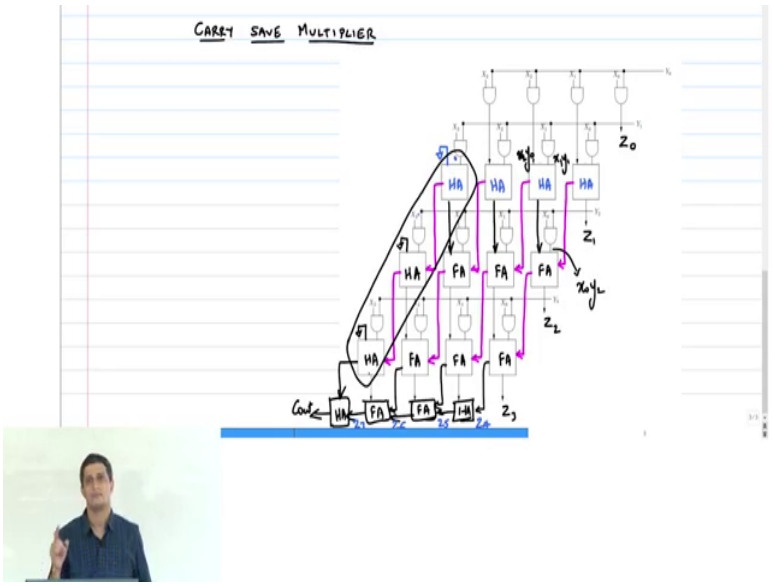
Digital IC Design
Prof. Janakiraman Viraraghavan
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture - 64
Carry Save Multiplier

(Refer Slide Time: 00:15)



So, the key point I am trying to make is instead of propagating the carry this way, I am going to now propagate the carry this way. I do not have to add the carry immediately to the left and that is what you are saying you can add the carry later, though it is not a functionality problem, it turns out it is only delay optimization that we get, ok. This is what I am going to do, right.

So, now, let us start filling out what should be my, you know half adder, full adder? First row all are only half adder, because it is only the partial products the 2 and I am going to add that

is all right therefore this is a half adder, half adder, half adder what about the last one? Just one and actually I can even bypass that go and add it to the next bit, you know I can bring this guy out of here, I will not do it for now ok, I will keep this as a half adder for now you will see the reason why.

So, I am going to call it a half adder and ground the other input ok, what about the next stage. Of course, now where should the sum come, the sum propagation has to be again like this, correct. One way to check what you are doing is correct or not remember I said that when you are trying to find z k, it has to be all the I y js such that I plus j is k. So, if you are for example, you get z naught z 1 z 2. In z 2 you should add only what naught Y 2, 1 y 1 and 2 y naught because the sum should always be. So, that is an easy check for you, make sure that that is being propagated correctly.

So, you have 1 y 1 coming here this is x 1 y 1 x 2 y naught and this 1 is what it is naught y 2. So, it makes sense these are simple ways to check that your answers you are doing the right thing. So, now second row, what are they, all full adders, third row? Yeah, actually you can make it a half adder if you are taking this definitely you can make it a half adder, right.

Now, clearly I have not taken into account those carry outs of my last row the sum has gone fast, right. I cannot say this is z 3 for example, because now the carry has got propagated downward, right. So, is this z 3, z 3 is fine you are right. So, the other guys is this z 4 yeah why because the previous carry I have not added now, right.

So, the same carry propagation is going to happen like this. And therefore, I need to have add more full adder blocks here yeah hold on, it is not that easy because this guy will now give me a carry. The last stage now I am down to the last stage I cannot do anything I have to propagate the carry side wards I cannot delay it anymore right.

So, these guys will give me a carry, this will give me a carry, ok. So, therefore, yes maybe this can be like what a half adder, this has to be a full adder, full adder, right. What about the carry

outs on this stage, I have not yet accounted for that, right. So, therefore, I need to put one more what half adder here and I will get a carry out here.

It turns out that is this last stage will give me all the sum bits will give me z 4, z 5, z 6, z 7, c out can be discarded, because we have now done extra adders we have put in right, that is c out is basically just you can discard it which is unsigned multiplication, any questions here?

How will c outcome? No, in the previous implementation the array multiplier z 7 was the c out of the last adder. Here the z 7 is the sum width of the last adder correct; unfortunately I have drawn it badly; so, s sorry, z 4, z 5, z 6, z 7, right. Any other any questions, oh what will happen to that carry out, why we will carry out always the 0 correct, yeah correct, you are absolutely, right. This is what I said you can take this product and simply plug it here right know. Yeah and therefore, I get rid of that block and actually instead of propagating to the carry, I am going to propagate that directly that this bit is 0, right.

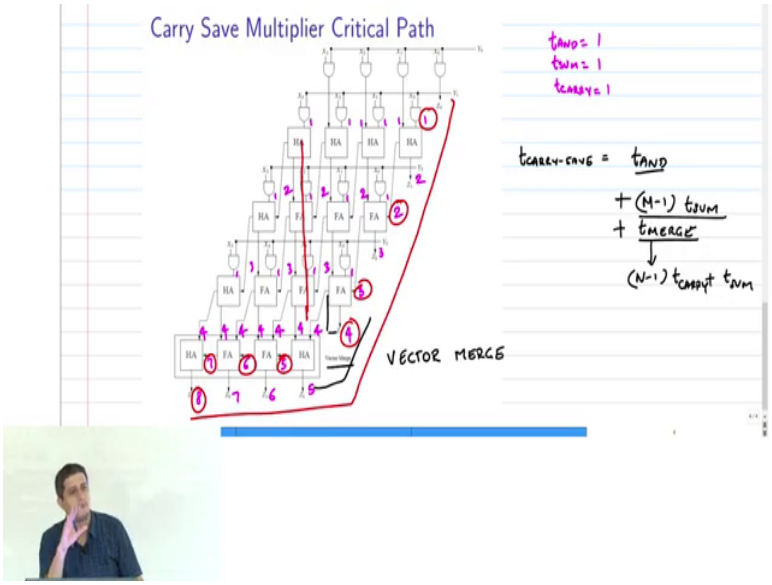
So, the output of this block will be what oh no sorry wrong, I am wrong this guy it is elf would have been x 3, y 1. Similarly this x 3, y 2 would have come here correct and thereby we would have avoided few more adder you are right, but I will show you why I am not doing that. Ok, I am going to retain this as it is ok, yeah.

How am I sure that c out will not matter? No, you are right I know. So, yeah what is saying is effectively yeah, so I think what they are saying is right, it is know to always be 0, because if you propagate this x 3, y 1 down and remove that half adder you can basically remove this set of half adders line. That is the idea that is what they are saying because the other input to those half adders were 0, right. See, this input is 0, this input is 0 and so is this input which means the sum is just one of the inputs no, wait, one of the bits is 0 right x 3, y naught there is no carry in here, correct.

I am removing all this, so that means, that half adder is just going to take the partial product and pass it through; do you agree with me, it will just go through. So, this x 3, y naught will appear here, this guy will appear here, this guy will appear here and this guy will appear here, that is all. So, if you propagate it like that you will find that final c out is very redundant ok,

but the reason I am retaining this is you will see in the next 10 minutes or so, ok. Any other questions on the carry same multiplier just the construction, right.

(Refer Slide Time: 11:01)



Now, we will go ahead and do this critical path analysis, ok. So, now, let us do the same exercise t AND equals 1 t SUM equals 1 t CARRY equals 1 just for an example here. So, let us now put the arrival times output of all AND gates is 1 arrival time. So, this will be 1, 1, 1, 1, 1, 1 right; this will also be 1, ok. Now, what about the first row when will the sum be ready z 1, 2 when will the carry be ready, 2 right. This guy what is the arrival time, 2; this is also 2, this also is 2.

Now next stage this z 2 will arrive at 3, this is 3; what about this guy, you are going to select between max arrival time which is 2, 1, 2. So, the max arrival time is 2 plus another unit time

you get 3 and then z 3 will be ready at 4, right. Now, what about the output of this guy here 4, previous guy I mean the next one 4, 4, 4 correct. What about the sum this guy 4, 4, 4.

So, now, we are going to propagate it to that final adder which is basically a ripple adder, right. So, when will z 4 be ready 5, z 5, 6, 7, 8, right. Now, I will ask you what is the critical path for this, simply take the highest guy then track the next sorry not this guy, you come here oh, we have to write that arrival time also. If 5, 5, 6, 7 then simply track this sorry, this will be my critical path, right. You go up it cannot be this right 3 is here and then you basically pick this guy and then come here, right.

So, the critical path now gets very well defined like this, one path is clearly longer than the other paths; which path yeah, from here you can even go here, you mean this way right, yeah. So, it turns out these two paths are the same the delay will be the same, effectively if you want to write the delay of this path CARRY save. It is going to be t AND plus how many things to propagate down? Why empty carry, I am asking only till here, till the input to the ripple adder finally, M minus 1 t SUM plus the final stage is called a vector merge stage.

This guy is vector merge, now observe what happens there, the inputs to this final adder stage all of them arrive at the same time, you see all the arrival times are 4, 4, 4, 4, 4. So, it is like basically a ripple adder where all the inputs are arriving at the same time. So, therefore, the final stage you can speed up by making it some other adder CARRY save, I mean a CARRY select or the CARRY you know skip or even a carry you carry adder right. So, we will simply call this as the merge delay.

Now, this depending on what architecture of adder you choose it will be O of N or O of N minus 1 if it is a ripple adder then this will simply be N minus 1 t SUM plus sorry, t CARRY, right. So, you will find in your project when you implement this and implement a ripple adder you will get some delay, if you replace this with a carry look ahead adder you will gain a lot of time because this t MERGE delay is as much as this delay. You look at the numbers N minus 1 t CARRY plus t SUM t AND plus M minus 1 t SUM they are very very similar numbers.

So, therefore, that final addition is on the critical path. So, that is why it does not even matter whether I came along this path or this path, because ultimately from there you are going to the t MERGE vector merge array anyway right. That was your question right why you I chose this path or not I think it does not matter, because I am now going to t merge after that, ok. So, with this we have you will find that you are able to optimize the multiplier very well.