

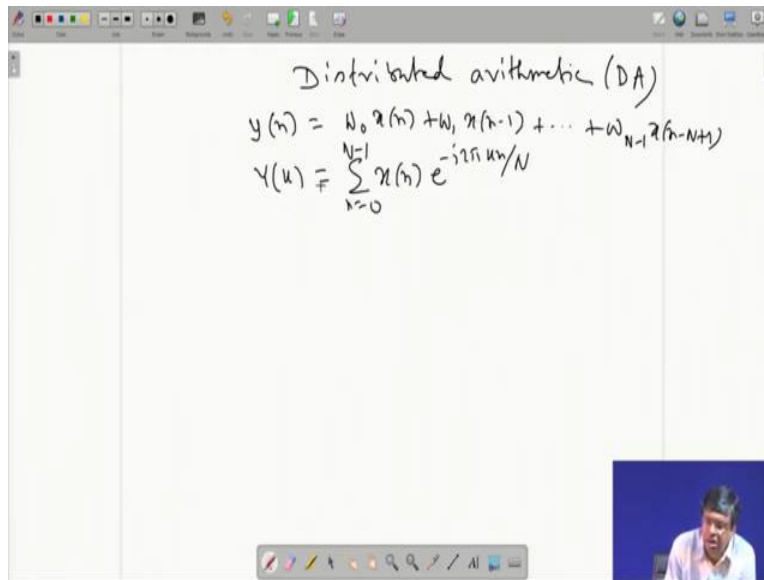
**VLSI Signal Processing**  
**Professor Mrityunjoy Chakraborty**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 39**  
**Distributed Arithmetic**

Okay, in the last class we discussed bit-serial realization of digital filters. And that time we were saying that the filter coefficients are encoded in a format called Canonic Sign Digit, CSD. The advantage of that format was that, since every position can help digit either one, minus one or 0. Using this it can be encoded such that, if there is non-0 digit one or minus one, then immediate neighbor to left and to the right will be 0. And by that process approximately half of the digits will be 0.

And presence of 0s will reduce the multiplying complexity a lot, because whenever you multiply anything with 0, you get 0. So, result is pre-known before hand, so need to dedicating hardware for that part. Because multipliers cause the big headache for us in hardware realizations not adders. You know if you multiplying a 16-bit number a 16-bit number, there are so many shift and add and all those stages. And it will be requiring a 16 by 16 array, carry ripple or carry save array, that takes more time that is power everything.

So, that is why it is a big challenge you know. It always the motivation for all VLSI rejoiners to bring down the complexity of multipliers, CSD was such thing. That today in this continuing lecture in a series, I will teach another technique by which, you can realize digital filters or even transforms, without recalling multipliers directly but using a ROM, Read Only Memory.

(Refer Slide Time: 01:58)



Distributed arithmetic (DA)

$$y(n) = W_0 x(n) + W_1 x(n-1) + \dots + W_{N-1} x(n-N+1)$$
$$Y(u) = \sum_{k=0}^{N-1} x(k) e^{-j2\pi uk/N}$$

And this is done by a method called distributed arithmetic. In DSP you come across linear combinations like say, when we have a filter FIR filter. Output is  $W_0 x(n)$ ,  $W_1 x(n-1)$  dot-dot any  $W$  and just FIR filters. It is a linear combination, it has got so many multipliers,  $W_0$  into  $x(n)$ .

Simultaneously,  $W_1$  into  $x(n-1)$ , simultaneously dot-dot-dot,  $W_{n-1}$  into  $x(N-1)$ , so, capital  $N$  number of multiplications and then add with a linear combination. Another linear combination is say, if you take DFT, DFT is summation  $x(n) e^{j2\pi kn/N}$ . So,  $x_0$  times  $e$  to the power something,  $x_1$  times  $e$  to the power something,  $x_2$   $e$  to the power something and they added, that is why it is linear combination. And any linear combination will have so many multipliers in parallel.

Like here  $W_0$  into  $x(n)$  simultaneously  $W_1$  into  $x(n-1)$  simultaneously  $W_{N-1}$  into this. So, you need to have capital  $N$  number of multipliers here also. Each multiplier is so complex, so we have lot of complexity. Question is how to bring down that requirement of multipliers here and by use of something else. And in this case that something else will be a wrong read only memory that is why this distributed arithmetic about. So, instead of considering either this FIR filter or this transform, will consider a general linear combination of this form.

(Refer Slide Time: 03:46)

The top screenshot shows the title "Distributed arithmetic (DA)" and the equation  $y = \sum c_k x^k$ .

The bottom screenshot shows a detailed derivation of the DA equation. It starts with the equation  $y = \sum_{i=0}^{N-1} c_i x_i$ . The input  $x_i$  is defined as a 2's complement  $w$  bit number. The expansion for  $c_0 x_0$  is shown as  $x_0/d = X_{0,w-1} + X_{0,w-2} + \dots + X_{0,w-i} + \dots + X_{0,0} 2^{-i}$ . Similar expansions are shown for  $c_i x_i$  and  $c_{N-1} x_{N-1}$ . The final summation is shown as  $(c_0 + c_1 + \dots + c_{N-1}) [c_0 X_{0,w-1} + \dots + c_i X_{i,w-i} + \dots + c_{N-1} X_{N-1,w-1}] 2^{-i}$ . The final output  $y_0$  is shown as  $(c_0 + c_1 + \dots + c_{N-1}) 2^{-i}$ .

And output  $Y$  which can be function of  $n$  is a linear combination of this stance.  $C_k$  or maybe  $C_i$  just a minute  $X_i$ . Each  $X_i$  is like a data like earlier I had like  $C_0$  could be in the case of filter  $W_0$ . Capital  $X_0$  could be  $X$  of  $n$ ,  $C_1$  could be  $W_1$ , capital  $X_1$  could be  $W_n$  minus 1 so and so in that example. So, there actually function of  $n$  time, but that  $n$  I am dropping from here, because I mean whatever I scheme proposed that has nothing to do with time  $n$ . That is why, I do not write anything as a function of  $n$  just to make things less complex.

So, I have got these numbers  $X_0$  is one number when  $i$  is 0. This is a  $W$  bit number  $2^i$ 's complement  $2^i$ 's complement  $w$  bit number.  $X_i$  so it will be like  $X_0 \dots$  because it is 0 MSB, then

binary point this 0 0 coming from 0 data x 0. There is this 0 transform this 0 comma w minus 2 dot-dot-dot  $X_0 W^{-1}$  minus j dot-dot-dot,  $X_1$  sorry  $X_0, 1 X_0, 0$ . In general  $X_i$  is  $X_i$  comma  $W^{-1}$ , i n data so i here. Then again i next bit dot-dot-dot again  $X_i$  next bit dot-dot-dot. Last one, so these are the w bit words at this linear combination coefficients are some numbers decimal numbers.

They can be any I mean in digital form, they can be any not necessarily w bit number. They can be any bit number, or bit number does not matter, it just a numbers. So, how to multiply decimal  $X_0$  by  $C_0$ , decimal  $X_1$  by  $C_1$  dot-dot-dot and add. These are jobs I have to do, but I have to increment digitally.

Now, Decimal  $X_0$  is minus this plus this into 2 inverse, this 2 inverse I am writing on top for some reason, then dot-dot-dot. Then this into 2 inverse you can check minus j plus dot-dot-dot, this into 2 inverse minus  $W^{-2}$  and this into 2 inverse this. Similarly,  $X_i$  is decimal value is I can write-in by d, by d, d for decimal, minus this plus this into same 2 inverse.

So, is 2 inverse is common for this, all these fellows, I am writing for one place on top in this column. So, this into 2 inverse, otherwise I could not have written in  $(\ )^{-j}$  2 inverse here, 2 inverse here, 2 inverse here that is not a good way. This 2 inverse is common for everybody here, then again this into 2 inverse j plus dot-dot-dot, this into 2 inverse minus  $W^{-2}$ . Plus this into 2 inverse minus bracket  $W^{-1}$  and again this one its decimal value is minus this.

Plus this time 2 inverse plus dot-dot-dot plus this time 2 inverse j dot-dot-dot plus this 2 inverse minus bracket  $W^{-2}$ . Plus this times 2 inverse minus bracket  $W^{-1}$ . I have to multiply this by  $C_0$  this by  $C_i$  there is multiply, multiply this decimal numbers and add. Now, one way is to calculate the inter-decimal value, then multiplying by  $C_0$ . Calculate the inter-decimal value, multiplying by  $C_i$  and so on so add. But I can also do this way, I can take say any one column, column means like this which have the same power 2 to the power minus j. So, this term will be multiplied by this is decimal one and decimal 0.

This is binary bit, but now in decimal domain, decimal one and decimal 0. This fellow will be multiplied by  $C_0$ , this fellow you multiplied by the  $C_i$ , this fellow will be multiplied by  $C_N$  minus 1. Finally, they have to be added because whether you calculate this first and multiplied by  $C_0$  and then calculate this first, multiplied by  $C_i$  and finally you add them.

So, essentially you will multiply this fellow by  $C_0$ , multiply this fellow by  $C_i$ , multiply this fellow by  $C_{N-1}$ . You can add them before end only, instead of calculating the decimal first decimal first and multiplying by  $C_0$  etc. I can as well hold this  $2$  to the power minus  $j$  common and multiply this fellow by  $C_0$ , multiply this fellow by  $C_i$ , multiply and add multiply by  $2$  inverse.

Do the same again in another column,  $C_0$  times this,  $C_i$  times this,  $C_{N-1}$  times this and add then multiply by  $2$  inverse. Here in the  $0$ th,  $C_0$  times  $X_0$ ,  $C_i$  times  $X_i$ ,  $C_{N-1}$  times  $X_{N-1}$ , add them and multiply by  $2$  inverse  $W^{-1}$ , do it for all of them. In the case of MSB, there is minus sign, so after you calculate  $C_0$  times  $X_0 W^{-1}$ , and so on and so forth, add them. No power of  $2$ , no  $2$  to the power minus something, but you have just to take  $2$ 's compliment of that one.

That is you have to take negative of that one. And then add for all the columns, column means this is the column, this is a column, this is a column. Each column has a power of  $2$  common. Then what is happening, if you consider this column which is not a MSB column, so there is no minus sign to start with. Then you have here an expression like,  $C_0$  times this poor fellow  $X_0$  because  $0$  is our data. Then  $C_i$ ,  $i$ th data this bit, this entire thing and to multiplied by  $2$  to the power minus  $j$  from top.

This I do for taking  $i$  equal to  $I$  mean taking the LSB column, next to LSB column, this column, this column all of them. Add and the row comes to MSB, do the same combination, no power of  $2$  just negation and add all of them. I can do this way also, if I do this way that advantage is, you see one thing this a bit and therefore in the decimal, this is a digit. But, digital be does not mean  $1$  or  $0$ ,  $1$  or  $0$ , this is in this case decimal  $1$  or decimal  $0$ , in this case decimal  $1$  or decimal  $0$ . So, this summation can have only few possibilities, say for instance, when all are  $1$ , then  $C_0$  into  $1$ , plus  $C_1$  into  $1$ , plus  $C_i$  into  $1$ .

So, when all the bits, all these bits among therefore digits are  $1,1,1,1$  all of them are  $1$ . Then the summation is  $C_0$  into  $1$ , plus  $C_1$  into  $1$  plus dot-dot-dot. So, summation will be then  $C_0$  plus  $C_1$  plus dot-dot-dot  $C_{N-1}$ . On the other hand, suppose all are  $0$ , then summation will be  $0$ , suppose all are  $0$ , suppose all are  $1$ , but all of this fellow is  $0$ .

Then it will be  $C_0$  plus  $C_1$  plus dot-dot-dot plus  $C_{N-2}$  only. When you come to  $C_{N-1}$ , this is 0. So, how many combinations are possible 2 to the power N, because this can take 1 or 0. If it is 1,  $C_0$  into 1 is  $C_0$ , if it is 0, it is 0. If it is 1,  $C_i$  into  $C_i$  otherwise 0. So, everybody can take every position I have only two possibilities, like  $C_0$  or 0,  $C_1$  or 0,  $C_2$  or 0,  $C_3$  or 0.

So, various combinations from these bits, because the bits can take only how many combinations? Either 0 or 1 here, 0 or 1 here, 0 or 1 here so 2 to the power N. So, for each pattern I will get a specific sum, which means this sum cannot take arbitrary values, it can take only these values. What is most important is, if I instead of doing the same exercise on this column, if I do it on this column. Even then I will have  $C_0$  times this,  $C_i$  times this,  $C_{N-1}$  times this.

But, since here also this can take only 1 or 0, 1 or 0, 1 or 0. The possibilities are same that is I can have the combinations of  $C_0$  to  $C_{N-1}$ , then you get same. Like if all are 1, once again I get this fellow, if all are 1 last one is 0 I get this so on and so forth. So, the set of combinations of this  $C_0$ ,  $C_1$  up to that does not change, because whether I am here or I am here or I am here every bit can take only 1 or 0, 1 or 0, 1 or 0.

So, this expression does not depend on this, does not depend on this. It can be 0, 0 the 0th location it can be 1 that is  $X_{0,1}$ ,  $X_{0,2}$ ,  $X_{i,2}$  sorry  $X_{0,1}$ ,  $X_{i,1}$  and so on and so forth. Or it can be location 2 and so and so. Here it matters, it varies from column to column, j is W minus 1, here is j, here is 1, here is 0. But, as per as this summation is concerned, its output is just one of those same 2 to the power N possibilities.

Because whichever column I have this fellow can take only 0 or 1, like here 0 or 1 or 0 or 1 or 0 or 1. Here also if you take this column, either this is 0 or 1, so I just  $C_0$  into 0 that is 0 or  $C_0$  into 1 is  $C_0$ ,  $C_i$  into 0 that is 0 or  $C_i$  into 1,  $C_i$ . Similarly here so summation is the set of values of this summation, then you get same, depending on the bit pattern.

Which means I do not need to, if I mean if I can program a ROM where depending on the bit patterns, 2 to the power N possibilities. This bit patterns will be like address will be address bus through which at bits will come serially. Through one line this bit is coming through another line, this bit is coming through another line, this bit is coming.

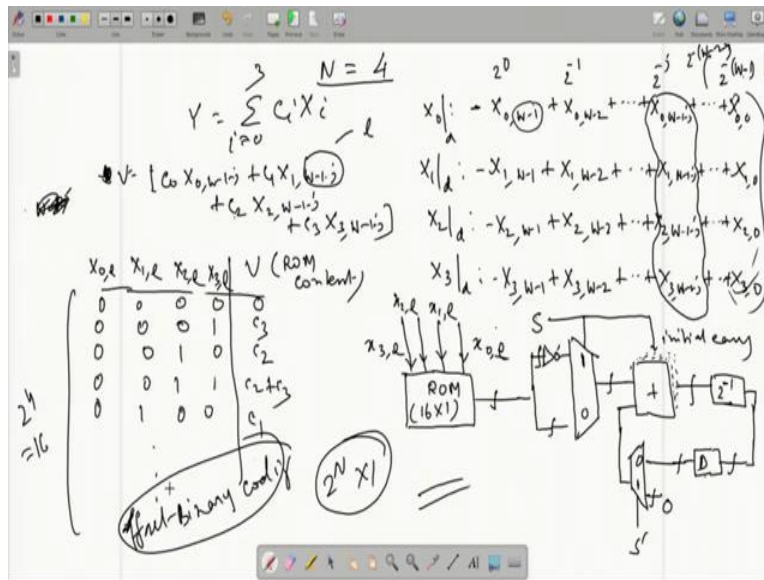
So, in this cycle this bit pattern will be presented to the ROM and this bit combination if it is all 1, I will store this value. If it is all 1 but 0, I will store this value. So, all the values of this will be stored in various locations, various addresses of the ROM, depending on what bit pattern I have. And so it will be bit serial like for this for  $X_0$ , there will be 1 line through which first this bit will come, then this bit will come, then this bit will come and like.

But, similarly there will be another line for  $i$ ,  $i$  n data, but single line, so bits will come serially. First this bit, then this bit, then this bit, so in a bit cycle these bits will be coming parallelly to the ROM that will point to an address. And that address depending on whether this 1 or 0, 1 or 0, 1 or 0 any particular value stores. So, that value will be common, common in the sense like if all are one here that value stored will be this.

If next time all are also 1, the address presented will be all are 1. Same fellow, same value will be faced, because address location does not change so and so. And row from the ROM this content will be come out, that content have to multiply by this power of 2 of course and add. And when it comes to MSB, ROM output whichever comes that is out of this combination,  $C_0$  into this,  $C_i$  into this,  $C_N$  minus 1 into this.

And this can be 0 or 1, this can be 0 or 1, this can be 0 or 1, so combinations will be the same thing. That output will be negative by again 2's compliment and then I will have to add the results for as I obtained from various columns. Then that I will get this  $Y$ , this is the philosophy so no multiplier is required. To explain it further, I have taken a case of 4-bit  $W$  is 4 sorry not  $W$  4-bit, I mean  $n$  4 numbers,  $n$  is 4,  $X_0, X_1, X_2, X_3$  w bit numbers. I will take as a case, I will explain this further.

(Refer Slide Time: 17:44)



Similarly,  $X_1^d$  (17:48) minus  $X_1^d$  data is  $X_1^d$  bit  $W$  minus 1 times  $2$  to the power  $0$ . Plus again data  $1$ ,  $W$  minus  $2$  to the power minus  $1$  dot-dot-dot and  $X$  to the power minus  $j$  this times this.  $X_2$  by  $d$  this into  $2$  to the power  $0$ , this into  $2$  to the power minus  $1$ . So, meaning is this into  $2$  to the power  $0$ , plus this into  $2$  to the power minus  $1$ , plus this into  $2$  to the power minus  $j$  so on and so forth.

So, I take a particular column this column, general column not MSB, when it comes to MSB no power of  $2$  multiplier but sign has to be inverse. So, accepting MSB I can be either here or here or here any of the columns. Not such columns have take general columns. I say,  $j$ th column there is  $2$  to the power minus  $j$  is the power and it is  $W$  minus  $1$  minus  $j$ .

In this case, I will have this summation  $C_0$  times  $X_0$  plus  $C_1$  times this summation. So, how many possibilities? This can be  $0$  or  $1$ , so either  $C_0$  into  $0$  is  $C_0$  or  $C_0$  into  $1$  is  $C_0$ , similarly here either  $0$  or  $C_1$ , here either  $C_2$  or  $0$ . Here also either  $0$  or  $C_3$ . Because these fellows can take only  $0$  or  $1$ ,  $0$  or  $1$  and that is independent of which columns I choose, because whatever be the column, each bit can take only  $0$  or  $1$ .

So, the possibilities of this summation, the various values it can take there same for all the columns that is the important thing. So, it can be form or truth table like this, I will not fully form. If I call this  $S$  dot  $S$ , suppose I call it say  $V$ , which is ROM content. And I have say  $C_0, C_1, C_2, C_3$  sorry not  $C_0$  very sorry in other way. This  $W$  minus  $1$  minus  $j$ , this  $W$  minus  $1$  minus





will happen is the output of the ROM is not a single lined bass, so the bass will denote by this. Either I have to negate when I am in the MSB, else not negate.

So, either way flag is a race is nothing, there will be counter is start counting, counting 0 when I am in the LSB cycle, counting 1 when I am in the next to LSB dot-dot-dot. When this  $W$  minus 1 is cycle that time this flag will be 1 else 0. And this what will this flag do is this, it go through a multiplexer 1, 0 and here is  $S$  this is coming here.

It means as long as  $S$  is 0, this data will go through this. There is as long as I am not in the MSB cycle, ROM output whatever be the one depending on the address given here, that will come in this way and come out of this bass. When I am in the MSB cycle, then this will go that is  $S$  is 1, so it will take this spot. Why inverter? Because you have to take negative, negative means whatever be the ROM output, I have to take 2's compliment. So, first have to compliment each bit so that not gate in each line in this line is bass.

And then one has to be added at the LSB that will do in a little later. So, otherwise first consider the other cycles not the MSB cycle, LSB, next to LSB or like that not MSB through what is now this. To start with suppose I am in the LSB cycles, so this data whatever the bits have come, LSB's of  $S$  corresponding ROM output has come. I have to multiply by 2 to the power minus  $W$  minus 1.

But what I will do? I will do multiply just by 2 to the power minus 1 and then put it in the loop. Again move it to the loop, next time again 2 to the power minus 1. Again moving it in the loop, next time again 2 to the power minus 1. So, I will move that result in a loop  $W$  minus 1 times, every time I want shift that is 2 to the power minus 1 shift.

So, eventually after  $W$  minus 1 cycle, it will go through this. The previous clock, the previous column  $V$  square 2 to the power minus  $W$  minus 2. I have not drawn the column here. When that data comes that is next to LSB, this column next to LSB, next to LSB, next to LSB, next to LSB again I go through this. That needs to be multiply by 2 to the power minus  $W$  minus 2. So, I will put that in a loop again but move it not  $W$  minus 1 times,  $W$  minus 2 times.

So, every time it goes through a loop, it will pass through a shifter one bit shifter cum sign extender so 2 to the power minus 1 and so on and so forth. So, eventually it will at the end  $W$

minus 2 iterations it will under go this loop and I will add all of them, this is the philosophy. So, what do? I will have an adder this not a single adder, it is a bass adder.

It is not a bit adder, it is a bass so actually there will be many full adders in parallel like this, bass will come. When I am in the LSB cycle, it goes in I pass it through a 2 to the power minus 1. 2 to the power minus 1 means, everybody will be shifted to the right by 1, and then there will be sign extension of the MSB by 1 bit to the left.

So, very easily done in hardware that is done and then I put it in a delay, I put it in a delay. It is not a single delay because it a bass so there will be shift of in parallel. And this time when it goes in, when it goes in a multiplexer, what is a multiplexer? It has another control is S prime. S prime will be one in the starting cycle one otherwise 0. If the starting cycle a 0, it is a bass with all 0s that will then go through here. Because so LSB cycle the corresponding ROM output comes goes through this. There is nothing to be added with, this is a studying case so I add with 0.

So, this 0s move in so this plus 0 come in and then they get real shifted by 1 bits staying in the loop cycle. Next time on what's? This is 0, if it is 0, it will follow this will go in. So, this will go in so you get here and next to LSB column result will come here, together they will add out of which is this will go through another loop.

So, again 2 to the power minus 1, but this will encounter 2 to the power minus 1 for the first time. Together if there in loop for together you know if I run this for W minus 1 time. The LSB column result will go through this W minus 1 times, so this power will come. The next LSB case came here not I the 0th cycle, but in the next to LSB cycle.

So, one is gun then out of total W minus 1 times iterations, one is gun because it is not coming not in the LSB cycle but next cycle 1 is gun. So, it will go through W minus 1 two times, every time 2 to the power minus 1 shift as it goes through this. So, this power will come but now they are getting added as again passing through. Then another result comes that requires W minus 3 times shift and I have already consumed two so that gets added. So, these fellows will rotate as usual, that fellow will be rotated W minus 3 times so on and so forth.

Only when the MSB comes, this S becomes 1 that time these fellows get complimented and I must add 1 in the LSB of that 1. Adding a 1 at the LSB and giving a initial carry when you are

adding 2 numbers, you want to add 1 at the LSB of one of the number is same as giving you add, giving a initial carry as 1.

So, what I will do is, I will give it as initial carry as long as it is 0, it does not change any result. Only if and it has to pass through this, S is 1 that 1 is going as an initial carry as a LSB position of this adders. So, you see this is a single implementation no multiplier required, but only it is bit serial. ROM is taking  $2$  to the power  $4$ , in general  $2$  to the power  $N$  into one locations.

There is a way to bring down the ROM size by a factor of 2. That is called binary off set binary coding, I think off set is giving in Parhi's book memory interesting technique, by which you know you add some extra logic gates outside, but ROM size can be brought down by 2. This is very widely used technique distributed arithmetic to bring down the complexity of multiplier. There is no multiplier directly used but this bit serial. So, 16-bit worked here in a faster clock, then the clock at which the data word is coming, but you have to pay that price.

So, that is all for this course, it is a good bye from me from here. I hope through this lecture series, I could impart some knowledge to you, because this is not something that is taught in everywhere. Resources are not widely available, only one great book is available, which is a compilation of all thrust techniques that is a book by Keshab Parhi.

All other rehearsal in a general preparation and all that. So, I just try to see that you know from here you can build up on your own. You know you read for that advance subjects, if that happens I will be very happy. Thank you very much. Wish you all the best, stay safe, bye-bye.