

Architectural Design of Digital Integrated Circuits
Prof. Indranil Hatai
School of VLSI Technology
Indian Institute of Engineering Science and Technology, Shibpur, Howrah

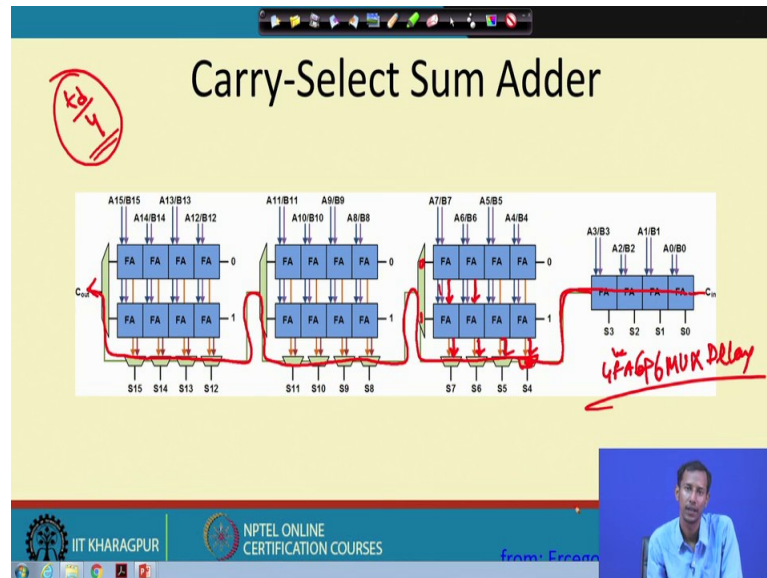
Lecture - 20
Efficient Adder Architecture (Contd.)

Welcome back to the course on Architectural Design of ICS. So, in the last class, we have seen that how we can optimize based on the restricted fan in and fan out option. So, how we can and that means, optimize the CLA whenever CLA in terms of. That means, whenever we are generating or the carry or we are writing the expression or implementing the expression for c carry, so, at the time how we can define the fan in for each of the levels. Whether, I can that means, if I restrict the fan in, so, at the time a obviously, it will increase the level.

So, what will be the group size and then what will be the level size based on that we are having we have seen four kind of topology. And based on that we have seen that the variable size implementation of the carry logic is it is better than the in terms of delay consumption, it is better than the fixed implementation of the same carry expression.

So again today, we will start with another architecture of this adder which is carry select adder. So, we will start this carry select adder by this name of if this carry select adder, it says that we have to based on this selection of the carry, can I do this addition operation; that means, what is the intention to make it faster.

(Refer Slide Time: 01:57)



So, how can we do this carry or what is that means, working principle of this carry select adder? So, in the carry select adder, in the ripple carry adder, what we do? Basically, we in the in each of this case we consider this full adder cell then this carry is basically propagated and at the output we get the final carry out. And each of this particular case, we got sum bit.

But in this carry select adder, what we do instead of that we do parallelly to that mean ripple carry addition operation. For what, one considering this carry input as 0 and another one considering this carry input bit to 1. That means, why I need to do that because in case of this ripple carry adder; that means, if my carry input bit.

So, at the time how can I design or how I can start with the design? That means, now what we are doing; we are making the 4 bit group. So, each of this is 4 bit and this is also 4 bit. So, at the very last of this; that means, had the very last group what we are doing, we are not following the corresponding carry select addition, what is the; that means, main idea behind carry select operation selection adder operation; that means, operation. So, that we are not doing.

So, what we are doing, from here, actually again if I just want to; that means, go back to this. So, if I just this if I just divide. So, at that time means what, I am having these particular four blocks. So, this carry is basically coming over here. So, this is but nothing but this ripple carry adder.

Now, based on this what I said, if I just want to add this particular value, suppose 1 0, so, in 1 case, if my carry input bit is 0 and if my carry input bit is 1 so, whenever this is 0, at the time what value I am getting I am getting 14 for this I am getting 15. So, that means, here from the beginning, I am calculating this 14 and 15 from the beginning.

Now, based on this carry input bit, sorry carry output bit of this particular LSB side, now I will choose what value I will select. So that means, if this bit is 1. so at that time I will select this 1 1 1, over here if this bit is 0. So at the time, I will go with 1 1 0 for this particular case.

Now, as this is also not producing any carry over here. So again, these value again it will have two of this sum bits. So, 0 means I will choose this particular case I will not choose this case. Again for this is also, there are having four of these sum bits and it is carry output bits; so, based on this carry output over here. So, it will select the corresponding things.

So; that means, what is the; that means, main; that means, fundamentals here? The idea behind this is that whenever this is over; that means, now I can I am doing what, I am parallelly basically computing all the values of this, sum bit of all this and computing parallelly ok. Considering because carry in that should be 0 or 1, so, both the case I am computing at once; that means, all the input bits are available to me, right.

So, in ripple carry adder, what is the problem? Unless and until, I am getting this carry input over here, I cannot start this operation. Again unless and until I am getting this

carry over here, I cannot start say its operation, but here what we are doing from the beginning or by putting two as we know that carry input if it as it is a binary in that we signal. So, it will have either 0 or 1. So, considering that fact from the beginning parallelly, all these blocks are running; that means, for 0, computation it is doing; for 1, considering 1, carry input bit to 1 also it is doing.

Now, the this the thing is that, whenever I will get this carry input bit there is a mux; based on the mux, so, I will get two of these sum. If I just go back to this, so that means, I will get for this A 7 B 7, I will get two sets of this particular sum over here. So, the sum, this is basically the sum for this considering carry equals to 0 and this is the sum for considering the 1. And again for this, this is for 0, this is for 1.

So, now if this bit is this carry out put bit if it is 1, then it will choose all the sum bits of all the some bits from 1 only. And again what is by is the carry output from this particular block, this is for 1, this is for 0. So, for this carry output, again it will come over here for the next 4 bit group.

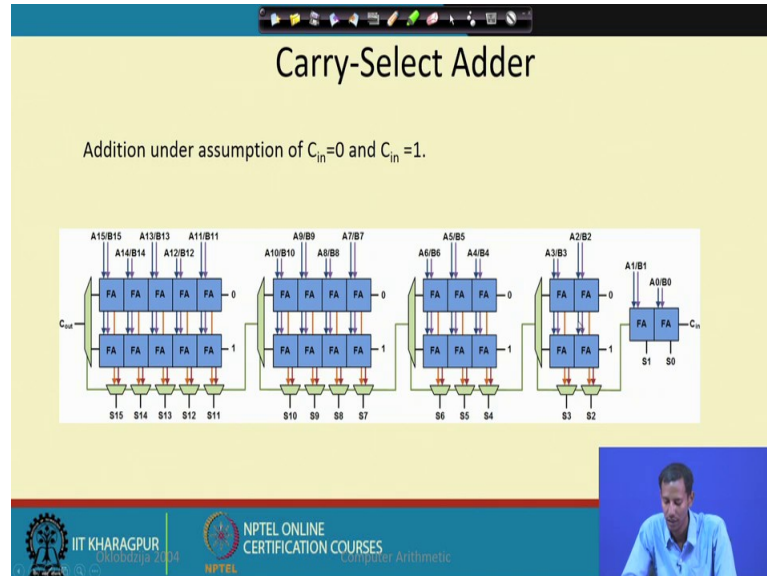
So, again this will choose based on the same logic. So that means, here then what is the problem here; the problem is that we are exploiting; that means, we are increasing the or; that means, we have increasing the speed or we are basically decreasing the critical path in corresponds to a carry look sorry this ripple carry adder by exploiting more number of hardware or more number of adder. That means, here what I have done? I have just replicated the 4 bit full adder twice.

So; that means, initially for ripple carry adder, I was needing only one set of this, but here I am needing two and then again I need at the output, I need some extra multiplexer. But here, the delay if I roughly estimate, I can reduce thus, so that means, initially what we are taking not actually this is roughly estimation. I am not telling this is the final one because as there is the mux. So, again it will take some of it is time.

So, I can reduce the corresponding delay. So, the maximum delay will be what? It will comes from this particular case to. So, the carry is basically now traversing to these particular paths, right. So, this is that means, corresponding delay for this particular path. So, here you see this is not that and this is connected to only; that means, one of this particular multiplexer; that means, as this is running in parallel. So, this bit is basically

connected. So, here I will get this 4 full adder delay and here, 1, then 2, then 3, we are 4, then 5, then 6, then 6 plus, 6 multiplexer delay so that I will get for this particular circuit.

(Refer Slide Time: 12:16)



So, if I just go to the next slide, so the same thing here you see in the; that means, in the up this in the previous slide, what you see that, we are considering 4 of this. Again, what we can do? We can just do it in a different way. That means, we can select in case of carry skip adder what we have seen we have seen that in the middle portion, if we chose the size of; that means, the more the size we can we get or we can reduce the delay critical path in a better way than if use fix number of; that means, groups for each of this; that means, fixed number of bits for each of this groups.

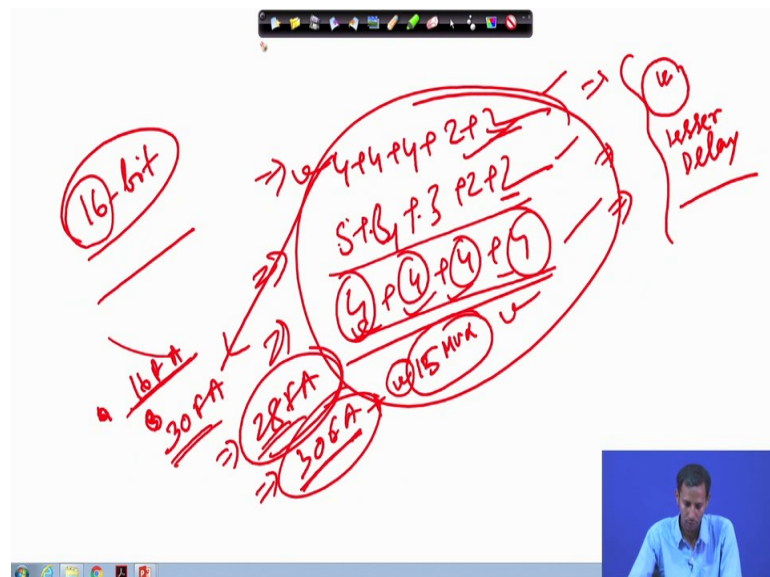
So, in this, here also we are doing this grouping right. In this particular case, can I do something do that kind of things to get the reduction in critical path. So, here also we can do. So, here would what we are doing? Actually, in the last if I just go back to the previous, so what I said that for this, I need 4 full adder delays and then I need this total 6 multiplexers delay.

Then if I just reduce this only 2 full adder cell; that means, the last level, then that 4 full adder cell delay, that will be reduced by 2. So, then that 2 is now passes through then 2, then 2, then again that is 3, then there is 4, then there is 5. So that means, the now the group is like something like this 2, 2, 3, 4, 5. That means, now the delay will be

something like this, what here 2 full adder delay then 1, 2, 3, 4, 5, 6, 7, 8, so 8 number of multiplexer delay we need to use.

So, now another thing also we can do, what we can do. So, here we are using this; that means, the last 4 bit that we have just initially, we have just distributed it in two way that was 2, 2. Then this is another 4 group that we have developed divide it something like this 5, 4, 3. So, 12 bit was there that has been divided in 5, 4, 3 or else you can try with that 4, 4, 4; here 2, 2 that also possible. That means, now it depends; that means it is basically.

(Refer Slide Time: 15:29)



How this 16 bit operation, how we can distribute this? You can distribute via like 4 plus 4 plus 4 plus 2 plus 2 or you can distribute 5 plus 4 plus 3 plus 2 plus 2 or you can 4 plus 4 plus 4.

So, that means, this 16 bit that you can group in combination of different groups of different that means, each group of different number of bits, that you can consider. So, whenever you will consider this, so considering this means, you will get for different-different of this that means, topology you will get different-different delay. And among them, which one will be the; that means, we from for what combination you will get lesser delay, lesser delay that will be your ultimate combination for implementing the carry select adder.

But here, what I said that I am increasing initially if I used this 16 full adder cell, here I need to use, here I need to use how many; that means, let considered for this particular case, if I just exclude 2 then for 28. So, 30 full adder cell for this particular combination.

So, for 4, 4, 4, so how much, so in it that means, 12, 28 number of full adder cell; so if I consider this, so at that time 5, 4, 3, 2, 2. So, then that means, 2, 3 5, 10. So, here for here also that is 30 full adder cell, ok. So that means, now I have increased this full adder cell to 30 or 28 or something along with this, I need more of the multiplexer over here.




So, if I am considering 2; that means 12 plus 1, 2, 3; so total 15 number of multiplexer again I need. So that means, I am increasing or I am as I said that whenever I, I need to gain something at that time, I have to lose somewhere. That means, exploiting extra hardware to the circuit I can reduce the delay in a very much in a certain way or that means, the delay reduction here that is in a good number.

So, based on or this following this particular that means, this logic or this idea, again the people have developed another architecture which is the that means, the fastest adder and possibly the fastest adder which the conditional sum adder. So, that we will see that architecture, we will see now.

(Refer Slide Time: 19:03)

Carry Select Adder:
combining two 32-b VBAs in select mode

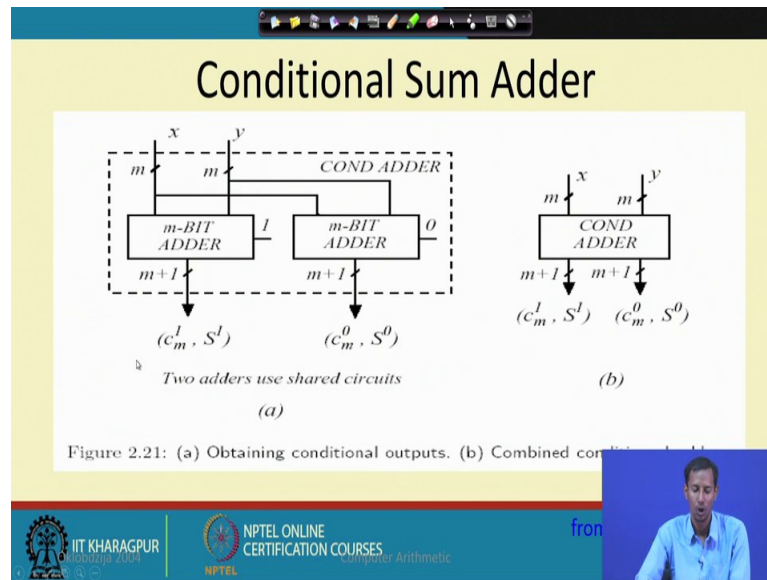
$Delay = \Delta_{VBA32} + \Delta_{MUX}$

So, here you see carry select adder that combination of 2 bit two 32 bit variable, this adder in select mode. That also if we can do, but, so what I said that based on this carry

select; that means, adder logic operation, we develop this conditional sum adder and this is the possibly the fastest adder. And if you just want to that means, know more on this condition some adder. You just follow this particular that means, papers which is in published in iee transaction on electronic computers in 1960.

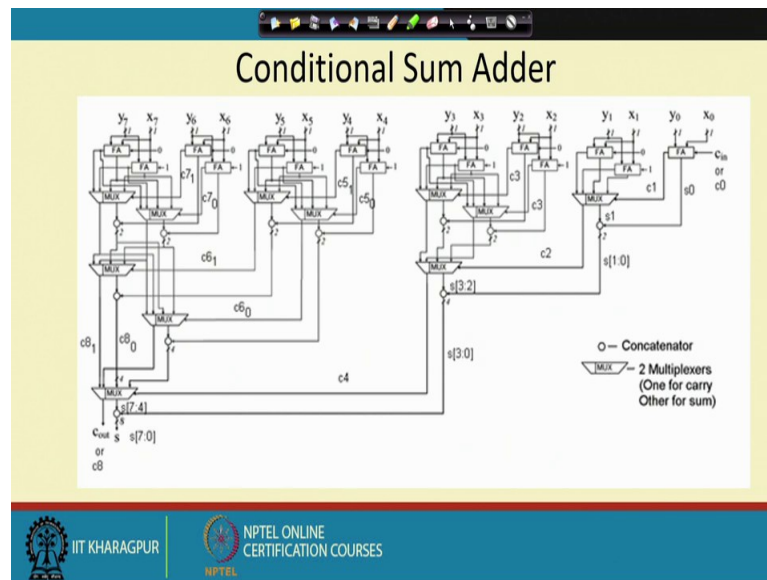
(Refer Slide Time: 19:50)



So, here what we actually we see that, for this input of x and y, considering this carry input of 0 and carry input of 1, I can get two different set of carry; that means, bit and sum bit; for this is for carry input 0 and this is for carry input 1. So that means, for this I am getting this c_m sorry, this c_m^0 , c_m^1 , s^1 and s^0 . That means, where is the difference here? So, in the in these particular case, I am not considering bitwise. That means, here I am considering 2 bit at a group; that means, this is a full adder 2 bit full adder, that is 2 bit ripple carry adder, this is a 3 bit ripple carry adder, this is 4 bit ripple carry adder, this is 5 bit ripple carry adder. But I have not considered individually to this particular full adder cell.

So, in conditional sum, so for each of these bits I can get to; that means, different set of sum and carry value considering the carry input to 1 or 0. So, that is why this is a conditional adder considering this x and y are the 2 inputs and I can get for this is for 1, sum this carry 1, and sum 1, this is for considering, this carry output is 0, there is sum c^0 and s^0 .

(Refer Slide Time: 21:42)



Now, if I just want to, that means, design one 8 bit conditional some adder then how can I draw that or how can I design that 8 bit conditional sum adder? So, this is the final architecture of 8 bit conditional sum adder. So, if I just explain or if I just want to. That means, how this is basically works, what is the working principle of this particular 8 bit conditional sum adder that you see, at the very of this at the very beginning of this, this is x_0 and y_0 and from here, considering the c_{in} , I can get the sum 0 right.

Now, this will produce the corresponding c_1 over there right. Now, in the next, what is? What happens I am having this 2 number of full adder, this conditional adder considering 1 for 0, another for 1. Why I have not consider for this particular LSB position because this c_{in} is the external signal which is directly coming to this particular point.

So, that is why I do not need this particular multiplexer operation over here. But I do not know here, what we are basically following, we are initially guessing that mind or we are predicting the sum and carry value considering the carry input bit to 0 1's and 1 1's. So, we are just getting the or obtaining the results of carry and sum in this particular from this 2 particular full adder cell.

So, whenever these full adders that means, the carry out from this full adder cell is 1, this is 1 means, it will choose the full adder output the carry and sum output from this particular full adder cell ok. That means, whenever we are computing the corresponding sum and carry, at the time I am also computing the sum and carry for this y_1 and x_1 too.

Only generation of this is now taking the decision which signal whether I need to pass the full adder cell from this particular; that means, path or I need to pass the signal from this particular full adder cell, whether that is for 1 or for 0, we are taking the decision through this multiplexers.

So, from here, now I am getting the s_1 . So, in the next what from this again I will get the sorry. So, in the next, I will get this sum that means the carry from these particular blocks too. So, whenever we will consider the second case, in the second case, I am I have to consider this I this for y_2 and x_2 ; that means, here we have considered x_0, y_0 , then we have considered y_1, x_1 . Then here, for y_2 and x_2 again, we are considering this 0 for this 0 and 1; for y_3, x_3 also 0 and 1, we are considering right ok. We will actually it needs more time to complete this particular that means, circuit for explanation.

So for today, we will just stopping and the lecture at this point. So, in the next class, we will start from this particular point on this conditional sum adder. We will see initially, we will see this how this architecture basically works. And then we will take one example and how the data is basically. Finally, compute the final result that also we will see in the next class.

Thank you.