$



57

The subtract instruction so, we have got this SUB normal subtract and the subtract with borrow so, that will come as SBB. So, subtract is similar as ADD. So, this should be register 2 minus register 1, ok. Because that is the destination so, that comes first. And so, that way we have got the subtract operation. Next we have got the, subtract with borrow so; there here it is fine so, this should also be register 2 minus register 1, this is register 2 minus 1 minus the carry flag.

(Refer Slide Time: 01:17)

The slide is titled "Arithmetic Instructions" and lists mnemonics for ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, and CMP. The INC section is shown below:

INC reg/ mem	
INC reg8	$(reg8) \leftarrow (reg8) + 1$
INC reg16	$(reg16) \leftarrow (reg16) + 1$
INC mem	$(mem) \leftarrow (mem) + 1$

DEC reg/ mem	
DEC reg8	$(reg8) \leftarrow (reg8) - 1$
DEC reg16	$(reg16) \leftarrow (reg16) - 1$
DEC mem	$(mem) \leftarrow (mem) - 1$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a man speaking.

So otherwise they are same, otherwise it is similar to the ADD operation, then increment operation. So, there is you can increment an 8-bit register, or you can increment 16-bit register or you can increment a memory location. So, this INC instruction, similarly we have got decrement operation so, decrement a 8-bit register 16-bit register or memory location.

(Refer Slide Time: 01:34)

The slide is titled "Arithmetic Instructions" and lists mnemonics for ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, and CMP. The MUL section is shown below:

MUL reg/ mem	
MUL reg	<u>For byte</u> : $(AX) \leftarrow (AL) \times (reg8)$ <u>For word</u> : $(DX)(AX) \leftarrow (AX) \times (reg16)$
MUL mem	<u>For byte</u> : $(AX) \leftarrow (AL) \times (mem8)$ <u>For word</u> : $(DX)(AX) \leftarrow (AX) \times (mem16)$

IMUL reg/ mem	
IMUL reg	<u>For byte</u> : $(AX) \leftarrow (AL) \times (reg8)$ <u>For word</u> : $(DX)(AX) \leftarrow (AX) \times (reg16)$
IMUL mem	<u>For byte</u> : $(AX) \leftarrow (AX) \times (mem8)$ <u>For word</u> : $(DX)(AX) \leftarrow (AX) \times (mem16)$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a man speaking.

Multiply so, we have got a multiply and register can be specified. So, in this case, AX will get AL multiplied by register the 8-bit register. And for word operation DX : AX will get

$AX \times$ register 16. Actually what happens is that if you multiply two 8 bit values so, you get a 16-bit value. So, in multiply operation one of the operand is always the AX or AL register. So, if you are doing 8-bit multiplication, then it is assumed that one of the operands is in AL registers, the other one is in the register that is mentioned here.

And the result is 16 bit and this AX register will hold that 16-bit value. On the other hand, if you are doing 16-bit multiplication, then whatever register you mentioned so, that will be it will be assumed that AX is the other source operand. So, the so, this is 16-bit register content will be multiplied by AX, and the result will be a 32-bit content, and that will be stored in DX and AX registers. Then the DX will hold the higher order 16 bit, and the AX will hold the lower order 16 bit.

So, similarly we have got this integer multiplication and a division operation. So, that way also it is similar to that multiplication division.

(Refer Slide Time: 02:54)

The screenshot shows a presentation slide with the following content:

Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

DIV reg/ mem DIV reg	<u>For 16-bit :- 8-bit :</u> $(AL) \leftarrow (AX) \div (reg8)$ Quotient $(AH) \leftarrow (AX) MOD (reg8)$ Remainder
	<u>For 32-bit :- 16-bit :</u> $(AX) \leftarrow (DX)(AX) \div (reg16)$ Quotient $(DX) \leftarrow (DX)(AX) MOD (reg16)$ Remainder
DIV mem	<u>For 16-bit :- 8-bit :</u> $(AL) \leftarrow (AX) \div (mem8)$ Quotient $(AH) \leftarrow (AX) MOD (mem8)$ Remainder
	<u>For 32-bit :- 16-bit :</u> $(AX) \leftarrow (DX)(AX) \div (mem16)$ Quotient $(DX) \leftarrow (DX)(AX) MOD (mem16)$ Remainder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Then there is a DIV instruction, so there assumption is that AX divided by the register, 8-bit register.

So, that will be done so, that will come to AL and will hold the remainder part so, quotient part will go to AL. So, AX is the dividend and register is the divisor and so, when this division is done quotient will be kept in the AL register and that remainder will go to the AH register, remainder will go to AH register. For 32-bit operation DX AX pair so, it is

expected to hold the dividend, that will be divided by the 16-bit register the divisor kept in this 16-bit register.

And after division AX gets the quotient, and DX gets the remainder. So, this way this division operation will be carried out.

(Refer Slide Time: 03:47)

Arithmetic Instructions
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

IDIV reg/mem IDIV reg	For 16-bit :- 8-bit : $(AL) \leftarrow (AX) \text{ :- } (\text{reg}8)$ Quotient $(AH) \leftarrow (AX) \text{ MOD } (\text{reg}8)$ Remainder
IDIV mem	For 32-bit :- 16-bit : $(AX) \leftarrow (DX)(AX) \text{ :- } (\text{reg}16)$ Quotient $(DX) \leftarrow (DX)(AX) \text{ MOD } (\text{reg}16)$ Remainder
	For 16-bit :- 8-bit : $(AL) \leftarrow (AX) \text{ :- } (\text{mem}8)$ Quotient $(AH) \leftarrow (AX) \text{ MOD } (\text{mem}8)$ Remainder
	For 32-bit :- 16-bit : $(AX) \leftarrow (DX)(AX) \text{ :- } (\text{mem}16)$ Quotient $(DX) \leftarrow (DX)(AX) \text{ MOD } (\text{mem}16)$ Remainder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We have similarly we have got this IDIV instruction.

(Refer Slide Time: 03:52)

Arithmetic Instructions
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

CMP reg2/mem, reg1/mem CMP reg2, reg1	Modify flags $\leftarrow (reg2) - (reg1)$ If $(reg2) > (reg1)$ then CF=0, ZF=0, SF=0 If $(reg2) < (reg1)$ then CF=1, ZF=0, SF=1 If $(reg2) = (reg1)$ then CF=0, ZF=1, SF=0
CMP reg2, mem	Modify flags $\leftarrow (reg2) - (\text{mem})$ If $(reg2) > (\text{mem})$ then CF=0, ZF=0, SF=0 If $(reg2) < (\text{mem})$ then CF=1, ZF=0, SF=1 If $(reg2) = (\text{mem})$ then CF=0, ZF=1, SF=0
CMP mem, reg1	Modify flags $\leftarrow (\text{mem}) - (reg1)$ If $(\text{mem}) > (reg1)$ then CF=0, ZF=0, SF=0 If $(\text{mem}) < (reg1)$ then CF=1, ZF=0, SF=1 If $(\text{mem}) = (reg1)$ then CF=0, ZF=1, SF=0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a comparison instruction. So, this comparison so, it will compare the operand like register 2, register 1 so, we will compare between them. So, if register 2 is greater than register 1, then all this is the this will be the setting CF equal to 0, ZF is equal to 0, SF is equal to 0.

And for different if register 2 is less than register 1 then the carry flag will be 1, ZF is 0 and SF is 1. So, this way this comparison will take place.

(Refer Slide Time: 04:19)

Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

CMP reg/mem, data	
CMP reg, data	Modify flags \leftarrow (reg) - (data) If (reg) > data then CF=0, ZF=0, SF=0 If (reg) < data then CF=1, ZF=0, SF=1 If (reg) = data then CF=0, ZF=1, SF=0
CMP mem, data	Modify flags \leftarrow (mem) - (mem) If (mem) > data then CF=0, ZF=0, SF=0 If (mem) < data then CF=1, ZF=0, SF=1 If (mem) = data then CF=0, ZF=1, SF=0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then other arithmetic operation like if register is greater than data, then this will be the setting, these are the various settings when we have got this comparison instruction. Logical instructions AND, OR, XOR, TEST, shift right, shift left etcetera.

(Refer Slide Time: 04:32)

The slide title is "Logical Instructions" and the mnemonics listed are AND, OR, XOR, TEST, SHR, SHL, RCR, RCL. The table shows the following entries:

AND A, data	
AND AL, data8	$(AL) \leftarrow (AL) \& \text{data8}$
AND AX, data16	$(AX) \leftarrow (AX) \& \text{data16}$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a man speaking.

So, AND is like just like we had got and in 8085. So, and A, data so, this is I can have 8-bit data or 16-bit data. So, with AX register they will be ANDed.

(Refer Slide Time: 04:48)

The slide title is "Logical Instructions" and the mnemonics listed are AND, OR, XOR, TEST, SHR, SHL, RCR, RCL. The table shows the following entries:

OR reg2/mem, reg1/mem	
OR reg2, reg1	$(reg2) \leftarrow (reg2) (reg1)$
OR reg2, mem	$(reg2) \leftarrow (reg2) (\text{mem})$
OR mem, reg1	$(\text{mem}) \leftarrow (\text{mem}) (reg1)$
OR reg/mem, data	
OR reg, data	$(reg) \leftarrow (reg) \text{data}$
OR mem, data	$(\text{mem}) \leftarrow (\text{mem}) \text{data}$
OR A, data	
OR AL, data8	$(AL) \leftarrow (AL) \text{data8}$
OR AX, data16	$(AX) \leftarrow (AX) \text{data16}$

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a man speaking.

Or we can have OR instruction, again the same thing. So, OR will be done and register 2 will get register 2 OR register 1, so in this case register 2 will get a register 2 OR memory. So, like that the OR operation will take place, and then we have got the string manipulation instructions.

(Refer Slide Time: 05:04)

The slide has a yellow background and a blue header bar. The title 'String Manipulation Instructions' is at the top. Below it is a bulleted list of points. A small note 'S represents string, SB string byte and SW string word.' is placed next to the point about string endings. At the bottom, there are logos for IIT Kharagpur and NPTEL, and a photo of a man in a blue shirt.

- ❑ String : Sequence of bytes or words
- ❑ 8086 instruction set includes instruction for string movement, comparison, scan, load and store.
- ❑ REP instruction prefix : used to repeat execution of string instructions
- ❑ String instructions end with S or SB or SW. S represents string, SB string byte and SW string word.
- ❑ Offset or effective address of the source operand is stored in SI register and that of the destination operand is stored in DI register.
- ❑ Depending on the status of DF, SI and DI registers are automatically updated.
- ❑ DF = 0 \Rightarrow SI and DI are incremented by 1 for byte and 2 for word.
- ❑ DF = 1 \Rightarrow SI and DI are decremented by 1 for byte and 2 for word.

So, string is nothing but a sequence of bytes or words. So, we can think of it as a sequence of bytes stored in the memory. So, 8086 instruction set includes instructions for string movement comparison scan, load and store. There is one REP instruction prefix so, it is used to repeat the string instructions. So, if you want to repeat it for some times. So, you can put it repeat prefix before that instruction. And how many times it will be repeated? So, that will be determined by the content of CX register.

So, string instructions end with S, SB or SW, where S represent the string SB is the string byte, and SW is the string word. Offset or effective address of the source operand is stored in the SI register, and destination is in the DI register. Depending upon the direction flag DF, SI, DI values will be updated. So, if DF is 0, a SI, DI will be incremented by one by one for byte and 2 for words.

So, if the instruction is ending with this B in that case, in that case it will be incremented by 1. If it is ending with W so, it will be incremented by 2. So, if it is if DF is equal to 0, on the other hand if DF is equal to 1, they will be decremented by 1 or 2. So, that way we have got these string value manipulation instructions. So, we will see in more detail some more instruction examples. And then this REP flag that we were talking about this REP prefix.

(Refer Slide Time: 06:36)

String Manipulation Instructions
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

REP	
REPZ/ REPE (Repeat CMPS or SCAS until ZF = 0)	While CX ≠ 0 and ZF = 1, repeat execution of string instruction and (CX) ← (CX) - 1
REPNZ/ REPNE (Repeat CMPS or SCAS until ZF = 1)	While CX ≠ 0 and ZF = 0, repeat execution of string instruction and (CX) ← (CX) - 1

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, this is repeat this CMPS or scan still ZF is equal to 0 so, 0 flag equal to 0. So, while CX is not equal to 0, and ZF is equal to 1, repeat execution of a string instruction and CX will be decremented by one every time. And then REPNZ so, if CX is not equal to 0 and DF equal to 0 repeat execution of string instructions, and CX will be made equal to CX is decremented it.

(Refer Slide Time: 07:20)

String Manipulation Instructions
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

MOVS	
MOVSB	MA = (DS) × 16 ₁₀ + (SI) MA _c = (ES) × 16 ₁₀ + (DI) (MA _c) ← (MA) If DF = 0, then (DI) ← (DI) + 1; (SI) ← (SI) + 1 If DF = 1, then (DI) ← (DI) - 1; (SI) ← (SI) - 1
MOVSW	MA = (DS) × 16 ₁₀ + (SI) MA _c = (ES) × 16 ₁₀ + (DI) (MA _c ; MA _c + 1) ← (MA; MA + 1) If DF = 0, then (DI) ← (DI) + 2; (SI) ← (SI) + 2 If DF = 1, then (DI) ← (DI) - 2; (SI) ← (SI) - 2

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

It is done till Z flag is not becomes equal to 1 as long as Z flag is 0, so, it will be continuing like that. So, this is the first instruction that we have moves B. So, moves byte. So, what it

does? If the first address is calculated the source address is calculated as $DS \times 16 + SI$. Destination address is calculated at as $ES \times 16 + DI$, then there this content is moved to this destination. And after that depending upon the DF flag setting DI and SI registers are updated. And MOVSW so, it is also similar to this only thing is that 2 bytes will be moved, and then depending upon the DF flag setting DI and SI will be incremented or decremented by 2.

(Refer Slide Time: 07:58)

String Manipulation Instructions

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Compare two string byte or string word

CMPS	
CMPSB	$MA = (DS) \times 16_{10} + (SI)$ $MA_t = (ES) \times 16_{10} + (DI)$ Modify flags $\leftarrow (MA) - (MA_t)$ If $(MA) > (MA_t)$, then $CF = 0; ZF = 0; SF = 0$ If $(MA) < (MA_t)$, then $CF = 1; ZF = 0; SF = 1$ If $(MA) = (MA_t)$, then $CF = 0; ZF = 1; SF = 0$ <u>For byte operation</u> If $DF = 0$, then $(DI) \leftarrow (DI) + 1; (SI) \leftarrow (SI) + 1$ If $DF = 1$, then $(DI) \leftarrow (DI) - 1; (SI) \leftarrow (SI) - 1$ <u>For word operation</u> If $DF = 0$, then $(DI) \leftarrow (DI) + 2; (SI) \leftarrow (SI) + 2$ If $DF = 1$, then $(DI) \leftarrow (DI) - 2; (SI) \leftarrow (SI) - 2$
CMPSW	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Then this CMPS instruction, the compare, so, compare we have got compare byte and compare word. So, otherwise it is similar to this MOVSB, but it is checking the, it is comparing between the strings. So, if this source string is larger, source byte is larger than the destination byte then CF will be made equal to 0. So, this will be the flag setting, otherwise this will be the flag setting so, it will go like this. Then we have got scan a string byte or word with accumulator.

(Refer Slide Time: 08:26)

String Manipulation Instructions
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS
Scan (compare) a string byte or word with accumulator

SCAS	
SCASB	$MA_e = (ES) \times 16_{10} + (DI)$ Modify flags < -(AL) - (MA _e) If (AL) > (MA _e), then CF = 0; ZF = 0; SF = 0 If (AL) < (MA _e), then CF = 1; ZF = 0; SF = 1 If (AL) = (MA _e), then CF = 0; ZF = 1; SF = 0 If DF = 0, then (DI) <- (DI) + 1 If DF = 1, then (DI) <- (DI) - 1
SCASW	$MA_e = (ES) \times 16_{10} + (DI)$ Modify flags < -(AL) - (MA _e) If (AX) > (MA _e ; MA _e + 1), then CF = 0; ZF = 0; SF = 0 If (AX) < (MA _e ; MA _e + 1), then CF = 1; ZF = 0; SF = 1 If (AX) = (MA _e ; MA _e + 1), then CF = 0; ZF = 1; SF = 0 If DF = 0, then (DI) <- (DI) + 2 If DF = 1, then (DI) <- (DI) - 2

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, it is basically comparison with the accumulator. So, it will compare AL value with this content of the memory location which is pointed to by ES : DI. And the other hand, if this is SCASW so, this will compare the word. So, it will be comparing AX register with the 2 successive memory addresses, which is pointed to by this ES : DI. And then DF will be depending upon the DF flag so, its DI value will be either incremented or decremented.

(Refer Slide Time: 09:05)

String Manipulation Instructions
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS
Load string byte in to AL or string word in to AX

LODS	
LODSB	$MA = (DS) \times 16_{10} + (SI)$ (AL) <- (MA) If DF = 0, then (SI) <- (SI) + 1 If DF = 1, then (SI) <- (SI) - 1
LODSW	$MA = (DS) \times 16_{10} + (SI)$ (AX) <- (MA ; MA + 1) If DF = 0, then (SI) <- (SI) + 2 If DF = 1, then (SI) <- (SI) - 2

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then we have got load so, it will load the string byte into AL or string word into AX. So, this LODSB so, it will be the source and the address is calculated as $DS \times 16 + SI$. And

this AL register gets the content of that memory location. If DF is 0 then this SI is incremented, if it is one, then SI is decremented. And this LODSW so, this will be moving one-word from a memory locations in to the AX register.

(Refer Slide Time: 09:35)

String Manipulation Instructions
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Store byte from AL or word from AX in to string

STOS	
STOSB	$MA_E = [ES] \times 16_{10} + [DI]$ $[MA_E] \leftarrow [AL]$ If DF = 0, then $[DI] \leftarrow [DI] + 1$ If DF = 1, then $[DI] \leftarrow [DI] - 1$
STOSW	$MA_E = [ES] \times 16_{10} + [DI]$ $[MA_E : MA_E + 1] \leftarrow [AX]$ If DF = 0, then $[DI] \leftarrow [DI] + 2$ If DF = 1, then $[DI] \leftarrow [DI] - 2$

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, just the reverse of these loads we have got STOS so, we have got STOSB and STOSW. So, this is a storing the content of AL register on to the memory address or it is called storing the content of a AX pair onto this two memory locations, MAE and MAE+1. So, that way this loading storing and loading will be done.

(Refer Slide Time: 10:02)

Processor Control Instructions

Mnemonics	Explanation
STC	Set CF $\leftarrow 1$
CLC	Clear CF $\leftarrow 0$
CMC	Complement carry CF $\leftarrow CF'$
STD	Set direction flag DF $\leftarrow 1$
CLD	Clear direction flag DF $\leftarrow 0$
STI	Set interrupt enable flag IF $\leftarrow 1$
CLI	Clear interrupt enable flag IF $\leftarrow 0$
NOP	No operation
HLT	Halt after interrupt is set
WAIT	Wait for TEST pin active
ESC opcode mem/ reg	Used to pass instruction to a coprocessor which shares the address and data bus with the 8086
LOCK	Lock bus during next instruction

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, then we have got a number of control instructions like this STC like set carry flag equal to 0. So, they are 0 operand instruction the CLC carry flag is 0, clear carry flag. CMC it will complement the carry, STD set it will set the direction flag like in the instruction.

So, you are telling if DF equal to 0, then this will happen, or this will happen. So, how to set the direction flag? So, that is given by this STD and CLD instructions, then we have got STI and CLI regarding interrupt flags enable and disable. So, we have got this NOP instruction NOP just like in 8085. So, you have got halt instruction for it is it is halt after the interrupt is set. And the WAIT, it is wait for the test pin to become active. So, as long as this test pin is high so, it will be remaining in the wait state, when the test pin will become low, then this condition is true so, it will be coming out.

Then there is an escape opcode so, that is used for the coprocessor. So, if you have an instruction with start with and starts with an escape opcode so, the 8086 will pass on the instruction to the coprocessor. And this lock prefix so, that will be lock the bus during the next instruction. So, this if you execute a lock instruction then in the next for the next instruction, so, bus will not be released by 8086 to others. So, next instruction will be executing in an uninterrupted fashion. So, this transfer control to a specific destination or target and they do not affect the flags. So, this control transfer so, we have got call instruction, return instruction and jump instruction.

(Refer Slide Time: 11:33)

Control Transfer Instructions

- Transfer the control to a specific destination or target instruction
- Do not affect flags

8086 Unconditional transfers

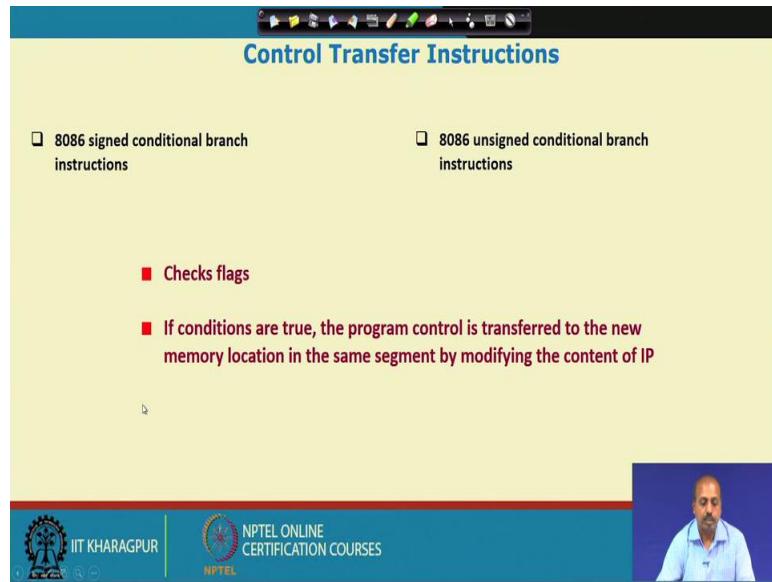
Mnemonics	Explanation
CALL reg/ mem/ disp16	Call subroutine
RET	Return from subroutine
JMP reg/ mem/ disp8/ disp16	Unconditional jump

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



So, call so, this is calling the subroutine. So, the address can be directly mention a 16-bit displacement a memory location address or a register. So, the register content will be used as the memory address. In this case, memory location content will be used as the target address. And here the address is specified directly. And similarly we have got the jump instruction.

(Refer Slide Time: 12:08)



Then in 8086 conditional branch instruction so, they will check the flags, and then if the condition is true the program control will be transferred to the new memory location in the same segment by modifying the content of the IP register.

(Refer Slide Time: 12:25)

Control Transfer Instructions	
<input type="checkbox"/> 8086 signed conditional branch instructions	
Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JG disp8 Jump if greater	JNLE disp8 Jump if not less or equal
JGE disp8 Jump if greater than or equal	JNL disp8 Jump if not less
JL disp8 Jump if less than	JNGE disp8 Jump if not greater than or equal
JLE disp8 Jump if less than or equal	JNG disp8 Jump if not greater
<input type="checkbox"/> 8086 unsigned conditional branch instructions	
Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JA disp8 Jump if above	JNBE disp8 Jump if not below or equal
JAE disp8 Jump if above or equal	JNB disp8 Jump if not below
JB disp8 Jump if below	JNAE disp8 Jump if not above or equal
JBE disp8 Jump if below or equal	JNA disp8 Jump if not above



So, this we have got JE so, JE displacement 8 jump if equal. So, if the equality condition is satisfied, then it is the 8-bit displacement specified. So, that will be added with IP and that will be making the 8 bit- jump. So, similarly we have got JNE or JNZ type of instruction. So, there also we have got this 8 bit, all this conditional branches so, they are related to the IP address with that to the IP value instruction pointer value. So, it will be jumping with respect to IP.

On the other end, this unconditional and unsigned conditional branch. So, we have got this JE displacement 8 and the things like that. So, this will be executing in a same fashion ok. So, we have so, this values may be unsigned, so, that is that is there.

(Refer Slide Time: 13:19)

The slide title is 'Control Transfer Instructions'. A bullet point says '8086 conditional branch instructions affecting individual flags'. A table has two columns: 'Mnemonics' and 'Explanation'. The rows are:

Mnemonics	Explanation
JC disp8	Jump if CF = 1
JNC disp8	Jump if CF = 0
JP disp8	Jump if PF = 1
JNP disp8	Jump if PF = 0
JO disp8	Jump if OF = 1
JNO disp8	Jump if OF = 0
JS disp8	Jump if SF = 1
JNS disp8	Jump if SF = 0
JZ disp8	Jump if result is zero, i.e., Z = 1
JNZ disp8	Jump if result is not zero, i.e., Z = 0

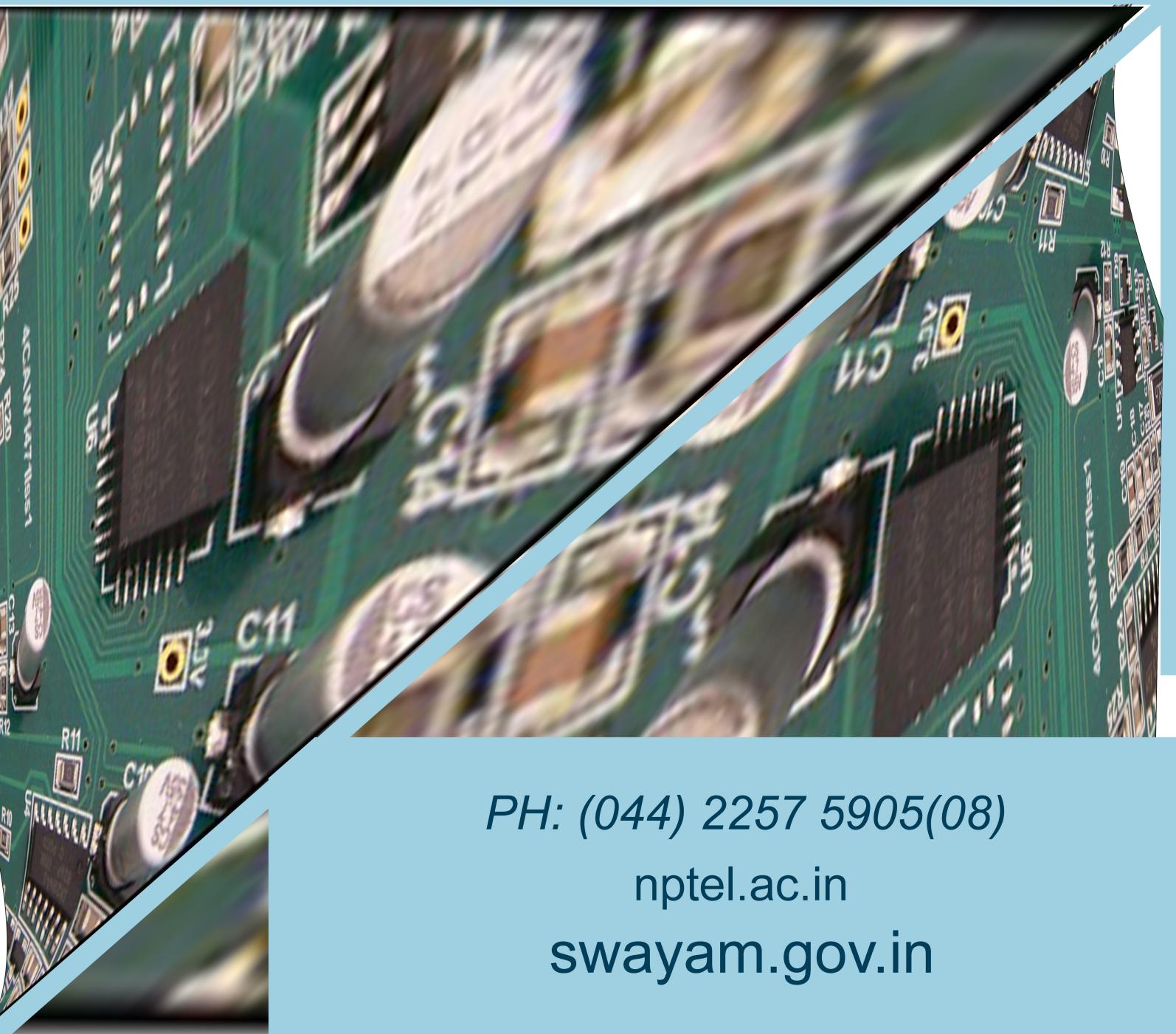
The footer includes the IIT Kharagpur logo, NPTEL logo, and text 'NPTEL ONLINE CERTIFICATION COURSES'. There is also a small video player window showing a man speaking.

So, conditional branch instruction that can affect flags like JC. So, jump if carry flag equal to 1 JNC jump if carry flag equal to 0 so, it will go like that. So, there are many such conditional branches that are available in 8086 that can be used for doing branching in that.

So, in this way in 8086 we find that we have got a much richer set of registers, we have got much richer set of instructions and operations that can be done as compared to 8085 and if you look into these further processors, then the structure will become more and more complex.

So, all these developments are done starting with digital circuits, where we know how to design individual logic modules or individual registers, individual combinational functions and all. And from there it is developed towards such complex systems. So, that gives a good idea like how we can design complex systems and what are the, what type of complex systems we can digital systems we can think about with using those basic gates and flip flops.

**THIS BOOK IS NOT FOR SALE
NOR COMMERCIAL USE**



*PH: (044) 2257 5905(08)
nptel.ac.in
swayam.gov.in*