Digital Circuits and Systems

Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture - 9
Code Converters

Having seen the techniques for reduction of Boolean functions by Boolean algebra as well as by using Karnaugh Maps we said we should use it in real circuit design that is where the object is the object of this is to exercise the program or the course as well as in real life in terms of digital design. The object is to design circuits which work according to specifications given. So we will look at a few circuits which are generally used in digital just to tell you how to translate a specification into a truth table and how to translate a truth table into a design using the Karnaugh Map.

In the beginning I said that there are2 types of digital circuits if you remember. One is called combinational circuits. A combinational circuit means circuits whose outputs depend on input of the current at the present time. The output depends on the input and if the input changes the output correspondingly change.

There is another class of circuits called sequential circuits. For this also the output depends on the input but not only on the inputs but also the previous behavior of the circuit. That means the output of the sequence circuit depend on the present inputs as well as the past outputs. So the techniques we have learnt so far the graphic the graph method and the Boolean algebra etc we will use for designing combinational circuits. Some of these generally used the commonly used digital circuits.

One class of circuits we will call this design of combinational logic circuits. One class of circuits in combinational logic is called a code converter. We represent data using codes, all of us know that. a simple binary representation of a digital number decimal number is also a code because I have only two symbols in binary 0 and 1 I need to represent a number which is larger than 1 so I need to use same0 and 1 in different patterns, that is a code. For example, 5 is written as 1 0 1 coding of the binary number into decimal number you call it conversion, decimal to binary conversion. But basically it is a code that represents a given decimal number. think of alphabets, when you write letters using computers, you send emails, you send SMS using cell ph1s, these are alphabets and special symbols, characters and so on.

Again the computer can only realize these are switches as I said, all the circuits inside are basically logic gates. I told you in the last class on how logic gates can be used for arithmetic operations. These are the 1s used for decision making, these are the 1s used for the control, all of them finally boil down to AND OR Inverter, I told you in the beginning AND OR Inverter are fundamental blocks fundamental gates by which you can design any circuit and I also said that can even be replaced by a whole set of NAND gates or NOR gates which are called universal gates.

That means when I have an English alphabet or any alphabet for that matter you want to represent with a computer you want to store it and process it and output it you need to represent these symbols of English characters or numerals or special symbols or punctuation marks by means of codes. A well known code for representing alphabets is called the ASCII code. ASCII stands for American Standard Code for Information Interchange. So anything is a code, any representation of binary pattern to represent a given data be it a decimal number be it a large number, be it a text, be it graphics also sometimes even graphics represent a binary number finally.

A television image you see or light intensity but a light intensity has to be represented as the intensity levels and intensity levels are translated as binary numbers. A larger binary number will be having more intensity. A larger intensity will be represented by a larger binary number and a lower intensity point will be represented by a smaller binary number. Therefore one class of circuits which are very commonly very standard are these code converters.

Sometimes we want to convert from one type of code to other type of code. some times for convenience, some times for security reasons you may want to have a code and you don't want somebody to understand so you have a key and only you know the key so you convert those numbers using that key to another set of numbers and use it. Only those people who are authorized will know the number will know the key and the rest of them will not. so that is one reason why you want to convert one type of code and number represented in one code to a certain numbers representing another code. Code conversion is required for many reasons. As I said one is for security reasons, we also want to do it for some hardware reasons. Sometimes by having a particular representation of 0s and 1s in a particular way the patterns of 0s and 1s falls in a particular pattern and the hardware implementation become simpler. We will see more of it later on.

(Refer Slide Time 9:22)

The class of codes which fall under this category for simplification of hardware are; one is called Excess - 3 code and the other is called the gray code. Excess - 3 code is a code which is used to convert a set of numbers to another set of numbers. By this extra Excess - 3 code a particular arithmetic operation becomes easier, the operation called complementing operation nine complement they say. We will see it later on when we talk about arithmetic circuits. So complementing of these numbers becomes easier if you represent those numbers in Excess - 3 codes. That is the reason why we want to go for Excess - 3 codes. There is no secret there.

You have decimal numbers represented in binary form and before you operate these numbers or before you use these numbers in arithmetic operation you convert them to Excess – 3 codes and you do this and then get it back into the binary code because the hardware for particular operations will be simpler if you do Excess - 3 coding. It is the same way with the gray code.

Gray code is used for reducing the switching activity they call it. Anytime we change one code to another code the0s are changed to be 1 and 1 to become0. Suppose I have a code I have a number 0 1 1 and 1 I have a four bit number a number with four binary digits with a 0 MSB0 1 1 and 1 that is 7 0 1 1 and 1 is a four bit representation of number 7 decimal, number 8 decimal is 1 0 0 0. Suppose I keep counting let us say count from 0 to 1, 2, 3, 4, 5, 6, 7 then 7 is indicated by 0 1 1 and 1 and my next count is 8, 1 0 0 0. Now all the four positions of the binary digits have to be changed from the previous values, first one has to become a 0, second one has to become 0, third one has to become 0, fourth 0 has to become 1 it is called switching, changing of the value of the binary from one value to another value is switching, we have just changed its value.

Why did I say switching? It is because we represent a binary operation by a switch. A closed switch is 1 and an open switch is 0 so a switch has to be opened to make a 1 a 0 and a switch has to be closed to make a 0 into 1. So four switches have to be closed or opened four switches have to be operated when I want to change a number from 7 to 8.
Suppose I need to switch only one switch at a time whatever be the magnitude of the number from one number to the next number if I have to make only one switching that reduces some activity. switching activity represents speed of operation, switching activity represents power consumption these are things which you have to take from me now but you can think of intuitively, you have to close and open more switches more energy is required and it is going to be less fast.

So when I say power saving speed improvement is possible by having a gray code representation. So in certain operations where this is important switching speed and power saving is important there also hardware minimization may become an important feature that may be a reason. The reasons may be many; code conversion is a standard operation in digital systems. Why it is done? As I said many reasons may be given for this. One is for the security reasons only you know the conversion nobody else knows the conversion so that may be a good enough reason it is called encryption.

Cryptography, cryptography is nothing but secret documents, then the second reason as I said is to reduce the hardware, some operations are possible with Excess - 3 codes by reduced hardware than by normal binary representation of decimal number. The decimal number has to be operated using computer or calculator or any digital circuit. I cannot feed the decimal number directly but I have convert it into binary and feed it. When I directly feed it as binary numbers it requires certain hardware, when I convert them to Excess - 3 and feed it I find certain operations are easier, certain operations require less hardware or less logic. That may be the reason for coding Excess – 3 as an example. The third thing is the switching activity and power saving, speed activity and all that which is the gray code.

These are only a few examples I am giving you. A representation of a number using 0s and 1s need not follow the sequence which are used to 0 is 0, 1 is 1, 2 is 1 0, 3 is 1 1, four is 1 0 0, 5 is 1 0 1 this called natural binary sequence normal binary sequence. Don't say that when I am using a computer always you need to follow the natural binary sequence. It is not necessary that you have to use a natural binary sequence for inputting numbers into a computer or any digital hardware. Into any digital hardware you can have a coded number perform the operation of the code in the converted code get the result in the converted code and reconvert it back to binary number with which you are familiar. That is why I thought I will give you some examples of code conversion how to design a code converter which is today's topic.

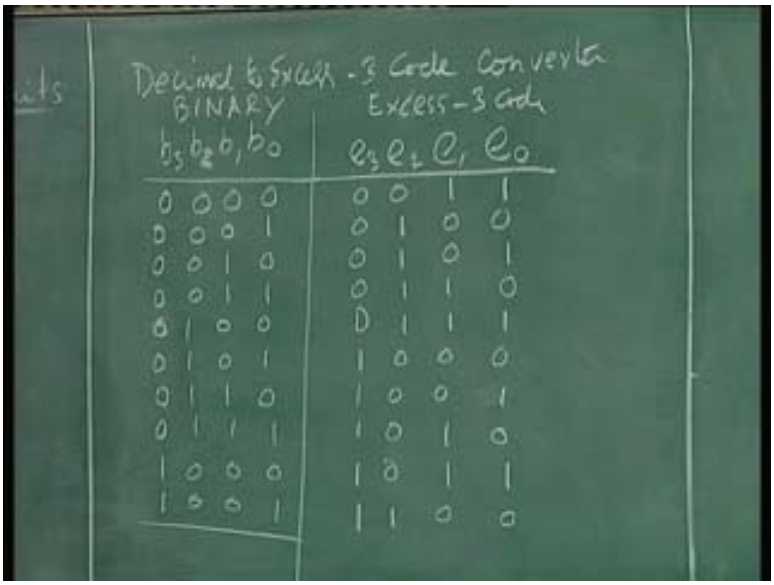
Combinational logic I want to give you practical circuits I told you last time. We only talked about techniques using all arbitrary sums of product and product of sum functions. I gave you min terms and max terms which are random whatever came to my mind I just gave as an example. But in a digital system design you have to start with a specification for a purpose so in order to get you to that sort of thinking we will have to take some examples of practical circuits and one such class of practical circuit is the code conversion. And I gave you a simple example of what a coding is, be it decimal to binary code, ASCII to character, character to ASCII codes or any of these Excess – 3 or gray codes or whatever the codes are.

Now we will first take an example of how to convert a decimal to Excess - 3 converter. Excess - 3 is nothing but add 3 as the name suggests. If I give you 0 you make it into 3, if I give 1 make it into 4, if I give 9 make it into 12 that is all. Only 0 to 9 is required in a decimal system. We will deal with this digit by digit. There are two ways of handling decimal numbers in a computer or in a calculator. I can convert the whole number into binary and have series of 1s and 0s mind boggling or I can take digit by digit convert them into binary and work on each individual digit and put them together finally.

So when I consider each decimal digit as one unit then each of those units will have four binary bits but out of four bits I can have sixteen combinations but we are only interested in ten combinations 0 to 9 and the other six are not required, not used. So the given binary code I will call it b_0, b_1, b_2, b_3 are the four binary bits of the binary code and the corresponding Excess - 3 code I will call it e_0, e_1, e_2, e_3 all of you know that. This is called a Least Significant Bit or LSB to the right, and the one to the left is called the Most

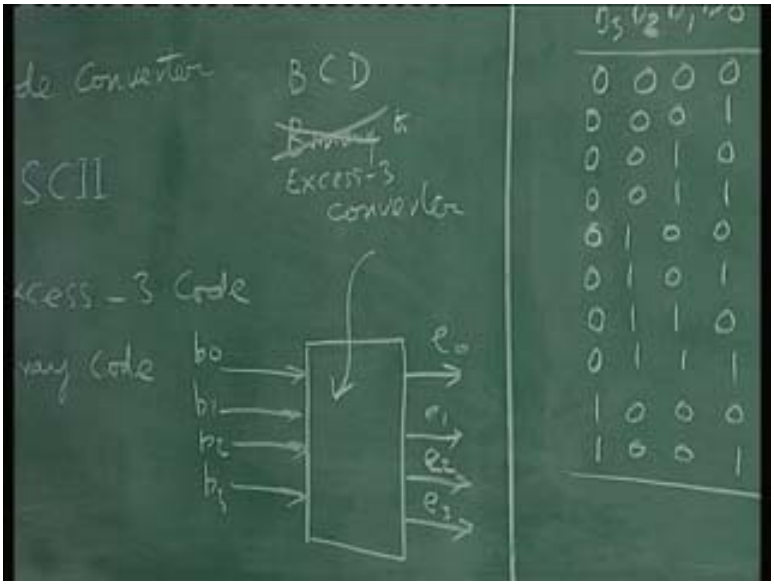
Significant Bit MSB because the value of this bit is much higher compared to this. This is the minimum value and this is the next higher, next higher, next higher value. So LSB to MSB there are four bits, here also LSB to MSB there are four bits so I am only interested in ten of these values 0 to 9, 0 1 2 3 4 5 6 7 8 9 decimal written in binary form ten values or ten patterns and corresponding Excess - 3 all I have to do is to add 3 and convert it back to binary. That means 0 is represented as 3, 1 is represented as 4, 2 as 5, 3 as 6, 4 as 7, 5 as 8, 6 as 9, 7 as 10, 8 as 11 and 9 as 12.

(Refer Slide Time 19:26)



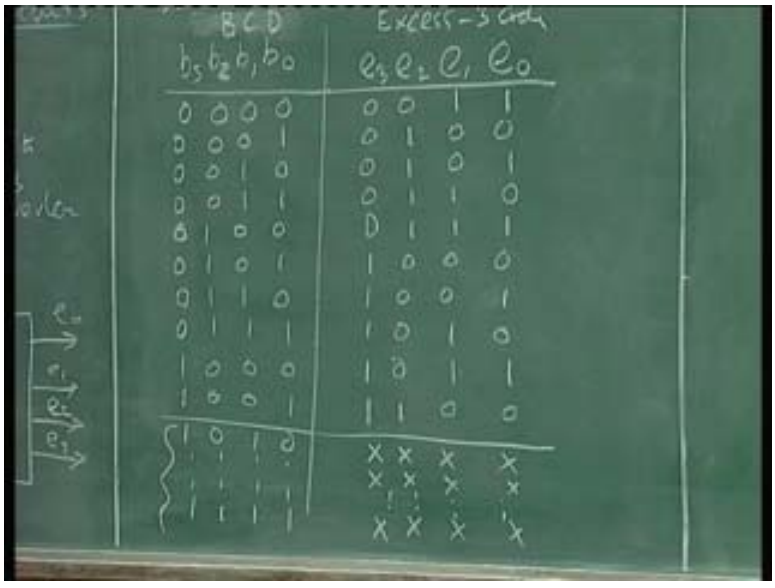
If when you are doing the converter what you are going to design is a converter binary to Excess - 3 converter for which I will give b_0, b_1, b_2, b_3 as inputs and you get e_0, e_1, e_2, e_3 as outputs. Strictly speaking I should not even call it binary because binary means it is natural it can go from 0 to infinity or to any higher value but I am going to restrict from 0 to 9 binary that means I am going to have only decimal numbers but represented as binary digits. Decimal number 0 to 9 is my interest but I cannot handle 0 1 2 3 4 etc in a computer, but I only need 0s or 1s in my computer so I convert my 0 to 9 into binary bits or binary bits as a redundant we are saying. Binary bit means binary digit. Since I am converting 0 to 9 into bits strictly speaking I should not even call this binary to Excess - 3 converter I should call it BCD to Excess - 3 converter already there is a coding, binary coded decimal.

(Refer Slide Time 20:58)



Actually these are decimal numbers represented as binary codes. So I will call it BCD to decimal converter so BCD number will be (Refer Slide Time: 21:09) and Excess - 3 number. So if b_0 b_1 b_2 b_3 have no possibility of getting values beyond this but since you are having four inputs this is a four input I have sixteen values of truth table I will have a Karnaugh Map with sixteen cells I will do minimization etc so I can conveniently use them as "don't care condition". Do you remember the don't care I talked about? So if anything happens from here on the last six entries corresponding to 10 11 12 13 14 15 for all these six entries the output will be "don't cares".

(Refer Slide Time 22:03)



I don't care about these outputs because these inputs are going to come but I can use this to my advantage for simplification, we talked about in the last lecture. Now my problem is stated this is how a design should start. Clear statement of the problem is possible by a representation like this schematic followed by a truth table, then use the Karnaugh Map do the simplification draw the logic circuit then go to the lab, get those gates, wire it up, and test it. That part we will do in the practical part of this course. There is a lab attached to this course we should do it later on.

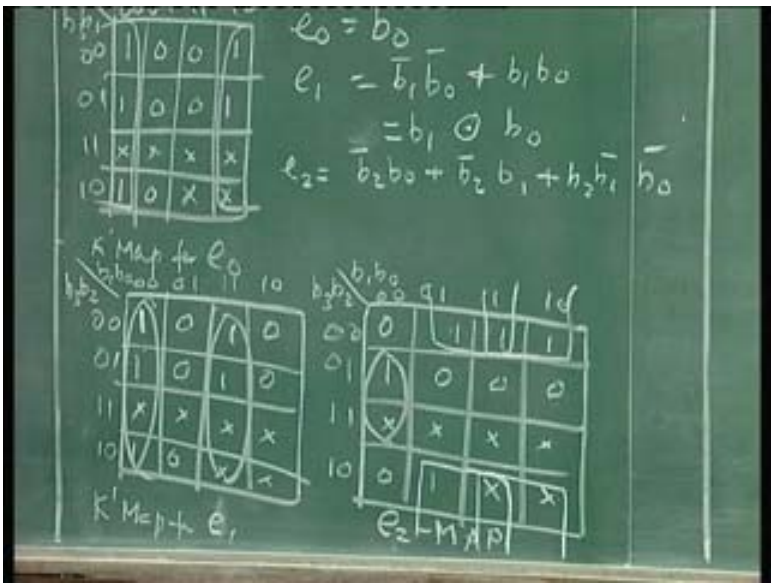
So now we will have to take each of these values e_0 e_1 e_2 e_3 and find out the logic combinations or logic functions which can represent e_0 in terms of b_0 b_1 b_2 b_3, e_1 in terms of b_0 b_1 b_2 b_3 etc we will have to write four functions, we will have to represent e_0 e_1 e_2 e_3 as functions of b_0 b_1 b_2 b_3 that is our job. Once you have this function and simplify it to minimum form then I know how to proceed because I can buy those gates and put it together and then get this circuit to work.

Let us quickly do this. We will first draw the Karnaugh Map for e_0 K' Map for e_0 this is b_3 b_2 b_1 b_0. All the sixteen values are here the corresponding e values are here so I have to put these e values in the map, so this is 1 0 1 0 1 0 1 0 1 0 and beyond that all are don't cares only ten entries are there all others are don't care conditions so I put a x x x x here, do you understand why I did this? This is the function I want to represent this e_0 is f of b_0 b_1 b_2 b_3, (Refer Slide Time: 24:56). That means for each input this is the output truth table and like that I have to do for each of these outputs I should draw a Karnaugh Map, for this I should draw a Karnaugh Map, for this I should draw a Karnaugh Map, for this I should draw a Karnaugh Map and then each of them I should simplify to minimum form and then draw the circuit out of it. So the K' Map for e_0 would be this, it is b_0 bar, please look at this, we need not have done this exercise because you look at b_0 you look at e_0 every b_0 entry is complemented in e_0 entry so b_0 e_0 bar you don't need an Einstein to do that.

This is the map for a_1 b_1 so again it is e_1, this is b_3 b_2 b_1 b_0 (Refer Slide Time: 26:20) more time is drawn by writing the map than the actual simplification process. So now I have to take the second entries here 1 0 1 0, 1 0 0 1, 1 0 0 1 1 0 (Refer Slide Time: 26:47). Now this would be I have two terms because this four I can combine as one and this four I can combine as one. This is the beauty of the don't cares. I use these two don't cares, this don't care I use, I use three of these don't cares and we care about the other three don't cares. So this is the advantage of the don't cares. Use it your advantage otherwise ditch it, that's what we do all the time.

You want some thing from somebody use it to your advantage and you don't forget about that fellow. What is that now? b_1 would be, this is b_3 b_1 b_0 bar b bar b_0 bar OR b_1 b_2 b_1 b_0.

(Refer Slide time 30:27)



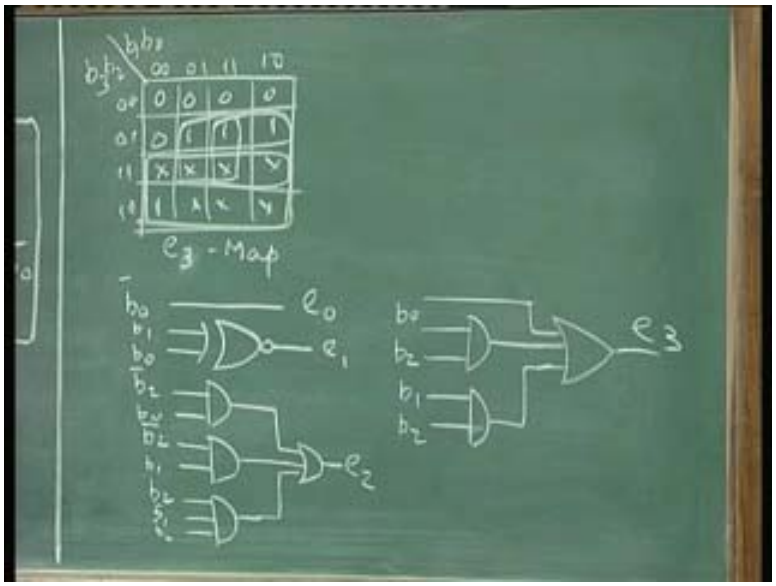
I don't know if you remember this. What was that as a gate? We wrote X NOR that is also called (Refer Slide Time: 28:06). Now the third one I will draw here 0 now the third row third column e_2 column has to be written this is the map for e_2, e_2 is map 0 1 1 1 followed by four 0s 1 and the rest are don't cares. this is slightly more involved these four form a group this 1 1 1, this 1 this 1 this form a group, second group even though there is 1 and 1 left uncovered I can use this, this and this to be simplified logic and finally this 1 is left so I will have to put it so it will be three terms, one term is considered of this, this, this, this, second term is considered of this, this, this, this, third term is considered of this two so that will be e_2 which would be, first we will take this, this would be b_2 bar b_0 then b_2 bar b_1 and finally this entry is b_2 b_1 bar b_0 bar.

Any questions? Any one who doesn't follow what is happening here barely straight forward but still you have to understand what you are doing. I am trying to put together a practical way of using whatever you have learnt till now and this is a useful circuit binary to Excess - 3 converter, binary to gray code converter.

A lot of this logic is used, in big digital machines, computers and all that you see some place something will be happening like this and this will be part of the whole system we are building. And then the last map will be for e_3 map. The e_3 map will be b_3 b_2 b_0 first five 0s then five 1s rest are don't cares. So this can be1 big thing I think I need two more terms this and this. Then b_3 would be this, first we will take this four this is b_3 b_2 bar, b_3 it is 8, b_3 then this will be b_2 b_0 and b_2 b_1. So these are the four simplified equations which will form the converter inside this block. So we give b_0 b_1 b_2 b_3 BCD values and decimal values written as binary and they get converted using these logic functions where you get e_0 e_1 e_2 e_3 and you get a Excess - 3 code which can be used for processing the computer and at the end you have to go Excess - 3 BCD code back to get the original

BCD numbers. Hence the circuit can be drawn easily you have e_0 as b_0 so you will have to assume that all the b_0 b_1 b_0 b_1 b_2 b_3 and their complements are available having to draw the inverters so I will assume that b_0 is available.

(Refer Slide Time 35:57)



The b0 bar is same as e_0 bar, e_1 will be b_1 and b_0 I am not showing that b_0 b_1 could have been directly taken from here to avoid too many connections, the same b_0 we are talking about here, this b_0 bar is coming from the same b_0 here. The next function e_2 is b_2 bar b_0, b_2 bar b_1, b_2 bar b_1 b_0 bar, b_2 b_1 bar b_0 bar all of them to together will be put into NOR gate, this is my e_2, e_3 is finally b_3 b_2 b_0 b_2 b_1.

Of course all of them will be in one place and then all these connections are same. For example this b_0 is same as this b_0, this b_1 is same as this b1 so you can take1 b_0 and take it like this so the connection will be a little bit complex so I thought you will just understand. you represent it like this that's all when you wire it up of course there is b_0 b1 b_2 b_3 coming into the circuit and you have to get them inverted if you want b_0 bar b1 bar b_2 bar b_3 bar so you will have eight such as b_0 b_0 bar, b_1 b_1 bar, b_2 b_2 bar, b_3 b_3 bar etc. Then you have all these gates, wire them up criss cross and then you get e_0 e_1 e_2 e_3 then the circuit is built then you give different codes and verify whether corresponding codes are coming as Excess - 3 codes. This is a simple procedure with a standard circuit which are commonly used circuits which are also easy to understand circuits, all code converters are designed this way to be more specific.

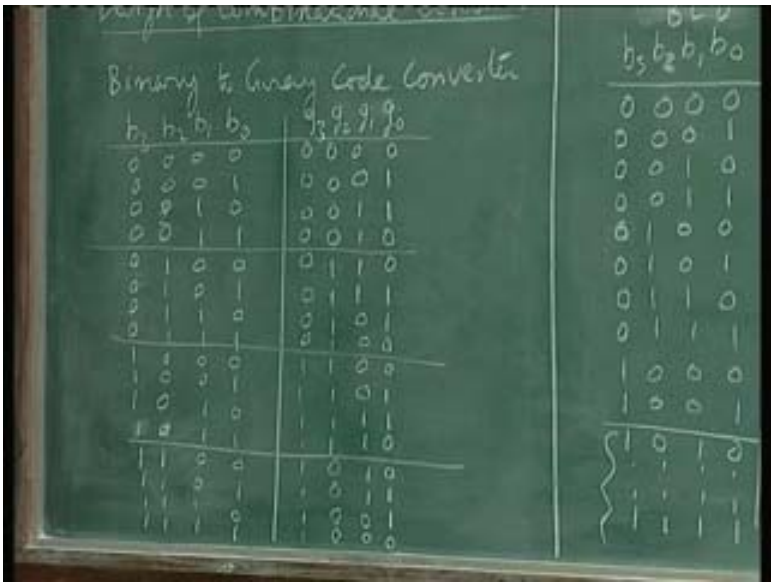
Are there any questions? What I am going to do is to give you a gray code, what a gray code is? I will tell you what a gray code is and I will draw the truth table and then you follow the same procedure, I will leave it is an exercise, do follow the same procedure to get the converted code that is binary code I will give you, gray code I will give you and the truth table I will give you. From the truth table you get you have the four values of the gray code using the Boolean reduction technique using K' Maps then draw the logic. So I

will give you the table and the result you have to draw the map and verify my results are correct and if the results are wrong then there is something wrong because I am going to give you the correct results.

So let us talk about gray code converter, binary to gray. Now this time I will do binary to gray code not BCD to this. As I said gray code has been designed to reduce the switching activity. That means from one binary value to the next binary value there should be only one bit change. Given a gray code for a given binary number the next binary number if you take the gray code can only change in one place so one of the 0s can become 1 or one of the 1s can become 0 so it is only one 0 to 1 transition between a gray code of a binary number and the gray code of the next binary number, that is the definition of a gray code, that is the characteristic of the gray code. As I said when you are reducing the switching activity when you are counting sequentially it reduces the switching activity thereby increasing the speed as well as reducing the power consumption.

So how does it go? We will do the full binary this time not only BCD, we will call it b_0 b_1 b_2 b3 again 0 0 0 1, 0 0 1 0, 1 1, I will put four and just make sure I don't make a mistake. There is no significance for these lines. When I write the code I will not write something, so I will call this g_0 g_1 g_2 g_3, the spelling of gray is this g r a y gray code, it is the name of the person not the grey color so we will do this, this is easy to write 0 1 1 0, 0 1 1 0, 0 1 1 0, 0 1 1 0. The next bit will be 0 0 1 1, 0 1 1 0, 0 0 1 1 and 1 1 0 0; g_2 will be 0 0 0 0, 1 1 1 1, 1 1 1 1, 0 0 0 0; g_3 will be 0 0 0 0, 0 0 0 0, 1 1 1 1 and 1 1 1 1. These is the binary list of the equivalent decimal values from 0 to 15 for four bits but you can do it for any number of bits. BCD means 0 to 9 so rest of them are don't cares. Here it is a binary I am saying. Just because I stopped to four bit that does not mean gray code is supplied only for four bits. Suppose I have a seven bit binary code I can get a seven bit gray code. The same rule will apply 0 1 1 0, 0 1 1 0, 0 0 0 0 four 0s, two 0s two 1s, four

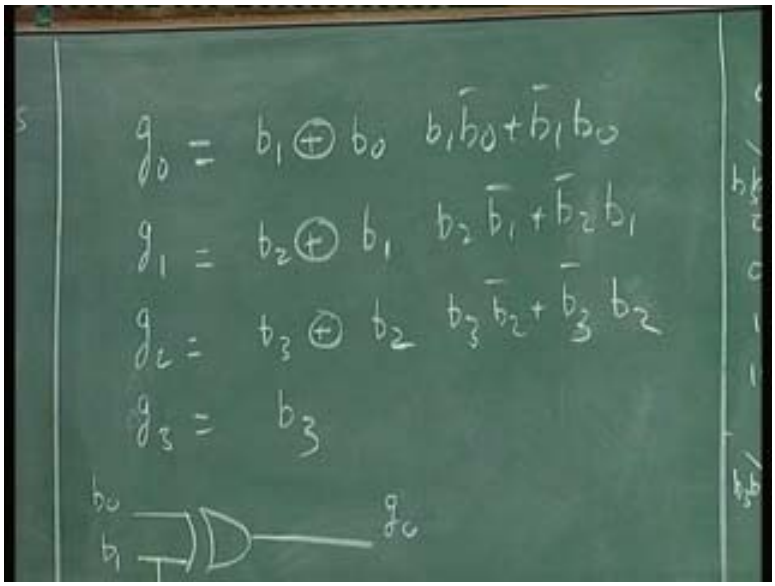
(Refer Slide Time 42:01)



0s four 1s, eight 0s eight 1s, sixteen 0s sixteen 1s like that you can go on writing, that is why I am calling it a binary to gray code conversion. Even though we are destined to hold binary digit bits it does not mean that you cannot do it for five bits or six bits or seven bits or eight bits. If you see now between one number and next number there is only one position for example there is only one change here. You see anything for the example this seven here it is seven to eight 1 1 1 becomes 0 1 0 0 four changes. Whereas here it is only one there is a 1 here 0 becomes 1 and the rest of them. Only one bit has changed from one number to the next number in a gray code.

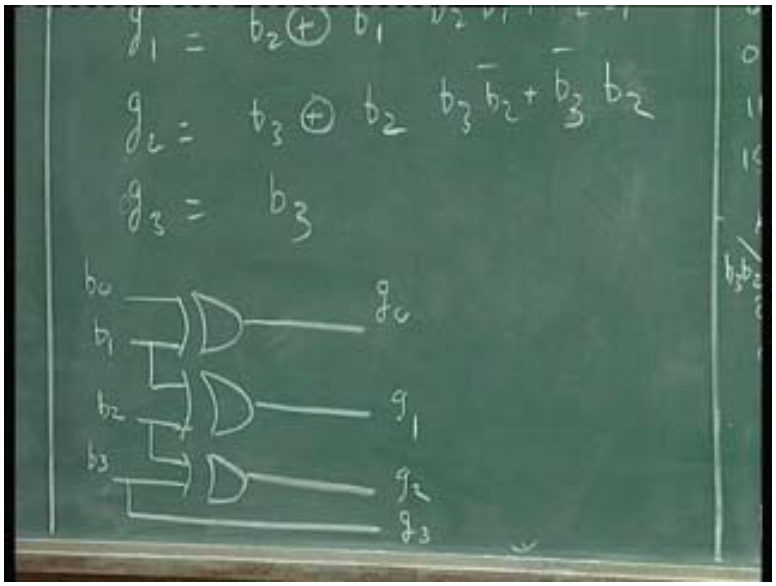
So I am not going to draw the Karnaugh Maps, it is your job as an exercise I am going to give you the answers though so you can verify this at home g_0 would be b_1 exclusive OR b_0; g_1 would be b_2 exclusive OR b_1 very simple; g_2 would be b_3 exclusive OR b_2 and g_3 is same as b_3 as you can see eight 0s followed by eight 1s eight 0s followed by eight 1s. Now you can very easily draw this b_0 b_1 this is g_1 g_0 g_1 g_2 g_3 a very simple circuit, four exclusive OR gates. Of course exclusive OR gate you won't read. When you read a Karnaugh Map you will get sum of products, you won't get exclusive OR but you can simplify it exclusive OR.

(Refer Slide Time 44:59)



This simplification is very easy. this term for example will not come like this, it will come like this (Refer Slide Time: 44: 44) you will get like this but you can see it like an exclusive OR gate because you won't get this direct expression from the Karnaugh Map because Karnaugh Map will get the minimum sum of product expression so this is a sum of product expression. Therefore this has to be converted and you will have to see that this is sort of connected properly. So with three exclusive OR gates I will get a four bit binary code into a four bit gray code, it is the same logic if it is five bits then it is one more exclusive OR b_3 b_4 it goes on so seven bit exclusive binary code we have we can get seven bit gray code by using six exclusive OR gates very simple elegant circuit and is practically used also.

(Refer Slide Time 45:35)



As I said many times people want to convert the binary code to gray code before doing the processing for all the switching activity and then get back the binary. This is a very elegant circuit commonly used circuit. Now, all I am asking you to do is to draw the Karnaugh Map for each of these values g_0 g_1 g_2 g_3 simplified in this form from this, this is evident this is same as this and then this I have already given to you. So these are examples of code conversion as one class of circuits. You can go on, anything as I said in the beginning of this course.

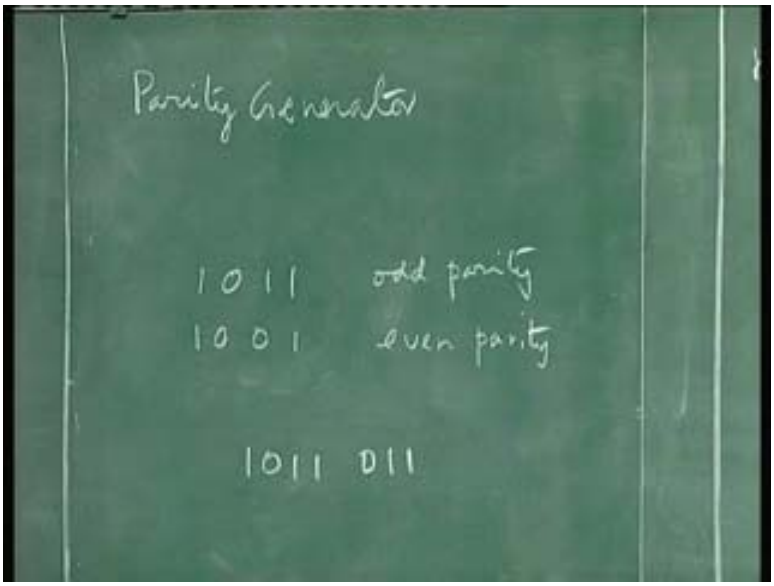
Anything you see is digital these days so you can think of applications have lots of inputs and lots of outputs there may be lots of inputs and one output whatever it is how are they connected what is the logical connection between the inputs and outputs, express it very clearly in words formulate it as a truth table, draw a functional block diagram, draw the truth table, write the truth table, get the map, do the simplification and draw the circuit, get the gates, go to the lab, wire it measure it and test it that is the simple design procedure of a digital system and I showed with two simple examples which are practical examples. We will see some more examples in the next lecture and finally we will go into the arithmetic operations or circuits which are the basis of computers.

Before that I will give you one more thing which probably can be though of as another homework. This is one of the assignments. The second assignment is, because I don't want to go on drawing the same type of thing again and again. We can do a reading assignment of codes. Every standard digital book will have sectional codes, ASCII code, Excess – 3 codes, binary codes, weighted code and all types of codes. You can take any of those codes they will give you the binary number and the corresponding codes. Just take any one of them and convert it into this logic that can be an exercise for you.

Then one other thing I want to talk about is the parity generator. I want to explain what a parity generator is, we will probably see an example of it later on. I sent digital data, digital data means 0s and 1s, a constant stream of 0s and 1s coming across, if it is a digital telephony, a digital transmission, internet traffic, ISDN whatever you name, the digital data that is coming over your transmission lines and then at the receiving end you can get these 0s and 1s and try to make sense out of it because you know the codes you decode it and find out but there may be errors occasionally. The 0 can get to 1 because of some static, a low level being static suddenly going to 1 and 1 bit instead of 0 got corrupted since it is a 1, similarly a 1 was because of some drop in the level of voltage to 0 suddenly, so there can be an error of 0 becoming 1 or 1 becoming 0 this can happen in several bits also but if only one bit is an error we can see if there is an error in one bit by checking the parity.

Suppose I have a bit stream the number of 1s in that bit stream is called the parity. let us say I am sending a four bit at a time I will say I will give information four bits at a time packets of four bits so I will say 1 0 1 1 so it's a four bit stream it has three 1s so this is called odd parity. The number of 1s in the bit stream I am talking about the four bit stream is odd I will call it odd parity but if on the other hand it is 1 0 0 1 I call it even parity. If number of 1s is odd it's called odd parity in a given bit stream. The bit stream is previously defined you can't just arbitrarily start at some point end at some point and say it is odd parity.

(Refer Slide Time 51:01)



For example, if you take an ASCII code it is a seven bit code, after every seven bit you will have to count the number of 1s see whether it is odd or even. So you should know the number of bits they are going to transmit at a given time so if you know that then you can say whether it as an odd parity or even parity, they will also put the parity along with that. Therefore for example an ASCII code seven bit code you see this is my code (Refer Slide Time: 51:04) now I will count my 1s 1 2 3 4 5 it's an odd parity I can put a 0 here

or 1 here and say it is an odd parity. So the extra bit I will to add to say that I have odd number of 1s. On the other hand if the number of 1s is even I will put a 0. So I will deliberately add a parity bit to my bit so instead of looking at every seven bit I capture eight bits at a time, seven bit is ASCII and the eighth bit is the parity so I verify whether the parity bit checks with the number of 1s. If the parity bit does not check with the number of 1s then I know it is an error. So this is one of the simple techniques of detecting simple errors.

So first we have to generate this parity add it along with data stream send it and when you receive it you receive the whole thing count the number of 1s and find out the number of parity bit and if the parity bit matches with the given parity bit it is fine and if the parity does not match with the given parity bit then you know. First you can it is an error so we can at least ask them to resend it. There are several techniques of error detection, error correction and then more than one bit in error what will happen, if two bits are on error what will happen and all those things are there. I am talking about error correction code that is a big topic by itself in communication. But can do simply, quick check and tell you there must be an error please send again, simplest thing you can think of.

So in the next lecture we will talk about how to generate the parity bit. That will be a good example for combinational logic. We will draw the truth table for a parity generator and implement it.