

Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar
Carry Look Ahead Adder

(Refer Slide Time: 0:18)

Can we reduce overall delay?



- Source of delay in adder
 - Sequential generation and consumption of carry
- Can carry be generated in parallel?



So, now let us ask one question that what is the typical size of adders we would require in regular computation. So, when we started doing computation when we started building computer then usually we used to work on 8 bit computers so that means usual data types used to be 8 bit ad then we started working on 16 bit then 32 bit then 6 bit. These days in our computers most of the computations are 64 bit computations.


Which essentially means that the delay of this particular adder would grow linearly as we grow the size of my addition. Which could be usually impacting things because this addition is a very fundamental unit in any digital design. You would require addition invariably in most of the circuits. So, if the delay will keep on increasing as the size will increase it is going to be a very costly operation.

So it is very very important that we reduce the delay of this operation. How can we reduce the delay of addition? To reduce the delay let us understand from where the delay is coming. So we have seen in our previous slide in previous information we see that the delay is actually because

of this carry and sequential generation of carry and sequential consumption of carry both of these things are important.


So unless my carry is generated it cannot be consumed and carry at i th stage cannot be generated unless carry of i minus 1 stage is there. So unless we break this sequentiality unless we are able to generate this carry in a faster way we cannot have a fast adder. So we have to, if we want to make, if we want to reduce the delay of a adder we have to generate this carry parallelly.

(Refer Slide Time: 2:37)



Closer look at carry generation

- Carry generation: When both A_i and B_i are '1'
- Carry propagation: When either A_i or B_i is '1'
- Carry kill: When both A_i and B_i is '0'
- $G_i = A_i B_i$
- $P_i = A_i + B_i$ Ideally $P_i = A_i \oplus B_i$
- A carry out a particular stage is '1' if either it is generated at that stage or propagated from previous stage



Digital Logic Design: Combinational Circuits. 64

How to generate this carry parallelly? So, let us look at how carry is actually generated. So in a single bit we see that carry is generated when both, we are talking about i th stage any particular stage. Let us say i stage carry would be generated when both A_i and B_i is 1, A_i and B_i are 1. Let us talk only about the primary input, so these primary inputs if both of them are 1 then carry would be definitely generated.

But some time carry need not to be generated in i th stage to be there in the i th stage, to be there in the i plus 1 stage. So sometime a carry is coming from behind or some of the stages i minus 2, i minus 3, i minus 4 some of the previous stages then also carry could be there as a output like C_i plus 1 could still be there because carry was generated somewhere in the previous stage.

So, how a carry which was generated in the previous stage will come as the output at i stage, because it would be propagated and for propagation the requirement is either A_i is 1 or B_i is 1. So, when either of them is 1 and there is a carry input from the previous stages then it would be

propagated. 1 plus 1 will become 0 so because there was a carry from the previous stages it will also be propagated.

So, and the third parameter is when this propagation would be halted we call it carry kill. So, when both A_i and B_i is 0 then whatever carry was coming from behind it will stop propagating. We can also say that we are killing that carry. The carry will not go to the next stage. So these 3 terms are very important to see how a carry is generated, how carry is propagated. So if we try to see these 3 piece of information and we try to analyze it further then we can say that yes how we would understand how we can generate this carry in a faster manner.

So, let us call this carry propagation signal, carry generation signal as G_i which is equal to A_i and B_i so when both of them are 1 then generation is 1. Propagation is equal to P_i at any particular stage, propagation is equal to A_i plus B_i , then you will ask me that we have said here that P_i is equal to propagation would be there when either of them is 1, so essentially P_i should be equal to A_i XOR B_i . So that means when either of A_i or B_i is 1 only one of them is 1 then propagation is true.

So why we are writing this? That is a valid question. Now, see this propagation has a slightly different definition. This propagation means this is also included generations so that means that even when both of them are 1 then also propagation would be there. So when this propagation either it would be generated or propagated so this P_i definition is more generic which means that carry would be propagated.

So, that essentially means if there is a carry from behind and if P_i is 1 then it would be propagated to the next stage. So, which definition of P_i should we take? So this definition of P_i is less costly in terms of delay, so here the delay of G_i is equal to 2 input AND gate and here the delay of generation of P_i is equal to 2 input OR gate, both of these delays are equal as well as small.

But the delay of generation of this XOR is little more than AND or OR, so that is why many a times we prefer this definition from the delay perspective. So, we can now say that a carry in a particular stage would be carry out in a particular stage would be 1. If it is generated in this particular stage or it has been propagated from previous stages. So, let us see if we can generalize it for 4 bit adders.

(Refer Slide Time: 7:30)



4 Bit adder

- A carry out a particular stage is '1' if either it is generated at that stage or propagated from previous stage
- $C_4 = G_3 + P_3 C_3$ P & G depends on primary input
 $C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$
- $C_3 = G_2 + P_2 C_2$
 $C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$ Carry Look Ahead Adder
- $C_2 = G_1 + P_1 C_1$
 $C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$
- $C_1 = G_0 + P_0 C_0$



So we have taken the definition here also, if carry is at a particular C out of a particular stage is 1 either it is generated at this particular stage or propagated from the previous stage. So I can write C4 is the last stage of my 4 bit adder, so C4 would be 1 if it is generated in third stage or it is propagated from third stage and there was a carry in from the third stage.

Similarly, we can also say that carry for the third stage is equal to, it is generated in the second stage or it is propagated from the second stage and there was a carry input onto the second stage. Similarly, we can also say that the carry at the second stage is equal to either it is generated at the first stage C out at the first stage would be equal to either it is generated at the first stage or it is propagated at first stage or there is a carry input to the first stage.

Now, this is actually in the terms of primary input so carry out at the 0 stage would be equal to if either it is generated at the 0 stage or it is propagated at 0 stage and there is a carry input. So this equation we can generate from the primary inputs C0 is a or primary input, P0 can also be calculated from the primary inputs a b c, a and b, similarly, generate can also be calculated from primary inputs.

Now, this C1 we can substitute here and we can calculate the value of C2 in terms of primary input. So, if I do that it become generation G1 plus P1 into G0 plus P1 into P0 into C0. Now,

whatever is the output here that we can keep we can substitute here at C2 and we can find out the value of C3.

So, C_3 is G_2 plus $P_2 G_1$ plus $P_2 P_1 G_0$ plus $P_2 P_1 P_0 C_0$. Now we can substitute the value of C_3 here and we can find out what is the value of C_4 . So C_4 is equal to G_3 plus $T_3 G_2$ plus $P_3 P_2 G_1$ plus $P_3 P_2 P_1 G_0$ plus $P_3 P_2 P_1 P_0 C_0$. So what does this equation mean? This equation essentially mean that at N stage my carry would be out if it is generated at N stage or it is generated at N th N minus 1 stage and propagated through N th stage or it is generated at N minus 2 stage and propagated through N minus 1 N th stage.

Similarly, we have to go till carry 0. So, this particular equation of C_4 at carry any particular N th stage essentially means that the carry depends on all these stages cannot ignore that. But we can compute it from the primary inputs we can calculate directly from propagation and generation which could be generated from AMP.

Now, there is another thing which we have to see that this is the equation for carry 4, so if I need to find out carry at any N stage the size of this equation will keep on growing as well as the number of input, maximum number of input to this equation will also keep on growing. So for example you see the delay or to generate C_4 would be equal to, what is the critical path for the AND gate?

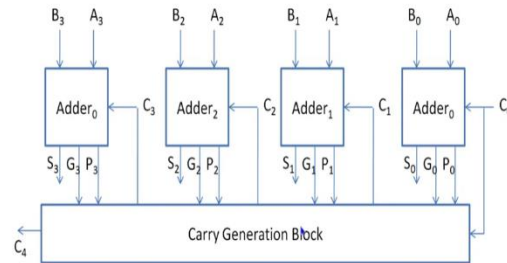
What is the maximum delay for the AND gate? Maximum delay will come because of this, so this would be 5 input AND gate and similarly there would be 1 2 3 4 5 there would be 5 input OR gate. So the total delay would be 5 input AND gate plus the delay of 5 input OR gate.

Now to make, so this particular type of adder where we are generating carry from our primary inputs P and G, P and G is calculated from primary inputs propagation and generate and then we are calculating the carry simultaneously we call it carry look ahead adder. So that means we are looking ahead for the carry before actually is carry propagated. So in short we also we are calling it CLA carry look ahead adder.

(Refer Slide Time: 12:57)



4 Bit CLA Adder



Delay analysis

- Assume simplistic model, One AND/OR gate $\rightarrow \Delta_G$
- Total delay = 1 Δ_G delay (P, G) generation + 2 Δ_G delay carry generation + 2 Δ_G delay for Sum

Digital Logic Design: Combinational Circuits.

66



4 Bit adder



- A carry out a particular stage is '1' if either it is generated at that stage or propagated from previous stage
- $C_4 = G_3 + P_3 C_3$ P & G depends on primary input
- $C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$
- $C_3 = G_2 + P_2 C_2$
- $C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
- $C_2 = G_1 + P_1 C_1$
- $C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$
- $C_1 = G_0 + P_0 C_0$

Digital Logic Design: Combinational Circuits.

65



Now, let us try to see how it would be in terms of circuit? So, now when I would like to design a CLA circuit what I would do is I will design a adder circuit in which I will modify it a little bit in the modification what I will do is I will have still 3 inputs $C_0 A_0 B_0$ or $C A B$ carry in A_i and B_i but the 3 outputs 1 is sum, another 2 outputs are generate and propagate at any i stage. So, propagate would be A_i or B_i , generate would A_i and B_i .

So, similarly for all the stages we will have generate and propagate and then there could be a carry generation block which can use all these equations to find out the carry value of C_1 , C_2 , C_3 and C_4 and all these carry values C_1 , C_2 , C_3 and C_4 , so all these values of C_1 , C_2 , C_3 , C_4 would be generated as output of this carry generation block. Now, C_1 would be given to the input of adder 1, C_2 would be given as a input to adder 2, C_3 would be given as a input to the adder 3, so this is 3 actually.

So this way we would be able to design it structurally and we can code it in modality using a Verilog also in a similar way. So let us try to analyze what would be the delay for this 4 bit CLA adder. So to simplify the delay analysis let us say that input, so the delay of AND and OR gate is going to be Δg irrespective of number of inputs. Irrespective of fan in or fan out we are assuming that there is a Δg delay for 1 gate.

So, we are talking about AND and OR here, we are not talking about XOR. 1 XOR delay could be from this definition it would be equal to $2 \Delta g$. So, the total delay would be equal to in 1 gate delay or in $1 \Delta g$ we would be computing all the propagation and generation. So all this P_i and G_i would be available after $1 \Delta g$ and after that there would be $2 \Delta g$ required, so $2 \Delta g$ would be required to generate all this carries that means C_1 , C_2 , C_3 , C_4 all of them would be generated in $2 \Delta g$.

So, once carry C_3 is generated then I would require another delay for calculation of sum. So this sum would require an XOR gate, so XOR gate we will again consider that a $2 \Delta g$ would be require. So the total delay would be $7 \Delta g$. You can also analyze that in case of a ripple carry adder the total was going to be the delay would have been $2 \Delta g$ for all of them plus 1 XOR gate delay. So, essentially there would be $4 + 8, 8 \Delta g$.

Now, you see the advantage is not that sufficient. So this way we are able to reduce the delay of a 4 bit adder. Now, at what cost we have increased the area? So this whole additional circuitry of carry generation block is the additional area consumption. The circuit which was require for all the adder 0, adder 1, adder 2, adder 3, it has not reduced, it is still there the same circuit is there. In addition, we have introduced this additional AND and OR gates to get this generate and propagate signals plus this carry generation block which is also an additional area. So, with this additional area we are able to reduce the delay somewhat.

(Refer Slide Time: 17:35)



Large CLA adders

- For example 16 bit adder
 - Carry generation will have large fan-in gates
 - To generate C16 critical path
 - 17 input AND gate + 17 input OR gate
 - Same gate delay will for large fanin is not applicable
 - Area implications
 - Not scalable
- Alternative
 - Use blocks of 4bit CLA
 - Multiple level CLA



Digital Logic Design: Combinational Circuits 67

Now, let us see, this was a 4 bit CLA adder what about if I want to design a larger adders. Let us say 16 bit adder, so ideally we should take 64 bit adder but 64 bit adder would become slightly complex for demonstration so let us take 16 bit adder we can generalize it to 64 bit adders also. So, the problem with 16 bit adder would be this carry generation would require now large fan-in.

So, for example if I want to generate C16, to get the C16 signal I would require AND gate with 17 inputs, OR gate with 17 input. The delay model which we have used in CLA, while talking about CLA we said that AND and OR gate would be same irrespective of fan-in but that will not scalable here. So, we can still say that using some transistor level techniques by properly sizing the transistors delay of 3 input AND gate and 2 input AND gate could be very similar but 17 input AND gate delay cannot be same as 2 input or 3 input AND gate.

So, that assumption of same gate delay for large fan-in will not be applicable at all. So that means this method is not scalable. Similarly, area implications is also going to be huge because to generate this 17 input AND gate or 17 input OR gate we would require multiple level logic and that multiple logic would contribute to delay. Similarly, it will also contribute to area so here area will also keep on increasing as we increase the number of stages in our look ahead.

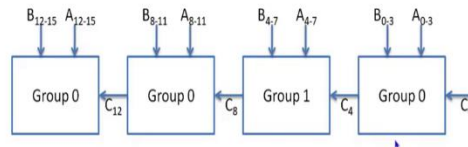
So, that essentially means that this CLA is not a very scalable approach. This gave us some idea how we can generate carry in a faster way, but if we go beyond 4 bit CLA then it may not be

scalable. We have to experiment so some may be 8 input CLA can also be okay, but if we go beyond then it is no scalable at all. What should we do? The alternative, so either we use the blocks in 4 bit CLA and then we use multiple of such blocks or we can also use multiple level CLA. So, let us see both of them one by one.

(Refer Slide Time: 20:17)



Large Adders: CLA Blocks



- Delay: $(n/2 + 3) \Delta_G$
 - Δ_G for propagate and generate values
 - $(n/4) (2 \Delta_G)$ for generating carry
 - $2 \Delta_G$ for calculating sum
- For 16 bit adder = $11 \Delta_G$



So, if we are using multiple blocks then what would happen essentially that you will have this as a group 0. In group 0 you will have A0 to 3 as a input, B0 3 as a input and similarly it would be a group of 4. So for 16 bit CLA I would require 4 such events and then the carry would be now rippling from 1 group to another group.

So, does it seem better? Is it faster than our regular adder or not? Let us quickly do this analysis. So, from a 4 bit CLA perspective the advantage was not much in case of ripple carry adder we were seeing the delay as 8 delta delays, 8 delta g delays. Here in 4 bit CLA it was 7 delta g delays but in case of 16 bit it could be substantial, because now generate and propagate of all of them could be generated at once.

So, there would be a delta g delay for generate and propagate at all the stages. While this carry propagation, carry generation here the carry generation would require 2 delta g and similarly after 2 delta g C4 would be available and another 2 delta g C8 would be available, at another 2 delta g C12 would be available. So the total number of transactions here would be like you will say N by 4 number of 2 delta g delay would be require to generate al the carries.

So, we are assuming C16 also need to be generated, so after C16 is generated. So the total amount, then afterwards the C12 is generated then you would also require sorry when C16 is generated after that you would further require 2 delta g for calculating this sum. The overall delay would become equal to N plus 2 plus 3 delta g. So, if it is a N is equal to 16, so that means it is a 16 bit adder. So a 16 bit adder would actually require 11 delta g delays.

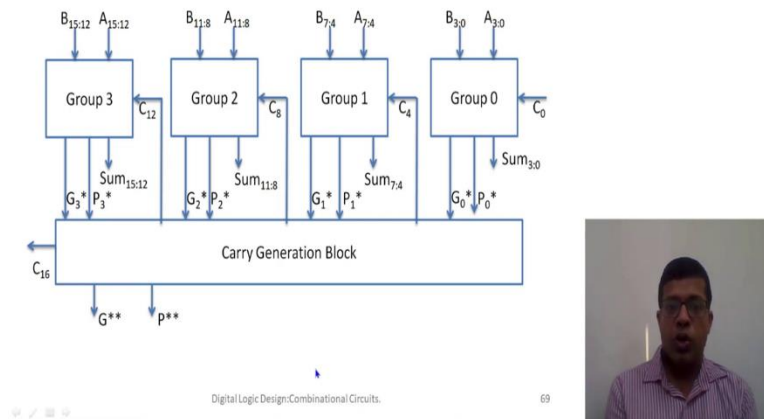
While in case of a ripple carry adder for 16 bit adder it would require 32 delta delays, delta g delays. So the advantage of CLA is clearly evident if we increase the number of bits. It would give us N by 4, so at least it would give us asymptotically we see the delay would be reduced by N by 4 or divide by 4.

So, if the delay for 64 bit adder in a regular case is x, then delay in case of CLA we are using block of 4 then it would be x by 4. So, delay is would be quite less but can we do slightly better or is there any better approach? Then we can use this carry generation and propagation in hierarchical manner.

(Refer Slide Time: 24:02)



Two Level CLA Adder



So, when we do it in a hierarchical manner it can look something like this, so we are applying it again and again. Now, this is the previous the same diagram we have drawn in a slightly different way. So there is a group 0 which is taking 4 bit input, similarly another group 1, group 2, group 3. So here we will have 4 groups, group 0, group 1, group 2, group 3 so we are talking this example for 16 bit addition.

Now, all these groups will have 4 bit addition taking place in all these groups. Now, if we apply that method again, we say that the output of this each of this group will generate sum which is 4 bit input additionally all of these groups along with doing the way what they were doing internally. So that means they had their own carry generation and propagation blocks which was based on the primary input A and B.

So based on that they also generate 2 additional signals which is called group generate and group propagate. So each group or each first level CLA is doing carry look ahead computation within itself, so when it is doing itself so that means it is generating propagate and generate signals and using those propagate and generate signals it is computing the sum.

So, along with generating its own propagation and generation signal it also generate group propagate and group generate signals. So each of this group is generating two additional signal which is group propagate and group generate. So, these group propagate and group generate signals can be given to another hierarchical block which is carry generation block, so this carry generation block the definition of this carry generation block is exactly similar to the carry generation block at the first level which is also present in all these groups.

But now this particular carry generation block is working on not only from the primary generate and propagate signal but it depends on group generate and group propagate signals. So what these group generate and group propagate signals I will tell you in the next slide. So, but for now let us try to understand that each group, so whether there is a there would be generation from this particular group or not whether there would be propagation so if there is a carry here whether it would be propagated as a C4 or not.

So that would be group propagate and similarly group generate would be whether there would be carry at this particular stage or not. So, whether there would be a carry at C4 or not. So this particular method can help us in generation of carry in a even hierarchical, even in a faster way. Now we can find the carry at C4 in a lesser time so that we will do quick analysis now.

Now, this hierarchical nature of this particular CLA where we can have a carry generation block here, which is at the second level. So there could be this group generation and group propagation can even be enhanced further. So let us say we would like to create a 64 bit adder using this 16

bit adder, so each 16 bit adder can generate another group generate and group propagate which can further be used to generate the carry for the next set of blocks.

So, if we do in this way then the total delay of any largest carry and any largest adder would not depend on linearly on size of adder but it would depend on the log of that end. So we are creating sort of a tree, so because of this tree structure now we are reducing the delay at each particular stage. So let us see what is this group generate and group propagate.

(Refer Slide Time: 28:41)

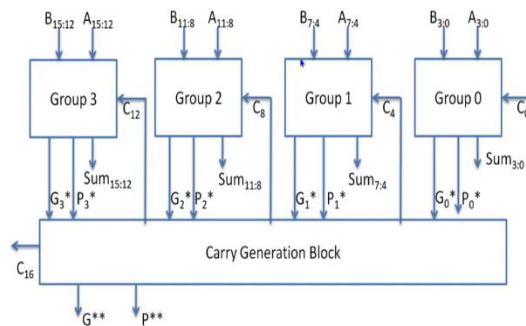


Two level CLA adders

- Use group carry and group propagate
 - $G^* = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$
 - $P^* = P_0P_1P_2P_3$
 - $C_4 = G^*_0 + C_0P^*_0$
 - $C_8 = G^*_1 + G^*_0P^*_1 + C_0P^*_0P^*_1$
 - $C_{12} = G^*_2 + G^*_1P^*_2 + G^*_0P^*_2P^*_1 + C_0P^*_0P^*_1P^*_0$
- Can be hierarchically designed
- Delay : $7 \Delta_G$



Two Level CLA Adder



So, group generate essentially means that a carry is generated in this particular group, it is not talking about. So, when we are saying group generate at this stage it does not know whether a

carry came from 0, whether carry is there from C_0 or not but within this group whether it is generated or not that is what is the important thing here.

So to know whether a carry is generated in that particular 4 bits, so we can use this particular equation which we have seen earlier. So that means carry is generate in the third bit or carry is generated in the second bit or it is propagated the third bit. Similarly, carry is generated at the first bit and propagated through second and third bit or carry is generated at the 0 bit and propagated through first, second and third bit.

So this is the group generate logic equation for 4 bit group generate. Similarly, if there is a carry from the previous stage whether it would be propagated out or not that means all of them has to propagate. So there has to be propagate in all the bits. So group propagate means that carry is propagated through $P_0 P_1 P_2 P_3$, so for all the 4 bits if carry is propagated that means for that particular group carry would be propagated from previous carry to the next carry.

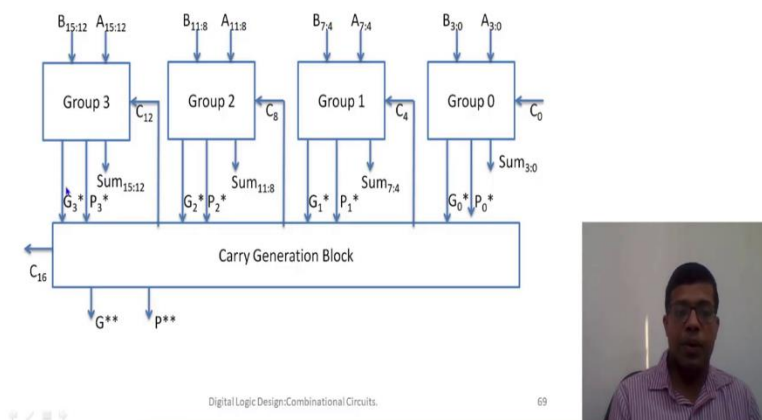
So if we have this G^* as a group generate and P^* as the group propagate then we can write the equation for our C_4 , C_8 , C_{12} and C_{16} accordingly. So, C_4 would be equal to if it is carry is group generated by group 0 and there were C_0 input and it was propagated through group 0. So, similarly for group 1 if it is generated at group 1 or it is generated in group 0 and propagated through group 1 or there was a carry input and it is propagated through group 1 and group 2.

So similarly, C_{12} is if it is generated in the 2 block or generated in the 1 block and propagated through 2 block, generated in the 0 th block and propagated through first block and second block. Or there was a carry in and it is propagated though group 0, group 1 as well as group 2.

(Refer Slide Time: 31:39)



Two Level CLA Adder



Two level CLA adders

- Use group carry and group propagate
 - $G^* = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$
 - $P^* = P_0P_1P_2P_3$
 - $C_4 = G^*_0 + C_0P^*_0$
 - $C_8 = G^*_1 + G^*_0P^*_1 + C_0P^*_0P^*_1$
 - $C_{12} = G^*_2 + G^*_1P^*_2 + G^*_0P^*_2P^*_1 + C_0P^*_2P^*_1P^*_0$
- Can be hierarchically designed
- Delay : $7 \Delta_G$



So let us look at this diagram again, so that means this C_0 would be propagated to C_{12} only if it is propagated through group 0, group 1 and group 2. Or C_{12} would be 1, if it is generated in group 2 or if it is propagated in group 2 and it is generated from group 1. So this way this method of carry generation is actually a generic method which could be applicable to any number of times we can apply it hierarchically.

Now, let us quickly see what would be the delay. So the delay of this 16 bit adder would be, now there would be 1 gate delay because of generation, there would be 2 gate delay because of

generating P0 and G0, so basically to generate group propagate signals there would be 2 gate delay and similarly to generate this C4, C8, C12 and C16 it would require another 2 gate delay.

So 2 plus 2 plus 1 and then finally there would be 2 gate delay to generate this final sum. So the total delay would be 7 gate delays which is even lesser than the previously we received 11 gate delay and further more interesting thing is that even there is a 64 bit adder. The total amount of delay in case of 64 bit adder would be 9 gate delays, which is substantially less than 128 gate delay in case of a ripple carry adder.

Or in case of a regular CLA or a block level CLA 64 bit adder will have a delay of the order of 64 gate delays. So this is how we can generate, we can see how optimally we can create adder which could be fast which could be efficient but of course there is a cost in terms of area. So, we have to pay in terms of area, we have to add more and more gates so that we can have an optimized circuit.