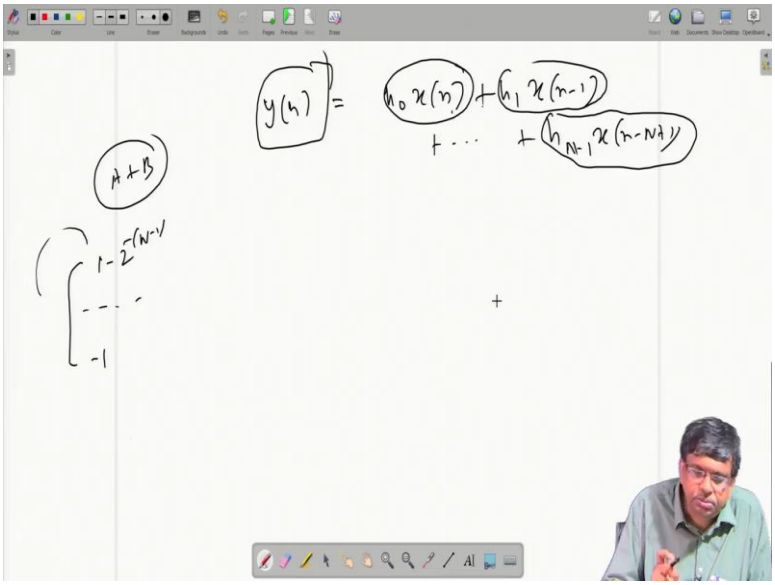
VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 34 - Multiplication of Two Binary Numbers

Now we will do something about a bin advantage of the two's complement number system.

(Refer Slide Time: 00:48)



Suppose you are adding like digital systems, digital filters and all that, you add like you carry out filtering operation. So find out y n is a convolution sum like h 0 x n plus h 1 x n minus 1 and dot dot dot dot up to maybe h, this is one term, this is another term, another term you are adding. You do multiplication and then you are adding.

Now if whenever you add two numbers say A and B, if the resulting value is within the range, what is the range of the two's complement system or w bit system? On the positive side, it is this upper half and lower half is minus 1, this is the range and the 0 between. If the sum is lying within this range then there is no problem, there will not be any overflow, w bits are good enough to represent the resulting one uniquely.

But in the resulting sum, has an overflow then 1 bit will go. When you add one bit from the most significant bit, we will go further to the left and that will be lost. So whatever w bit result you get that will be wrong. So overflow is a problem, you need to check overflow and then make corrections accordingly, is a problem. In general, in addition. Now here what happened? Suppose you are carrying out a summation like this and it is somehow told that final summation that is this result is within this range for w bit representation within this

range. So this is fine, so if you give me w bit, you can always represent this nicely without any overflow.

Suppose this is given to you, but the way you are calculating, you are calculating so many additions. So it is quite possible that some intermediate sums because you always take two at a time as an adder, adders are two input, one output device. It can take only this adder for that button. You can take only two fellows as input and give one fellow as output and that would take and with another fellow and like that. So many intermediate summations. It is quite possible that even if the final result somehow you guarantee to lie within this range, intermediate summations, so you do not have any control, maybe they will lead to overflow.

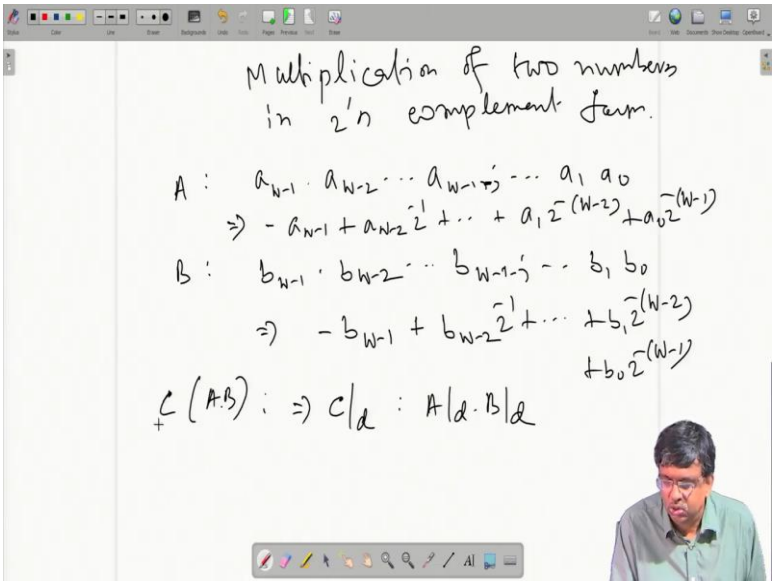
Ultimately, somebody I mean cancel out some something and finally, result will come back in this range. But some intermediate summations can give rise to some overflow. If that overflow is not checked there will error, there lies advantage of two's complement system that if you follow two's complement number system, even if some intermediate sum here on this side has an overflow, there is a theorem that states that you just ignore that overflow as we did.

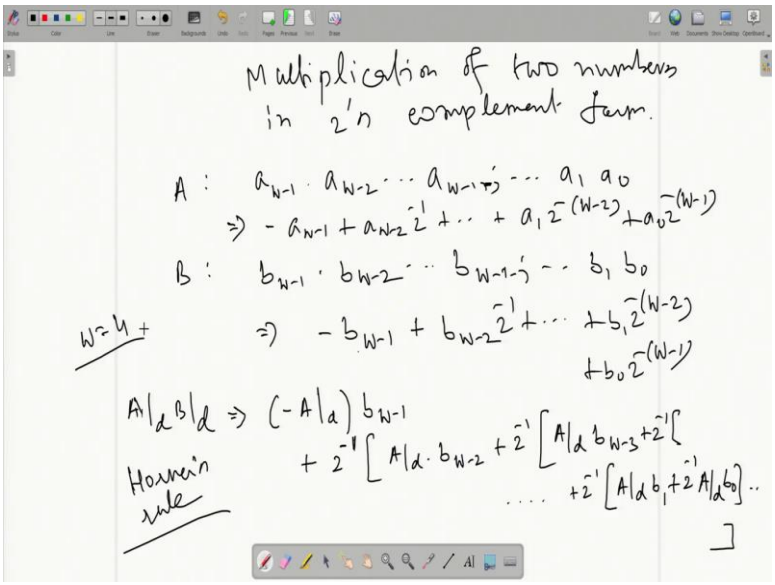
Whatever result you are getting that may appear to be wrong, but please move with that result, carry on with the entire summation, eventually the result that we will get that will be the corrective again. There is a theorem which looks at advanced mathematics but that is the beauty of this two’s compliment number system.

That is whenever you are adding too many numbers, if the final number is guaranteed to lie between minus 1 to this 1 minus to the power minus w minus 1, there is the range of a w bit number. If that is guaranteed then even if there are intermediate overflows, that you are doing so many additions, even if some intermediate overflows and some intermediate sums, you can just ignore them and carry on with summation as it is, you do not have to worry, final result will be as it is, will be correct.

This is the property of two's complement number system, which is very useful in digital signal processing because here you come across convolution sum, so too many things we added or you do a transform, discrete cosine transform or DFT, where again in the summation you have too many sums to added and like that. You are guaranteed to have overflow free operation result. This is the main advantage.

(Refer Slide Time: 04:44)





Now we consider multiplication, multiplication of two numbers in two's complement form. Suppose you are given A, which is a, w minus 1 dot a w minus 2 dot dot dot. You know the decimal value under two's complement system. And you know this intermediate thing a 1 2 to the power. Similarly here, now digital multiplier means you develop a circuit, develop some digital hardware, where one input you give as A w, there is a bus, w bit bus that carries the all bits of A. Similarly, another w bit bus that goes in, that carries out all the bit of B and then you do some processing inside, that result should be such if you call it C, which is actually A, B. A dot B, A and I want the product.

This should have, this should be a w bit number, this would be number somewhat, whose decimal value should be equal to product of the decimal of A into product of this.

There is this C should be such that C of decimal, I want to generate as output C so that the decimal value of C should be A by d times B by d. Then that hardware will be called a digital multiplier. That is it takes A w bit numbers it takes B w bits numbers, does some kind of processing, generates an output C so that the decimal value of that C under two's complement system will be nothing but product of decimal A and decimal B.

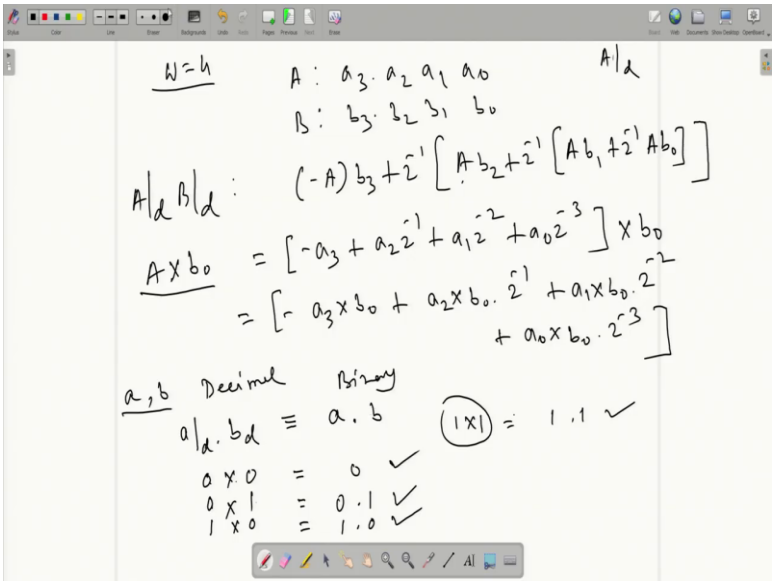
If you can do that, then that is called a digital multiplier. So that is what we will try to work out. We will follow what is called Horner's rule A by d, into B by d, B by d I write like this and A and this entire thing to be multiplied by A by d. So A by d times this which is also you can write as minus of A by d times this decimal thing.

Whenever you write this expression, everybody treat it as a decimal digit, so value remains same. It is originally binary 1, now it is 1 decimal 1, if it is originally binary 0 now decimal 0 but it is decimal digit now, that is why I can put minus here I can multiply by 2 inverse and all those. This we cannot do when it is purely binary B then operations are very limited, 1 plus 1 is 0 with carry 1 or and gate all those things, you cannot write like this.

So product will be the decimal value of A by d into the entire decimal expression, which is A by d times this. I take minus with A by d this, then 2 inverse I take common, so A by d times this then I had 2 to the power minus 2, 2 to the power minus 3, so again I take another 2 inverse common, so again A by d times next bit, again take 2 inverse common and dot dot dot dot dot final you will have 2 inverse common A by d times b this 1 1 and again, just last two inverse times A by d times b 0, You can put this also under a bracket but no need, dot dot dot this. So it is a nested step function, this rule is called as a chain rule called Horner's rule.

There is function within a function is nested. 2 inverse A by d times b w minus 2 plus again 2 inverse A by d times b w minus 3 plus again 2 inverse and so on and so forth, similar thing. Now to show how this can be used to build up a digital multiplier, we take, we will take an example of 4 bit numbers. That is w will take to be 4 as an example. So it will be a 3, a 2, a 1, a 0, b 3, b 2, b 1, b 0.

(Refer Slide Time: 10:46)



So w 4, I got A is nothing but a 3 dot a 2, a 1, a 0; B is nothing but b 3 dot b 2, b 1, b 0, I repeat whenever I write this expression, these are binary so that time it is binary 0 or binary 1 binary 0 binary 1 all and the arithmetic they employ is binary arithmetic. Here I cannot put a minus and all that but the moment I take the decimal expression under the two's complement system formula, there I can take them as decimal variables and do everything I want.

Now, so I just simply apply the Horner's rule here for this for example, so A by d times B by d this product will be what minus A times b 3 plus 2 inverse A times b 2 plus 2 inverse A times b 1 plus 2 inverse A times b 0. This is what I have. Now let us start with A b 0, here this A actually A by d.

But, I am not writing it, is understood that is A by d is a decimal value. So these values are actually A by d. But I am just not writing the by d part here. I hope you understand that in this decimal expression so whenever I write capital A, they are all decimal capital A. Similarly, b 0, b 1, b 2 they are all decimal values, decimal digits.

So let us start with A b 0, decimal A times into this into decimal b 0, but decimal A this decimal A, this is again now minus a 3 plus a 2 into 2 inverse, a 1 2 inverse 2, a 0 2 inverse 3. This into b 0 which is minus a 3 into b 0, plus a 2 into b 0 times 2 inverse, a 1 into b 0 times 2 inverse 2, plus a 0 into b 0 2 inverse 3.

Now before I proceed further, let us see one thing, this is the decimal expression. What is small a 3 or a 2, similarly b 0. There are all, originally they are binary bits. Now in this they will treated as decimal digits but value remain same, binary 0 means here it is decimal 0;

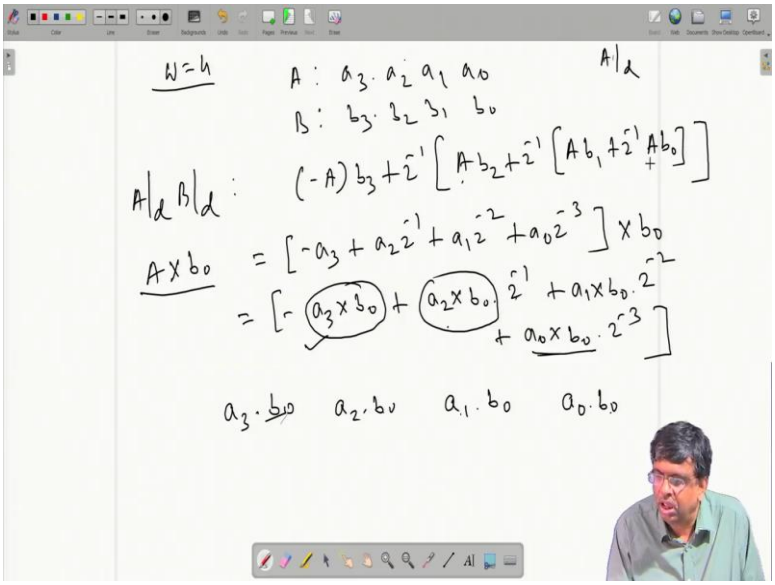
binary 1 means decimal 1 so on and so forth. Now, let us see one property, suppose I give you two binary bits a and b, if it is a by d times b by d, my claim is this is equivalent to, in binary domain this is decimal domain.

And in binary domain, it is equivalent to and operation this and of binary a and binary b, that is if I do binary a dot binary b whatever bit I get, its decimal value will be same as decimal of a times decimal of b, you can easily see suppose a and b both are 0. Suppose a and b both are 0, so a by d is 0, b by d is 0, so left hand side is 0 into 0, 0 decimal. Here it is binary 0 binary 0 but 0 AND gate 0 is the binary 0, its decimal value will be 0. So that is satisfied, this is 0 and this is 0, satisfied. Suppose this is 0, this is 1, decimal value is 1 decimal value is 0.

If decimal value of a is 0, that means in binary it is 0; if decimal value of this is 1, then in binary it is binary 1 and AND gate of them 0 dot 1 is binary 0 and binary 0 its decimal value is decimal 0. So decimal 0 on this side and 0 into 1, 0 on this side so this is satisfied. Similarly, if this is 1 by decimal 1, this is decimal 0, if you multiply you get 0, but if decimal 1 here it is binary 1, if it is decimal 0, here it is binary 0. And 1 dot 0 is a binary variable whose value is logic 0, binary 0. So its decimal value is again simply 0 only. So this is satisfied and last one if both are 1, then 1 cross 1, 1 cross 1 on this side, but if it is 1 here it is binary 1; if it is the decimal 1 it is binary 1.

So and 1 dot 1 is a binary 1, binary 1 has decimal value equal to decimal 1 only. So again, this side, decimal 1 this side its decimal value is decimal 1, so this is satisfied, which means if I give you two binary bits A and B in decimal domain, if you multiply their decimal values, decimal value means if it is binary 0, remains as decimal 0; if it is binary 1 remains as decimal 1. If you multiply them, it means in binary domain you treat them as binary bits and do AND gate first and operation between them, whatever binary variable you get out of it this a dot b, that will leave the same decimal value as the product of the decimal of a and decimal of b. So in binary domain, I simply have to represent that as a dot b.

(Refer Slide Time: 17:00)

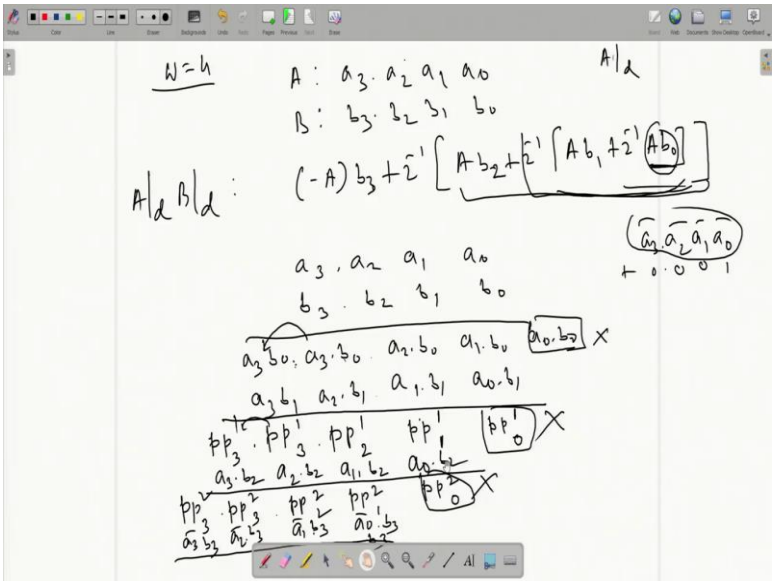


That means this is the decimal expression. This is corresponding to a binary word, what word? This is decimal bit, decimal digit, decimal a 3 into decimal b 0; in binary domain, it will be binary a 3 dot binary 0. This will be here it is decimal a 2 into decimal b 0, in binary domain it will be a 2 dot b 0, I am just first writing them. Similarly, a 1 decimal a 1 decimal b 0 product, it will be binary a 1 dot binary b 0 and in a 0 cross b 0, a 0 cross b 0 means binary a 0 binary b 0.

And if I put them side by side, I get a 4 bit word. What will be the decimal value of this under two's complement formula? It will be, this will be treated as a decimal digit, so it will be decimal a 3 into decimal b 1, sorry decimal b 0, decimal b 0 with minus sign. So it is already there, plus 2 inverse times so 2 inverse. Decimal of this will be a 2 decimal into b 0 decimal is there into 2 inverse plus a 1 dot b 0 in binary domain. In decimal domain, it will be decimal a 1 times decimal b 0 present into 2 inverse 2 plus again it is a 0 dot b 0 in binary domain.

So in decimal domain, it will be decimal a 0 times decimal b 0 into 2 to the power minus 3 that is what I have here. So any A times b 0 decimal value of A into decimal b 0 will give rise to a 4 bit word here, it will be just like this. If A is capital A so a 3 dot b 0, a 2 dot b 0, a 1 dot b 0, a 0 dot b 0, so that is what I have.

(Refer Slide Time: 19:02)



Here, so start with A b 0, capital A b 0 means if I form a table, A b 0 means a 4 bit word, a 3 b 0, a 3 b 0 means a 3 dot and AND operation. dot I am just dropping it is there. If you want, I can put also. a 2 dot b 0, a 1 dot b 0, a 0 dot b 0, alright. The 4 bit word, but this is to be multiplied by 2 inverse, this decimal value to be multiplied by 2 inverse, which means in corresponding binary word, what will happen?

Everybody will get shifted to the right by 1 and this is the MSB that will get the sign bit. Whether there is binary point here, there is sign bit that will get extended. So I should shift them to the right and extend it or I can do this way also, I put the binary point from here to I move here and extend this.

I will get the same thing, you can easily verify, a 3 b 0 should become should go to the right of binary point and again come back. So it has gone to right of binary point and again come back, a 2 b 0 was just to the right of this. It will be shifted further, so binary point here so it is shifted further and everybody shifted further. And this guy has gone out of my register because register is 4 bit. This is what I have. So there is 2 inverse a b 0, then a b 1 same logic again, a 3 dot b 1, a 2 dot b 1, a 1 dot b 1, a 0 dot b 1.

You add whatever you get, this is called and I get some intermediate result that is called partial product pp, pp first stage first pp. So it is pp1, but is the LSB, you add the two. Incoming carry is 0 to odd adder, result is here that will generate a carrier that will move to this side. So this carry plus these two will be added that will give rise to a result pp1 next bit and again a carry will move to this side. This will be added, that will give us pp1 but second

next bit and they will get added. So you have pp1 3, this will be this result, then this has to be multiplied by 2 inverse.

So my binary point was here and this should be shifted to the right by 1 and then this should come back here. You can as well move the binary point to this side and we extend this and become 5 bits. So LSB it is checked out, see what this result. There is this summation. Now again multiply by 2 inverse means sorry, now this is it is this much. Now A b this A b 2 this got erased A b 2. So we took again binary word a 3 dot b 2, a 2 dot b 2, a 1 dot b 2, a 0 dot b 2. So a 3 dot b 2, a 2 dot b 2, a 1 dot b 2, a 0 dot b 2, we add the two, incoming carry is 0.

So this partial product of the second next stage, so stage number 2, 0 is bit, carry generated moves here. These two plus carry will give us two new bit pp2 1, again carry moves to this side, these two plus carry it will be pp and carry moves this side pp and that is all. And then 2 inverse times that, so there is a binary point here. Instead of shifting everything to the right, I move the binary point to the left as before and this is pp3 2 this guy comes here.

And for sure I do not have space here. So next one is minus a into b 3, minus a 2 into b 3 means, what is minus a? In case of minus a in binary word it will be complement each bit and add 1 at the LSB. If I come to that is, it will be a 3 bar, a 2 bar, a 1 bar, a 0 bar, plus 0, 0, 0, 1, there is a binary point here and these together to be AND gated with b 3.

So I will take AND gate of this part with b 3 and AND gate of this part with b 3 and add. If I add AND gate of b, this will be a 0; by the way first let me erase this, that this is chucked out, so should be a 3 bar b 3, this is b 3, a 2 bar b 3, a 1 bar b 3, a 0 bar b 3 dot gate and then again here, 0 dot b 3 is 0, 0 dot b 3 is 1 dot b 3, so only b 3 will come here. You add, you get the result.

So this is usage of Horner's rule. Now, the question is we have to develop a hardware. So what is the dependence graph here, you remember in the very early part of this course, I said that there are three graphical representations; one is called single flow graph, one is called data flow graph and the other one is dependence graph, dependence graph. So what is the dependence graphs of this? This I considered, this is a big thing so I will not start that here, I will consider in the next class. Thank you very much.