

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

**Lecture – 18**  
**Efficient Adder Architecture (Contd.)**

Welcome back to the course on Architectural Design of ICs. So, we are seeing the different Adder Architecture. So, we have seen a ripple carry adder, we have seen carry skip adder ok. So, after that we have seen that the carry skip adder if we just make it a group for if the number of  $n$  is large. So, at the time if we make it a group and then using that group if we just use the; that means, the carry skip adder for each of this group then we can get the benefit or that delay has been reduced by the factor of  $k$ .

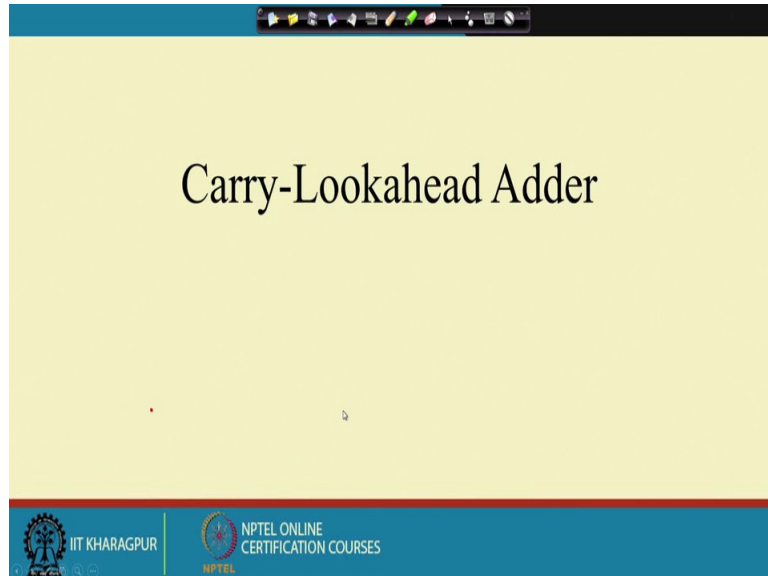
So; that means, whatever the length of the group we have the corresponding bit position of each of this group we have consider, that delay has been reduced by the factor of that. So, after that again what we have seen that you instead of that as this in the LSB group, if we if we produce one that carry. So, that has to be passed through the; that means, LSB and unless and until we are reaching to the MSB that carry will not be assimilated.

So, that is why what we have what we see as it has to; that means, traverse towards the; that means, from LSB to the MSB. So, that is why what another technique we would we have used that is variable that grouping we have done. So, instead of keeping that  $k$  fixed to each of this group we have chose dynamically; that means, changed value which is that for each of this group for LSB and MSB keeping the LSB side and MSB side, the group the; that means, the word length for the group at the lower values.

Whereas the middle portion where I will group the middle portion of this corresponding bit position. So, at the time I can I will choose more on that; that means, the word length will be more on that and if I do that at that time what happens I am getting the corresponding this delay reduced by a factor of square root of  $n$  whereas, instead of in the earlier case it just variable that block adder if I use at the time I can get the; that means, delay variation that is  $n$  by  $k$  so; that means, I can reduce the delay by more if we choose this multi-level variable block adder operation or architecture.

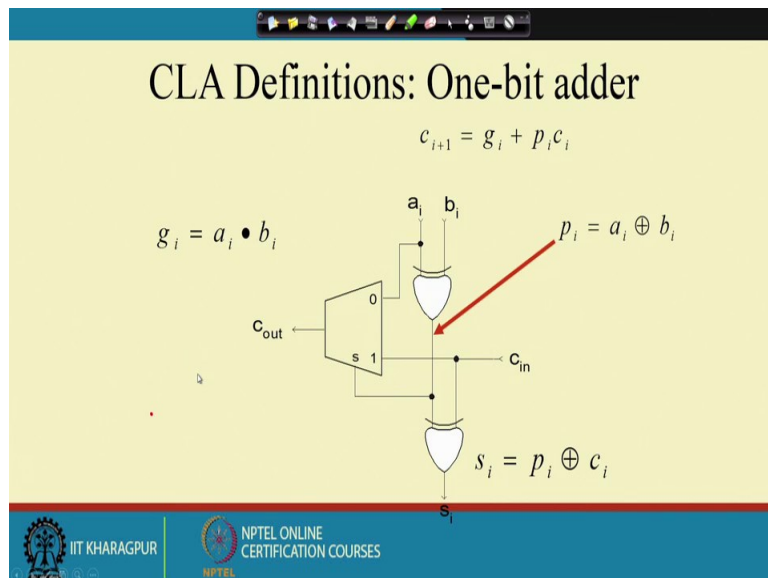
So, again today we will see; that means another architecture which is carry look ahead adder.

(Refer Slide Time: 03:08)



So, in the carry look ahead adder.

(Refer Slide Time: 03:12)



We have already; that means, seen this that, based on this the input bit the carry is generated and the carry is propagated. So, this is the; that means, this is the logic a XOR b that is the logic for carry propagation and a AND b that is the logic for carry generation. So, using one marks easily I can just do the; that means, I can skip this; that

means, at what bit position whenever this propagated value of this is 0, at that time it will select any values of a or b and whenever this value is 1 so, at the time it will select the corresponding carry in bit after considering the operation table of full adder cell that we have seen.

(Refer Slide Time: 04:07)

First we should examine a realization of a one-bit adder which represents a basic building block for all the more elaborate addition schemes.

Operation of a Full Adder is defined by the Boolean equations for the sum and carry signals shown in this slide:  $a_i, b_i$  and  $c_i$  are the inputs to the  $i$ -th full adder stage, and  $s_i$  and  $c_{i+1}$  are the sum and carry outputs from the  $i$ -th stage, respectively.

From the above equation it is clear that the realization of the Sum function requires two XOR logic gates.

The expression for Carry function could be rewritten using the Carry-Propagate  $p_i$  and Carry-Generate  $g_i$  terms.

If Carry-Propagate is 1, the Carry out of the stage will be equal to the Carry signal into the stage:  $c_{i+1} = c_i$  regardless of the carry inside the stage.

If Carry-Generate is 1, there will be a Carry signal out of the stage will be 1 regardless of the value of the incoming Carry signal.

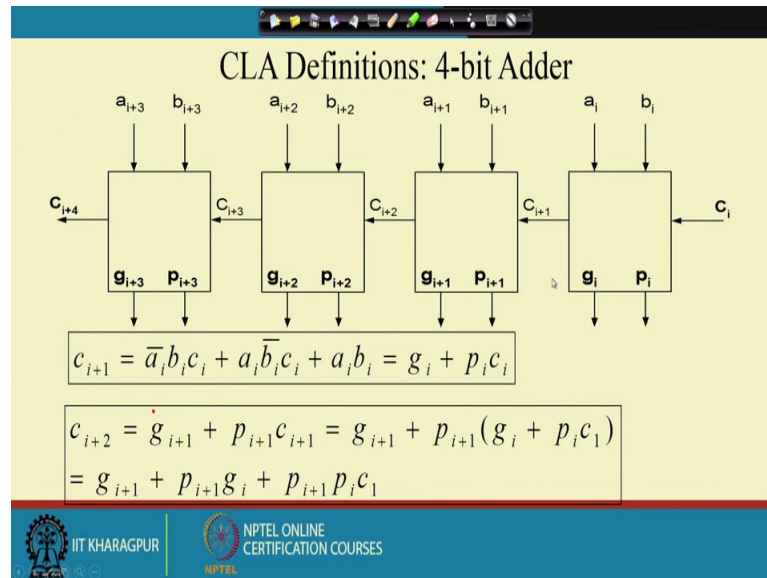
Given that the multiplexer block is often faster than a single gate, using multiplexer in the critical path helps to achieve better performance.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 40

So, what we have done. Firstly, what we have done we should examine a realization of one bit adder which represent a basic building block for all the more elaborated addition operation, and operation of a full adder is defined by the Boolean equation for the sum and carry signal shown in this particular slide which is  $a_i, b_i$  and  $c_i$  at the inputs of the  $i$ th full adder stage, and where is this  $s_i$  and  $c_{i+1}$  at the sum and carry output from  $i$ th stage respectively. From the above equation it is clear that the realization of sum function require two XOR logic gates as sum function is what  $a \oplus b \oplus c_i$  ok.

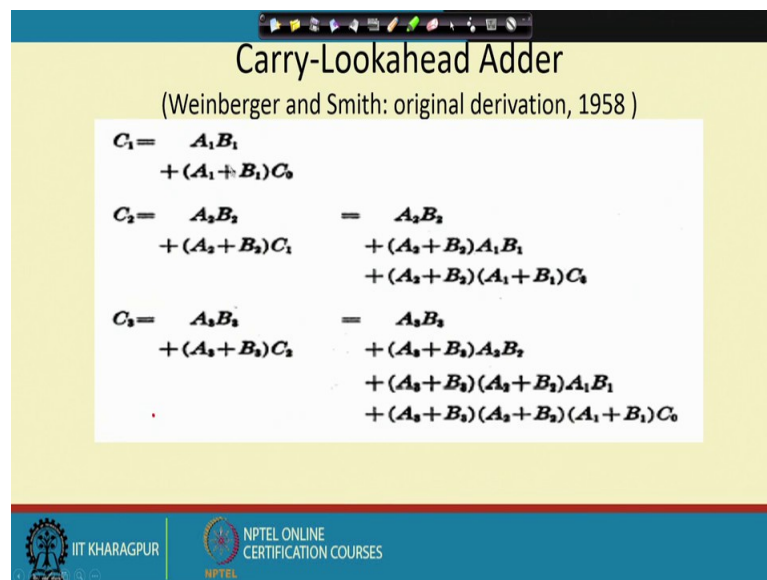
So; that means, two input XOR gate I need two level; to two input XOR gate; that means, I need two levels. So, the expression for carry function could be re written using the carry propagate  $p_i$  and carry generate  $g_i$  terms if carry propagate is one and the carry out of the stage will be equal to carry signal into the stage  $c_i$  plus 1.

(Refer Slide Time: 05:18)



So that means, the that is we have generated and so, what happens actually if we take this example? So, if we take ok.

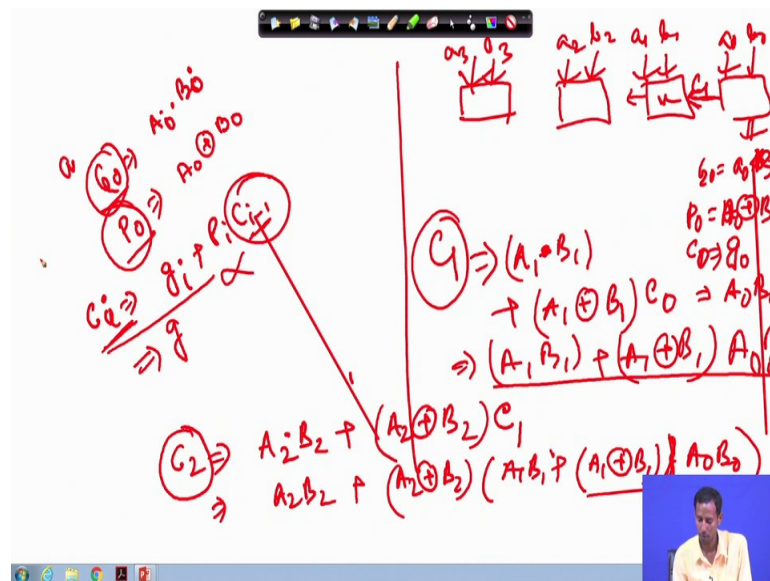
(Refer Slide Time: 05:25)



So, this is the carry which is generated at the very first level ok. So, what is the carry what I written that is carry is  $AB$  plus  $A \oplus B$  into  $C$  in. So, for the first level for the first level; that means, first level means after the LSB. So, where I need  $A_1$  and  $B_1$  is the input. So,  $A_1 B_1$  plus  $A_1$  plus  $B_1$  sorry this is not plus this is XOR ok

So,  $A_1 \text{ XOR } B_1$  into  $C_0$ ; so, then  $C_2$  is written as  $A_2 B_2$  plus this is OR operation and this is the XOR operation. So,  $A_2 B_2$  XOR with  $B_2$  into  $C_1$ ; so, now, if I put this  $C_1$  over here if I just in this particular equation if I put  $C_1$  over here. So, at the time what it will be? It will be  $A_2 B_2$  plus  $A_2 B_2$  into this  $A_1 B_1$  plus  $A_2 \text{ XOR } B_2$  into  $A_1 \text{ XOR } B_1$  into  $C_0$  but what we are doing basically here we from this. So, how we can develop this?

(Refer Slide Time: 06:58).



So; that means, for each of the stages we know that for  $P_0$  and  $G_0$  that is for zeroth stage or for the LSBs position right. So, for  $P_0$  and  $G_0$  for this propagation that is a XOR b th  $b_i$  and for generation that is a into b i right. Now if I just write that for  $G_0$  and for  $P_0$ . So, for generation what is that that is a i into b i so; that means,  $A_0$  into  $B_0$  for  $P_0$  that is  $A_0 \text{ XOR } B_0$  right now for  $C_0$  what I can write that is  $G_0$  plus  $P_0$  into  $C_0$ .

Sorry this is  $C_{i-1}$  if I write and this is  $C_i$ . So, if  $i$ ; that means, from the beginning now suppose I am having let us consider 4 bit. So, this is a 0 b 0. So, So, for this. So, for this is a 0 b 0 this is a 1 b 1 this is a 2 b 2 this is a 3 b 3 ok. So, for  $C_0$  and  $P_0$  of this what is that that is  $G_0$  is a 0 into b 0 for  $P_0$  that is a 0 XOR with  $B_0$  this is  $A_0$  and this is the carry in which is basically coming over here for this particular stage ok. So, then the carry that will be what this if I consider this; so, then for this  $C_i$  is this for sorry for  $C$

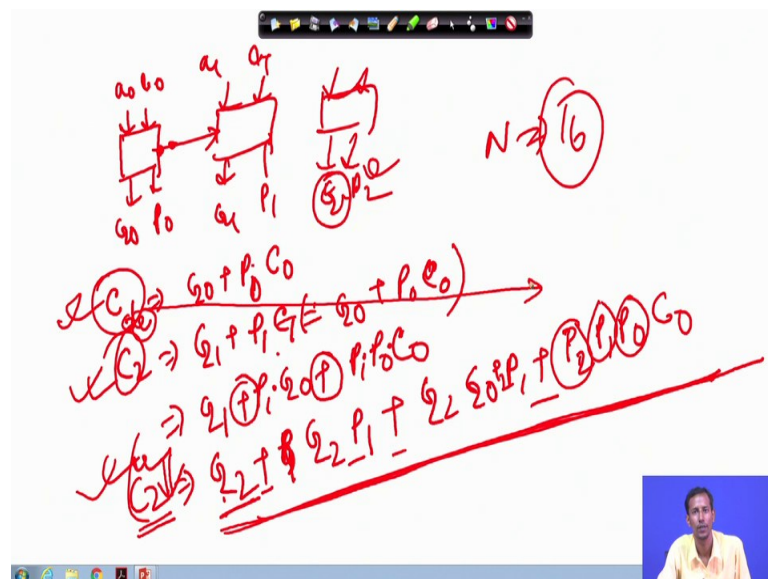
1 that is  $A_1$  for this particular case as for I am not considering this because this  $C_i$  minus 1 is 0.

So, this will not be come over here. So, I can calculate this  $C_1$  from here. So, at the time what will it will be  $G_i$  is  $G_i$  is nothing, but this  $A$  and  $B$  plus  $A_1$  XOR bit  $B_1$  along with this  $C_i$  minus 1 so; that means, which is nothing, but this  $C_1$ ,  $C_1$  is this now if I just  $C_1$  will be what  $C_{i-1}$  will be again if I just put this as. So, there is nothing, but the  $0$  ok.

So,  $g_0$  that is  $g_0$  is  $A_0 B_0$  so; that means, now what I can write this is  $A_1 B_1$  plus  $A_1$  XOR  $B_1$  along with  $A_0 B_0$  if I in the same fashion if I just want to calculate  $C_2$ . So, at that time what it will be? It will be like  $A_2 B_2$  this  $g_2$  plus this will be  $A_2$  XOR  $B_2$  along with this  $C_1$  this is  $C_0$ , this is  $C_1$  and this is  $C_0$ . So,  $C_1$ ; so now,  $C_1$  is what  $C_1$  is nothing, but this particular value. So, if I just put them  $A_2 B_2$  plus  $A_2$  XOR  $B_2$  into  $A_1 B_1$  plus  $A_1$  XOR  $B_1$   $A_0 B_0$  sorry not  $B_0 A_0 B_0$  after that.

So; that means, here I do not I have generally this  $C_2 C_1$  from the very beginning why how? They are generating this  $g_0$  and  $P_0$  term in each of this case.

(Refer Slide Time: 12:34)



So, what we are doing; that means, for each of this stage we are generating for a 0 and b 0 we are generating two case which is  $g_0$  and  $p_0$  or each of this case we are basically for a 1 and b 1 we are generating  $G_1$  and  $P_1$ . For each of this stages we have already

generated the corresponding  $G_2$  and  $P_2$  now the carry for the carry for carry over here at this particular stage for  $C_1$  I need that will be  $G_0$  plus  $P_1$  sorry  $P_0$  into  $C_0$ .

For  $C_2$  it will be  $G_1$  plus  $P_1$  into  $C_1$ . So,  $P_1$  into  $C_1$  means, again this will be replaced by this will be replaced by  $G_0$  plus  $P_0$  into  $C_0$ . So, ultimately if I just put it like this  $G_1$  plus this is  $P_1 G_0$  plus  $P_1 P_0$  and  $C_0$ . So,  $C_2$  will be something like this it will be  $G_2$  plus this one. So,  $G_2$  plus this one means this  $G_1$  sorry this will be  $G_2 P_1$  plus  $G_2 G_0 P_1$  plus  $P_2$  and  $P_1$  then  $P_1 P_2 P_1 P_0$  and  $C_0$  something like this it will be continued.

So in each of this case you will see that it is having  $g_2$  I have already from the beginning I have calculated that then  $P_2 P_1$  and  $P_0$  I have already calculated. So, now, whenever I am just increasing this values. So, at the time what I need? I need some of the AND gates for this and some of the; that means, OR gate for this because this particular operation from the beginning I have done. So, this  $C_1$  and  $C_2 C_3$  that are available it is not that whenever I am computing this  $C_2$  I have to wait unless and until this  $C_1$  computation is over it is not like that. I am computing each of this carry signal at a time simultaneously here ok.

So; that means, now the problem is there; that means, whenever I am computing this  $C_2$  at the time the levels of this that is increasing; if I consider this more on of  $n$  if I increase the number more. So, at the time the levels of addition over here that will increase, but the thing is that I do not have to wait unless and until this is basically finished which is happens in ripple carry adder. So this is the; that means, operations of  $k$ th if you just see that this is the 4 bit look at carry look ahead adder.

(Refer Slide Time: 16:25)

### CLA Definitions: 4-bit Adder

$$c_{i+1} = \bar{a}_i b_i c_i + a_i \bar{b}_i c_i + a_i b_i = g_i + p_i c_i$$

$$c_{i+2} = g_{i+1} + p_{i+1} c_{i+1} = g_{i+1} + p_{i+1} (g_i + p_i c_i)$$

$$= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i$$

So, here this  $a_i$   $b_i$  and it produce this  $p_i$  and  $g_i$  then again  $p_i$  plus 1  $g_i$  plus 1  $p_i$  plus 2  $g_i$  plus 2  $p_i$  plus 3  $g_i$  plus 3. So, then  $c_i$  plus 1 how it basically computes that is  $g_i$  plus  $p_i$  into  $c_i$  then  $c_i$  plus 2, that has been computed using  $g_i$  plus 1 plus  $p_i$  plus; that means, there is a nothing, but this.

So, instead of  $i$  that will be  $i$  plus 1 why  $c_i$  is now this  $c_i$  plus 1 that will come over here. So, something like this I will get of this expression.

(Refer Slide Time: 17:07)

### Carry-Lookahead Adder: 4-bits

$$c_{i+3} = g_{i+2} + p_{i+2} c_{i+2} = g_{i+2} + p_{i+2} (g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i)$$

$$= g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i$$

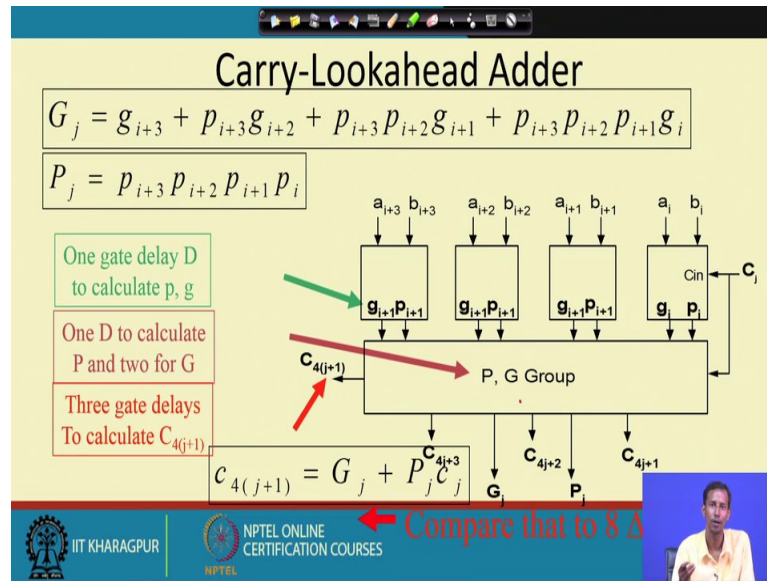
$$c_{i+4} = g_{i+3} + p_{i+3} c_{i+3} = g_{i+3} + p_{i+3} (g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i)$$

$$= g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i c_i$$



So,  $c_{i+4}$ ; that means,  $c_{i+4}$  what will be the expression? That is  $g_{i+3} + p_{i+3}c_{i+3}$ . So, this is up to this is for the generation of the carry and this particular  $p$  what we have seen that  $p_2 p_1 p_0$  into the last  $c_0$  this  $c_0$ . So, that is this particular expression is for propagation of the carry ok. So, more the; that means, if I consider is 16. So, 16 number of this  $p_i$  signal that will be consider over here.

(Refer Slide Time: 17:54)

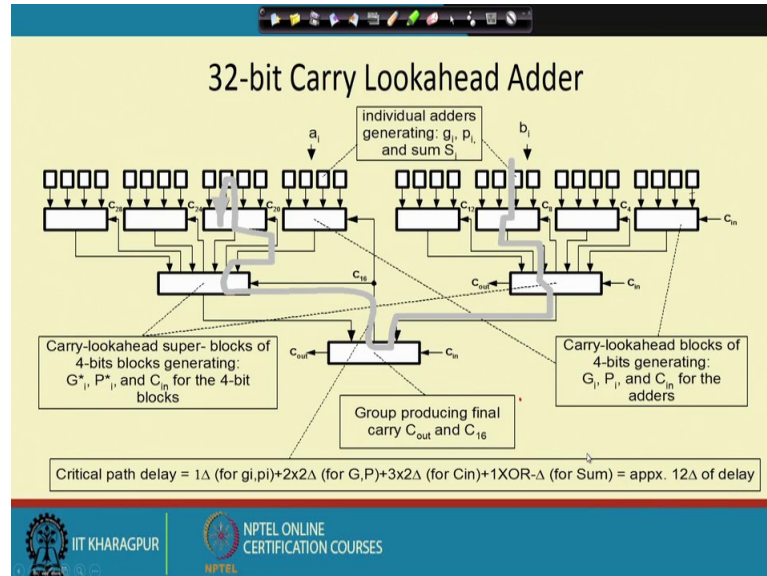


So, this is for carry look ahead adder and. So, here you see this one gate delay for calculating this  $g_{i+1}$  and  $p_{i+1}$  why? Because XOR gate I require for this for generation of this sorry for propagation of this for generation I need AND gate so; that means, one gate delay for this particular thing and for this  $p$  and  $g$  group one  $D$  calculate  $P$  and 2 for calculating  $G$  and for the final  $C_{i+j}$  it needs 3 gate delays to calculate  $C_{4j+1}$  why if you just if you just go back to this. So, if you see; that means, 1 2 3 4 if I just consider this. So, at the time this will give me; that means, what is the levels of gate I need for computation of this ok.

So, this indicates that thing only. So, the total number of; that means, gate delay is one here one for calculate the; that means, 1  $D$  to calculate  $P$  and two for calculating the  $G$  group and 3 gate delays to calculate  $C_{4j+1}$  so; that means, the total is required that is 1 plus 2 plus 2 that is 7 sorry 1 plus 1 2 then 4 plus 3 which is 7 gate, but in; that means, in carry look ahead adder that will be 8 gate delay. So; that means, I can reduce the corresponding levels for 4 bit only if you consider more on of this at the time I can

get more the savings, but for 4 bit I can say I can save one gate delay in compare to ripple carry adder.

(Refer Slide Time: 20:30)



So, So, this is this is the; that means, 32 bit carry look ahead adder ok. So, we have grouped into 4 bit look ahead carry adder and then this 4 bit look 4 bit whatever; that means, that carry we are considering then we have the subdivided into this corresponding to considering 4 of this is 16 bit adder and this is 16 bit adder than this two finally, basically added again and we are getting the corresponding final c out.

So, the total; that means, the delay will be any of this particular group either it may be comes from this to this to this then there a or this to this to because it has to pass through 1 2 3 4 5 ok. So, five of this particular block; so, that is why the delay for this 32 bit carry look ahead adder will be something like this. So, this group is basically used for carry look ahead adders of 4 bit blocks generating this  $G^* P^* I$  and  $C_{in}$  for 4 bit blocks and this is; that means, group final carry  $C_{out}$  and  $C_{16}$  and here carry look ahead adder this 4 of this 4 bit generating for  $G_i P_i$  and  $C_{in}$  for the adders.


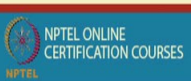

So, the total critical path delay for this one gate delay for cal computing  $G_i$  and  $P_i$  so; that means, for generation of the carry and propagation of the carry 2 into to get delay for again  $G$  and  $P$ ; that means, the computation of that we have already seen and  $C_{in}$  into 2 gate delay for  $C_{in}$  and; that means,  $C_{in}$  means the last one and then for one XOR gate

delay for sum total approximately 12 gate delay we will get for this particular 32 bit carry look ahead adders.

So, this is the; that means, expression here the; that means, there is a mistake which is this will be XOR operation. So, in this manner you can just express or you can just extend the corresponding C 1 then C 2 then C 3 then C 4 in this something like this ok.

(Refer Slide Time: 23:11)

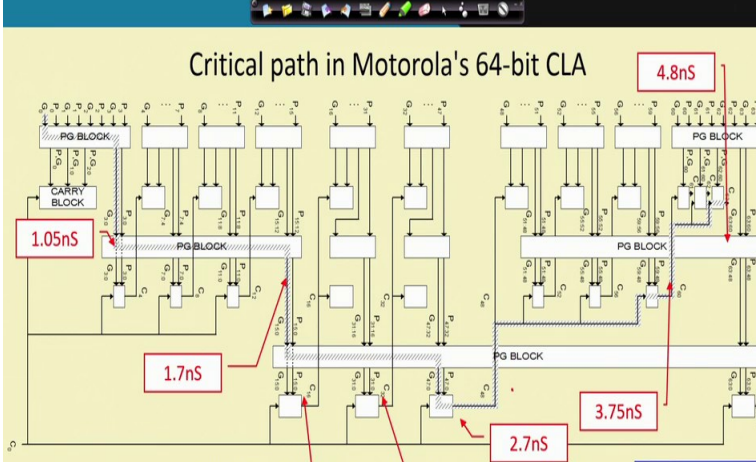
### Carry-Lookahead Adder (Weinberger and Smith: original derivation )

$$\begin{aligned}
 C_4 &= A_4 B_4 &= A_4 B_4 \\
 &+ (A_4 + B_4) C_3 &+ (A_4 + B_4) A_3 B_3 \\
 &&+ (A_4 + B_4) (A_3 + B_3) A_2 B_2 \\
 &&+ (A_4 + B_4) (A_3 + B_3) (A_2 + B_2) A_1 B_1 \\
 &&+ (A_4 + B_4) (A_3 + B_3) (A_2 + B_2) (A_1 + B_1) C_0 \\
 &= A_4 B_4 &+ (A_4 + B_4) A_3 B_3 \\
 &&+ (A_4 + B_4) (A_3 + B_3) A_2 B_2 \\
 &&+ (A_4 + B_4) (A_3 + B_3) (A_2 + B_2) (A_1 + B_1) (A_1 + C_0) (B_1 + C_0).
 \end{aligned}$$







So, this carry look ahead adder.

(Refer Slide Time: 23:16)

### Critical path in Motorola's 64-bit CLA



Critical path: A, B - G<sub>0</sub> - G<sub>3,0</sub> - G<sub>13,0</sub> - G<sub>47,0</sub> - C<sub>48</sub> - C<sub>60</sub> - C<sub>63</sub> - S<sub>63</sub>

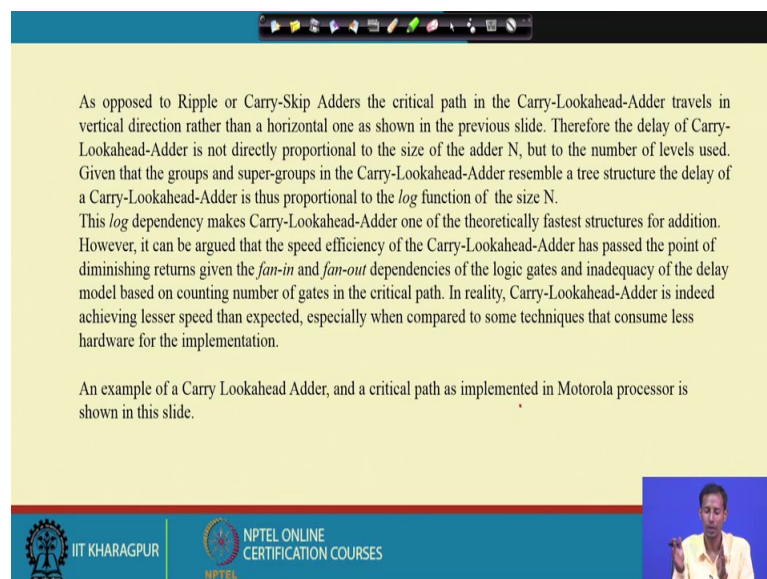
That are used there is used in Motorola one and; that means, one in Motorola's particular circuit. So, at the time it says that this critical path as it follows different; that means, this blocks. So, at the time it has different of this delay element; that means, for this G 0 to for 4 bit of this pg block.

The delay here that is 1.05 whereas, the same thing whenever this pg block it passes through the pg pg block; so, the time it is 1.7 nanosecond at this particular point. So, then for P 15 and G 15 it is 2.0, then for 31 it is 2.35 or 47 this is 2.7 for 50 the for 51 that is 3.7, ultimately for 63 the total time is 4.8 nanosecond and this is the; that means, corresponding architecture for this.

So, the critical path is that which one is the critical path it started from the G 0 then it is coming from this that particular that is 3 2 15; that means, for this 4 group 4 group then again it is coming to this, then coming over here passes through this then again it is going through this particular block and this ended at this C 63 which is basically producing the this C 64 which is the final output.

So, this is the critical path A [noise B A; that means, the 2 G 0 to G 3 0 G 15 0 G 47 0 then C 48 C 60 C 63 and then S 63 ok; that means, which C 63 is used for S 63. So, that will be the final critical path or for this particular system.

(Refer Slide Time: 25:33)



As opposed to Ripple or Carry-Skip Adders the critical path in the Carry-Lookahead-Adder travels in vertical direction rather than a horizontal one as shown in the previous slide. Therefore the delay of Carry-Lookahead-Adder is not directly proportional to the size of the adder  $N$ , but to the number of levels used. Given that the groups and super-groups in the Carry-Lookahead-Adder resemble a tree structure the delay of a Carry-Lookahead-Adder is thus proportional to the  $\log$  function of the size  $N$ . This  $\log$  dependency makes Carry-Lookahead-Adder one of the theoretically fastest structures for addition. However, it can be argued that the speed efficiency of the Carry-Lookahead-Adder has passed the point of diminishing returns given the *fan-in* and *fan-out* dependencies of the logic gates and inadequacy of the delay model based on counting number of gates in the critical path. In reality, Carry-Lookahead-Adder is indeed achieving lesser speed than expected, especially when compared to some techniques that consume less hardware for the implementation.

An example of a Carry Lookahead Adder, and a critical path as implemented in Motorola processor is shown in this slide.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this gp is basically used in one of the motorolas cheap ok. So, this says as opposed to the ripple carry adder the critical path in carry look adder travels in vertical direction rather than in horizontal as it; that means, in the ripple carry adder the carry is basically follows from this particular site ok.

So, there is the horizontal one, but here in case of this carry look adder it is in the vertical direction; that means, the how much level I need for computing the carry ok; for C I I need to carry; that means, consider this g and p considering this. So, whenever I am increasing more on that. So, at that time this levels of computing the carry so, that will be the most critical path ok.

So, that is why here the this critical path basically varies in the vertical direction not in the horizontal direction which happens in the case of carry look ahead adder. So, therefore, the delay of carry look ahead adder is not directly proportional to the size of the adder in, but to the number of levels used.

So, given the groups and super groups in carry look ahead adder resemble a tree structure the delay of carry look ahead adder is does proportional to log function of size n so; that means, if I use a 8 bit. So at the time either that the corresponding levels which I require that is  $\log_2 n$ ; so, that is 3. So, the log dependency makes carry look ahead adder one of the theoretical fastest structure for addition; however, it can be argued that the speed efficiency of carry look ahead adder has pass the point of diminishing return given the fan in and fan out dependency of the logic gates and inadequate of the delay model; based on counting number of gates in the critical path. In reality carry look ahead adder is indeed achieving lesser speed then expected especially when compared to some techniques that consume less hardware for the implementation.

So, where it will take; that means, what is the condition of fan in and fan out dependency what are those that we will see in the next class. That means, how this fan in and fan out dependency theoretically I can get this the corresponding that; that means, the corresponding critical path should be  $\log N$  ok, but in practical we do not get that why you do not get that? Because of this fan in and fan out dependency what is that fan in fan out dependency that we will see in the next class so.

Thank you for today.