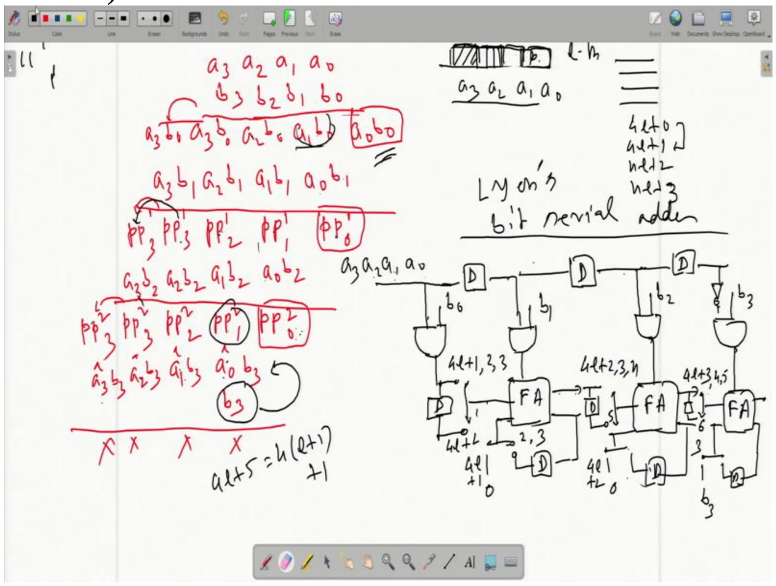
VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 36 - Bit Serial Multipliers

Okay, in the last class we are considering the multiplication of two binary words, w-bit words A and B. And, how to get C? We followed Horner's rule. But, then as an example we took 4-bit words, A and B and we carried out the multiplication. There was a multiplication table which is again for your sake I have re-written here.

(Refer Slide Time: 00:57)



So there was an array, dependence graph was an array in which 4 bits- a 0, a 1, a 2, a 3, they were coming parallelly through one side. And, four bits are b; b 0, b 1, b 2, b 3, they were coming primarily through another side. So, that was bit parallel or word serial structure.

That is, if the bits are coming in clock, like at Lth clock, I have got a word, whose 4-bits are a 0l, a 1l, a 2l, a 3l. That is in Lth clock LSB, next to that, next to that 4 lines, 4 bits. Similarly, b 0l b 1l, b 2l, b 3l. In the very next clock, l plus 1 bits will change. Here also, bits will change for A and B both. And, again you carry out another multiplication. So we will do multiplication again, again clock after clock. That was the purpose.

Now, we will consider another system, where instead of it is bit serial, bit parallel or word serial architecture, we will have a bit serial architecture where bits of either A or B, may not be both, either A or B, they will come serially in the system. There will be only one line through which first because we follow the LSB first convention, so, a 0 will come, then a 1, then a 2, a 3 like that.

We may consider a situation like this, where suppose you are calculating a filter output, y n as w 0 x n plus w 1 x n minus 1 and dot dot dot dot. Consider any multiplication. This is like A. This is like A. This is like B. This will be 16-bit. This will be 16-bit, or 4-bit, 4-bit, 8-bit, 8-bit, similarly here.

Consider this multiplication. Here, you see data is coming. So, nth clock, if it is 16-bit there will be 16-bit lines, 0 1 0 1 0 1 0 1. That will be our 16-bit word in two's complement form. But, multiplied by w 0 which may be a 16-bit word again; but that is constant. That does not change from clock to clock.

So, here if we call that B, so all the bits of B, they are time invariant. They have been fixed internally, built-in. And, bits of A; that is function of n, okay. If it is 16-bit, all the 16-bits, each of them contains 0 or 1, 0 or 1. But, at nth clock we will get one word; nth plus 1 clock, another word. So, at nth clock we have one product; nth plus 1 clock we will have another product value and that will…

We will consider this situation where one of the words is dynamic. It is coming at its clock from outside and that we are taking, that we take to be A. And, the other one, say B; it is built-in. So, it is constant. It does not change with time, okay.

For our example, we again consider w equal to 4; this one. Okay, this case where we consider 4-bits for A; 4-bits for B, all right. So, what happens when we convert a word serial i.e. bit parallel structure into bit serial, suppose you had originally 4 bit lines in parallel, one for a 0, one for a 1, one for a 2, one for a 3. But, now I have got one line through which I will first send a 0, then a 1, then a 2, then a 3.

So, if this is one clock period, originally, this was used and I had four lines. So over this entire duration, I had a 0 on this line, a 1 on this line, a 2 on this line, a 3 on this line. But now I have got only one line. So, this clock will be divided into four parts, this clock. So, this is one sub-clock. During this time, okay. This is one sub-clock.

So, in the first sub-clock I will say b 0. So, b 0 will come out first. Then, here I will send b 1. So, b 0 will be followed by b 1, then b 2. Not b, say a; because b is built-in. So a 0, a 1. Then, a 2 here followed by a 1. Then, a 3 followed by a 3. a 0, a 1, a 2, a 3. Let me draw again.

So, during this clock we will send a 0. So, through this line a 0 will be the first guy. Then here I send a 1, then a 2, then a 3, alright. So, if it is Lth original clock, original clock Lth, so I have got zeroth, first, second. So Lth, that will be the four sub-clocks or bit clocks. So, this will be 4 l plus 0. Then, this will be 4 l plus 1; then 4 l plus 2, 4 l plus 3. So, in 4 l plus 0, a 0 will come out. In 4 l plus 1, a 1 will come out. In 4 l plus 2, a 2 will come out. 4 l plus 3, a 3 will come out.

Then again, from Lth clock we go to l plus 1th clock. But, all the four bits again comes through serially; through one line. So, again, l plus Lth clock will also be divided into four sub-clocks and through them we will send out. So, it will be 4 into l plus 1 plus 0, 4 into l plus 1 plus 1, 4 into l plus 1 plus 2, 4 into l plus 1 plus 3. And, through them again we will send out new a 0, new a 1, new a 2, new a 3. And it will go on like that. So, it will be bit serial system then.

Now, keeping that in mind and keeping also this in mind that all the bits of B are built-in, they are not changing with clock. They are built-in, constants. Look at this a 0 here. a 0 is arriving here at 4 l plus 0. I am doing a 0 dot b 0. But, not including it; not counting it. I am throwing it away. So, this is of no use.

So at 4 l plus 0, a 0 comes. I do dot but then no use. At 4 l plus 1, a 1 comes. So, a 1 dot b 0. Then, at next clock, at 4 l plus 1, a 2 comes. I do a 2 dot b 0. Then, 4 l plus 3, a 3 dot b 0. And, then I sign extend it. And with a 1 dot b 0, what I add? a 0 dot b 1. a 1 dot b 0, I add a 0 dot b 1. Now, you see if there is a line, through this we are coming a 0. In fact, I should write a 0 bracket l, a 1 bracket l like that. But, that l I am dropping. Otherwise, it will become very clumsy here. But, actually I should write a 0 l, a1 l, like that. The l I am dropping but you keep that in your mind. Then a, okay.

a 0 came at 4 l plus 0; of no use. Then, a 1 came at 4 l plus 1 and then dot AND gate. AND gate with whom? All of them AND gate with b 0; a 1 dot b 0, a 2 dot b 0, a 3 dot b 0. So, other input is b 0, built-in. This is built-in. So what I get here is a 0 dot b 0, which I do not use. Then at 4 l plus 1, a 1 dot b 0. At 4 l plus 2, a 2 dot b 0; 4 l plus 3, a 3 dot b 0, right.

Now, start with a 1 dot b 0, this guy. This is to be added a 0 dot b 1. But, a 0 came here at 4 l plus 0, okay. And, I have to add with, a 0 came here at 4 l plus 0. a 1 dot b 0, this is formed at what clock? When a 1 came. When a1 came? At 4 l plus 1. At 4 l plus 1, I am having a 1 dot b 0. So, at 4 l plus 1 I should also have a 0 dot b 1. That means a 0 which came at 4 l plus 0 should be available at 4 l plus 1 also. That means I should put it in a delay. And then I need a 0 dot b 1.

So now a 1 dot b 0 at 4 l plus 1 it has come. This will get added with a 0 dot b 1. a 0 came at 4 l plus 0. After one cycle delay, it is coming here at 4 l plus 1. So, at 4 l plus 1 they will be added. So, there will be an adder, full adder. So, a 0 dot a 1 dot b 0 at 4 l plus 1. That time a 0 has come because after one cycle delay here. So, that when it is a 1dot b 0, that time it is a 0 dot b 1. They are to be added with 0 initial carry.

So this full adder, I am giving a switch. Through the switch it is touching here. Touching here at what time? 4 l plus 1; that time it is touching this. And, that time I have got a 0 dot b 1, there carry input. Carry will start here. Initial carry is 0. So, this switch will touch it at 4 l plus 0. Sorry, 4 l plus 1. That is the time 4 l plus 1. That is the time addition is starting here.

You touch it at this point 4 l plus 1. That is the time I am starting this addition. So, at 4 l plus 1, a 1 dot b 0 here and a 0 dot b 1. They take initial carry 0, produce a result, produces a carry. This carry will be now moved here in the next cycle. Next cycle we have a 2 dot b 0. a 2 comes here at 4 l plus 2. So, in the next cycle a 2 dot b 0 comes. That time a 1 has moved here, a 1 dot b 1. So a 2 dot b 0, a 1 dot b 1. They get added and with previous carry. So, this carry comes here with a delay. So, whatever carry got generated in this cycle, that in the next cycle moves here.

So, it is 4 l plus 1. It is 2. I am not writing 4 l again. It is 2 comma, sorry into 4 l plus 2. I do the addition. Again, result comes out; new carry comes, then is a 3 b 0. a 3 came here at 4 l plus 3. That time a 2 has moved here. So, a 3 b 0 and a 2 b 1; they get added with the previous carry. This addition is taking place at 3; okay this is 1, 2, 3, alright. Again, new result, new carry.

But, now I have to do the sign extension. a 3 b 0 should again come to the this input; come to the adder input again, repeated. The way this sign extension is done, we put it through a delay and move it here. So, it is 4 l plus 1 here, 4 l plus 2, 4 l plus 3. So 1, 2, 3, during these occasions, switch is touching this. There 4 l plus 3 I have a 3 dot b 0; a 3 dot b 0 here. Then, at 4 l plus 4 it moves here. So, switch also moves here, 4 l plus 4.

Remember, 4 l plus 4 is nothing but 4 into l plus 1 plus 0. So, for the l plus 1 cycle, next cycle it is zeroth. So that time a 0, new a 0 should come because l plus 1 is what? The zeroth bit cycle. That time new a 0 would come. But, my switch is moving here, not here. So new a 0 times, new a 0 dot b 0 even if it is formed here that will not go into full adder. This will go into full adder. That is how we are ignoring this. So this ignoring, that is how we are ignoring this.

Now this addition is done, addition is here. So I am getting this result, this result, this result, this result. This is coming at 4 l plus 1. But 4 l plus 1, this will be discarded. So, 4 l plus 2 that times a 0 dot b 2. So a 0 came here at 4 l plus 0. It came here at 4 l plus 1, now coming here at 4 l plus 2. So it is and then AND gated with b 2 and FF full adder. So what will happen?

This will touch this at 4 l plus, this I will discard 4 l plus 1. So 4 l plus 2, 3, 4; 4 l at 4 l plus 2 a 0 came at 4 l plus 0, then here at 4 l plus 1, here at 4 l plus 2. So I have got a 0 dot b 2 and adding with result that is at 4 l plus 2; new result, new carry. This is the carry input. Again, initial carry here, when you do these addition initial carry is 0. And this addition you start at 4 l plus 2, okay.

At 4 l plus 3, a 1, a 1 came here at 4 l plus 1, so it is at 4 l plus 3 here, a 1 dot b 2 with this result and the previous carry. New result; new carry. Then, this came at a 2 b 2. a 2 came in at 4 l plus 2, now it is 4 l plus 4, is not it? 1, 2, 3, 4. So, 4 l plus 4 a 2 b 2. So, this result comes here, new result, new carry goes on. Then, this carry again comes but this result is repeated. So this result will be repeated here, it is a sign extension.

So, 2, 3, 4 it is at 5; so 4 l plus 5. And, 4 l plus 5 means 4 into l plus 1 plus 1. Okay, 4 into l plus 1 plus 1. So for the l plus word cycle per word, bit cycle 1. Okay, bit cycle 1. That would have come but that we are ignoring. Okay, that we are ignoring. Let me, alright, that gets ignored. When it comes to 4 l plus 5, instead of taking this previous result we move the result here. So, whatever result is generated, that is ignored. That is how we ignore this result, and this result.

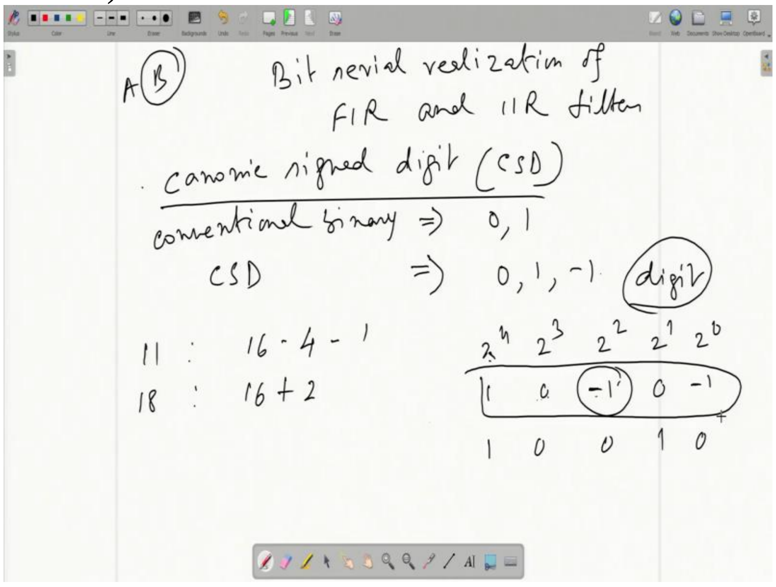
And now last one, we put a delay again. So, a 0 came here at 4 l plus 0, 4 l plus 1, 4 l plus 2, 4 l plus 3. And, you need an inverter. These all are NOT gated, b 3, once again you have got adder; full adder, carry. So, addition starts at what? This was at 4 l plus 1, 4 l plus 2, 4 l plus 3 because this one we will ignore, 4 l plus 3.

So at 4 l plus 3, this previous result will move in here. So they will be switched. It will be here at 4 l plus 3, then 4, then 5, a 0 bar, then a 1 bar, a 2 bar. a o came here at 4 l plus 0, here after three cycles 4 l plus 3 followed by a 1, followed by a 2, followed by a 3 and they are NOT gated.

And when it comes to 4 l plus 3, that time this result, previous result will be extended. Okay, 4 l plus; sorry. This is 0, 1, 2, 3, 4, 5, 6. So 3, 4, 5 here. When it comes to 6, this will move here, alright. So, this is the result, these are previous carry. Now here when you do the addition, this will be given as initial carry. So, initial carry will neither be 1 or 0. It will be b 3. If b 3 is 1, it will be 1; otherwise 0. And the addition starts at 3, so it will be at 3.

So this is a bit serial adder. It is called Lyons's serial adder. There are other forms of bit serial adders also, various, I will just consider these. This will give you an idea about what is meant by bit serial adder. Our next topic will be, I mean going beyond this multiplier. I mean, why not take the whole digital filter itself and have a bit serial realization? That is the next topic, bit serial realization of FIR and IIR filters.

(Refer Slide Time: 19:13)



A (B)

Bit serial realization of FIR and IIR filter

· canonic signed digit (CSD)

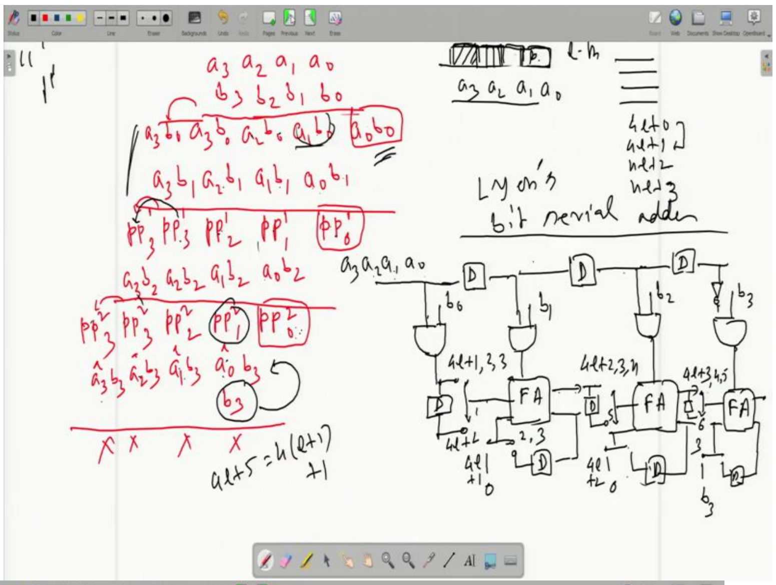
conventional binary ⇒ 0, 1

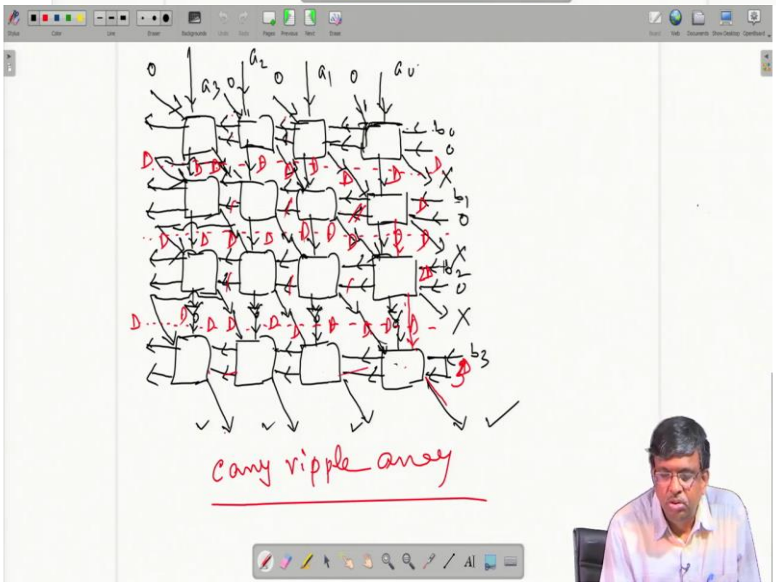
CSD ⇒ 0, 1, -1 (digit)

11 : 16 - 4 - 1

18 : 16 + 2

2^4	2^3	2^2	2^1	2^0
1	0	-1	0	-1
1	0	0	1	0





Here one thing we will teach, I will just touch upon called canonic signed digit, CSD. Suppose we are doing this multiplication. If say b 1 is 0, binary 0. Then you see, we know the AND gate result of, I mean any bit with b 1. If b 1 is binary 0, a 0 dot 0 is 0. This is 0, this is 0, this is 0. And when you add, no change; you get the same result which means this entire addition step here can be skipped. So if you have more and more 0s, this is addition, intermediate addition steps can be skipped which gives rise to a lot of you know, saving in hardware.

Similarly, if you go further beyond, see this array. What does the array do? Array, if you have got some of the bits 0, the entire operation here is not required. So this row can be skipped, okay. For instance, suppose b 0 is 0 or say b 1 is 0. So b 1 dot a 0 which is 0 plus incoming, so incoming remains as it is. And, carry? No carry gets generated and same here.

So, these four easily go away. So, you get setting in hardware which means, if it is like this that out of two numbers, you multiply A, B. At least one of them has got many bits equal to binary 0, then your multiplication complexity will go down hugely because many rows in the overall multiplication table by Horner's rule will not be required. Because you already know the result, no composition no addition is required here. Because nothing changes when you multiply your word by binary 0 and add with something incoming, alright.

Therefore, why not do something so that one of the numbers multiplicand has lot of 0's. Now, here you can ask me the question that look B is not in your hand, B is in the hand of the customer. Whatever B is, or your design, whatever B you get, you have to accept that. How can you say it will have lot of binary 0's? There this mode of this canonic signed digit representation system, it is a data format; it is a way of encoding a number, that comes in play.

CSD, it gives you instead of by the common binary system which gives you 0 and 1, that is conventional binary, you will have 0, 1. But CSD will give you three things: 0, 1 and minus 1, okay. And using this, and that is why call a CSD digit because bit means only two possibilities. Anything that has more than two possibilities, we have to call it a digit. So that is why we say CSD digit, canonic signed digit not bit. Cannot be canonic signed bit, not CSB but CSD.

But using this provision of having a negative 1 also (minus 1), there are powerful ways by which given the decimal number you can encode that into this digits, using these digits into a word which have lots of 0's. In fact, you can make sure that at least 50 percent of the digits in the representation are 0's. So, those 0's will not contribute anything in the multiplication.

When it is minus 1, minus 1 times some number is nothing but minus times minus of original product, okay. So first you carry out the normal product having a binary 1. Whatever is the result just take a two's complement of that, that will give you the negative of that. I am showing a way. Suppose we have 1, 11. So I am not considering fractional number, for your convenience I am taking full integers. 11, I can write like, you know, 16 minus 4 minus 1. 16 means that is if I have got 2 to the power 4 or 2 to the power 3, 2 to the power 2, 2 to the power 1, 2 to the power 0, so 5-digit word.

So it is 2 to the power 4, I will give plus 1. Then 2 to the power 3, 8; there is no 8 here. So 0 into 2 to the power 3, 1 into 2 to the power 4, minus 4 means 2 to the power 2. So minus 1 times 2 to the power 2, see minus is coming. Then 2 to the power 1 means 2, there is no 2 here, so 0 into 2 to the power 1. And, again minus 1 means, 2 to the power 0 is 1, so minus 1 times 2 to the power 0. So this will be the word.

So, you see I have coded so that there are two 0’s. In CSD, it is always I mean the coding technique is such that whenever you have got one non-zero digit, it is always guaranteed that the left neighbor and right neighbor will be 0’s at least. I mean, neighbors further to the left, further to the right also could be 0’s. But at least immediate left, immediate right will be 0’s. Here I have got only right; there is nothing in the left. And here I have got only to the left, nothing to the right. Here I have got both immediate left, immediate right; they are 0’s.

Take another example, say 18. 18 can be written as 16 plus 2 to the power 4, it is very easy, 2 to the power 1. So this will be 1, 0; so 1 into 2 to the power 4, 0 into 2 to the power 3, 0 into 2 to the power 2, 1 into 2 to the power this.

So see it has got two neighbors, left and right both are 0 and neighbor further to the left also 0. So on and so forth this way you can carry on. This is a powerful way, actually it is an algorithm by which the encoding can be done.

But here purpose is this that we can have plus 1, minus 1 and of course 0. But we can make sure that every non-zero digit has got at least one 0 to the left, one 0 to the right which means approximately 50 percent of the word will be filled with 0’s. And, presence of the 0’s simplifies my hardware because when I do a multiplication those 0’s contribute to nothing. No, you know, no hardware is required to do multiplication using 0’s; so that gives rise to huge saving.

So CSD is a powerful technique. When we go for bit serial realization of filters, we will assume that the filter coefficients have been given in CSD form and then we will proceed. This we will do in the next class. Thank you very much.