Digital Circuits and Systems

Prof. S. Srinivasan
Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture # 14

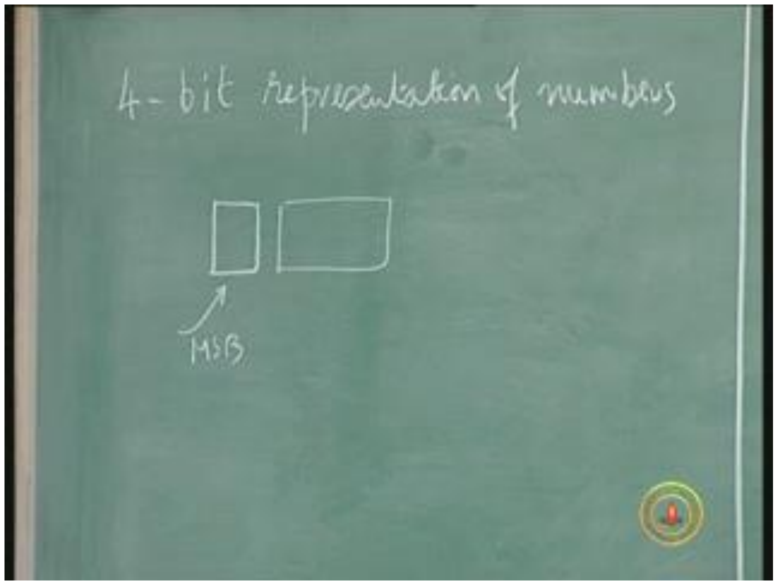
2's Complement Subtractor and BCD Adder

So, if you remember the last time we discussed subtraction and said subtraction is nothing but addition of complement numbers and in the case of binary it is 2's complement addition the subtraction is carried out by adding 2's complement of the given number which has to be subtracted from the original number. We also found out how to get the 2's complement. there is a 1's complement plus 1 and 1's complement is nothing but a inversion bit by bit and that can be carried out using bunch of Exclusive OR gates or group of Exclusive OR gates which are controlled by the subtraction bit, if you want to subtract you pass it through them pass it through the invertor, if it is adding then the bits do not get inverted and if you are going to subtract the bits get inverted.

One thing I should have mentioned earlier actually I would like to mention now is that we have to be very careful, we are talking of subtraction even addition for that matter. You should be careful about the number of bits you are working with because when you are talking of negative numbers and positive numbers and the most significant bit being 1 for negative number and 0 for positive numbers which bit you look at really?

For example if I have number 3 when I ask you the binary representation of 3 you will say 11 then do you think 1 is the most significant bit and because it is 1 it is negative? I have not told you what is the total number of bits and in that system you will have to fill in all the positions and then locate it. Suppose I have a 4-bit number system I have 4-bits represent a given number and in that if I assume the most significant bit to be a sign bit so 3 will be represented as 0 011 since MSB is 0 the number is positive so plus 3. That is something which I did not mention explicitly but it is important to do that. when you are doing an arithmetic operation because your hardware is designed for a specific number of bits, you have a 4-bit adder, 8 bit adder, 4-bit multiplier, 4-bit subtractor, Arithmetic Logic Unit is 4-bit ALU, 8 bit ALU so the hardware is defined for the specific number of bits and you have to fill in all the positions before you can do the arithmetic operation.

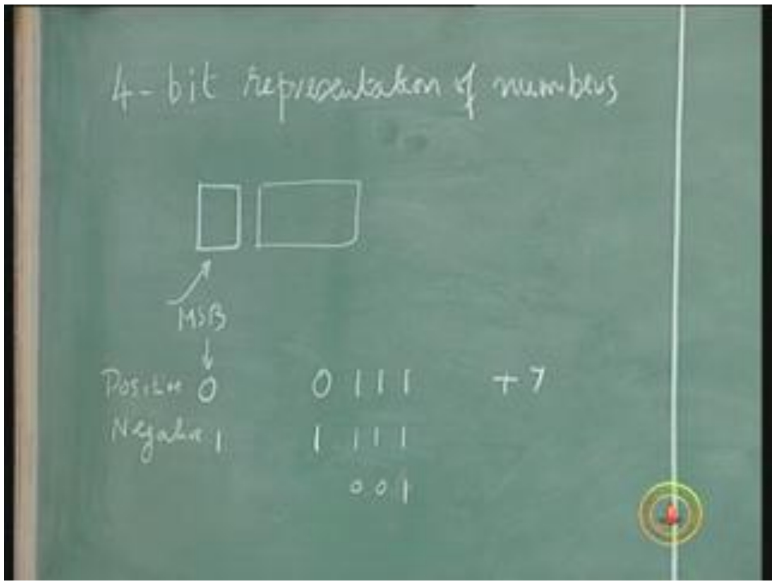
(Refer Slide Time 5:28)



So let us assume of 4-bit number representation and I would consider in this 4-bits, this as MSB (Refer Slide Time: 5:30) the sign bit and these are the bits which represent the value of the number so I can represent a maximum of only three positions and in addition to that there is a sign bit so if 0 is positive, 1 is negative so the maximum of positive number is 0111 which is plus 7 what is the minimum negative number or maximum whatever you want to call it because the number is negative which has the minimum value but what is the largest number you can represent in negative? Somebody says it is minus 7 and somebody says minus 8 let us see because these 2's complement comes into the picture now.

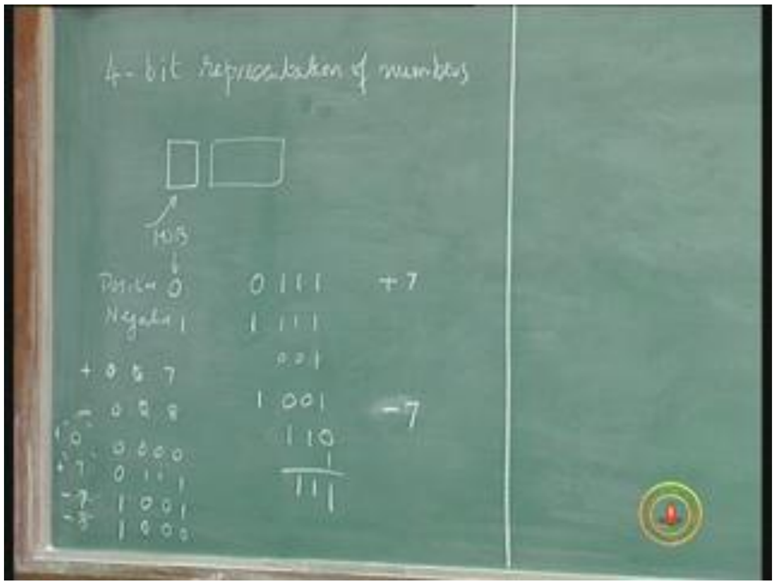
So if I put the number now 1111 because the number is 1 this is negative so I will have to take a 2's complement and the 2's complement of this is 000 of course I need not worry about this bit any more because this is only a sign bit and having taking the 2's complement of this these are the magnitude bits and for the magnitude bits have to take the 2's complement 0000 plus 1 so this is one extreme (Refer Slide Time: 7:14).

(Refer Slide Time: 7:10)



This 1 111 is 1 extreme and the other extreme is 0000 so 1 followed by 000 again this is a negative number you take 1’s complement 7 but if you add 1 to this then this will represent as the same number in 2’s complement 1000 and this is actually minus 8 because if you put let us say 6 suppose I have 1001 is a negative number what is the 2’s complement of this add 1 so minus 7 it is 1001. So 1001 is minus 7 and 0111 is plus 7 I still have 1000 which is a negative number because MSB is 1 but already we have reached minus 7 so I can use it to represent minus 8. That means positive numbers goes from 0 to plus 7 and negative numbers goes from 0 to 8.

(Refer Slide Time: 9:41)

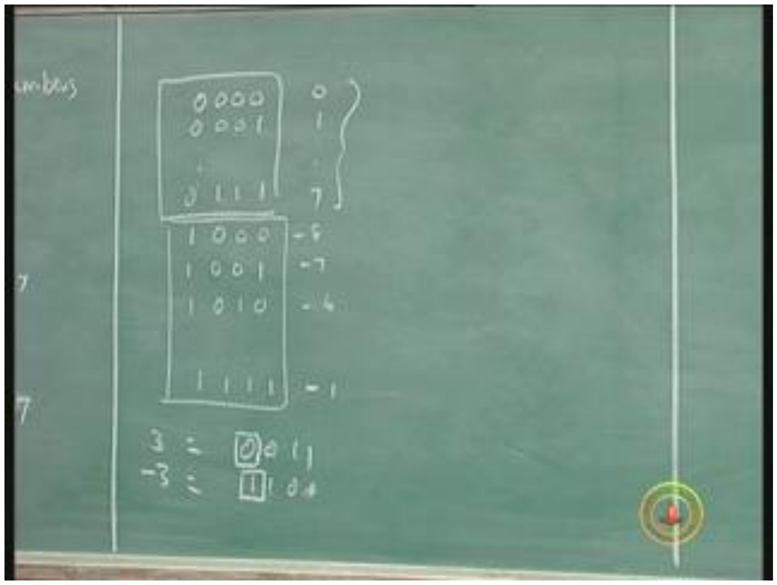


So 0 positive is 0000, 7 positive is 0111, 7 negative would be 1001 and minus 8 would be 1000. It is like you start with 0000 actually it is not very difficult 0000 is 0 and 0001 is 1, this is 017, what is the next number you know? It is 8, what is this number 1001, minus 8, this is 8 minus 8 minus 7 minus 6 (Refer Slide Time: 10:35) and 1111 will be what? It is minus 1 because if you take 2's complement of this 111 will give you 0000 1's complement add 1 to it so it becomes 0001 so minus 1 so I have 0 to 7. I have 0 to 7 and minus 8 to minus 1 the reason is I do not have to represent 0 two times that is why I am getting one extra number for negative. I don't have plus 0 and minus 0 I have only one 0 so 0 is only once. Hence 0 is represented as 0000 0000 is 0 so 1111 is minus 1 so if the numbers are 0 to 7 and minus 1 to minus 8 range of numbers with 4-bits is 0 to 7 positive minus 1 to minus 8 negative.

Now in order to represent the number I will have to tell you how many bits are there and give you the sign bit clearly that means the arithmetic operation is carried out only with magnitude bits, sign becomes automatically taken into account based on the addition process. So now if you want to represent two numbers, we will do two types of additions no problem, the result will always be 3 bits. That means I have range of numbers I can represent only 0 to 7 or minus 1 to minus 8 which means I can only represent the magnitude of these numbers using 3 bits. So all the arithmetic operations should also result, the sum or the difference should always be less than 3 bits in magnitude, this 4-bit number with sign magnitude form a 2's complement representation I can only 'carry out' additions and subtractions whose results does not exceed 3 bits whether it is positive or negative.

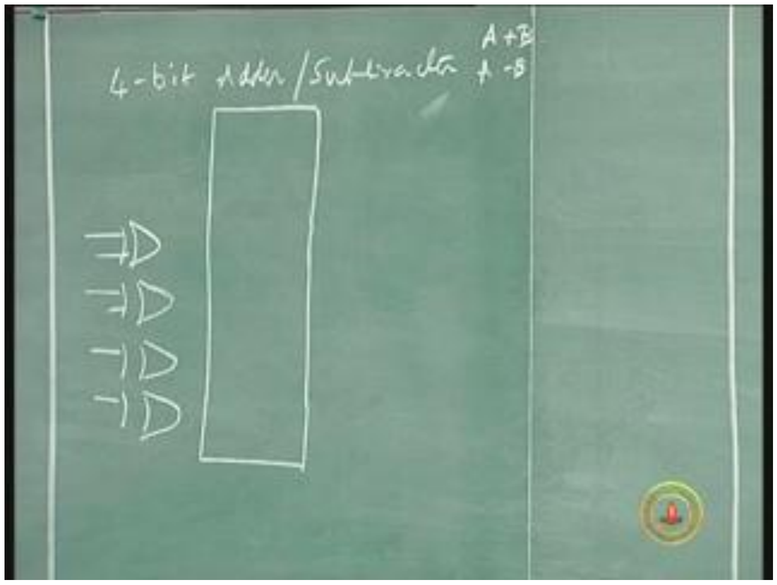
With that background if you look at the number if you have 3 the first thing you have to ask me is whether it is plus 3 or minus 3 you are talking about and you say plus 3 it is 0011 so I will say 4-bit representation I already said 4-bit representation so 3 would be represented as 0011, minus 3 will be represented as I will have to take a 2's complement of this which is 1100 add 1 to that and it becomes 1101 so this is a positive number this goes to the negative number 101 1011.

(Refer Slide Time: 13:37)



Now if you look at our addition or subtraction being a 2’s complement addition which we showed in the previous lecture and 2’s complementing can be done by passing the bits through a excusive OR controlled by a bit and you decide whether the inversion is required or not required then I can draw a general 4-bit add subtract 4-bit adder subtractor using these.

(Refer Slide Time 15:15)



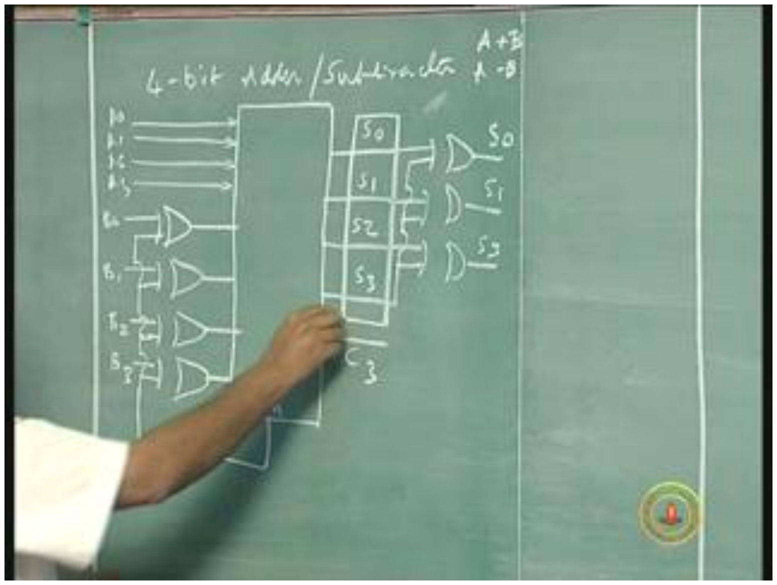
These are the numbers to be added As and Bs drawn into A plus B or A minus B so A will be directly given A_0 A_1 A_2 A_3 A_3 being the sign bit of number A that can be positive or negative A_3 is the sign bit. Similarly B_3 is the sign bit B_2 B_1 B_0 and this will be my add bar subtract I will call this signal add bar subtract what does it mean? If it is 0 addition operation, 1 if it is subtraction operation, 0 for add operation and 1 for subtract operation. if this bit is a 1 this gets inverted and if this bit is 0 it does not get inverted that's what we saw Exclusive OR gate, the bit gets inverted if the other bit is a 1 the other bit is 0 the bit does not get inverted. So, for subtraction we want addition but we do not want inversion. So B will go through this Exclusive OR gates and become plus B or minus B as required for adding or subtraction. And also the subtract signal will be used as the 'carry in' signal. For addition the 'carry in' is 0 and for subtraction 'carry in' is 1 because of the 2's complement we said, this can only give you 1's complement we have to add one more to the 1's complement to get 2's complement so in order to get a 2's complement I have add extra 1 which is given here.

Now the result would be S_0 S_1 S_2 S_3 is carry significant or not now, because the 4-bit adder will give you four sum bits S_0 S_1 S_2 S_3 and a carry bit, we shall call C_3 but does C_3 has any use? No, because we are only talking of 3 bits now and the fourth bit is a sign bit and our number will always be, the result will always be 3-bit magnitude of the sing bit, the S_3 bit will be the sign bit, S_3 bit will be sign bit, S_2 S_1 and S_0 would be magnitude bits. If the number is negative S_3 would be 1 and if the result is positive S_3 would be 0.

So we can now 'carry out' addition subtraction A and B can any take any values within 0 to 7 or minus 1 to minus 8 and the result can be only either 0 to 7 positive or minus 1 to minus 8 negative and that will be represented by the sign bit S_3 and if you want to get the original number you don't really need it because we wanted 2's complement representation we have given the number we represented it and made a subtraction, 2's complement of number is given to us. The number was given to us and we complemented and subtracted it adding extra 1.

If you want to get the original number back in positive form whether the result is positive or negative you want the magnitude only what I should do is to do a 2's complement one more time at the output and that can be done using the S_3 bit. If the S_3 bit is 1 again Exclusive OR it if S_3 is 0 I will pass it through that. Supposing I will pass it through another set of Exclusive OR gates just only 3 bits and this would be used to complemented if required. If the number is negative C_3 would be 1 so I can use it to complement the S_0 S_1 S_2 otherwise it goes as it is like the magnitude alone. You don't need it actually as I said. you would like to keep it in this form because later on you want to go to the next computation again you have to get the 2's complement representation of the given number so it is good enough to have this result (Refer Slide Time: 20:44).

(Refer Slide Time: 20:40)



But in case you want the magnitude of this number take the 1’s complement and then you have to add 1 and I need one more adder to this so I will have to put this into an adder it is nothing but a 3-bit adder I need to have a fourth bit a 0 input and then one input, this is S_1 and this is S_2. This will be the magnitude of the negative number. if you had negative number is the output there are two ways of representing it, either you can represent as it is S_3 S_2 S_1 S_0 will give you the negative number in 2’s complement form you can keep or store it as it is use it as it is in the next stage of addition or subtraction that is what computers do, computers do not keep on changing every number to sign magnitude, we are the ones interested in the sign magnitude. We want to know whether the number is negative or positive and the magnitude, we would rather be happy in knowing that some result is minus 17 rather than giving the 2’s complement version of it.

But a computer doesn’t have to, in computers if minus 17 is again used in next stage why should you convert it into minus 17 and I again convert it into 2’s complement and then use it so most of the time this is valid. if this is final stage you want to display positive number or negative number so you want to again convert it back to magnitude and some of the sign has to be represented here, this would be the sign, so the sign would be represented as minus sign some sort of display in which you can say this is minus because S_3 is 1 and I will use this minus to invert this 3 bits to get 1’s complement add 1 to it and so on. This is an extra circuit which is unnecessary but if you really insist that my final result should be displayed as a sign magnitude representation not a 2’s complement representation then you can do that.

Anyway the crux of the matter here is, you should know the number of bits before hand, you cannot just start taking a number given as it is to you, taking inverse and adding 1. I gave you the example of 3 when I say 3 you are going to say 11 and think it is a negative number and to start again the 2’s complement is not required.

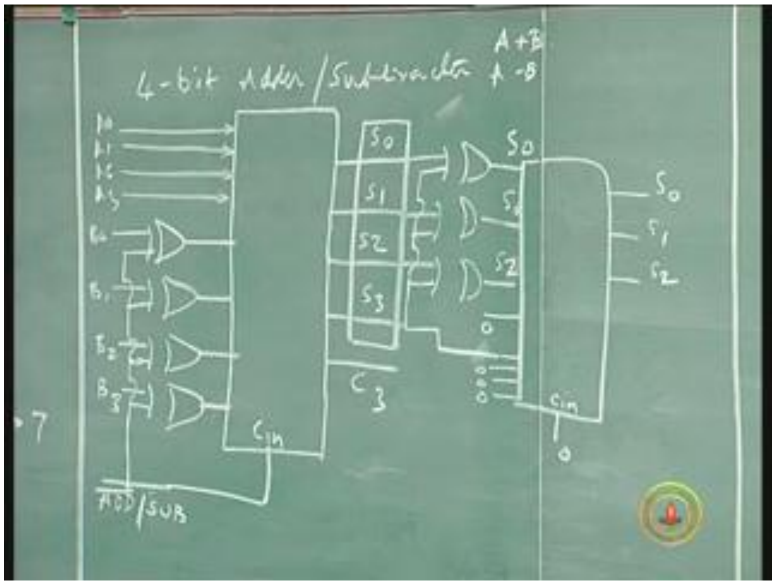
If I tell you 3 using a 4-bit representation in which the fourth bit is a sign bit then you know that it is 0011 so it is a positive number you won't do anything but retain 11. So first question you should ask is how many bits you have to represent my numbers and does it include sign bit or not. if I say 4-bit including sign bit then you know one bit is represented for sign my numbers can be represented for only 3 bits and if I say further it is a 2's complement representation I know then I can only represent 0 to 7 and minus 1 to minus 8. Then I know this is how I represent. Any number outside this range cannot be represented using this representation of 4-bits with one sign bit. So if I tell you add 10 to 6 you will say I cannot add 10 because I cannot know how to represent 10 because I can only represent 3 bits and 10 is outside my range. Or even if I say add 5 and 4 you will have to say that even though 5 and 4 both represent 3-bit representation both are positive numbers when I add it allow a overflow I have no way of handling it.

If I try to add 5 and 4 it would be 9 and 9 will be 1001 it will be misinterpreted as negative number 1001 you would think as a negative number because sign bit is 1 and you will simulately start taking of the complement so if you add plus 5 and plus 4 plus 5 is 0101 and plus 4 is 0100 so you add and get 1001. So immediately I see this is a negative number because I have 4-bit representation and the fourth bit is the sign bit so immediately I start again 2's complement of this which will be 0111 and this is 7 (refer Slide Time: 25:57) which is wrong obviously this has to be 9, why is it happening? It is because this exceeds the number, the number 9 exceed the 3 bits allotted for the magnitude, 9 is 4-bit magnitude I have allotted only 3 bits for this magnitude representation and since exceeds that, it is the overflow error. Of course we will not talk about those here, it is called overflow.

So, first question is how many bits are there, second question is in what form is the number represented. So it is 4-bits and one of them is sign bit then you say 2's complement representation of S then the negative number you convert it into 2's complement and represent it then do the addition or subtraction always making sure the result is less than what can be represented using the 3 bits magnitude and one sign bit.

And further if the result is negative you can leave it as it is. The best way is to leave it as it is and say it is the negative number because S_3 is 1. People understand it is the negative number so the value is not that, if you want to know the actual value you will have to get the 2's complement. But for some reason you have compulsion to display it as a sign magnitude number you have to do one more 2's complement which will involve extra hardware.

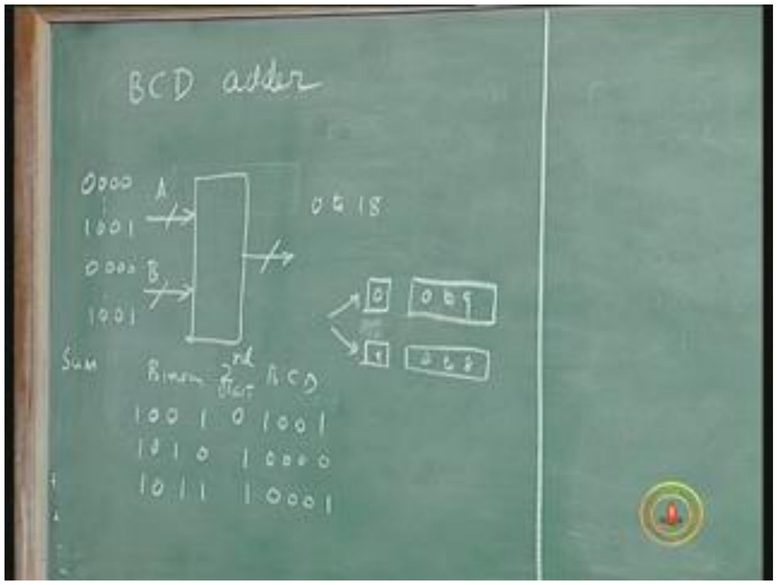
(Refer Slide Time: 27:20)



So if you want you can always look at this as a special case. S_3 will be anyway there. There will be 1000 and normally you look at this as the magnitude and in the case of 100 alone we will look at S_3 also. This is the subtractor adder, we almost completed the adders we talked about half adders, we talked about full adders, we talked about multiple bit adder multi-bit adders, we talked about carry propagation, we talked about carry look ahead to speed up, we talked about subtraction, we talked about complementing numbers, we talked about 2's complement and 1's complement we said how 2's complement addition is same as subtraction, we also saw how a common hardware of adder can be used for addition as well as subtraction by taking the 2's complement of the number to be subtracted and one variation is this is the BCD.

Suppose I am talking about BCD, BCD I already mentioned to you earlier about Binary Coded Decimal. Suppose I have only 9 I have used decimal number system and because the computer or the hardware that we design deals with binary numbers I have to necessarily convert my numbers into binary. But I am not comfortable with numbers beyond 9 because I am used to decimal representation. I will feed in all the numbers as decimal numbers the computer will have a hardware which will convert my decimal into binary and do the addition, subtraction or whatever it is and I will like to have my result again back in decimal representation by means of a 7-segment display. That means with the result of the BCD addition I will also like to have a BCD. That means if I have two numbers whose values are from 0 to 9 I will like to have a result which is from 0 to 18 and not beyond that. Hence this is called the BCD adder.

(Refer Slide Time 34:08)



Suppose I have two numbers I will put A B you know what I mean by this arrow with a slash, this means that there is more than one bit so let us say 4-bits so 0 to 9. Therefore the number can go from 0000 to 1001. Similarly this can go from (Refer Slide Time: 30:21) so what is the number now 0 to 9? For 0 to 9 we have the result as 0 to 18 that's what I like to have. But on the inside it is a binary adder so up to 9 there is no problem, at 10 it becomes 1010 but I would like to have a 10 as 1 followed by 0. If the result is 11 suppose this is 8 and 3 so as far as this is concerned this is a binary adder it is not a BCD adder.

BCD adder is only in my mind, my concept is BCD but the adder is the binary adder, this is 3, this is 8 so the result is 8 plus 3 is equal to 11 and what is 11 in binary? It is 1011 so it will come as 1011 whereas I would like to represent it as 1 followed by 1. Thus I would like to have 1 followed by, I don't have to worry about 2-bit because there is 1 bit either a 0 or 1 here. So I like to have a bit which is 0 or 1 and if it is from 0 to 9 it is no problem and from 10 onwards I will have to have 1 followed by 0 to 8. That means whenever the number exceeds 9 I will have to do a little bit of something to bring it to what I want.

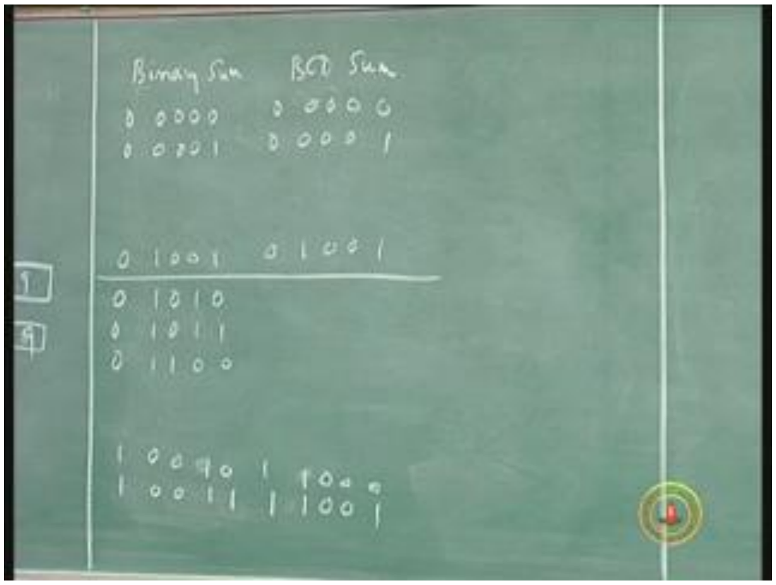
Now what can I do to get a BCD representation of a binary sum? If the number is 9 binary BCD I am only talking about sum, and BCD also is 9 so no problem. And when the number is 10 I would like to have it as 1 followed by 0000, 1 followed by a 0 this is my BCD second bit (Refer Slide Time: 33:01) this is my second digit here, the second BCD digit, 1 followed by 0000. So from 9 it has to come to this, this would have been in 10 but I want it as this 1 followed by this.

If you take the decimal equivalent of this number what will be the number? It is 00001 is equal to 16 so 1011 is 11 but I want to have this displayed as 1 followed by a 1 so it is again 1 followed by 16. Hence if I add beyond 9, for 10 and above 10 all I have to do is to add a 6 to the output. If I add 6 to the sum beyond 9 I will get my two representation in

BCD form and this I can represent separately by an LED or something saying that when it glows it is more than 10 and when it does not glow it is less than 10, I can say so. This can be a simple LED which cannot glow for numbers 0 to 9 but glow for numbers 10 to 18. So a simple LED will do. But this number I have to modify from 1010 to 0000 1011 to 0001 which can be done by adding 6 to that.

So if you want write the full truth table for this binary sum BCD sum you can complete this table yourself I will just tell you what it is, 0 followed by 0001 so up to 9 there is no problem. Binary sum will be same as the BCD sum. The binary sum is 0 followed by 111 etc up to 18 maximum and beyond 18 we do not have to do, what is 18? 18 would be 10011, this is 3 plus 15 18 so actually 19, 19 also can come back, it is not going to happen, up to 19 I have a display because this can be 1 followed by 9.

(Refer Slide Time 36:51)



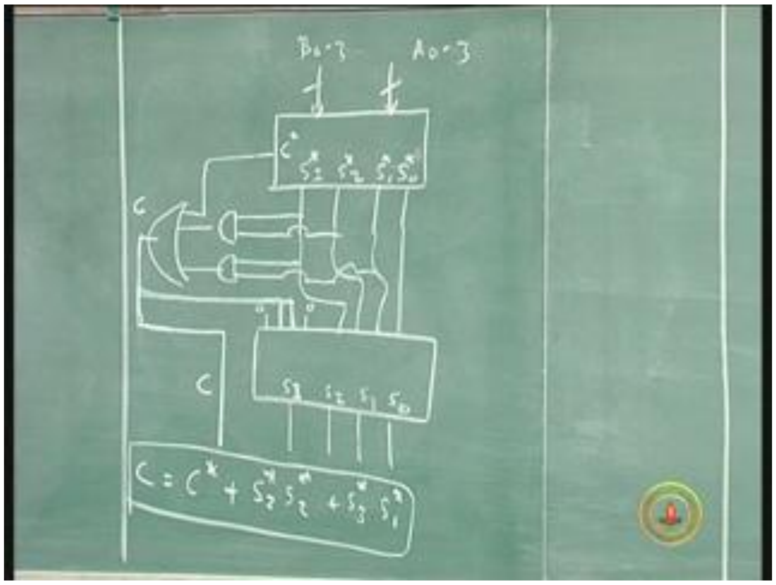
Binary Sum	BCD Sum
0 0000	0 0000
0 0001	0 0001
0 1001	0 1001
0 1010	
0 1011	
0 1100	
1 0010	1 1000
1 0011	1 1001

$\boxed{1}$
$\boxed{9}$

But in practice I will not have 19 because I cannot represent…., each of them can only be 0 to 9 when two numbers of 9 9 plus 9 can be 18 but in practice it can only be 18 and in theory it can be 19. So for 19 this will be 1 followed by 9, if it is 18 it is 1 followed by 18 and so forth. This 10 would be 1 followed by 0000, 11 would be 1 followed 0001, 12 will be 1 followed by 0002 and etc.

It is this portion which has to be taken care of and this portion is no problem. I am going to give you this circuit and later on explain how it works to save some time. We have the normal adder in which I gave the two BCD numbers A_0 to 3 B_0 to 3 and the numbers will be anywhere from (Refer Slide Time: 37:47) we will call this S_0. I will put a S_0 S_1 S_2 S_3 this is not a final sum so I will not qualify this, I will put a star here, may be final sum will come later on after adding 6 to that and then a carry.

(Refer Slide Time 46:21)



Now I am going to have one more adder and add 6. I need to add a 6 to that. Let me first draw the circuit and explain. So this will be added with another adder in which this is this, so to do that I would add 6 and because this is 4-bits this 6 has to be added so it is a two 4-bit adder. Therefore instead of 5-bit adder I am going to use two 4-bit adders but I cannot add always I can only add whenever it generates a carry here. Whenever the 6 has to be added this has the original sum (refer Slide Time: 39:02) that has to be first modified and then added so this will be 2.

When do you have to add 6? I have to add 6 whenever the 6 sum exceeds 9. So how do you know that the sum exceeds 9? It is by this part of the truth table. If you study this part of the truth table it happens whenever these two bits binary sum this is binary sum S_3 S_2 S_1 S_0 S_3bar S_2bar S_1bar S_0bar this is binary sum so either it is this carry which is the binary carry this is C bar. So I have to add 6 for all these cases where C star is 1 or when these two are one, for this and this the bits are one (Refer Slide Time: 41:12) or when this and this are one. These first two terms are taking care of this and all the other terms are taking care of these two and last term is taking care of these two. So I have to add this, we will call this the final carry which is what is required, the final carry is the temporary carry with the binary sum or S_3bar S_2bar or S_3bar S_1bar and this carry is also required for us. This carry will be required for the entire addition. Whenever I need to add 6 to the original sum I have to add 6 to the sum when the number or the sum exceeds 9.

As soon as the sum exceeds 9 I have to have a carry addition of 6 which means I have already got into my second digit. So whenever I have 6 I also should get my extra digit this is that extra digit I am talking about which can be 0 or 1. If it is 0 the number is less than 9 and if it is 1 the number is greater than 10, for 10 to 19 this would be 1 and for 0 to 19 this would be 0 that is what I meant by putting a simple LED to show that the number is less than 9 or greater than 9.

When do we add 6? We add 6 whenever the number exceeds 9. How do you know when number exceeds 9? It is by looking at this. So this is the combination. Whenever the number exceeds 9 we are going to have this combination of either the last bit is 1 carry bit, 1 that is 16 17 18 19 and up to 15 this will be 0, this is 16 17 18 19 (Refer Slide Time: 43:30) and the last four rows of my truth table, I want you to complete the truth table as an exercise this truth table will have 9 here, another 9 here that is 10 to 19 so there are 10 entries that is from 0 to 9 it is 10 entries and another 10 entries. Here in the last four rows anyway you have to add 6 or when this and this that is the last two bits of the original sum are 1 then we need to add 6 or when the first S_3 and S_1 this and this are both 1. So for this if you write the full truth table you will find that the pattern will emerge.

Whenever there is a 1 here or whenever there is a 1 here and here or whenever there is a 1 here and here these are the three conditions where you have to add 6 and that is what is represented here.

If the original addition results in a carry which is for a number greater than 15 this C star takes care of 16 17 18 19 because 4-bits can only represent 0 to 15 but the number can be anywhere from 0 to 19 theoretically and practically 0 to 18 so I need to have four conditions four possibilities of the number being 16 17 18 19 taken care of by this or whenever the number is greater than 9.

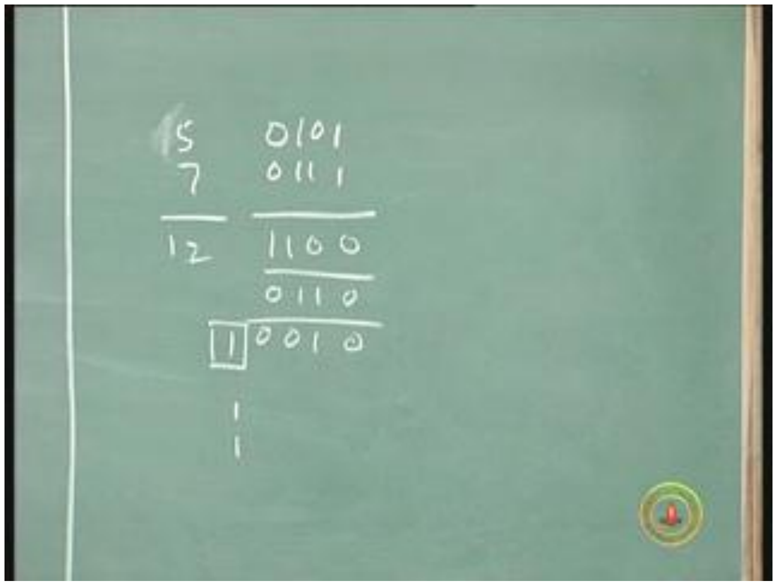
How will you represent the number greater than 4-bits, number greater than 1010? You start from 1010 right up to 1111 you will find that in this case I can't put 1 here directly because this 1 will also be here so when this is 1 and this is 1 or when this is 1 and this is 1 so for these two conditions also it is taken care of the number exceeding 9. So this is the condition for which I need to add 6 so 6 is nothing but 0110, this 0 I will put permanently and this 1 I will take whenever this output is there so this OR gate implements this function, this Boolean function should implement this OR gate.

Thus whenever these OR gate has an output 1 we will have one adder and at the same time we will have this as the carry. So the 18th number is S_0 S_1 S_2 S_3 plus carry that is the BCD addition will come out of this. The 6 has to be added, no problem. I have to take the binary addition I need to take another binary adder 4-bit adder and add 6 but it is not that simple. I can only add 6 under certain circumstances. What are the circumstances under which I have to add the 6? It depends on whether there is a 'carry in' the original addition or whenever the number exceeds 10. So whenever the number exceeds 10 so these two terms (Refer Slide Time: 46:55) represent the condition when number exceeds 10, number is 10 to 15 in these two terms. This term exceeds 16 17, 16 17 18 19 that are taken care of by this term, these two terms take care of the fact that the number is between 10 to 15 that is 10 11 12 13 14 15.

So once one of these three conditions are met I need to add 11 0110 is 6, the original number plus 6 gives this. Otherwise this will be 0 and if the result is 0 to 9 this C will be 0 so I will be adding 0000 that means the original number will come out as the final number and this also will be 0. This is BCD addition this is what happens really because I need to have a BCD representation only for display purpose again.

If you are using this as an intermediate result for the next processing you can treat with all binary numbers. But if you are taking digital numbers decimal numbers and treating them digit by digit, then in the calculator for example I need to punch in decimal digit since I want the result as decimal but inside that there may be a binary digit but I only want decimal input and decimal output. This is what happens in the calculator.

(Refer Slide Time: 49:35)



Suppose you try do a simple calculation of 5 and 7 and the number is to be 12 and if 5 will be converted as 0101, 7 would be converted as 0111 the result would be 1100 no problem. But I want the result to be displayed as not 1100 but I want the result to be displayed as 12 that means I have to add 6 to that and it becomes 18 and out of that 18 that 1 will come separately and that last one is the 16 number, if you remove that 16 the remaining is 2 and that 2 will be converted as 2 and it will glow in the 7-segment display. So it is 12 is equal to 5 and 7 so no problem but I do not want this so add 6 to that and this one will come as 1 by your 7-segment display, these two will come as (Refer Slide Time: 49:42) in a 7-segment display and that is why you are able to see 12 in your calculator and not 1100, this is the point. So I should add the 6 because this is more than 9 and for 10 and above I need to add 6 and that’s what this is.