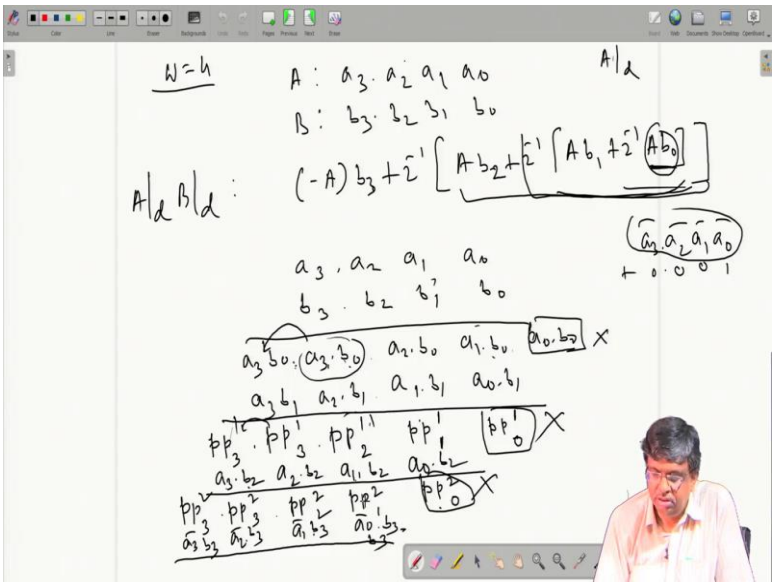
VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 35 - Carry Ripple and Carry Save Array

(Refer Slide Time: 00:28)



So the last class we are considering multiplication of two numbers using Horner's Rule and numbers were represented in two's complement format. We explain the Horner's rule, the chain rule and then we took a special case of 4 bit numbers. What is capital A, what is B, I applied Horner's rule and then led to a tabular of multiplication where you are truncating, I mean throwing LSB and then there is a sign extension and all these things and that is how you get the final result.

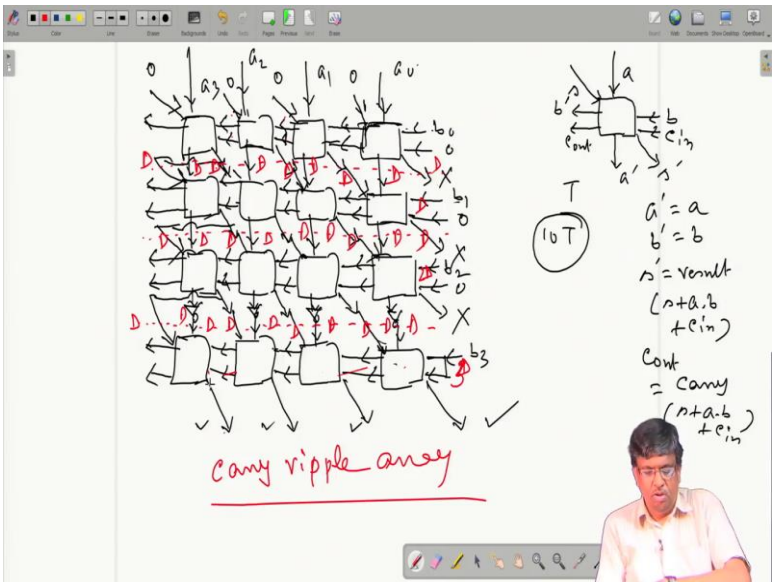
Now the question is, if I have to draw an architecture, I should draw first a dependence graph here, dependence graph. How to draw a dependence graph in this case? So what is the basic operation we have here? You see take any just this operation suppose here, you have got something called pp2 1, pp2 superscript 1 that is given to you. a 11, sorry a 1 and b 2 they are coming in, so you do AND gate. So a 1 dot b 2 so first you do AND gate of them, then you add with this, there is an incoming carry from the previous edition that is from this row.

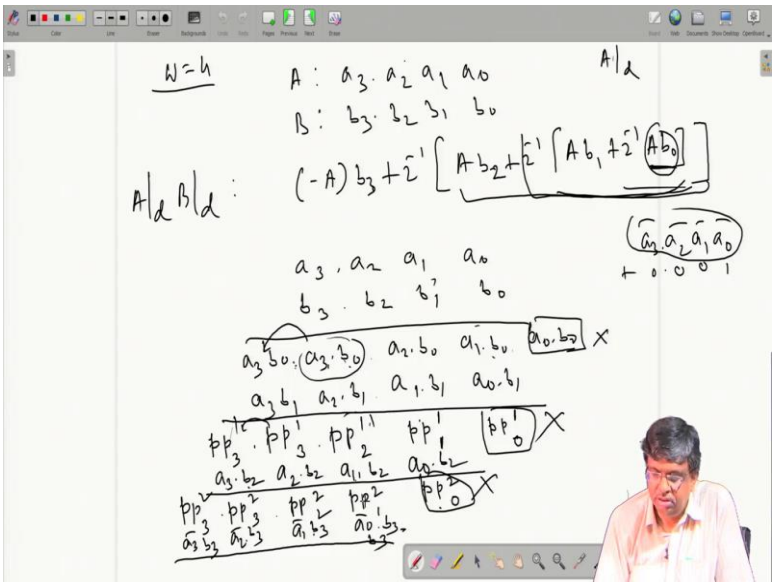
So incoming carry a 1 dot b 2 and pp1 2, they get added, real comes out, new carry moves out. So three things are added; one is coming from previous stage this pp, result of previous stage, another is a input, another is b input. a dot b plus that thing which is coming from the previews state. They get added with the incoming carry, result output carry. These are basic processing and this is common everywhere.

There I put some deviations for instance, there is in time, how to do that in architecture? We will see because this is a departure. The same thing gets, I mean repeated here, how to implement that or there is an a 3, there is a NOT gate here. Suddenly, a 0 bar, a 1 bar, a 2 bar, a 3 bar comes. So long it was not a 0, a 1, a 2, a 3; a 0, a 1, a 2, a 3.

So those things have to be incorporated in the architecture, but first a basic dependence graph where the basic operation is this, a input dot b input plus some incoming stuff say pp with an incoming carry gives two output, one is real output another is new carry output. So that will be our basic process in cell.

(Refer Slide Time: 02:52)





So suppose we design a processor. There is an incoming stuff is like the partial product pp s. So the orientation is very important in DG, I am making it angular because this is actually be,

where in the final circuit. I am making it angular because when I draw the full dependence graph, you will see it has to be angular then only it will make sense.

When I draw the DG then we will see why it is angular. This is s, this is a input, this is b input, this is c in, carry in. So what it does, a dot b plus s that comes out as s prime real comes out s prime, carry goes out as c out. Either a which moved in, I will take the same a out which is a prime, the b which moves in that will come out as b prime which is same as b.

So we write the equation, a prime will be same as a, this output a prime is same as input a, output b prime is same as input b and s prime is a result part of what addition? s plus a input dot b input plus incoming carry and what is the other output c out? It is the carry out, carry of the same thing, same addition. This is my job, so I define my basic processor. Then I draw the array, let me draw and then you will see that whatever we are doing is the correct thing.

These are the cells, the same processor which is shown by this box we have it here in plenty. I will give a 0 here that will continue, I will draw it later this part. Similarly, we have a 1 here and similarly, here you have got, I give the b input here b 0 carry input, initial carry is always 0. And b will continue to b 0, as you know whatever a input comes a 0, same a 0 will come out, same a 0 will come out here and here also.

Similarly, a 1 whatever a 1 comes in, the same a goes out as a prime. So here is a 1 that a 1 comes in, again comes out as a 1, the a 1 comes in and comes out as a 1.

Similarly, a 2 goes in, a 2 goes out, a 2 goes in, a 2 goes out, a 3 goes in, a 3 goes out, like that. Similarly for b 0, b 0 went in, b 0 came out because b comes in, it comes out as b prime, which is b prime is same as b. So b 0 in b 0 out, again b 0 in b 0 out, so on and so forth, initial carry is 0. Then this at s, initial s is 0, whatever going is this, we are giving a 0 from vertical direction then a 1, a 2, a 3 and b 0 we are pushing from right, a 0 dot b 0.

But it does not have to be added with anything from behind because nothing has computed so far, so that is 0. That 0 we are providing at the initial incoming stuff. So it will be a 0 dot b 0, a 1 dot b 0, a 2 dot b 0, a 3 dot b 0. So a 0 dot b 0, a 1 dot b 0, a 2 dot like that, so a 0 dot b 0 will add with 0 at 0 initial carry.

So result will come but this result we discard. Then b 0 goes here, a 1 dot b 0 that will get added with 0, this we have seen a 1 dot b 0 it comes as it is, it does not get added with anything from top, so added with 0 actually, 0 plus this. Carry from this side actually no carry. So in our circuit, there is a provision of carry but for this row no carry actually is

generated. Here 0, the same 0 will come out, so a 1 dot b 0 plus 0 plus 0, so result will come new carry. Carry is 0 again because this is 0, this is 0, this is a 1 dot b 0, that is the result.

So only a 2 dot b 0, this is 0 carry and 0 in, so it will be just a 2 dot b 0 that is the result that will come out and then again, a 3 dot b 0 plus 0 and here also the 0 carry, so that will come out. Now, this we have discarded, a 1 dot b 0 is to be added with, we start with b 1 now. So a 0 dot b 1 that a 1 dot b 1, a 2 dot b 1, a 3 dot b 1, so a 0 dot b 1 then a 1 dot b 1, a 0 dot b 1 is added with this carry generated w moves to this side, result will be here which again we will discard.

So a 1 dot b 0 then will be a 2 dot b 0, the previous the whatever result we obtained in this stage, that is a 1 dot b 0 with that a 0 dot b 1. a 2 dot b 0 was obtained at this stage with this a 1 dot b 1, a 3 dot b 0 was obtained in this stage with that a 2 dot b 1 and then sign extended with that a 3 dot b 1.

That is what this we will discard. So a 1 dot b 0 whatever be the plus 0 plus 0, this is result that was generated. With that we must have a 0 dot b 1, so we give b 1 here and again, 0 initial carry. So a 0 dot b 1 that gets added with this. So you see this is the incoming edge, this is the outgoing edge, they join hand in hand, s is prime, they join hand in hand.

So previous result from that stage plus a 0 dot b 1 plus initial carry that will be the new result, that we come out here, again we discard. But it will generate and same b 1 will come out, it will generate a new carry that carry will be here, that carry. So now it will be a 1 dot b 1 plus whatever result came out. That and this carry that will generate new result, it will not be discarded and will generate a new carry and this b will continue as b 1. Then again, a 2 dot b 1 plus previous result plus this incoming carry that will generate new result and new carry and b will continue that is b 1 will continue. Here as if I need sign extension that I am coming to.

Now you see here a 3 dot b 0 that was a previous result. But it is a 3 dot b 1, a 3 dot b 1, that a 3 is one input, b 1 is one input first with your AND gate but then we have to add with this result, which was obtained I mean not on that column, we obtained here. So a 3 dot b 1 it will up to be added with this result. Already a 2 dot b 1 plus this result we did, but this result now gets extended and then a 3 dot b 1 it gets added and of course with this carry.

So this extension we do like this. This is drawn and brought back here like this. So a 3 dot this b 1 plus the same result, which is brought back here is extended form and this carry that

will generate new carry and this will continue. They are of no use here but result. Now again, this result will come here, we have got a 0 initial carry 0.

Since a 0 dot b 2, now b 2 a 0 dot b 2 and previous result they get added with initial carry 0, new result which is discarded but carry moves in, then previous result a 1 dot b 2 and so on and so forth. So a 0 dot b 2 and this previous result comes here, it gets added and 0 initial carry, so this result comes which is discarded, b 2 continues, carry of their summation is important, result is discarded but carry moves.

This carry that a 1 dot b 2 plus this previous result and this carry that will generate another new result, new carry, b continues as b 2. Then again, a 2 dot b 2 plus previous result plus this carry that will generate new result, new carry and this continues. Then a 3 dot b 2 plus again, it will be this result repeated, you can see easily this result.

This result it gets repeated and with a 3 b 2. So it was used in the previous stage when you had a 2 b 2, that time it was used to add, the same is repeated here, which is the sign extension went from a 2 dot b 2 I have a 3 dot b 2. So when I have got a 2 dot b 2 here, I use this result for adding, the same result will continue again whenever a 3 dot b 2, That is why I bring it back here. This is the sign extension, this is how it is to be done in practice in hardware and b continues as c continues, I am not bothered about them. These are result.

Then there is one small change down, next time when I go this is discarded again, but I do not have, I will not have a 0 dot b 3. Earlier I had a 0 dot b 0, next row a 0 dot b 1, next time a 0 dot b 2 but now you have a 0 bar dot b 3, a 1 bar dot b 3, a 2 bar dot b 3, a 3 bar dot b 3. That means you have to be NOT gated. That is why this a will pass through this, then come here, b will pass through this then come here. This will pass through these then come here, and this will pass through this then come here.

And then I give b 3, so I give about b 3, alright. a 0 bar b 3, and this previous result, initial carry is 0 but remember I am adding b 3, so I can take b 3 as initial carry. So these and these added with b 3 being the initial carry. So same b 3, I can bring back here same b 3. What is initial carry that gets added with this previous result, new result this I will use.

So b 3 continues new carry, that gets added with this, so a 1 bar dot b 3 now, earlier it was a 0 bar dot b 3, now a 1 bar dot b 3 with previous result new carry, result goes here. New carry continues, this continues b 3. Again, a 2 bar b 3 plus previous result and new carry that will be the new result, this continues new carry now after this addition.

Now a 3 bar dot b 3 plus it was this result again, you can easily check, so this has to be repeated which is a sign extension. You get a result. These two fellows go out and my final result are here. So 16 processing sensor required and it is a 2-dimensional DG. One is maybe x-axis, y-axis. One is for the a fellows, the index of a 0, 1, 2, 3, so it is going in this direction; index of b 0, 1, 2, 3, is going in this direction. So this will go this direction, this direction x, y.

Now, what is the critical path of this DG? What is the critical part of this DG? You can see one thing. If I start from here, this goes here no delay, this goes here no delay, this goes here no delay, then I come down by this arrow here, here and again this way, this way no delay. This will be the longest path. So in this, cell takes T time it will be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10 T. That is the critical path.

What you can do, you can do (())(19:37). They are all forward path, same direction. So what you can do, you can add a delay here at all the edges. So obviously, they do not get added, there is a cut. This delay is no longer, this path is not delay free. Now delay has come and because you have got delay, you have given delay, here this because everything was coming now suddenly one cycle delayed, so you should have a delay here also. Then again, you can apply cut set here, I can bring D D D in all the edges.

Now because it is now delayed twice, it was coming otherwise straight, but now delayed here, delayed here, twice delayed. So this should be also given after two cycles, twice delayed. This should be given after one cycle, then again here you can cut it like this, all forward path. So there will be delay, you do not have to take that delay out from anywhere. So now this will be 2D, this will be 3D but here it is three cycle delayed, so I should give b 3 at all those after three cycles here, after two cycles here after one cycle here. Now I can walk to this, now question is, can I have a cut? Can I cut like this and bring delay here?

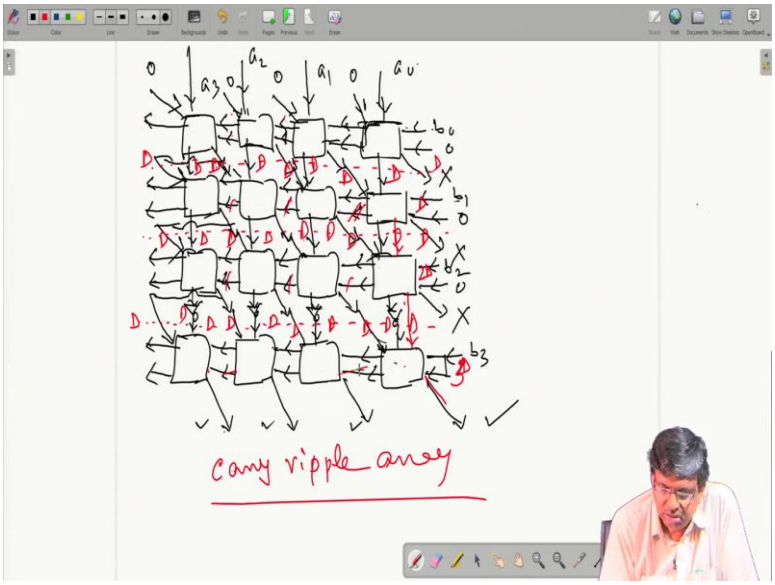
Now, the problem is if I have to bring delay here, this is in one direction right to left that see the arrows from right to left, and this is from left to right. From left to right I have got delays. So these delays will go, have to be taken out and have to be brought in here. Which means whatever delay free path I got here along this line, again whatever delayed path, whatever delay I could bring in this path here, those delays will disappear. This will go here, here; this will go to provide delays here and like that.

So I cannot go anymore. This array is called carry ripple array. This is a famous array but multiplication array, but this problem is you see the critical path. Like, you know, you can do that job here only when this carry is available and for that you must complete this job

beforehand so that this carry is available. But to do this job this carry is required, which means before you do this job you must do this.

And again, this requires this carry, which means before you do this job, you need this and so on and so forth. So they are in sequence. So that is why there is a dependence. This depends on this, this depends on this, this depends on this, this depend on this, this depends on this, this depend on this and so on and so forth. That is the problem of this array. But otherwise is a famous array.

(Refer Slide Time: 23:26)



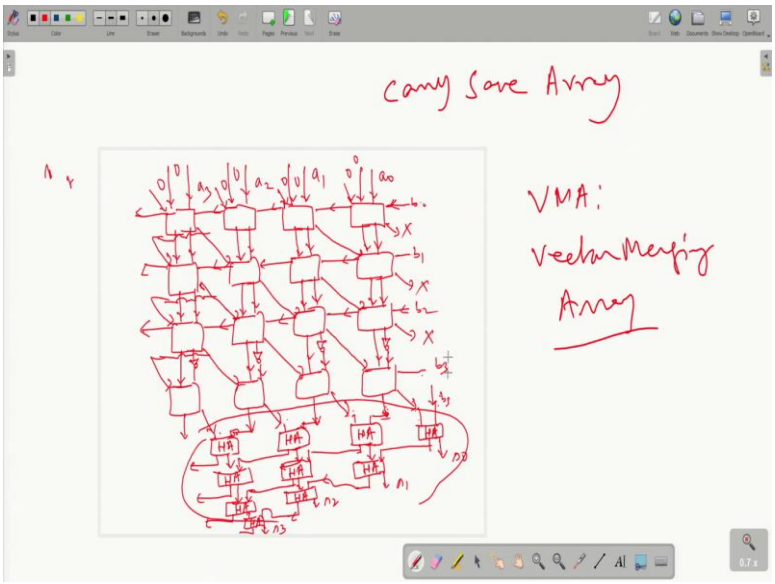
Now, there is another array called carry save array where it is a tricky version of this. There what we do? Let me explain this way, consider this, this carry. There is a problem, this carry is causing the dependence because you must have this job done first then only you can start this job. And then you must have this job done first. Then this carry available, then only you can take up this job. So on and so forth.

We can break this dependence, suppose I do not add this carry. So this result will be incorrect, but this carry I bring back here, this carry. So this gives correct result. But if I do not add it, the new carry, this real will be incorrect, finally it will corrected here. This carry will not have the contribution of this carry, fine.

So this result we have some something incomplete but it will proceed like this, but this carry when it moves here, it will get added with this, this additional component will be having some contribution here, that will come, that will get added here. It can be shown that correct

result will come out. So I move this carry and this I do at every places, so I moved these carries, this I cut. This I cut, this I cut, this I move here, this I cut and this I cut, like this.

(Refer Slide Time: 24:44)



So what I get, I draw another figure. This will continue as before. These carries, in fact this 0 initial carry instead of writing there, we can as well we can bring here. So 0 initial carry and there this carry I will instead of doing this way I will do this way. Similarly, this 0 carry this whatever carry I had in this direction that I bring back here. This also 0 carry whatever I would have had here, this I bring back here and then again 0 carry whatever I would have had here, the same I bring back here.

Then b 1, b 1 will and here of course, I give 0's initial value 0 this is absolutely unchanged. This results go as it is. This carry moves downward, instead of going this way I bring it back here, only b moves, result comes here. Whatever carry I would have had, I bring it here downward, this moves so result comes here, here also result comes. Whatever carry would have gone this way, this I bring back here. This output, this carry goes downward. Then again, this result we discard.

Here we get b 2 of course, this carry will move downward. This carry will move downward, this carry will move downward and this carry also move downward, this further continues. And here I have got b 3, now b 2 I have got b 3 to be, I was adding b 3 here additionally, that will have to be added with, along with this I have to add b 3, these two to be added.

Then similarly, this and this they get added, it is a result. But the carry is here and here again then this result, this result and this carry should be added, is not it? Because this should move

to here. Carry means what? It should move here then only get added. So these two should be added, these two should be added, these two, then this carry and this carry should get added here because it should move to higher position. There is a meaning of carry, these two should be added. So these two, these two and here I got, I forgot to add the sign extensions.

These are all what you did last time, sorry. So this generates a result, this carry comes, these two should be added. So this and this, this and this, this and this, this and this, this carry I believe there is no overflow so this should be 0. So, what we do? We first passed these two and half adder, half adder because two input adder is half adder, so what would be the utput? This is the sum output, the LSB but….Similarly, these two I add with an half adder but this half adder can generate a carry, that carry should get added with result here. So this result further gets added with this, these two gets added here.

So this half adder again generates a carry, that carry should come here. So its result was coming here and this is coming here from here. So this will again generate next s 1 but its carry, this carry should be added to the result coming out here. So one more half adder. So now this will generate a result, which is s 2 but this will again generate a carry.

But before that let me come here, these two again will get added. Half adder it generates a result, that result will get the carry from here. This carry will go out, there is no overflow so I am assuming them to be 0. This also will move out carry, but the result here will have to be added with the carry coming from here, then only it will be a correct result. So HA still a carry is coming. So here this will go out.

This will be the final result, s 3. This part is an additional thing, we have to add and is called VMA, Victor Merging Array and this array is called carry save array. These are purely bit parallel or what serial structure because all the bits say 4 bits are coming parallelly. There is a whole word is coming serially. What word? They will coming in terms of clock.

So maybe in the Lth cycle I will have a 0l, ,a 1l, a 2l, a 3l, will form 4 bits of a word, Lth word that will come in. Similarly, b 0l, b 1l, b 2l, b 3l, they are the 4 bit of a Lth bit, Lth bits of a 4 bit word b. In the Lth cycle, I do this kind of thing. So basically, it is a word serial or bit parallel structure but the other extreme is bit serial multiplier, which is what we will consider in the next class. Thank you very much.