Digital Circuits and Systems
Prof. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology Madras
Lecture - 10
Parity Generator and Display Decoder

In the last lecture we saw some examples of designing combinational circuits starting from specifications through the truth table and the Karnaugh Map. We took the examples of code converters which are a class of or type of circuitry in digital electronics used frequently. We talked about a BCD to Excess - 3 converter also a binary to gray code converter. I also mentioned about parity generators. I mentioned parity is a concept used to detect errors. A single bit error can be detected using parity. so what we do in a parity generator is to send a bit stream of a given number of bits and you also send an extra bit that bit will tell whether the total number of 1s in the bit stream that you have sent earlier is odd or even.

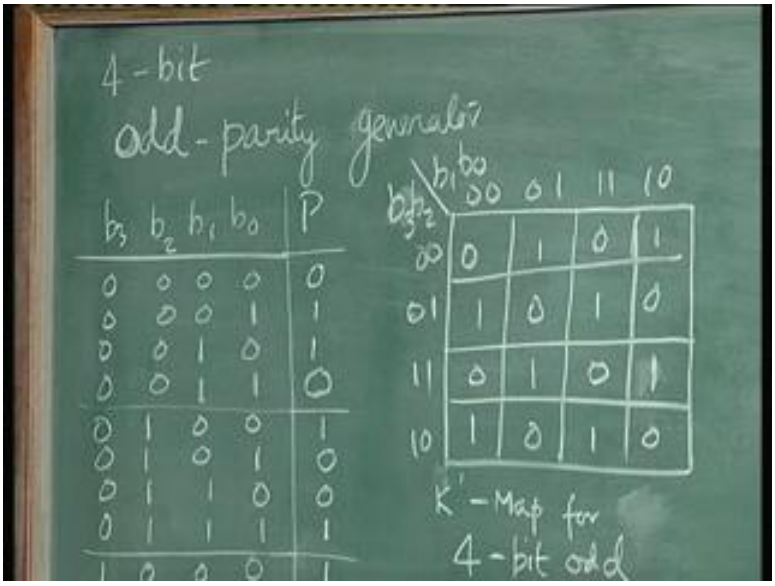
So, for example, if a four bit stream bits are sent in groups of four bits the total number of 1s in a group of four bits is 1 and a fifth bit will be sent called a parity bit. The number of 1s in the four bits data bits originally sent is odd we send an extra one. If the number of 1s which is sent is even you send a 0 indicating the number of 1s is even. So this extra bit is called parity bit you can have an odd parity or even parity again you have to define it. So the receiving side should be able to receive the parity bit and check whether your data matches with that. it is not necessary should always send an odd parity as 1 and even parity as 0 you can also do it even parity as 1 and odd parity as 0. As long as there is a complete communication understanding between the sender and the receiver the sender this is called protocol.

Protocol is defined as, they know what will be the total number of bits we are going to send in each group of data and they should also know whether you are going to send a parity bit or not at the end of group of data, they should also know whether you are sending an odd parity bit or even parity bit. So once you understand the protocol on both sides it is used for communication and as I said in the last lecture it is not a very sophisticated scheme it is a very simple scheme. all you can do is a detect if there is a single bit error in your data stream, if there is a double bit error then you can't do it two bits have changed you can't do anything about it. So this is one simple scheme.

The idea of introducing this is whatever is the protocol you define is always a way of designing digital circuits for that. You give an example of all of those things you see in communication and telephony in control everything is finally digital and everything boils down finally to design of digital circuits using the specification, truth table, Karnaugh Map, gates the procedure and to highlight that I am taking a simple example. So in this example what we will do is a four bit odd parity generator. That means I am going to send four bits at a time and the parity bit will be 0 if the number of 1s is even that is why it is called as odd parity generator error. Odd parity will be generated or a parity bit will be generated whenever the parity is odd. In this case parity is odd because the number of

1s is 1 so I will have 1 at the output. So whenever the number of 1s in the group of four bits is odd a 1 will be generated by the parity generator and whenever it is even it will not generate a bit that means it will be a 0. And this is for a four bit I can do it for a five bit six bit seven bit as I said ASCII code is a seven bit code, you are sending ASCII characters ASCII data and I would like to have an eighth bit as a parity bit because seven bit ASCII code eighth bit will be a parity bit. These are all various schemes or protocols.

(Refer Slide Time 6:22)



Now given this you have to look at this table and see in each of the sixteen rows using four bits this is the binary bit we are sending b_0 b_1 b_2 b_3 find out the number of 1s if it is even put a 0 here if it is odd put a 1 here and 0 is assumed to be even. And now I want to design a circuit which will make this generation of parity bit possible from this data. that means my hardware is 4 - bit odd parity generator and the inputs of these are the four binary bits and output is p odd and this is the Karnaugh Map 0 1 1 0 I just mapped this truth table into this four variable K' Map b_2 b_3 b_1 b_0 being the four input bits.

Now the next step is to simplify this reduce that using grouping of the 1s. Is there any grouping possible at all in this map? All are individual 1s so how many terms will be there in the final output? So if you write P odd will be, each of these 1s is an individual term it cannot be combined because all the neighbors all the adjacent cells are 0s for any given 1. For any given 1 all the adjacent cells are 0s so you cannot group them at all together so there will be eight terms. But as you see here there is a nice pattern here an alternate one, checker board pattern they are called chess board or checker board pattern.

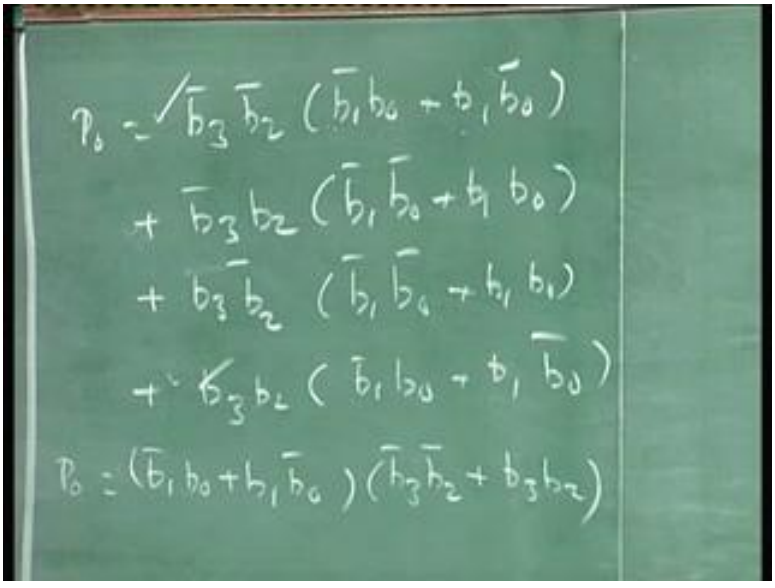
So whenever it is a chess board or checker board pattern there is a hardware simplification possible using Exclusive OR gates. Even though you cannot directly read an Exclusive OR gate from a Karnaugh Map one can do a further simplification of the minimum sum of products using Exclusive OR gates. And if you are experienced enough you can even read the pattern of the Exclusive OR from the map. Since it is a first

exercise you are doing of this type I will write it in the conventional way sum of products and then simplify it to show that this pattern is nothing but the Exclusive OR pattern so each of these 1s is an essential prime implicant which has to be in the final expression. All the essential prime implicants have to find a place in the final expression so each of these terms can be written quickly b_3 bar b_2 bar b_1 bar b_0 plus b_3 bar b_2 bar b_1 b_0 bar first two terms, the second two terms are b_3 bar b_2 b_1 bar b_0 bar plus b_3 bar b_2 b_1 b_0 plus these two I can write as b_3 b_2 bar b_1 bar b_0 OR plus b_3 b_2 bar b_1 b_0 and finally these two b_3 b_2 b_1 bar b_0 plus b_3 b_2 b_1 b_0 bar. Therefore all the eight terms have been written. This is if you want to do a minimum sum of product expression with traditional Karnaugh Map simplification Karnaugh Map reduction this is what the solution is.

It looks like we need eight input OR gate each of the eight input OR gate will come from an AND gate with four inputs and all the variables b_0 b_1 b_2 b_3 have to be inverted. So there will be four inverters to invert variable b_0 b_1 b_2 b_3 and eight AND gates to combine these eight terms followed by an eight input OR gate, it is too much logic. So let us see we can simplify it. As you can see you look into the pattern you can simplify this. This is common so I can combine these two terms as b_3 bar b_2 bar (b_1 bar b_0 plus b_1 b_0 bar). You know that b_1 bar b_0 plus b_1 b_0 bar is Exclusive OR b_1 b_0. Likewise we can say for this as b_3 bar b_2 (b_1 bar b_0 plus b_1 b_0) and this is Exclusive NOR.

Next is b_3 b_2 bar (b_1 bar b_0 bar plus b_1 b_0) and finally b_3 b_2 (b_1 bar b_0 bar). This is grouping, I have grouped each of these, two terms in each row, that's all, very simple. Now this is common to these two b_1 bar b_0 bar b_1 b0 bar which is Exclusive OR b_1 and b_0 is common to these two terms so I can take it out and say (b_1 bar b_0 or b_1 b_0 bar) (b_3 bar b_2 b_3 b_2). This is combining this and this.

(Refer Slide Time 14:06)

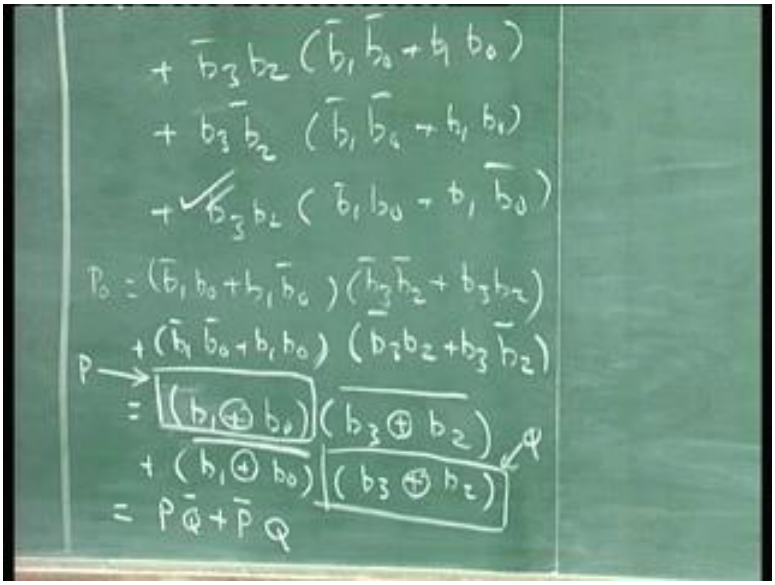


These two terms are common (b_1 bar b_0 bar OR plus b_1 b_0) which is Exclusive NOR (b_3 bar b_2 plus b_3 b_2 bar). What we can do is; further start writing it as Exclusive OR b_1

Exclusive OR b_0 b_3 Exclusive NOR b_2 but I am going to write it as Exclusive OR b_2 bar. Exclusive NOR is same as Exclusive OR bar so I am taking this term and writing as Exclusive OR (Refer Slide Time: 15:34) OR (b_1 Exclusive OR b_0 bar) for this term this is (b_3 Exclusive OR b_2).

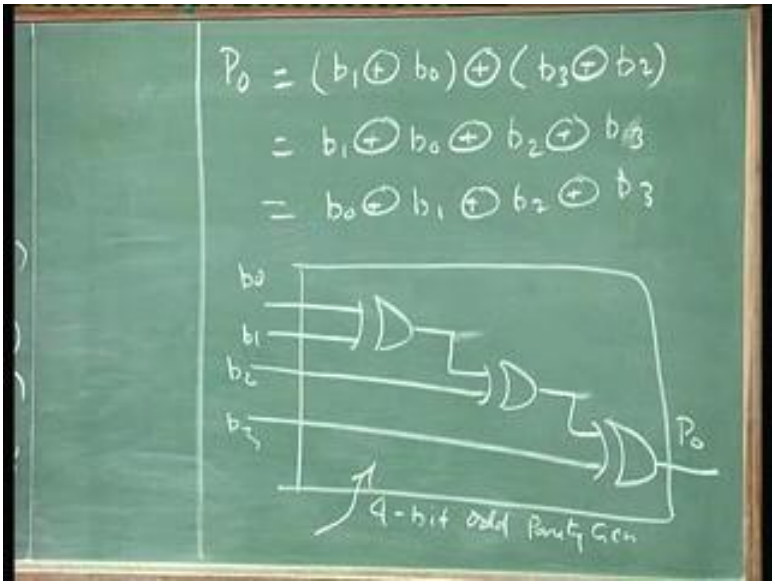
Now if you call this b_1 Exclusive OR b_0 as another P, this is Q same as P Q bar OR P bar Q where P is b_1 Exclusive OR b_0 and Q is b_3 Exclusive OR b_2, P is this and Q is this. Just identify them just for the sake of identification. That means really it is b_1 Exclusive OR b_0 this is nothing but P Exclusive OR Q, this is P Exclusive OR Q PQ bar OR P bar Q so P is nothing but b_1 Exclusive OR b_0 Q is nothing but b_3 Exclusive OR b_2 so the whole P odd is b_1 Exclusive OR b_0 whole Exclusive OR (b_3 Exclusive OR b_2) by law of commutation. I can also write it as (Refer Slide Time: 17:40) b_0 Exclusive OR b_1 Exclusive OR b_2 it is a very nice pattern so this pattern can be identified as Exclusive OR pattern.

(Refer Slide Time 16:34)



I think we had this in the gray code somewhat similar to this the gray code simplification that you saw in the last lecture some what similar pattern so now the hardware is so simple all we need is three Exclusive OR gates so I can give b_0 and b_1 and that will be given to b_2 this will be given to b_3. So if I give four bits this is my hardware the logic that we had inside this box. The four bit odd parity generator here is nothing but three Exclusive ORs contained this way, this is the 4 - bit odd parity generator and by induction we can prove that I can use any number of bits 7 - bit ASCII code so all I need is six Exclusive OR gates to produce the parity. Even parity may be Exclusive OR or Exclusive NOR combinations slightly different you will have to work it out whether it is odd parity or even parity.

(Refer Slide Time 21:14)



The beauty of the Exclusive OR Exclusive NOR combinations is one is the complement of the other. So if you take an inversion of an Exclusive OR you get Exclusive NOR and when you take the inversion of Exclusive NOR you get Exclusive OR.

Hence because of that whether it is odd parity or even parity is a question of one inversion again. So these are some of the practical circuits we use all the time in data communication; transmission and parity checking. Parity generation parity checking is an integral part of any data communication. As I said this is a very simple communication protocol for error detection of a single bit correction but if there is more than one bit and there are multiple errors and what you do with the error being detected how do you correct it? So as I said it is a huge communication subject by itself where error correction, error codes, error detection, error detection techniques which type of codes are most amenable for detecting errors and all those things, that is not the subject of this course so we will not dwell too much on it. Finally everything reduces to logic hardware, gates, etc that is the point I want to make.

Are there any questions on this? So you can now do it for different number of bits odd and even parity and for all those combinations. Let us see one last example in this series and then we will move on to arithmetic circuits. Whatever examples I have been discussing or standard circuits are used repeatedly in digital system design. The application may be different application may be communication or as I said error control or switching activity reduction there may be different reasons coding, security but these are standard circuits used again and again. So I am picking up those circuits but the concept of design is the same. Solving the problem, specification is very clearly defined going to the truth table to the Karnaugh Map with simplification and hardware.

Sometimes from the minimum sum of product you can directly implement this logic but sometimes you may have to do some extra manipulation as in this case. We also saw this in the case of the gray code generator, some sort of a grouping helps in reducing the hardware further. This looks so formidable eight gates each with four inputs and inverters for each one of them followed by eight into OR gate this is so elegant. So those types of things also sometimes are possible. Hence as I said the last one from this example we will take today which is called as the 7 - segment display decoder.

Again we will say BCD to 7 – segment display decoder what do you mean by this? we are used to decimal numbers in arithmetic, in data presentation, in displays and everything, we are standing in front of a counter in a bank they keep putting the display of the number and what is the number that is being currently service it is a decimal number and I don't think you would like if the number is given in binary form. But then as I said digital logic is always for binary number. So some way we are converting decimal to binary that is what we do all the time using four bits and BCD means 4 - bit binary we will take for each of the digits rather than convert the whole number of whatever magnitude convert into binary by repeated division by two that is the technique used instead of that you take each digit of the decimal representation and convert into corresponding binary this is called BCD representation.

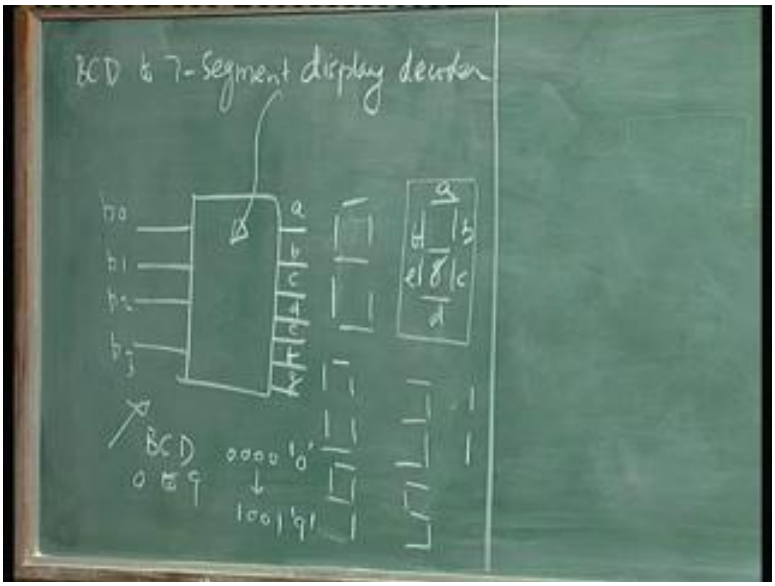
So, when you do a BCD representation you give the binary numbers bits either for code conversion or arithmetic all that we will be only doing binary, digital logic, the circuits inside, gates and other sequence of circuits that we are going to discuss later on so all of them are going to process binary numbers but I want to give input as a decimal so I have a BCD converter.

Similarly, after the result is out I need to have a decimal display rather than a binary display. So this is called BCD to binary. once I have done this process and it is a BCD, at the output of my process there is a decimal number coded in the binary form but I want a decimal display so a four bit binary number from 0 to 9 using four bits has to be displayed either in a clock or any other place you want to display number as a decimal number so we need to have sort of an interface or hardware for that. So this is what it is going to be. Thus I have a BCD number b_0 b_1 b_2 b_3 this is a binary coded decimal number 0 to 9. That means it will be 0 0 0 0 to 1 0 0 1 ten combinations will occur here and each of these combinations should produce a display, (Refer Slide Time: 25:22) this is the display decoder we are trying to design. What is the display look like? You have seen the displays all over the place.

So, depending on the number you want you can use this display. for example, if it is three you want to display or if you want to display five the standard display we will use for displaying by any BCD number from 0 to 9 if 0 would be this then 9 would be this, some people will do it this way for their understanding this is good enough and 8 would be all these segments which would be right up. So each is a segment which is independently controlled, these are LEDs Light Emitting Diodes. These are Light Emitting Diodes which are individually controlled to light up for the particular number you want to display.

Suppose I want to display 3 I need to light up this LED, this LED, this LED, this LED, and this LED (Refer Slide Time: 26: 45). If I want to display 1 I need to light up these two LEDs. If I want to display 0 I need to light up all LEDs except this bar. If I want to display eight I need to light up all the LEDs. So I should produce the corresponding signal so the particular LED will be lit up or not, that is the function of this BCD to 7 - segment display converter. This is called 7 - segment display because there are seven segments. We will call these segments a b c d e f g, a to g has seven numbers so each of these segments we call it seven segments, a display can be made up using seven segments and each of these segments can be controlled by an LED which can be independently turned on or turned off depending on the requirement. If a particular segment has to be lit up the LED should be turned on. if a particular segment should not be lit up then that particular LED should not be turned on it should be turned off.

(Refer Slide Time 29:51)



So how many outputs do I need from this? I need seven outputs called a b c d e f g depending on the code of the number I am giving here 0 to 9 this is 0 this is 9 depending on the code of the number I gave the corresponding LED should light up. If I give 0 0 0 0 I want a b c d e f to glow g not to glow. If I give five 0 1 0 1 that is five 0 1 0 1 is five so I want LEDs a c d f and g to glow and all others not to glow. If I want eight 1 0 0 0 is eight then I need all a b c d e f g to glow. So I like to write the truth table for each of these segments.

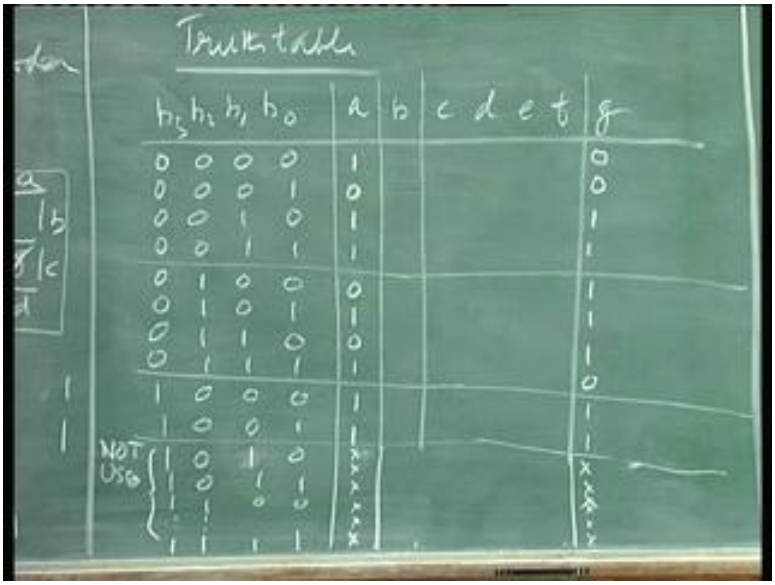
What are the combinations for which ‘a’ should glow? What are the combination for which ‘b’ should glow? What are the combinations for which ‘c’ should glow? ‘d’ should glow, ‘e’ should glow, ‘f’ should glow, ‘g’ should glow and then draw a Karnaugh Map for each of these segments, simplify it and do the logic then I have a 7 - segment display decoder which is used everywhere. You can see in an elevator you have the 7 - segment display, clocks have this as I said in the waiting rooms and other places wherever you go

you have the seven, these are only the numbers, you also have the ASCII display decoders. When you want to have messages which has non numerals also alphabets and special symbols, punctuation marks it's all possible, same logic here too, give the code tell me what are the LEDs I should make to glow and what are the LEDs I should switch off then I will have the final count.

So we will write the truth table for this, the inputs are as I said b_0 b_1 b_2 b_3 and the outputs are 7 in number a b c d e f g. this is a four input seven output truth table four input will have sixteen rows out of which you will be concerned only with first ten rows 0 to 9 and 11, 12, 13, 14, 15 are going to come because for one single display I cannot put a 10 or 11 or 12 or 13 so all we need is 1 0 0 0, 1 0 1 0, 1 0 1 1, 1 1 0 0 these six rows (Refer Slide Time: 31:38) are not used so they will have big don't cares. I want to put one big X all the columns are don't cares for all of these rows. For the six rows 'a' don't care, 'b' don't care, 'c' don't care so I don't want to go on writing that so this big X means I don't need any of these a b c d e f g is not defined for these last six rows.

So let us quickly do this for one segment. What are the numbers for which 'a' should glow? Let us list them and then put the corresponding table. 'a' should glow for 0,2, 3, 5, and depends on how you define 6 so we will have 6 like this. If you also have alpha numeric characters then this will be used as a small 'b' and in that case I will put 6 like this. If I am only doing numerals I can have 6 like this also. We have both numerals and alphabets this may be considered as a small 'b' so you can put a cap on the top. so to make it simple I will have six of this type so 'a' will not glow for 6, and 'a' will glow for 7, 8 of course all of them will glow, and 9 because 9 will be like this, this is my 9. So 'a' will glow for 0, 2, 3, 4, 5, 7, 8, 9.

(Refer Slide Time 35:54)

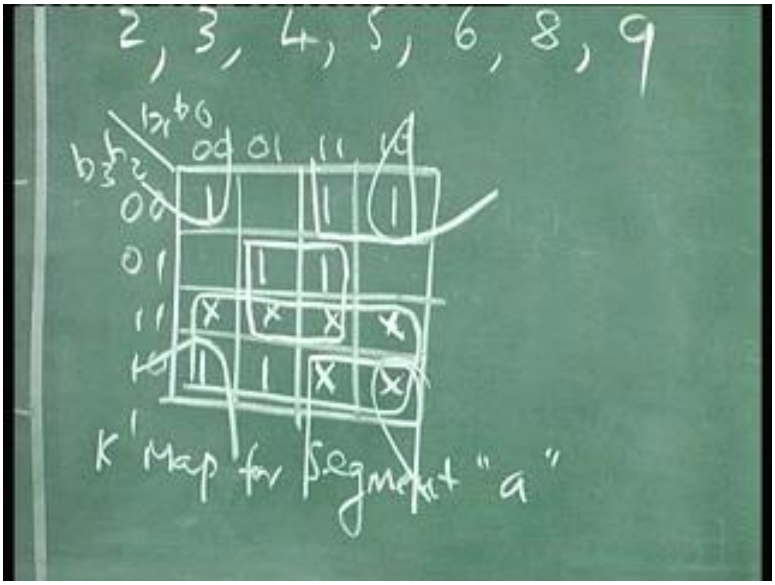


Any questions does any one not understand what I am saying? It looks simple to me at least but if it is not clear please tell me. We will do one more segment to make sure that you understand.

Let us say what are the numbers for which the segment ‘g’ should glow? The segment ‘g’ need not glow for 0, for 1 it need not, 2 yes, 3 yes, 4 yes, 5 yes, 6 yes, 7 no, and 8, 9. This is how we can finish for a b c d e f g and then map it on the Karnaugh Map. Now in the Karnaugh Map corresponding to ‘a’ you put one for these segments 0, 2, 3, 5, 7, 8, 9 and rest of them are don’t cares as I said.

Similarly we will write for ‘g’, for 0 it doesn’t glow, 1 it doesn’t glow, 2 it glows, 3 it glows, 4 it glows, 5 it glows, 6 it glows, 7 it does not glow, 8 it glows, 9 it glows. We can finish the rest of the table identifying the segments, identifying the codes or the numbers from 0 to 9 for which ’ b’ should glow, ‘c’ should glow, ‘d’ should glow and so forth. So having done that our next job is to do the Karnaugh Map for ‘a’ identify the simplest possible solution and then implement it using gates then do it for ‘b’ then do it for ‘c’ then do it for ‘d’ ‘e’ ‘f’ ‘g’.

(Refer Slide Time 38:25)



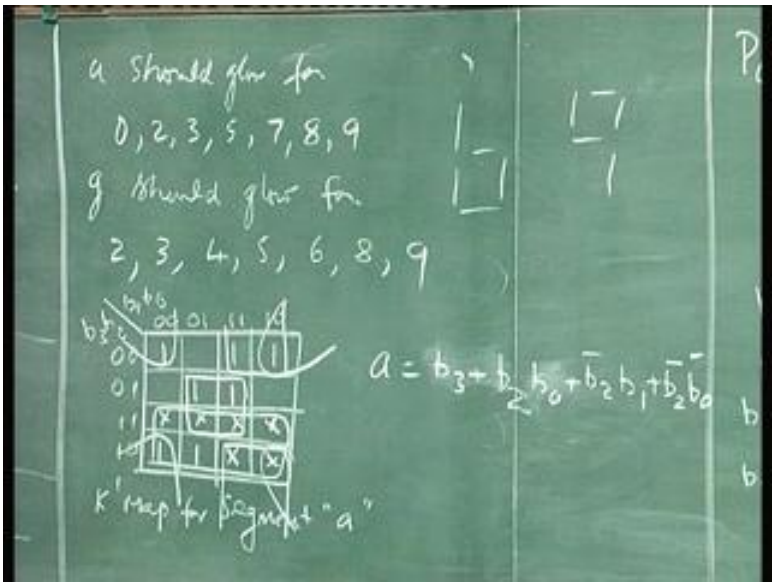
So let us do it for ‘a’ map for segment ‘a’ K’ Map for segment ‘a’ b3 b2 b1 b0 it is 1 0 1 1, 0 1 0 1, 1 1 this is the map for ‘a’ segment. Now let us simplify this. These four corners (Refer Slide Time: 37:32) can be one grouping and all these four corners the power of don’t cares you can see here.

I did not know the hardware simplicity at that time. for some reason we made 6 like that as I said small ‘b’ and all that numerals, if we put an extra 1 here you mean then it will become simpler fine but I am assuming ‘a’ will glow for 6. This is the second iteration. You design for a customer, the customer tells you my 6 should look like this. You come to the drawing board and do the design and tell this fellow if you can let me do this 6 like

this I can save you half a cent by reducing some hardware then he may fall for it. So it is a second iteration. Having specified the problem I already defined 6 as this and 9 as this so let us not go back on that. It can be done I am not saying it should not be done but there should be a reason for doing that.

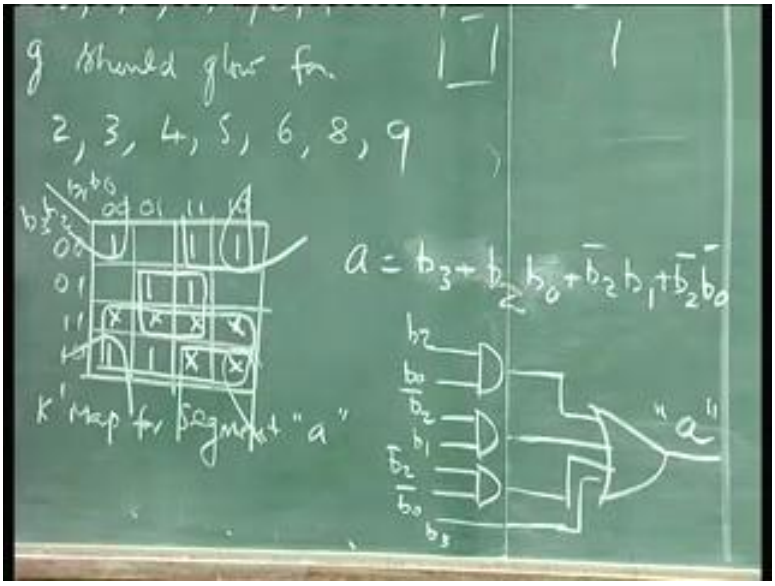
Now what is the expression for 'a' minimum sum of products? First you use this four this will be b_2 b_0 then this will be b_3. Usually it is the convention not a rule to write the smallest number of literals and then larger number of literals progressively. So b_3 then this is b_2 b_0 this is this, and these four will be b2 bar b_1 and then these four would be b_2 bar b_0 bar.

(Refer Slide Time 41:41)



What is the last one? b_2 bar b_0 bar. So we will draw the gate structure for this. Display 'a' would be b_2 b_0 bar, I am assuming b_2 bar b_1, b_2 bar b_0 bar and I am assuming b_0 b_1 b_2 b_3 have been inverted already because all of these segments will require b_0 b_1 b_2 b_3, b_0 bar b_1 bar b_2 bar b_3 bar we don't have to show inverters every time. Inverters can be done once in the beginning to cater to all the segments and the last one is b_3. Therefore whenever these conditions occur 'a' will glow and then 'b' will glow and 'c' will glow and then finally the combination of them will make different digits glow.

(Refer Slide Time 43:09)



So we will quickly do it for ‘g’ may be because you have already written ‘g’ just complete the example. Map for ‘g’, what are the segments for which it is 2 3 4 5 0 1 2 3 4 5 6 7 8 9 this is map for ‘g’. So ‘g’ is, this is one term (Refer Slide Time: 44:24) another term, another term, another term, this is b_2 b_3 then what is this term it is b_2 b1 bar, this term the last one b_1 b_0 bar and this would be b_2 bar b_1. So you have to check these terms when you do it one more time to check all the terms are right both in terms of the literals and the bars. So this would be again a simple b_2 b_0, b_1 b_0, b_2 b_1 so this is b_2 b_1 bar, b_1 b_0 bar, b_2 b_1 and finally b_3 all of them together.

So likewise you can complete this exercise for other segments b c d e f and also check this ‘a’ and ‘g’ to make sure that all the entries are right. We can do a little improvision if you want to save them. Sometimes what you can do is now this circuit will have four inputs and seven outputs we treated them as one circuit even though we are doing it in seven different circuits.

Each Karnaugh Map gives a logic circuit with gates so there are seven individual circuits inside but as far as consumer is concerned the buyer is concerned you go to the market and ask for a BCD to 7 - segment display decoder and he gives you one IC in which there is seven of these circuits independently. So sometimes it is easier when you are putting so many circuits together it is possible to combine terms. There may be slightly different way of implementing if you can identify similar terms or same terms and draw all the seven maps together and see whether there are any common terms between all the seven at least many of them, if two or three of those maps have common terms that part of the hardware can be shared between those three segments. You are drawing a map for a b c d e f g suppose there is a group of 1s common for let us say three or four of those maps, if you identify that term that part need not be repeated in everything, you can take that part as one of the inputs to the OR gate for that particular segment, each segment will have an OR gate into which many AND gate terms will be feeding in, many product terms will be

feeding into the OR gate and if one of these product terms is common to three or four or even seven or whatever that part of the hardware AND gate combination need not be repeated for every segment.

You can go one step further and see even deliberately re-read the map not the most optimum way. Do not re-read the simplest possible way, you see how many common terms I can make, make as many common terms as possible and rest of the terms which these displays.

So these are all some of the design techniques you can use but here we are talking about concepts and using a truth table, Karnaugh Map in order to design circuits instead of taking arbitrary sum of products, arbitrary min terms, sequence as I did in the first few classes. I have now given you practical circuits like code converters and parity generators and 7 - segment display decoders. So that is the development of subject but if you are a practical designer getting paid by a guy who hired you to give the best possible design you will have to go one step further and see whether I can find common terms and reduce the cost of the hardware.

We will not do that exercise now but remember that can be done if you want to. So as I said we are going to stop with this type of general combinational circuit. Of course all circuits can be treated as combinational circuits, many more applications are there. I want to go to the arithmetic circuits. So in next lecture we will start talking about arithmetic circuits because these are the basic units building blocks of computers or calculators, all computational functions require arithmetic so we will start designing arithmetic circuits from the next lecture.