

Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar
Lecture 35
Fast Adder: Carry Select Adder

(Refer Slide Time: 0:25)





Adders – previous lecture

- We learnt about
 - Ripple carry adder, and a faster design of carry look ahead adder
- We observed
 - How Boolean equations are formed
 - To generate C4 Boolean equations had 9 variables
 - No minimization was performed
 - Delay and area trade-off

Digital Logic Design/Combinational Circuits 17



Hello everybody, today we are going to continue with our discussion on design of adders. So, let us first summarize what we have done in previous lecture. In previous lecture (0:30) of it we have studied, we have seen how to design an and bit adder using ripple carry technique as well as faster design using carry look ahead adder.

So this as done at top level but on a side perspective we have also observed during the previous lecture that how Boolean equations were formed. We have seen to create Boolean equations we have created truth table and secondly we have analyzed the circuit and we have created Boolean expression based on the functionality whatever we have observed.

We have also created Boolean expression using the recursive technique where we were replacing one particular variable with the other variable. So these are the various methods that we would be generating Boolean expression so that we can design a Boolean logic or a Boolean system. So this one question which have been carrying for about from last many lectures that how do we create input for a Boolean system, for a digital logic.

So these Boolean equations essentially came from the functionality or the behavior. So, in previous lecture we have also seen that there could be many variables in one particular expression. So, for example in 4 bit CLA we have seen that to generate Cout of 4 bit CLA this carry look ahead expression contains 9 variables $P_0 P_1 P_2 P_3, G_0 G_1 G_2 G_3$ and input carry, so 9 variables if it was a 4 bit CLA.

If the number of if instead of a 4 bit CLA it would have been 16 bit CLA the number of variables in this expression would have been 33. So, yes so this point also signifies that, yes number of variables in a particular expression could be very very large and this also reinforce one point that, if number of variables are very-very large this our method which we have studied previously like Quine McCluskey or Kmap will not be very very sufficient so that is why when we studied these particular expressions, when we did these expressions we put no effort in minimizing.

Although they were not minimize able also that is a side effect, but side point but yes from our side also we have not put any effort to minimize, to use K maps or Quine McCluskey method because if we had use then it would had taken exponential amount of time. The other thing which we have also learnt from the previous lecture is yes there is a trade of between area and delay.

So we have seen that if ripple carry adder is very small in area, but if we keep on adding more area so for example if we use carry look ahead adder the delay would minimize and to further minimize a delay we can use a 2 level CLA or multiple level CLA. So with multiple level of carry look ahead carry generation unit area would keep on increasing but our time will also reduce to certain extent.

So this trade-off also tells us that yes if we are willing to put in more area there would be some chances that delay would be reduced. We cannot infinitely reduce the delay but yes to certain extent delay can be reduced if we keep on adding area or some technique which is little more complex but they can help us in reducing area.

(Refer Slide Time: 4:32)

The slide is titled "Fast adders" and features the NPTEL logo in the top right corner. It contains a bulleted list of adder types:

- Parallel Prefix adder
 - The Brent Kung adder
 - The Ladner-Fischer adder
 - The Kogge-Stone adder
- Ling adders
- Carry select adders ←
- Carry skip adders
- Carry save adders ←

At the bottom of the slide, there is a small video inset showing a man speaking, and the text "Digital Logic Design Continued Lecture 10" and the number "10" are visible.

So, because this addition is very important topic and it is a very fundamental thing for any digital system for any kind of a processing system. So that is why in last 50 - 60 years there has been a huge amount of research in creating very like fast addition circuits. So other than this carry look ahead adders or ripple carry they are the basic fundamental techniques, but using those techniques so in CLA in carry look ahead what we have seen that carry can be generated parallelly by looking ahead.

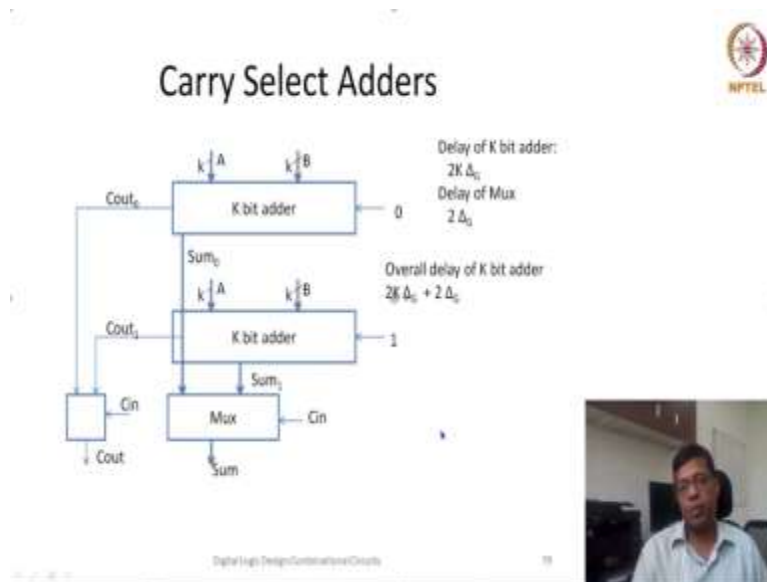
So, basically we can have a carry for some individual blocks and then we can use the other blocks carry to finally generate the final carry. So all these continuations are sudden kind of a we call it parallel prefix adder. So why parallel, because now carry is being generated parallelly so it is called parallel prefix adders. So their various combinations people have thought of and people have created so some of the popular ones are Brent Kung, Ladner Fischer or Kogge Stone. Ling adders are also sort of parallel prefix adders.

So, these parallel prefix adders are some variation of carry look ahead but in a more general sense, they are the combination how this carry would be generated in a look ahead manner would be slightly different in each of them and they would have one advantage of the other advantage over the others. So, other than that there also many other type of fast adders which have been which have evolved during last half a century - carry select adders, carry skip adders, carry save adders and we can also create hybrid of all of them.

So, let us say we can use for 4 bit we CLA and for rest of them we use carry select adder or carry skip adder, so all those combinations are also there. So in this lecture we are not going to talk about all different all these options, but we will focus on these 2 special ones, carry select adders and carry save adders. So why we are selecting these 2 this carry select adder is will be using some different functionality and that functionality that type of optimization we can think of applying somewhere else also, if it is required.

So the idea is to know what kind of digital techniques or digital optimizations we can apply given certain circumstances and carry save adder is a fast adder not for 2 operands but for a very special case, but it is important to know that particular scenario. So let us first understand what carry select adder is?

(Refer Slide Time: 7:28)



This carry select adder, the idea is that the problem there is that we did not know in any particular adder we did not know what would be the carry in and because we did not know carry in that is the major problem and because of that we cannot compute what would be the carry out. So what carry select adder do that instead of knowing what would be the carry value it assumes, it computes the sum so here it computes the sum assuming that carry is 0.

So this is a k bit adder where A is also of k size, B is also of k size and sum_0 is produced based assuming that input carry is 0 and similarly $Cout_0$ is produced and it also computes the area sorry the hardware is duplicated so it will produce 2 different instance of k bit adder.

1 which will take 0 as a carry in another which will take 1 as a carry in and so it will also produce that is why sum_0 as well as sum_1 , sum_0 means when carry was 0, $Cout_0$ means when carry was 0. Similarly sum_1 when carry was 1 and $Cout_1$ when carry was 1. So we have to note that this is completely replicated circuit, so area is simply doubled. But not the delay, delay of both of them would be same.

Now why do we do computation, this duplicate computation? Because at the end what we do is as soon as we know when this C_{in} is there, then we can see that okay based on C_{in} , if C_{in} is 0 then it will select some 0, if C_{in} is 1 then it will select 1. Similarly, it will also do the similarly mux thing here if C_{in} is 0 then it will take $Cout_0$ as the $Cout$ and if C_{in} is 1 then it will take $Cout_1$ as the output.

So the idea is that by the time this computation is done C_{in} would be available. So this way this multiplexing can help us, so we can go ahead in computing the addition and in parallel there could be other adder which could be doing its work so that it can generate C_{in} and when C_{in} is generated then it will we can mux, we can create the output and see that which one to be selected.

So when this C_{in} to be generated, when the C_{in} to be given for that let us do a quick delay analysis. So delay of let us say which k bit adder let us say we take the simplest one ripple carry adder. So, if we take ripple carry adder as a input, so as the implementation of this k bit adder that delay of k bit adder we understand in a ripple carry is $2k \Delta g$. Δg is the gate delay, so gate delay into 2 into k that would be the delay of k bit adder.

Now, to get the output there would be a delay of mux also. So this mux will consume $2 \Delta g$ delay, so the final delay is going to be for K bit adder it is going to be $2k \Delta g$ plus $2 \Delta g$. Now, we have to ensure that this C_{in} is available, this C_{in} is available at least at this this $2k \Delta g$ time. So, if C_{in} is to be available at $2k \Delta g$ time, so that means that the adder which is preceding this adder so basically this computing some k bits.

The adder which is doing from where this C in is being computed that should generate this C in or C in as C in by that time this particular sum is available. So that should also take $2k$ plus $2k$ delta g time and from this equation time it seems that let us say that particular adder is also using this carry select adder combination. So that means that should be k minus 1 bit adder.

(Refer Slide Time: 12:05)

Carry Select Adders

Delay of K bit adder: $2k \Delta_{g1}$
 Delay of Mux: $2 \Delta_{g2}$
 Overall delay of K bit adder: $2k \Delta_{g1} + 2 \Delta_{g2}$

Digital Logic Design/Combinational Circuits 79

Carry select adders – Unequal groups

K Bit carry select adder

K-1 Bit carry select adder

K-2 Bit carry select adder

Forms an arithmetic progression

Largest size adder \sqrt{N}
 Delay of adder : $2 \sqrt{N} + 2$

For $N = 32$, 9 groups would be required
 1, 1, 2, 3, 4, 5, 6, 7, and 3

Digital Logic Design/Combinational Circuits 80

So, that means we are creating sort of groups, so 1 group which is using k bit adder the another group which is using k minus 1 bit adder so on so forth. So this delay is selected in such a way

that the delay when sum is there is equal for all the groups. So, if I have an n bit number so first highest order k bits I will use for this carry select, k bit carry select and then the lower significant bits I can have $k - 1$ bit carry select adder and then $k - 2$ bit carry select adder.

So this looks like arithmetic progression. So that means I can create an arithmetic progression and I create groups in such a way and the other thing what I am ensuring that at a particular group, so for example at this the sum as well as the C in from the previous group is computed at the same time.

Similarly, sum at this particular group as well as carry in from the previous group are computed at the same time so that is why it formed sort of an arithmetic progression. So, if it is arithmetic progression it also means that, let us say if the total addition is for N bit the largest k value would be equal to or the largest value would be equal to ceiling of under root N .

So, let us say if N is 32, so that means the value the maximum value of k is going to be 6. So the total delay the maximum total delay will also depend on the largest value of k that means 2 under root square N plus 2 would be the delay of this carry select adder. Now, if N equal to 32 these are going to be the groups, so we have to start with 1, 1 then 2, then 3, then 4, then 5, then 6, then 7 and whatever remaining would be given to 3, would be given and would be the last row.

So, this way this particular type of adders although it looks little orthodox that we are doing computation twice, but it seems to be very effective because by the time we are doing computation some other guy is calculating and whatever was our depending thing so we were here dependent on C in, so this C in can be computed in parallel and we are duplicating the hardware and doing it twice.

So this is very general optimization technique which can be used in designs, so basically in designs as well as in coding algorithms or wherever. So the idea is that if we are able to do 2 copies like and then finally choose at which one should we select. It can reduce the delay but it is increasing our effort of design, it is also increasing the amount of area or cost also.


But delay can be minimized, so let us say very example from our assignments so we did not know what kind of assignment professor is going to give but we know either he will give this assignment or that assignment. So we know the 2 options which he will give. So what we can do

is we can parallelly work on both the options as soon as professor will give the assignment we can give him the answer.

So this and similarly in other many scenarios wherever we did not know what our dependency is going to be, we can be prepared but this preparation can help only if the number of options are finite. So here the number of options were only 2. So either carry in could be 0, carry in could be 1, so if it is 2 or finite number of options then we can still think of doing it in parallel but if the, it is not so then we did not know.

So this is also called speculation in some cases, speculation means that we speculatively assume that if the output is going to be 0 then we can do this and if output is going to 1 then we can do this particular processing and then when we came to what is the output based on that we can select our final output.

(Refer Slide Time: 17:08)




Carry select adders – Unequal groups

K Bit carry select adder	K-1 Bit carry select adder	K-2 Bit carry select adder
--------------------------	----------------------------	----------------------------

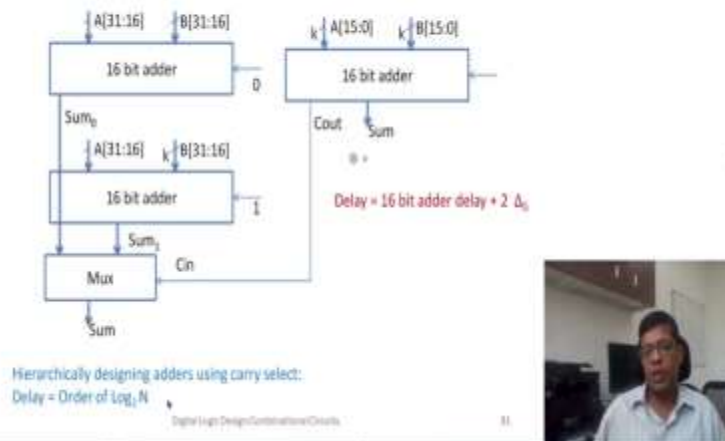
Forms an arithmetic progression
Largest size adder VN
Delay of adder : $2VN + 2$

For $N = 32$, 9 groups would be required
1, 1, 2, 3, 4, 5, 6, 7, and 3

Digital Logic Design: Combinational Circuits 88



Carry Select Adder equal groups



So this was using unequal groups but with equal groups also this particular algorithm is equally effective. Let us quickly see how equal groups carry select adder can be created. So let us say we created 2 adders of like we divide the, let us say we are designing a 32 bit adder we divide it into 2 parts 16 bit adder and 16 bit adder. So first 16 bit adder is a standard one and we did not know how it is inside implement whether it is CLA, whether it is carry select adder or carry skip order or parallel prefix adder we did not know.

But let us assume that it is a 16 bit adder, the other one we are implementing we are designing and implementing it using carry select adder. So that means there are 2 combination of this 16 bit adder, 1 which is taking 0 as carry in, the other which is taking 1 as a carry in. The upper significant bits, upper 16 significant bits of the B is part of input here.

Similarly, upper 16 bits of A are part of this input here and we are adding both of them. We are getting sum 0, we are also getting sum. So when this C in would be available? The C in would be available only when this particular addition would finish. So this addition is the lower 16 bits of A and lower 16 bits of B. So, as the Cout would be available that would be given as a C in to this particular adder and then we can decide what would be the sum, whether it is sum 0 or whether it is sum 1.

Now, if we see that whether this would be advantageous or not, it is certainly advantageous because the delay of this particular 32 bit adder, what is the delay of this 32 bit adder? Whatever

is the delay of 16 bit adder plus delay of this mux. So the addition, this 16 bit addition and this 16 bit addition can be done in same amount of time. So this sum and this sum would be produced in same time, but here an additional mux delay would be required so the total time would be delay of 16 bit adder plus $2 \Delta g$.

So using this carry select adder we are able to reduce the delay to half, approximately half. Now, can we do it better? Yes, so this inside implementation of this 16 bit adder is not defined here so we can do the way we want. So in previous combination when we were having unequal groups, we thought of using only ripple carry adder because the size of each group was non uniform.

So one of the group was 7, then 6, then 5, then 4, then 3 so if the size of group is variable then using some other effective adder like carry look ahead adder or carry parallel prefix adder will create some sort of a confusion. We cannot sure when the would be what delay would actually match the 2 parts. But here if we are using carry select adders with equal size of groups, groups of equal sizes then lot of optimizations are possible. So fun optimization is certainly this that we can make sure that this 16 bit adder is a fast adder.

The other possibility is that this particular technique of carry select adder we can hierarchically apply. So, we can use, we can design this 16 bit adder again using carry select adder of 8 bit and 8 bit and then again 8 bit carry select adder we can design using 4 bit carry select and 4 bit carry select and the leaf level 4 bit carry select 4 bit adder we can implement using carry look ahead adder.

So the only thing we have to ensure here that the 2 parts which we are implementing should have same delays to have the optimized results. So this way this carry select adder can also lead to a overall delay of order of $\log n$. Why order of $\log n$? Because we can break them into 2 and then break them into 2 and then break them into 2. It is actually order of $\log n$, it is not equal to $\log n$ because this $2 \Delta g$ delay would be there in every iteration.

So, for example we are creating this 16 bit adder using a carry select adder then it would be a delay of 8 bit adder plus again $2 \Delta g$. Still the order would be equal to order of $\log n$. It is a very effective technique and it is a very generic thing also which can be applied to different kind of circuits.