

System Design Through VERILOG
Prof. Shaik Rafi Ahmed
Department of Electrical and Electronics Engineering
Indian Institute of Technology, Guwahati

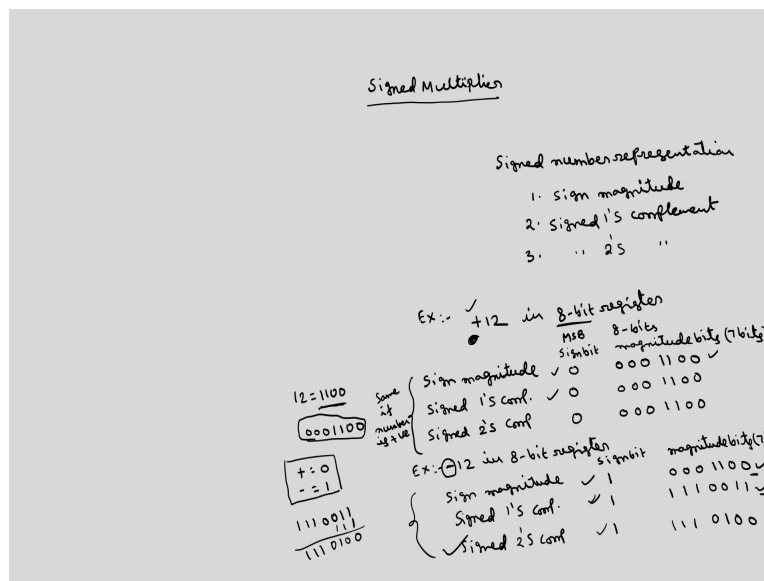
Case Studies
Lecture - 25
FIR filter implementation

In the last lecture, we have discussed about the Implementation of FIR filter. So, in that we have used the unsigned multiplier called Braun multiplier and then, we have shown the simulation results. So, the next case study that I am going to take is IIR filter implementation; but this time, I will use signed multiplier. Because in most of the applications, the data that we are going to handle can be negative also. So, a signed multiplier is very much useful in many applications.

So, here, I will take the implementation of IIR filter. So, in that a multiplier is involved ok. So, that multiplier I am going to implement by using a signed multiplier which is called as Baugh Wooley multiplier. So, before going for the implementation of the IIR filter, first I will discuss about the signed multiplier.

So, what is the mathematical background of this signed multiplier? This signed multiplier is not as straightforward as the unsigned multiplier. Here, you have to develop some mathematical analysis to implement a signed multiplier.

(Refer Slide Time: 01:52)



So, what is signed multiplier? So, we know that there are three ways to represent the signed numbers ok. So, first, I will discuss about the signed number representation. So, in that we have sign magnitude, signed 1's compliment, signed 2's complement. So, if I want to explain this three number representations, if I take an example, suppose if I want to represent plus 12 and I want to store in 8-bit register. So, how to represent in sign magnitude and how to represent in signed 1's complement and signed 2's complement?

So, what is the magnitude of this 12? 12 can be represented as 1 1 0 0. So, this plus will be represented by. So, in sign representation plus will represented by 0 minus will represented by 1 because the computer can understand only 0's and 1's it cannot understand plus and minus. So, we will represent this plus with 0 and minus with 1.

So, in signed magnitude representation or signed 1's or signed 2's, so total I have 8-bits. Because I am going to store in 8-bit register. Out of this 8-bits, so MSB bit is called as sign bit. There will be 1 sign bit which is normally most significant bit and there will be some magnitude bits, remaining 7 bits; so, this will be 7 bits.

So, in general, if you want to represent in n bit register, 1 bit we have to used it for sign; n minus 1 bits, we have to use for magnitudes. So, sign bit is because this is positive number;

so, 0. So, regardless of whether it is signed magnitude, signed 1's or signed 2's, this sign bit is always 0.

Then, coming for the magnitude, so what is the binary equivalent of 12 is 1 1 0 0, but I want to store in 8-bit register; total 7 bits has to be there. So, I can append 3 0's here; so, 1 1 0 0. This is 0 0 0 1 1 0 0. One important point here is if the number is positive, all the three representations are same, same if number is positive.

So, in signed 1's also this is 0 0 0 1 1 0 0 only; in signed 2's also 0 0 0 1 1 0 0 only; the only difference is in representing the negative numbers. So, if I take another example of say minus 12 and I want to store in same 8-bit register. So, if I take sign magnitude, signed 1's, signed 2's, so there will be 1 sign bit and 7 magnitude bits.

So, because this is negative number, negative minus 12, so sign bit is 1 in all the three representations. The magnitude bits will differ. So, what is this binary equivalent of 12? Is 1 1 0 0.

So, in sign magnitude representation, we will append 3 0's because I want to store as a 7 bit magnitude. So, the magnitude bit remains 0 0 0 1 1 0 0. This is same as this except for the sign bit. Magnitude bits are same in sign magnitude representation, if the number is positive or negative, the only sign bit will changes; whereas, in case of signed 1's complement representation, the sign bit anyhow will changes.

In addition to this, magnitude bits will be the 1's complement of this. So, the 1's complement of this one is 0 becomes 1, 1 becomes 0. So, this is the representation of signed 1's; 1 is sign bit followed by magnitude bits are 1 1 1 0 0 1 1. This is simply the 1's complement of the magnitude of signed magnitude representation.

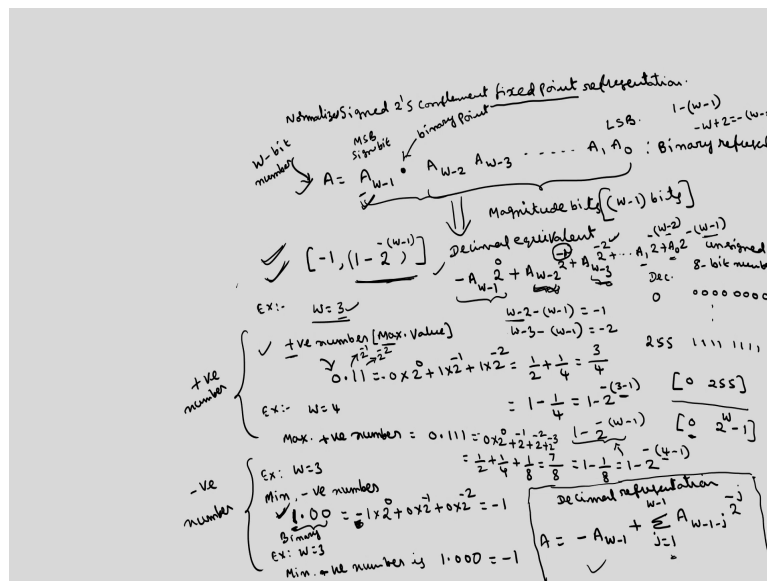
And signed 2's sign bit is 1; the 2's complement of this. So, 1's complement is this, we have to add 1 to the 1's complement. So, 1 1 1 0 0 1 1 plus 1. So, this is 1 plus 1 is 1 0 1 plus 1 is 1 0 1 0 1 1 1. So, this is 1 1 1 0 1 0 0. So, in order to represent the negative numbers, we have three representations. In sign magnitude representation, sign bit is 1 and magnitude bits are simply the 7 bit equivalent in this case of 12.

In signed 1's complement representation, we have the sign bit as 1 and magnitude bits will be the 1's complement of the magnitude bits of sign magnitude representation and in sign 2's complement representation, sign bit is 1 and the magnitude bits are the 2's complement of the magnitude bits of the sign magnetic representation.

So, out of these three, most commonly used representation is signed 2's complement representation. So, why what is the reason is? So, if you add an array of numbers. So, if the final result is within the limits, we can neglect the intermediate overflows or under flows. That is the one of the beautiful features of signed 2's complement representation.

That is the reason why this signed 2's complement representation is more popular and this will be used in many microprocessors and DSP processors. So, I will explain that property of the signed 2's complement representation which is the under flows or overflows can be neglected as long as the final result is within the limits.

(Refer Slide Time: 10:05)



So, before going for this one, I will represent this sign 2's complement representation. Normally, signed 2's complement representation fixed point representation, we have floating point also, we are not discussing here the floating point representation.

In floating point representation, we will be having exponent and mantissa, we need two registers to store a single number; one register for storing the mantissa, one register to store

the exponent. So, we are not going to that floating point representation. We will take the fixed point representation.

So, if A is a W -bit number; A is a W -bit number. So, in that A W minus 1 is the MSB bit. This will be as it acts as sign bit. So, in fixed point representation, after this sign bit, there will be a binary point. This is binary point. Then, followed by W minus 1 magnitude bits.

So, in the order of this weights, the next bit is A W minus 2, A W minus 3, so on up to will get A 1 and A 0. A 0 is LSB and this W minus 1 bits are magnitude bits. So, this is the standard normalized signed 2's complement fixed point representation.

If I use unsigned representation for W -bit number, what is the range of the number that we can represent? It is 0 to 2 raised to the power of W minus 1. If it is 8-bit number unsigned representation, what is the range? All 8 0's and the maximum number is all 1's. This is in decimal, this is 0, this is 255. So, the range is 0 to 255. So, range will be normally represented by parenthesis. This is range 0, 255. So, this is the range of the number that can be represented in unsigned format. So, in general, this is in general in case of unsigned numbers, this is 0 to 2 raised to the power of W minus 1; whereas, in case of signed numbers, this range will be minus 1 because here we are going to represent the negative numbers also.

In unsigned, there is no negative numbers. So, the minimum bit is 0 only; whereas, here the minimum bit is minus 1 and the maximum bit is 1 minus 2 raised to the power of minus of W minus 1. This I will show with an example, how this will be this range. Suppose, if I take W is equal to say 3 bit. So, if I take the positive number, I want to find out the maximum number.

Maximum number will be positive number. So, for maximum number, this is positive, so MSB bit is 0 point. So, in this representation MSB bit is 0 point. So, will be having W minus 1 bits means 2 bits. So, for maximum value these 2 bits will be 1 1. Because in positive numbers, we are not going to take the 2's complement. So, this is the maximum possible 3-bit number in sign 2's complement representation.

So, what is the decimal equivalent? How do you convert this? 0 into 2 raised to the power of 0 is the weight of this bit; this weight will be 2 raised to the power of minus 1, this weight

will be $2^{-2} + 1 + 2^{-1} + 1 + 2^{-2}$. So, this is equal to $1 + 2 + 1 + 4$. This is $4 + 3 + 4$.

So, this can be represented as $1 - 1/4$ or in other words, this is $1 - 2^{-2}$ raised to the power of $3 - 1$. Of course, 2^{-2} can be written as 2^{3-1} . So, I want to represent in this manner. So, that I can explain the range of the numbers that can be represented. So, this is in the form of W is equal to 3 . So, this is in the form of $1 - 2^{-W}$ raised to the power of $W - 1$. This is the maximum positive number in case of W is equal to 3 .

If I take another example say W is equal to 4 , what is the maximum positive number is equal to 0 point because positive number sign bit is 0 , magnitude bit should be all 1 's. Then, only you will get maximum number. So, what is the decimal equivalent of this one? $0 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3}$.

So, this is equal to $1 + 2 + 1 + 4 + 1 + 8$. This can be written as $8 + 4 + 2 + 7 + 8$. This can be written as $1 - 1/8$ and again, this is $1 - 2^{-3}$ raised to the power of $4 - 1$. So, it is also in the form of this, W is 4 . So, in place of W , we have 4 .

So, in general, the positive number can be a value which is $1 - 2^{-W}$ raised to the power of $W - 1$, where W is the number of bits in the given number. Then, coming for the negative numbers, these are examples for positive numbers. For negative numbers, if I take two examples again. If I take W is equal to 3 .

So, what is the minimum possible negative number is sign bit is 1 , followed by minimum is we should have all 0 's; 2^0 's. So, what is the equivalent of this one? This is -1 because this is sign bit. So, the decimal equals this is binary. So, decimal equivalent will be? So, this will be -2^{-1} ; even here also, you can call as -1 because this is -1 bit, but anyhow 0 will become 2 .

This is -1 because this is sign bit is 1 . So, this is $1 + 2^0 + 2^{-1} + 2^{-2}$. So, understand this why this -1 sign is because this value is negative. So, to get the negative

value, if this bit is 1, you will get minus 1 as a result. If this bit is 0, anyhow minus also if you take, you will get 0 only because the result is positive.

So, this will be overall value is minus 1. If I take W is equal to 4 also, same. The minimum negative number is 1.000, this is equivalent to minus 1. So, this is how the range of these numbers is limited. But the advantage of this one is we can neglect the intermediate overflows if the final result is within this range.

So, this is actually binary representation of a W -bit number in signed 2's complement fixed point representation of the W -bit number. So, what is the corresponding decimal? What is the corresponding decimal equivalent of this one? We can see that here for negative numbers, what you have taken? You have taken minus sign.

So, similarly I can take here also this is minus $A W$ minus 1 into 2 raised to the power of 0, we are going to use the same logic that we have used here in the examples. This is minus sign; plus what are these weights? $A W$ minus 2, 2 raised to the power of minus 1 plus $A W$ minus 3, 2 raised to the power of minus 2 plus so on up to $A 1$ 2 raise to the power of plus $A 0$ 2 raised to the power of what are this the exponents of this $A 1$ and $A 0$?

We can see here that this W minus 3 means here 2 W minus 2 here means here 1. So, this is minus of; so, whatever the value here plus 1 ok; plus W minus 1, we are going to subtract W minus 1. So, here to get this minus 1 W minus 2 minus of W minus 1 is the value which is equal to W W will get cancelled, we will get minus 1. So, this minus 1 is the exponent of this.

Similarly, here W minus 3 minus of W minus 1, we will get minus 2 so, this minus 2. Now, we can tell here this one will be if I assume that this is x . So, 1 minus of x ; 1 minus of x should be equal to if I assume that this is x , this is W minus 2, this is 1. So, 1 minus of W minus 2 minus of W minus 1.

Sorry, this is 1 minus of W minus 1 will be the coefficient of this one. So, this will be equal to minus w , this minus 1 plus 1 get minus W minus plus 2. This is equal to minus of W minus 2. Is it clear? This is minus of W minus 2. And similarly, what about this? We will get minus of W minus 1 because the exponents are decreasing minus 1, minus 2, if I consider the negative

sign. So, minus 1 minus 2 so, 1 up to minus of W minus 2, the last coefficient is minus of W minus 1.

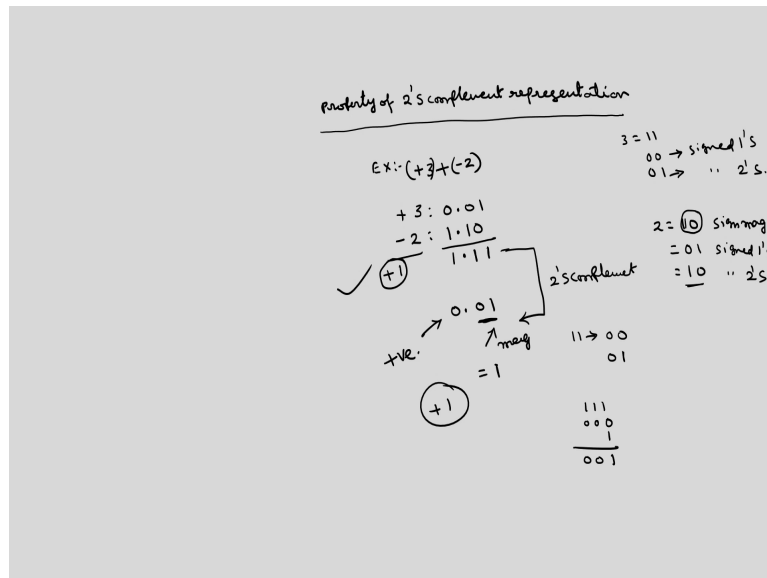
So, this is the decimal equivalent. So, we can alternatively use this binary representation as well as decimal representation to represent the negative numbers. So, this decimal representation can also be represented as decimal representation can be simplified as this is minus of A W minus 1 2 raised to the power of 0 is 1 plus I will make this as sigma, sigma j is equal to 1 to W minus 1, A W minus 1 minus j into 2 raised to the power of minus j . Is it satisfying this?

So, this anyhow this is minus a suffix W minus 1 2 raised to the power of 0 is minus A W minus 1, this is ok. Now, for j is equal to 1, what happens? This becomes A W minus 2 A W minus 2 and coefficient is 2 raised to the power of minus 1. For j is equal to 2, W minus 3, 2 raised to the power of minus 2; for j is equal to W minus 1, the last one, this becomes W minus 1, W minus 1 get cancelled.

So, A 0 and the coefficient is 2 raised to the power of minus of W minus 1. So, this is A in decimal form. This is one of the important representation that we are going to use in the remaining mathematical analysis of this signed 2's complement multiplier.

So, this is some background and coming for the property that as I have told, if I add a series of numbers, if the final result is within the range, we can neglect the intermediate overflows or underflows. So, that also I will explain with an example.

(Refer Slide Time: 25:10)



This is property of 2's complement representation. If I take an example, where there is overflow. Suppose, if I take 3 minus 2. So, 3, how do you represent? So, in order to represent 3, how many bits are required first of all? This is plus 3, minus 2.

So, in fact, we can call as here all the numbers are I mean you have to represent the numbers along with the sign; plus 3 plus of minus 2 which is 3 minus 2. 3 is how many bits are required? The magnitude bit is 0 because positive number and how many magnitude bits are required? 3 is nothing but 1 1, but in sign 2's complement representation, what will be this? This is 0 0 in signed 1's and 0 1 in sign 2's. So, the magnitude is 0 1.

And minus 2, how do you represent? 1 dot what is the binary equivalent of 2? It is 1 0 in sign magnitude representation, but then we want sign 2's complement representation. So, this will be equivalent to 0 1 in signed 1's and this will be 1 0 in signed 2's. So, this is 1 0.

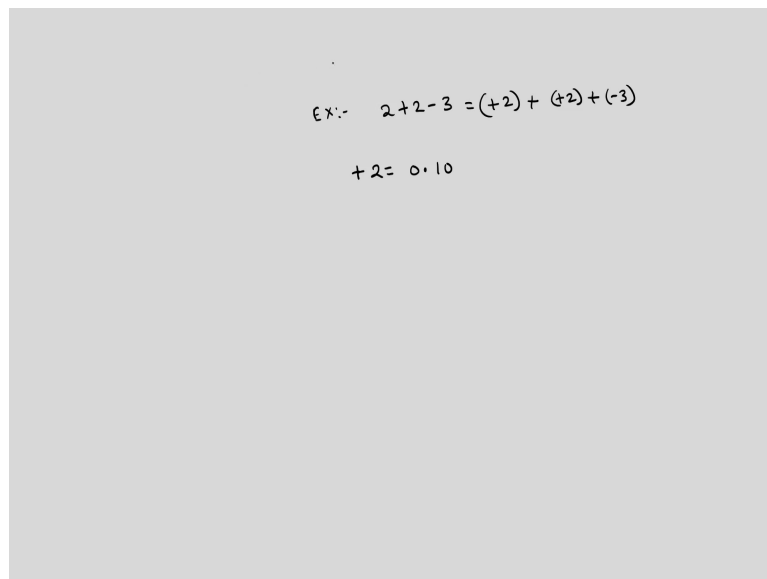
Of course, this 1 0 signed magnitude and signed 2's also same only for this particular example; this need not be same for all the numbers right. So, if I want to perform this addition 1 plus 1 is 1, 1 plus 0 is 1, 0 plus 1 is 1 point this one is 1. So, what should be the answer in decimal? It should be plus 1, but this is not the final answer.

So, final answer can be obtained by taking the 2's complement of this one because we are representing in 2's complement; 2's complement of this one is final answer ok. So, what is

the 2's complement of this? So, 1's complement is 0 0 for 1 1; 1's complement is 0 0; 2's complement will be 0 1 and for this 0.

This is 1 1 1 is 0 0 0 plus 1 is 0 0 1. This is 0.01. So, this will represent positive number and this one is magnitude. Magnitude this is equal to 1. So, this is plus 1; so, this is plus 1. So, this is how we can perform this signed 2's complement addition, there is no subtraction at all here because we are going to represent the number along with the sign.

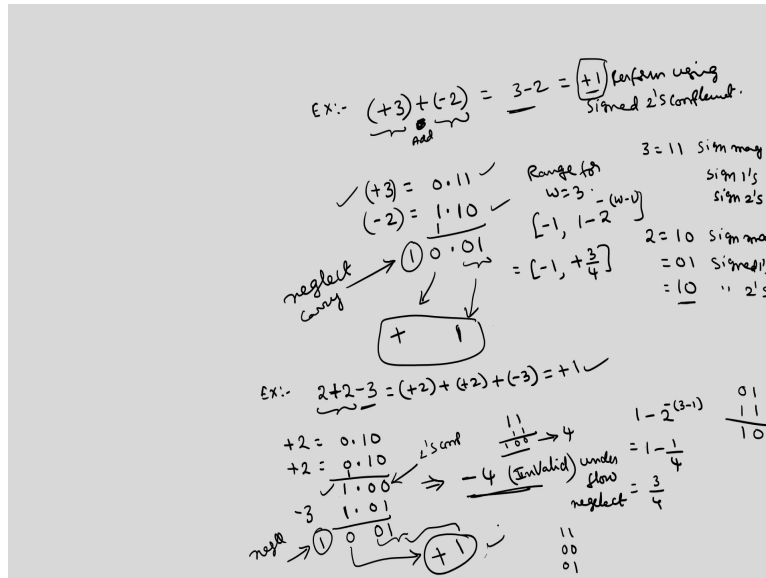
(Refer Slide Time: 28:57)



EX: $2 + 2 - 3 = (+2) + (+2) + (-3)$
 $+2 = 0.10$

So, in order to study that property, intermediate overflows and all if I take the second example, if I want to add 2 plus 2 minus 3. So, how do you represent? This is nothing but plus 2 plus plus 2 plus minus 3. So, how do you represent plus 2? Plus is sign bit is 0 magnitude bits is 1 0 because for positive numbers all the three representations are same. If you want to explain this with the help of examples.

(Refer Slide Time: 29:46)



I will initially take example of plus 3 plus of minus 2 this is nothing but actually 3 minus 2. But normally in 2's complement, we will take along with this sign. So, what is the result? Expected result is plus 1. So, if you want to explain this in sign 2's complement, how this addition will be formed? Plus 3, how many bits are required to represent plus 3?

So, this is equal to plus represent the sign bit is 0 and the magnitude bits are 3. How many bits are equal to represent 3 is 1 1 in binary equivalent; but in case of positive numbers, this is sign magnitude or signed 1's or sign 2's, all the three representations are same. So, this is 0.11.

Minus 2 this is equal to sign bit is 1 and what is the sign magnitude representation of the 2? It is 1 0 is sign magnitude representation; but because this is negative number, we have three different representation. So, this is 0 1 in signed 1's and this is we have to add 1 to this 1. So, 0 1 plus 1 is 1 plus 0 is 1 0 1. So, 1 0 sign 2's. Because here you are going to perform this using sign 2's complementary representation.

So, you have to take sign 2's complement representation which is 1 0. Then, this only additionally because I have taken along with the signs. So, the subtraction becomes addition. This is 1, 1 plus 1 is 1 0, 1 plus 1 is 1 0. So, this over flows you can neglect because. So, how many bits you have taken? 3 bits. This is 3-bit representation.

So, what is the range of the number that can be represented in 3-bit? The range is for W is equal to 3 is minus 1 is the negative number that is the minimum number; positive number is 1 minus of 2 raised to the power of minus of W minus 1. So, this is nothing but for W is equal to 3; 3 minus 2 becomes 2. So, this is 1 by 4.

So, this is equal to this is equal to minus 1 2 plus, what is the value? 1 minus of 2 raised to the power of minus of three minus 1 which is equal to 1 minus 1 by 4, 3 by 4, this is plus 3 by 4. So, what is the result of this 3 minus in fact? It is plus 1. We can neglect this carry, remaining what is the magnitude this represents? Plus. In positive numbers, we have directly taken the magnitude. What is the magnitude equivalent of this 1? Is 1; so, result is plus 1, this is plus 1.

So, if I take the second example, if I want to perform 2 plus 2 minus 3. So, you see nothing but plus 2 plus plus 2 plus minus 3. So, what is plus 2? Its sign bit is 0, magnitude bit is 1 0 because positive number. Magnitude sign magnitude, sign 1, sign 2's is same. Another plus 2 is 0 point again 1 0.

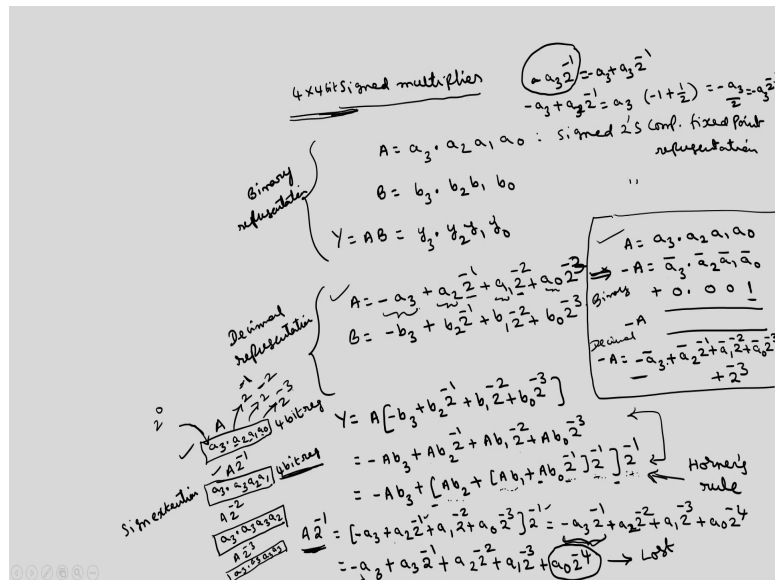
First, I will perform addition of these two numbers. So, 0 1 0 point 1. So, what is the magnitude of this one? This represent minus sign; this is minus so, minus. What is the magnitude 2's complement of this? What is the 2's complement of this one is so 0 0 is 1 1 plus 1 is 1 0 1 0, this is 4. But this is not valid result. We know that this should be plus 1 so, within the range.

So, within the range of this 3-bit representation so, even though if you have the intermediate overflow or underflow, this is called underflow. We can neglect this. You neglect this intermediate result and you proceed with the final this thing, you take minus 3 also. So, minus 3 is sign bit is 1; magnitude bits are for minus 3, this is 1 1 is signed 1's representation, sign magnitude representation.

Signed 1's is 0 0; sign 2's is 0 1 so, 0 1. If I add this one 1 0, 1 plus 1 is 1 0, 1 you neglect. So, this represent positive and this represent plus 1. So, this is the final result is correct. In between we have got some invalid result; but the final result is correct. So, it is then one of the important feature of the sign 2's complement representation. Even though if you get some underflows or overflows in between, as long as the final result is within the range. So, we can

neglect the intermediate overflows and under underflows. This is the reason why this sign 2's complement representation is more popular.

(Refer Slide Time: 36:11)



Now, coming back for this signed multiplier, how to perform the multiplication. Let A is equal to a 3 dot a 2 a 1 a naught. This is signed 2's complement fixed point representation. a 3 is the sign bit; a 2 a 1 a 0 are magnitude bits and B is another 4-bit number. We are discussing about the 4-bit by 4-bit sign 2's complement multiplication ok.

b 3 dot b 2 b 1 b 0; this is in binary form. So, how to perform multiplication as Y is equal to AB. Of course, if you multiply two 4-bit numbers, result can be 16-bit; but here, I am going to truncate the lower order 8-bits and I will store only the 4-bits of the result only. If you want, you can retain the remaining bits also; but it will increase the complexity.

So, there is a compromise here again between the complexity and the accuracy. Here, I am restricting this result to 4-bit only. So, this will be something like y 3 is sign bit, y 2 y 1 y 0 are magnitude bits. So, how to perform this multiplication? So, in order to perform this multiplication, if I write this in the decimal form, this is binary representation.

If I take in decimal form, so what is A is, minus a 3, this we have already discussed plus a 2 2 raised to the power of minus 1 plus a 1 2 raised to the power of minus 2 plus a 0 2 raised to

the power of minus 3. B is minus b_3 sign bit plus $b_2 2$ raised to the power of minus 1 plus $b_1 2$ raised to the power of minus 2 plus $b_0 2$ raised to the power of minus 3.

So, if you want to perform AB, I will fix 1 number and I will then, I will represent the other number in the decimal form. And I will use a rule called Horner's rule. So, Y is equal to A, I will fix that is as it is; B, I will represent this number minus b_3 plus $b_2 2$ raised to the power of minus 1 plus $b_1 2$ raised to the power of minus 2 plus $b_0 2$ raised to the power of minus 3.

So, this is nothing but minus $A b_3$ plus in fact $A b_2 2$ raised to the power of minus 1 plus $A b_1 2$ raised to the power of minus 2 plus $A b_0 2$ raised to the power of minus 3. This I am going to write cleverly so that the complexity of the multiplication architecture will become less complex.

I will write this as $A b_2$ plus I will write again $A b_1$ plus $A b_0 2$ raised to the power of minus 1, then 2 raised to the power of minus 1 and 2 raised to the power of minus 1. We just check whether these two expressions are same or not. Minus $A b_3$, $A b_3$ is as it is; $A b_2$ will be 2 raised to the power of minus 1, this $A b_2$ this overall this bracket and this bracket.

So, this will multiply this $A b_2$ will multiply with only this 2 raised to the power of minus 1 and what about $A b_1$? This will multiply 2 times. This with this as well as this. So, that this becomes $A b_1$ becomes 2 raised to the power of minus 2. And what about $A b_0$? This will multiply 3 times this one 2 raised to the power of minus 1, another 2 raised to the power of minus 1, another 2 raised to the power of minus 1.

So, these two are same. This particular representation is called the representation using Horner's rule. This is very much useful representation to get a low complexity architecture for signed multiplier. Now, here we have minus $A b_3$, $A b_2$ and all. So, A is this one. What is minus A? if A is this implies what is minus A? If I know minus A, then b_3 you have to multiply with each coefficient with b_3 .

So, in signed 2's complement representation, given A how to obtain minus A. Minus A is nothing but 2's complement. So, minus A can be obtained using the 2's complement. So, A is

in binary form; this is a 3 dot a 2 a 1 a 0 minus A becomes the 1's complement is a 3 bar a 2 bar a 1 bar a 0 bar plus 1; 1 has to be added the least significant bit. This is equal to minus A.

If font in decimal form this is binary representation. In decimal form minus A is nothing but this is minus a 3 bar plus a 2 bar 2 raise to the power of minus 1 plus a 1 bar 2 raised to the power of minus 2 plus a 0 bar 2 raised to the power of minus 3 plus this extra term will come this 2 raise to the power of minus 3.

So, this is the complete representation of minus A in both binary form as well as decimal form. So, these results we are going to use to obtain the final architecture. This is one important thing. So, another thing is given A, we require A into 2 raised to the power of minus 1 also; this is A into 2 raised to the power of minus 1.

So, given A, how to obtain A raise to the power of A into 2 raise to the power of minus 1, what is A 2 raise to the power of minus 1? So, this is equal to what is A in decimal form? Minus a 3 plus a 2 2 raised to the power of minus 1 plus a 1 2 raised to the power of minus 2 plus a 0 2 raised to the power of minus 3 into 2 raised to the power of minus 1.

This is equal to minus a 3 2 raised to the power of minus 1 plus a 2 2 raised to the power of minus 2, this minus 1 is there another minus 1 minus 2 plus a 1 2 raised to the power of minus 3 plus a 0 2 raised to the power of minus 4. But I want to store this A into 2 raised to the power of minus 1 in 4-bit register only. This is A I have stored in 4-bit register. How do you store this A in 4-bit register? a 3 is the MSB bit followed by dot a 2 a 1 a 0.

So, what is the weights of this one? The weight of this one is 2 raised to the power of 0; weight of this one is 2 raised to the power of minus 1; weight of this is 2 raised to the power of minus 2; weight of this is 2 raised to the power of minus 3. But I want to store A into 2 raise to the power of minus 1 also in the 4-bit register. So, what is expected? Here, there will be sign bit followed by 3-bits with weights of 2 raised to the power of minus 1 minus 2 minus 3.

So, there is no question of minus 4. So, that is why if I write this in equivalent form, I will write this as minus a 3 plus a 3 2 raised to the power of minus 1. So, this single term, I am

going to write equivalent to this. We can easily verify that this minus a $3 \cdot 2$ raised to the power of minus 1 is equivalent to minus a 3 plus a $3 \cdot 2$ raised to the power of minus 1 ok.

Because if I take minus a 3 plus a $2 \cdot 2$ raised to the power of minus a $3 \cdot 2$ raised to the power of minus 1, if I take a 3 as common, this is equal to minus 1 plus 1 by 2. So, this is nothing but minus 2 plus 1 by 2. So, this is minus 2, this is minus 1 plus half is minus half. This is minus a 3 by 2 that is equal to minus a $3 \cdot 2$ raised to the power of minus 1.

So, minus a $3 \cdot 2$ raised to the power of minus 1 can be written as minus a 3 plus a raised to the power of a into a 3 into 2 raised to the power of minus 1 plus remaining terms are a $2 \cdot 2$ raised to the power of minus 2 plus a $1 \cdot 2$ raised to the power of minus 3 plus a $0 \cdot 2$ raised to the power of minus 4.

But I want to store in 4-bit register; this A into 2 raised to the power of minus 1 value also. So, this value will be left, this will be lost. Because I do not have that position. The positions are only 2 raised to the power of minus 1 position, minus 2 position, minus 3; minus 4 position is not there in 4-bit register. So, this will be lost. So, then what will be stored? There must be minus sign.

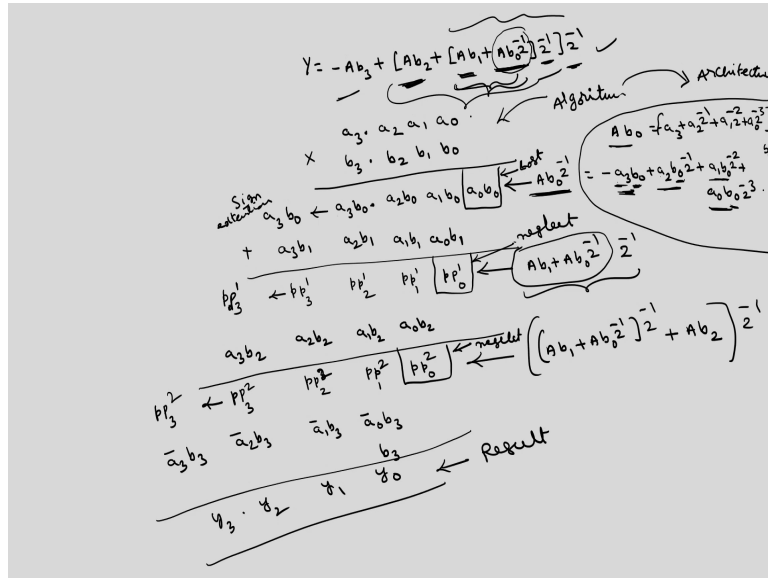
So, A is with minus sign; the a 3 here also minus a 3 that a $3 \cdot 2$ raised to the power of minus 1 term. What is the 2 raised to the power of minus 1 term? Here we have minus a 3 . So, the first bit is a 3 and then 2 raised to the power of minus 1 term is a 2 ; so, a $2 \cdot 2$ raised to the power of minus 2 term is a 1 ; so, a $1 \cdot 2$ raised to the power of minus 3 term is a 0 .

Here, what is 2 raised to the power of minus 1 is a 3 only; minus 2 is a 2 ; minus 1 is a 1 and a 0 is lost. So, this is one of the important result. If I have A as a $3 \cdot 2 \cdot 1 \cdot 0$, then A into 2 raised to the power of minus 1 will be having a $3 \cdot 2 \cdot 1$ and a 0 will be lost. This is what is called the important property which is called sign extinction property. So, the sign 2 's complement representation will be having sign extinction property.

So, given a as a $3 \cdot 2 \cdot 1 \cdot 0$, A into 2 raised to the power of minus 1 becomes a $3 \cdot 2 \cdot 1$, a 0 will be lost. In a similar manner, you can easily derive that if I want A into 2 raised to the power of minus 2 in 4-bit register, so the sign bit will be a 3 itself, but it will be copied twice; a 2 and a 1 , a 0 will be lost.

And if I want write A into 2 raise to the power of minus 3, then all a 3s; a 2, a 1, a 0 will be lost. So, this is equal to a 3 dot a 3 dot a 3 dot a 3. So, this is another important result. Using this results, we are going to derive the architecture for 4-bit by 4-bit sign multiplier.

(Refer Slide Time: 49:08)



Now, coming for this algorithm, so what is value of Y, we have obtained is minus A b 3 plus A b 2 plus A b 1 plus A b 0 2 raised to the power of minus 1 whole into 2 raised to the power of minus 1 whole into 2 raised to the power of minus 1. This is what is the Horner's rule.

So, now, if I take the multiplication of a 3 dot a 2 a 1 a 0 into b 3 dot b 2 b 1 b 0. So, first I will develop the algorithm using this Horner's rule, then I will map this algorithm on to the architecture. Here, I am going to develop algorithm. Later, I will map this algorithm on to the architecture.

So, the first thing is we will start with the A b 0 2 raised to the power of minus 1. So, what is A b 0? A b 0 is nothing but A b 0 is nothing but A is a minus a 3 plus a 2 2 raise to the power of minus 1 plus a 1 2 raised to the power of minus 2 plus a 0 2 raised to the power of minus 3 into b 0. So, this is nothing but minus a 3 b 0 plus a 2 b 0 2 raised to the power of minus 1 plus a 1 b 0 2 raised to the power of minus 2 plus a 0 b 0 2 raised to the power of minus 3.

So, what is A b 0 is simply you have to multiply with each and every bit. So, this is a 3 b 0 dot a 2 b 0 because this is a to b 0 after dot which is 2 raised to the power of minus 1 term

and a 3 b 0 is 2 raised to the power of 0 term and 2 raised to the power of minus 2 term is a 1 b 0, 2 raised to the power of minus three term is a 0 b 0. So, this will represent A b 0. This A b 0 in this total term.

Suppose, if I want A b 0 into 2 raised to the power of minus 1 also. So, in the previous, we have shown that if A is this, A is this, what is a into 2 raise to the power of minus 1? Is simply a 3 a 2 the sign bit will be copied. So, if I want this one, this will be lost; this will be lost to get A b 0 2 raised to the power of minus 1, this bit will be copied; a 3 b 0, this is sign extension.

So, this will be if you remove this a 0 b 0 term and if I keep this a 3 b 0, a 3 b 0, a 2 b 0, a 1 b 0; this will be a b 0 2 raise to the power of minus 1, this entire thing. Then, what you have to do? You have to add this to A b 1. So, A b 0 means a 3 b 0, a 2 b 0, a 1 b 0 a 0 b 0; if I want A b 1 means, so a 3 b 1, a 2 b 1, a 1 b 1, a 0 b 1. So, this plus because I want to perform up to here now. A b 1 is nothing but so a 0 b this is a 3 b 0 is there and this is last one is a 0 b 0. But now, now we want A b 1. So, in place of b 0, we have b 1; this b 0 b 1, this b 0 b 1, this b 0 b 1.

So, this is a 0 b 1, a 1 b 1, a 2 b 1, a 3 b 1. So, if I add this, I will call this partial products. This is not the final product. So, I will call this as partial product 1 and this is zeroth bit partial product 1, this is 1th bit partial product 1, this is second bit partial product 1, this is third bit.

This is the notation that I am following. So, what will be this value now? This complete value is A b 1 plus A b 0 2 raised to the power of minus 1, this is the total value of we have obtained. This is total this. This you have to again shift. So, this if you want to shift with 2 raised to the power of minus 1, then you have to neglect this, then you copy this sign bit p p 3 1. So, this will be now into 2 raised to the power of minus 1 up to here, up to here. Then for that, we have to add A b 2.

So, A b 2 is nothing but again, so a 0 b 2, a 1 b 2, a 2 b 2, a 3 b 2. So, if I perform this, I will get another partial product which will call as p p 2, second partial product 0th bit, p p 2 1 bit, p p 3 second bit, p p 2 second bit, p p 2 third bit. So, this result is total result is this A b 1 plus

$A b_0 2$ raised to the power of minus 1, this into 2 raised to the power of minus 1 plus $A b_2$, up to here.

This entire thing is again we have to shift with 2 raise to the power of minus 1. Finally, we have to add this. So, this if you want to shift by this entire thing if you want to shift by 2 raise to the power of minus 1, then you neglect this and you extend the sign of this $p p_3 2$. Then finally, you have to add minus A_3 . We know minus A_3 is if I know a as this one, minus a is this. So, minus $a_3 a_2 a_1 a_0$ plus you will get one extra term called 2 raised to the power of minus 3.

So, what will be minus $A b_3$? Is $a_3 a_0 \bar{b}_3, a_1 \bar{b}_3, a_2 \bar{b}_3, a_3 \bar{b}_3$ because 2 's complement, then we will get one b_3 here because there is 2 raised to the power of minus 3.

For minus A , if you multiply with minus A with b_3 , then all the terms $a_3 \bar{b}_3, a_3 \bar{b}_3, a_2 \bar{b}_3, a_1 \bar{b}_3, a_0 \bar{b}_3$ and then finally, simply b_3 because here only one. So, this is this b_3 . Finally, you will get this result. You are calling as this is $y_3 \text{ dot } y_2 y_1 y_0$. This is the final result. This is the algorithm so, how to map this algorithm onto the architecture, we will discuss in the next lecture.

Thank you.