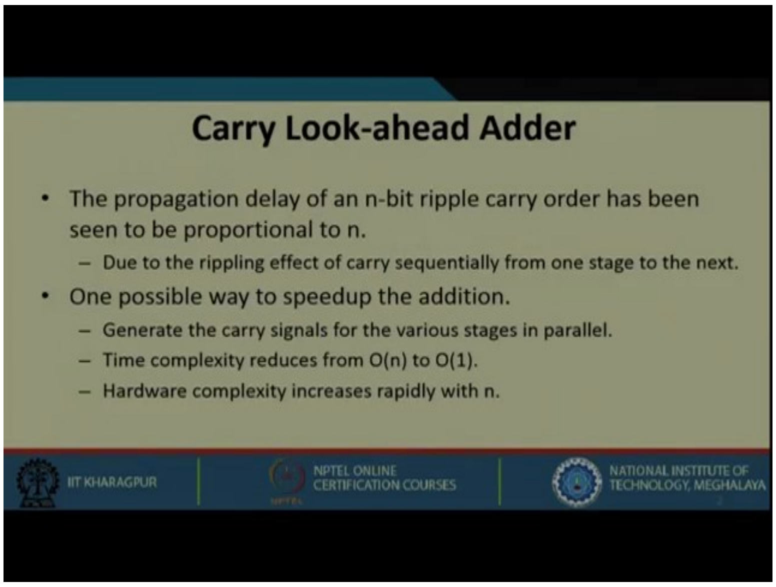
Computer Architecture and Organization

Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
Design of Adders (Part II)

We continue with our discussion on Adders. If you recall in our last lecture we looked at one kind of adder called ripple carry adder. And in the ripple carry adder we have observed one very distinct characteristic; that in the worst case the carry input ripples through all the stages up to the final carry output. This results in an overall worst case delay of the adder that is proportional to the number of bits or the number of stages.

(Refer Slide Time: 00:59)



Carry Look-ahead Adder

- The propagation delay of an n-bit ripple carry order has been seen to be proportional to n.
  - Due to the rippling effect of carry sequentially from one stage to the next.
- One possible way to speedup the addition.
  - Generate the carry signals for the various stages in parallel.
  - Time complexity reduces from O(n) to O(1).
  - Hardware complexity increases rapidly with n.

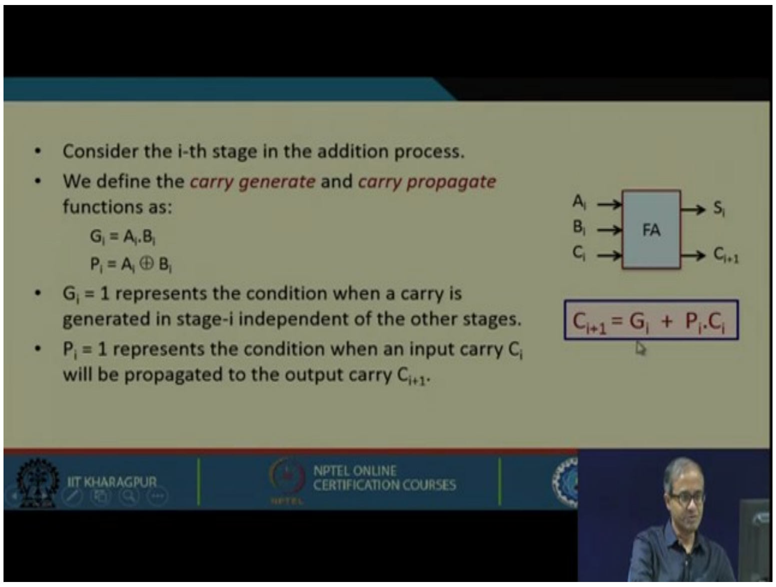
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, we continue with our discussion. What I just now mentioned is that the ripple carry adder that we have seen so far has total propagation delay in the worst case proportional to n, and the primary reason was the rippling effect.

Suppose I consider the full adder at any particular stage; full adder is unable to start the addition until or unless its carry input is available. Now in the ripple carry adder this carry input is coming sequentially through a rippling effect, because of that the different full adders was having to wait till the carry input comes. So, this second approach is another kind of adder that we look here is called carry look ahead adder. So, as the name implies we are doing some kind of a look ahead to determine the values of the carry.

If you can do that, then the total time complexity of the adder can reduce from order n which is roughly proportional to the number of stages to order 1 which means addition can be done in constant time independent of the number of stages, but this is possible if somehow we can generate the carry bits in parallel. If you can do that then all the full adders can perform their addition simultaneously. Here there is no need to wait for the previous full adder to complete; this is the basic idea behind the carry look ahead adder. But the only trouble here is that the hardware complexity the amount of gates that will be required will increase rapidly with the value of n. This is the drawback of carry look ahead adder.

(Refer Slide Time: 03:18)

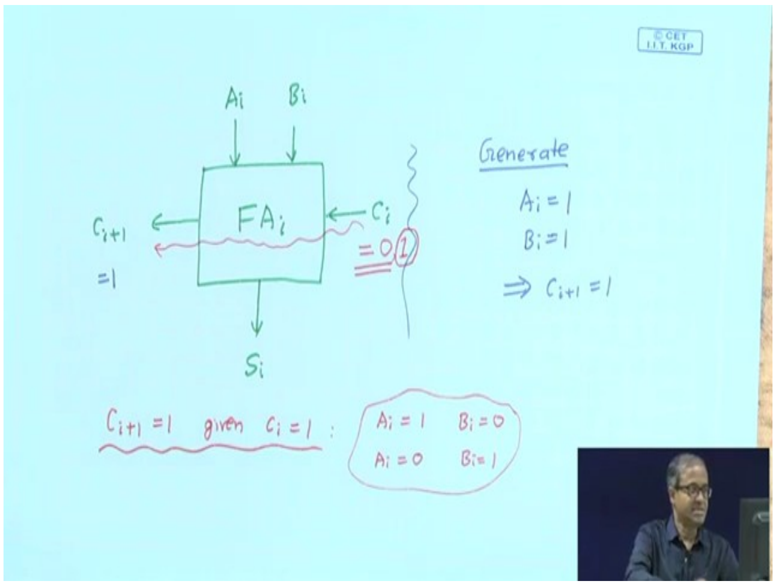


- Consider the i-th stage in the addition process.
- We define the *carry generate* and *carry propagate* functions as:
  - $G_i = A_i.B_i$
  - $P_i = A_i \oplus B_i$
- $G_i = 1$ represents the condition when a carry is generated in stage-i independent of the other stages.
- $P_i = 1$ represents the condition when an input carry C_i will be propagated to the output carry C_{i+1}.

$$\boxed{C_{i+1} = G_i + P_i.C_i}$$

So, let us see the basic idea, we look at a full adder in terms of how it works let us consider the full adder in the i-th stage of a ripple carry adder say where the inputs are A_i, B_i and C_i and the outputs are S_i and C_{i+1} . Now we define 2 different kinds of events one we call as the carry generate, other we call the carry propagate. The idea is similar to full adder.

(Refer Slide Time: 04:01)



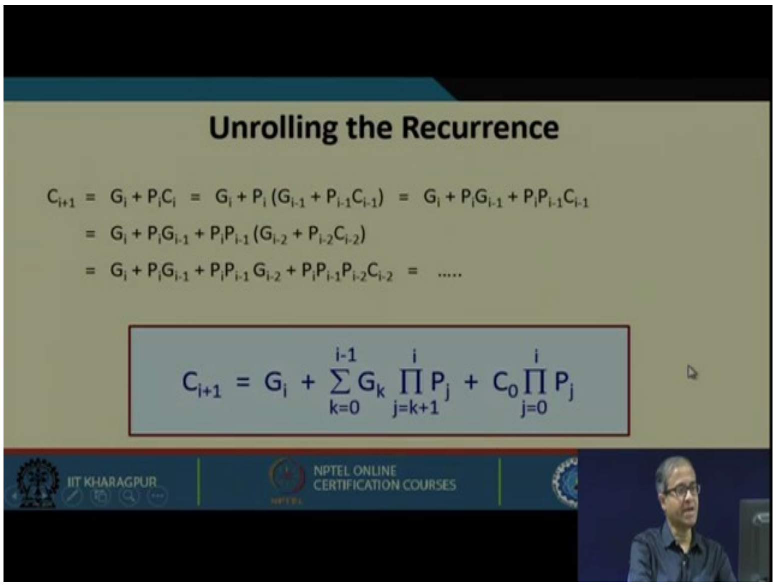
Let us consider full adder at stage i. So, the inputs are A_i and B_i , sum is S_i and input carry is C_i and the output carry is C_{i+1}. Carry generate means that this full adder will be generating a carry; meaning C_{i+1} will be 1 independent of what has happened in the previous stage.

So, I do not care what the value of C_i is; C_i can be 0, C_i can be 1 whatever it depends on the previous stages, but this A_i and B_i are the present inputs from the numbers which are coming. So, we can say if A_i is 1 and B_i is 1 then definitely there will be a carry out this will imply C_{i+1} will be 1. This is the scenario for carry generate a carry is generating in the i-th stage itself irrespective of what has happened in the previous stage. So, you see the generate function G_i is defined as A_i and B_i.

The second scenario is called carry propagate. Suppose my input carry has come as 1. So, my question is under what condition this input carry will be propagating to the output; that means, C_{i+1} will be equal to 1 given C_i equal to 1 what are the other conditions for that? This carry will be propagated under the condition when A_i is 1, B_i is 0 or A_i is 0, B_i is 1 because you add this 1 to 1 and 0 it will generate a carry you add this 1 to 0 and 1 it will also generate a carry. So, this condition actually talks about the exclusive OR function.

So, the carry propagate function is defined as $A_i \oplus Bi$. So, this is what I have explained so far G_i equal to 1 represents the condition when a carry is generated in this stage itself independent of the previous stages, and propagate function will be 1 when an input carry C_i will be propagated to the output carry C_{i+1}. Now with respect to this G_i and P_i sub functions we can write down an expression for $C_{i+1.}$ we can write the carry output will be 1 if either a carry is generated or the propagate function is true and there was an input carry. So, if either of these two conditions are true then C_{i+1} will be 1. So, I can simply write down the expression for this C_{i+1} in terms of G_i and P_i like this.

(Refer Slide Time: 08:29)



Unrolling the Recurrence

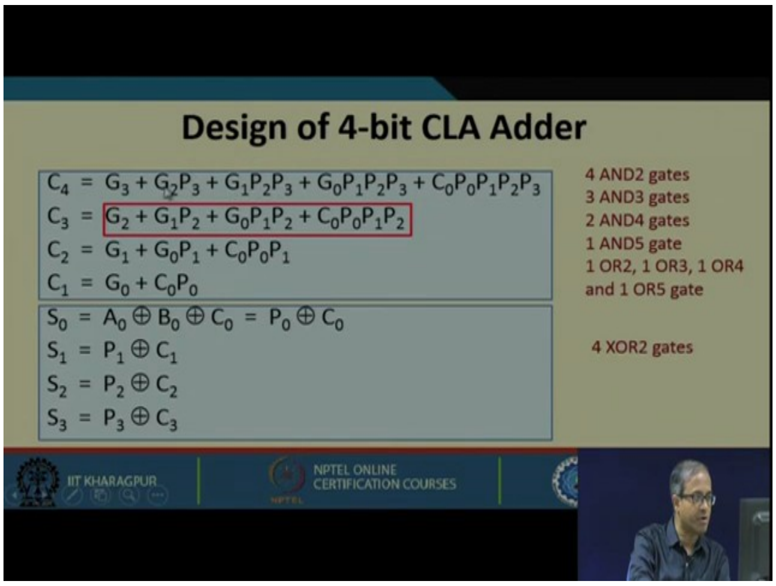
$$C_{i+1} = G_i + P_iC_i = G_i + P_i(G_{i-1} + P_{i-1}C_{i-1}) = G_i + P_iG_{i-1} + P_iP_{i-1}C_{i-1}$$
$$= G_i + P_iG_{i-1} + P_iP_{i-1}(G_{i-2} + P_{i-2}C_{i-2})$$
$$= G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + P_iP_{i-1}P_{i-2}C_{i-2} = \ldots..$$

$$\boxed{C_{i+1} = G_i + \sum_{k=0}^{i-1} G_k \prod_{j=k+1}^{i} P_j + C_0 \prod_{j=0}^{i} P_j}$$

Now, this is like a recurrence relation, I am defining the C_{i+1} in terms of the input i. So, I can recursively expand this function like C_{i+1} is $G_i + P_iC_i$; in a similar way this C_i we can expand as $(G_{i-1} + P_{i-1} C_{i-1})$, again I have a C_{i-1} ,expand the C_{i-1} as G_{i-2}, plus $P_{i-2}C_{i-2}$ multiply them out; you continue this multiplication process till you reach C_0. So, in the last term where there is C_0 will be containing the product of all the P's.

And there will be several other terms with $G_{i-1,}$ there will be a single P_i, G_{i-2} there will be 2 P_i's, G_{i-3} there will be 3 P_i's and so on. So, it can be expressed in the form of expression like this. So, if you work this out you will see that this expression and this generalization they mean the same thing. So, actually I am working this out for a specific case for a 4 bit adder.

(Refer Slide Time: 10:06)



Design of 4-bit CLA Adder

$C_4 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$

$C_3 = \boxed{G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2}$

$C_2 = G_1 + G_0P_1 + C_0P_0P_1$

$C_1 = G_0 + C_0P_0$

4 AND2 gates
3 AND3 gates
2 AND4 gates
1 AND5 gate
1 OR2, 1 OR3, 1 OR4 and 1 OR5 gate

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

$S_3 = P_3 \oplus C_3$

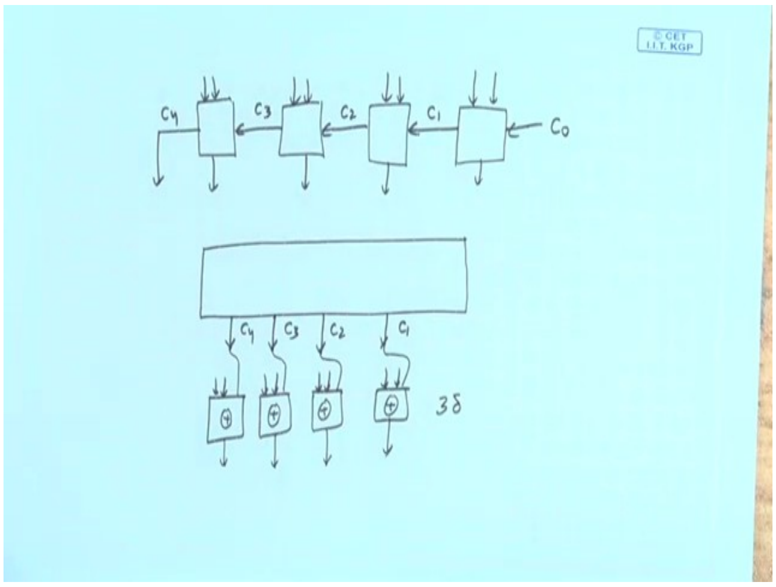
4 XOR2 gates

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Let us do it. So, let us consider a 4-bit carry look ahead adder where using the same expressions whatever we have written down here. So, using the same expression C_4 can be written as $G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3$; this actually follows from this way expression. C_3 can be similarly expressed like this, C_2 can be expressed like this, C_1 can be expressed like this.

Using the carry propagate functions, the sum outputs can be expressed as shown.

(Refer Slide Time: 12:46)



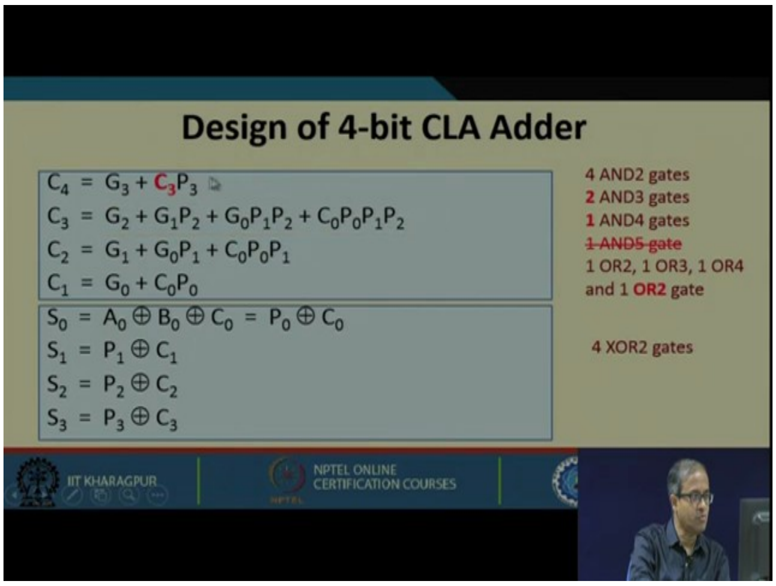
Now, if you do this what you are actually gaining; let us try to understand in normal ripple carry adders that have seen earlier.

So, the inputs were coming; carry in was there, and carry outputs were rippling through. So, if this was your input carry C0. So, C1, C2, C3 and C4 --- because of the rippling effect we have to wait till the carry input was available then only it can do the computation and generate the sum. Now you see what we are saying is that we propose to design a separate combinational circuit that will be directly generating the C1, C2, C3 and C4 values, and then we will be having 4 full adders; they will be fed with the A,B inputs as usual, and the third carry input can be directly fed from here. So, you see there is no ripple effect between the full adder stages.

So, after these carries are available, you need just an XOR function here; that means, just a 3 delta delay. So, when all the sum will be available, this will be independent of the value of n, this is what we try to show in these expressions. So, if we use these expressions to generate a combinational circuit we can directly generate C1, C2, C3, C4, and if we can do that we can directly feed those carry inputs. So, we do not need the full functions of the full adders; we need just XOR gates to generate the sums, and this can be done in parallel with no rippling effect.

So, this expression can be simplified; you just look at it one thing this sub expressions G2 G1 P2; this one is appearing here. A lso if you take P3 common from these four terms, you will see that this same sub-expressions appearing.

(Refer Slide Time: 15:39)



Design of 4-bit CLA Adder

$C_4 = G_3 + C_3P_3$

$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$

$C_2 = G_1 + G_0P_1 + C_0P_0P_1$

$C_1 = G_0 + C_0P_0$

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

$S_3 = P_3 \oplus C_3$

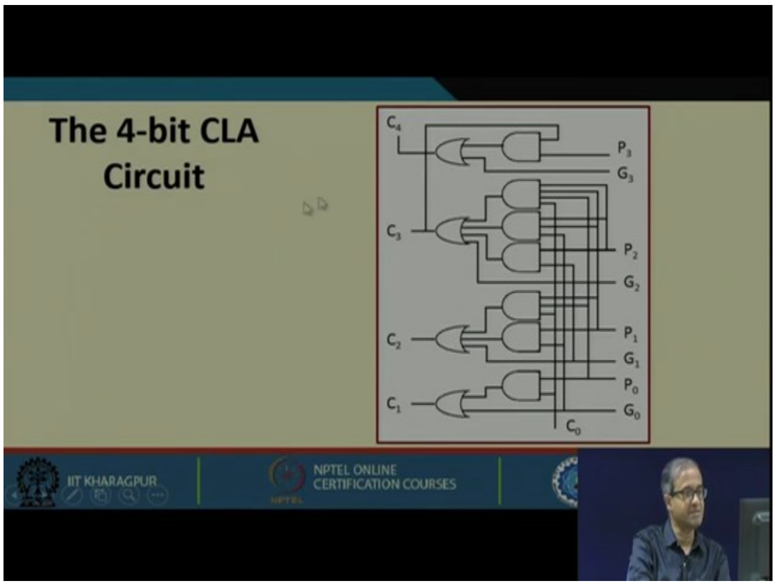
4 AND2 gates
2 AND3 gates
1 AND4 gates
~~1 AND5 gate~~
1 OR2, 1 OR3, 1 OR4 and 1 OR2 gate

4 XOR2 gates

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, if you substitutes C3 here, this expression gets simplified to C3 P3, and if you do the simplification the advantage you gain is that see here you needed 3 AND3 gates and 2 AND4 gates; here the number of gates are reducing.

(Refer Slide Time: 16:11)



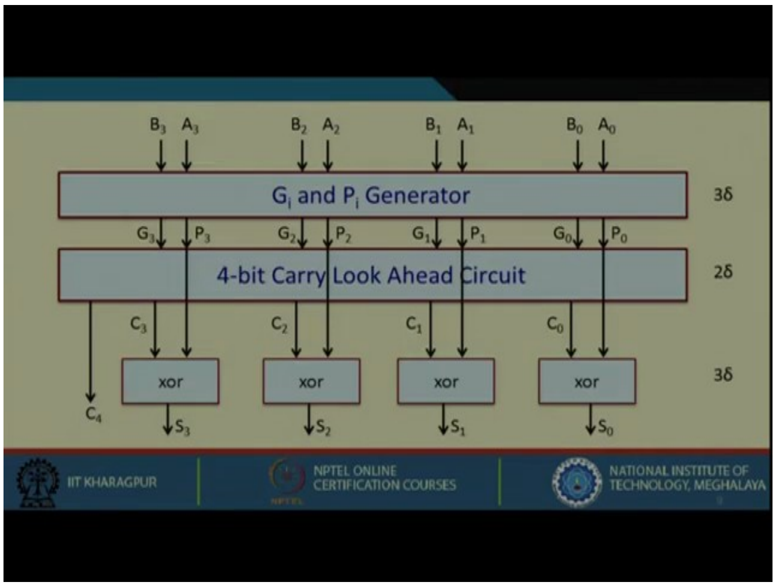
The 4-bit CLA Circuit

So, your circuit becomes simpler. So, if you simply realize these functions your circuit will look like this. So, suppose I have my Pi Gi functions, you can generate the carries using this kind of a 2-level circuit and you recall that Pi is nothing but the XOR of the inputs; that means, 1 XOR gate and Gi is nothing, but the end of the input; that means, 1

AND gate. So, you need this much hardware, but one problem is that if we look at this recurrence relation; as the number of n increases this the complexity of this function C1, C2, C3, C4, C5, C6 they will go on increasing.

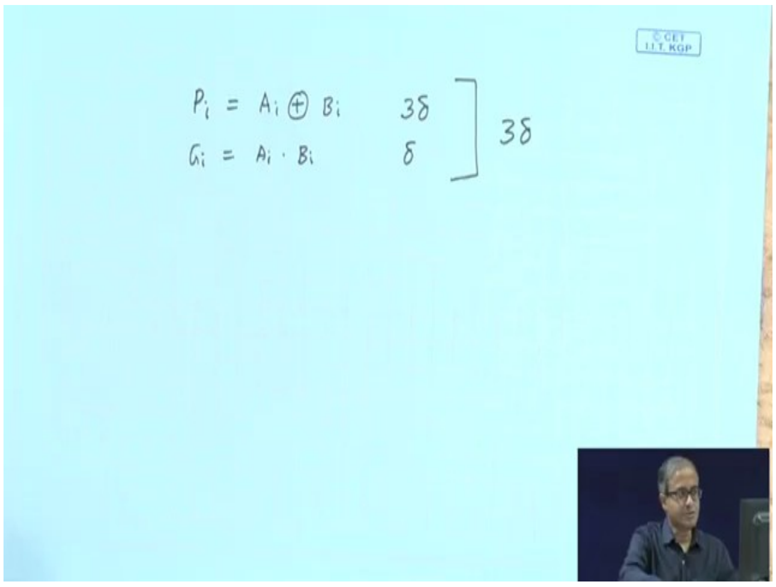
So, the number of gates will also increase. So, it will become increasingly difficult to implement or realize the function. So, you may know that as the number of inputs the size of a gate increases, the delay also increases very rapidly. So, you need to control the number of inputs limited to a small value, 2 or 3 let say. So, this is one drawback.

(Refer Slide Time: 17:38)



So, the circuit of this 4-bit carry look ahead adder will overall look like this; there will be one circuit which will be generating Gi and Pi.

(Refer Slide Time: 17:55)



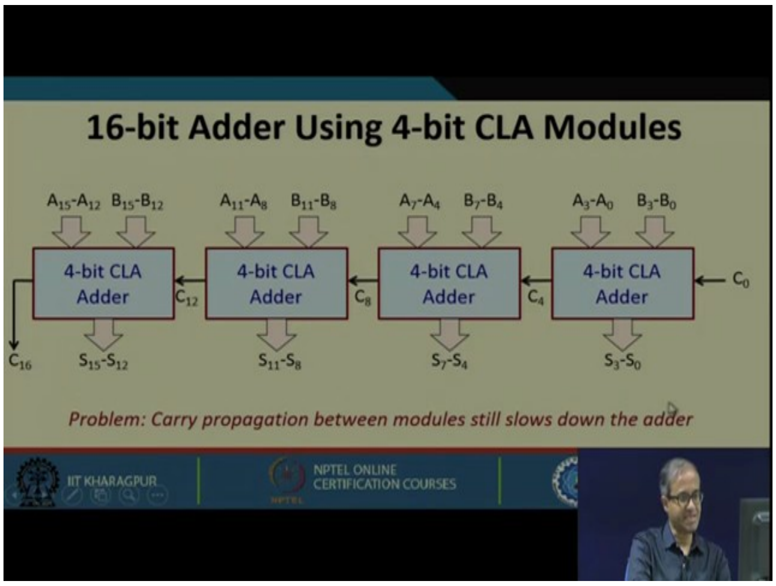
$$\left.\begin{array}{lll} P_i = A_i \oplus B_i & & 3\delta \\ G_i = A_i \cdot B_i & & \delta \end{array}\right] 3\delta$$

How just as I have said Gi is nothing but AND; Pi is nothing but XOR. So, how much delay will take? This XOR takes a delay of 3 delta as I said earlier.

AND gate is a single gate delay. So, the worst-case gate delay is 3 delta. So, this Gi and Pi generator requires the time of 3 delta to generate all the Gi, Pi values. And this 4 bit carry look ahead circuit will look like this; you say this is just a 2-level AND-OR circuit, but if you implement like this the delay increases, but you implement it in the original form, it becomes worst-case delay of 4 for generating C4, but I am assuming the previous approach where you are not doing that simplification. So, it will be 2 delta, but using the simplification this will become 4 delta.

And once you have done this all the carry values are available in parallel. So, you simply do an XOR with the P values to generate the sums, and this XOR again will require 3 delta. So, you see after a constant time of 8 delta, you have generated the sum bits in parallel. So, that is why we mentioned earlier that we are aiming to reduce the addition time from O(n) to O(1); that means, proportional to n to constant time addition. So, carry look-ahead adder helps us in doing this.

(Refer Slide Time: 19:48)


16-bit Adder Using 4-bit CLA Modules

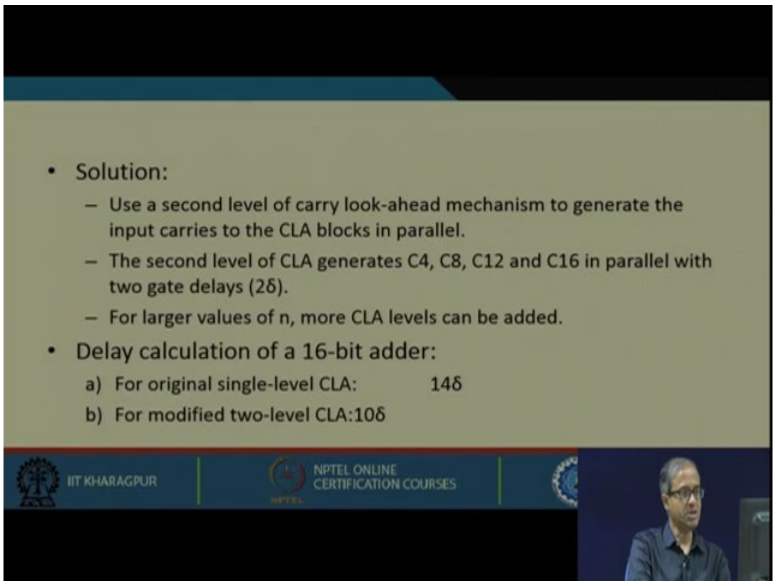
Problem: Carry propagation between modules still slows down the adder

But 4-bit is too less a number. Let us say we want to design a 16-bit adder; having a single 16-bit carry look ahead adder will become impractical as I have just now said; because the expression for the carries will become exceedingly complex.

So, the product terms will become larger in sizes, number of terms will also become very large requiring a very large OR gate. So, one alternative may be you can have as a basic building block of 4-bit carry lookahead adder modules, just like the once you have designed. They will be taking 8 delta time for addition, and you chain them just like a ripple counter between these stages. So, this is a mix of carry look ahead and ripple carry adder. So, every 4 bit you can do in parallel, but this C4 will be generated only after you just recall these carries will be generated only after 3 delta + 2 delta = 5 delta time. So, you will have to wait that time for the carry to be generated; similarly C8, similarly C12, similarly C16.

So, unless C4 is available you cannot start addition here, unless C8 is available you cannot start addition here, but of course, you can do this Gi, P i generation in parallel, but for this carry look ahead circuit you have to wait 2 delta, and only after that carry look ahead circuit C4 will be generated, C8 will be generated, C12 will be generated. So, carry propagation between modules is still slowing down the adder here.

(Refer Slide Time: 21:48)



- Solution:
  - Use a second level of carry look-ahead mechanism to generate the input carries to the CLA blocks in parallel.
  - The second level of CLA generates C4, C8, C12 and C16 in parallel with two gate delays (2δ).
  - For larger values of n, more CLA levels can be added.
- Delay calculation of a 16-bit adder:
  a) For original single-level CLA: 14δ
  b) For modified two-level CLA:10δ

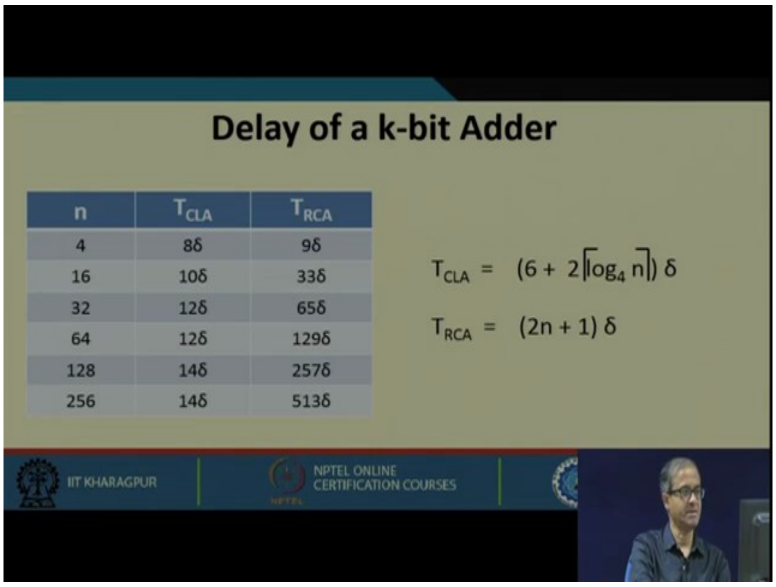
So, we propose that we use a second level of carry look ahead like you see here. We have 4-bit carry look ahead modules, and inside these modules we have a circuit like this. So, we already have Gi, Pi generator and carry look ahead circuit.

So, what we are saying is that at a higher level let us have another carry look ahead circuit for which you can derive the equations in a similar way, that will be generating C4, C8, C12 and C16 in parallel in constant time. So, in that case you can have a 2 level carry look ahead adder module, this is what we are saying here use a second level of carry look ahead mechanism to generate the input carries for the carry look ahead blocks in parallel, and this second level block will be generating the carries C4 C8 C12 and C16 together in constant time, but if n is larger maybe you will be requiring more number of levels.

Let us say I need a 64 bit adder, I build 4-bit modules and combine to build a 16-bit adder, 4 such 16-bit adders with another 4-bit carry look ahead module I can use to build up 64-bit adder. This is how so-called fast adders are built in many of the recent computer systems. So, delay calculation is fairly simple for original single level carry look ahead adder for 16-bit, the delay was 14 delta, but for 14 delta means I am talking of this circuit because this carry generation will take 2 delta additional time because of this carry look ahead time 2 delta because this Gi, Pi can be generated in parallel and it will become 14 delta.

If you do a calculation, but if you do this modified form then you need only 10 delta. So, I am not showing the detail of this 2 level multi-level design; conceptually the idea I mentioned talking of a general k-bit adder, I am showing a quick comparison with a ripple carry adder.

(Refer Slide Time: 24:18)



Delay of a k-bit Adder

n	T_{CLA}	T_{RCA}
4	8δ	9δ
16	10δ	33δ
32	12δ	65δ
64	12δ	129δ
128	14δ	257δ
256	14δ	513δ

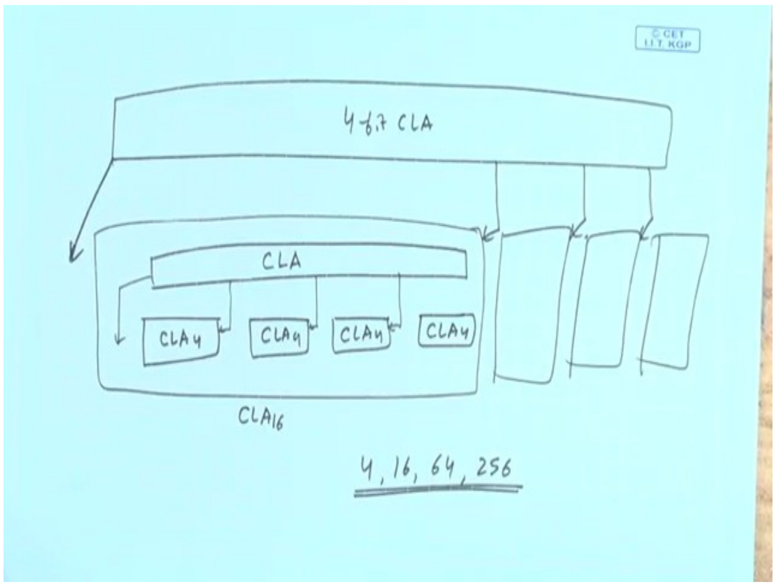
$$T_{CLA} = (6 + 2\lceil \log_4 n \rceil)\,\delta$$

$$T_{RCA} = (2n + 1)\,\delta$$

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

And a carry look ahead adder with this kind of 4-bit 4-bit look ahead. 4-bit look ahead means I am just repeating once more that my idea is like this.

(Refer Slide Time: 24:37)

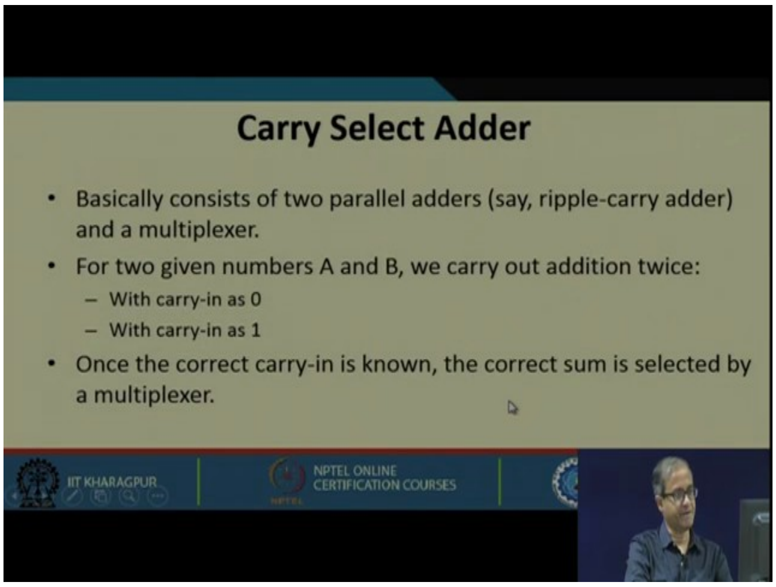


So, I will be using carry look ahead adder, four CLA4 blocks; that means, 4-bit carry look ahead adder. So, I use a higher-level carry look ahead module for this will be parallely generate these carries. So, what I have now is a block which I can call as CLA16 16-bit adder.

So, I can have 4 such blocks 16-bit adders, and above that I can have another higher level 4-bit CLA module. So, these will be generating the carry inputs for these high level adders. So, in multiples of 4 I can do, in 4 bits, then 16 bits, then 64 bits, then 256 bits, and so on. This means assuming that 4 is a reasonable number, beyond 4 it will be too expensive to construct a carry look ahead adder this is our assumption. So, from this calculation the delay for a carry look adder comes like this 6 + 2 log n to the base 4 if you make a simple calculation it will come.

And this will be the number of look ahead stages So, if I compare you see therefore, ripple carry adder as n increases the delay increases rapidly, but for carry look ahead adder it does not increase that much appreciably; it remains more or less in the same order.

(Refer Slide Time: 27:01)



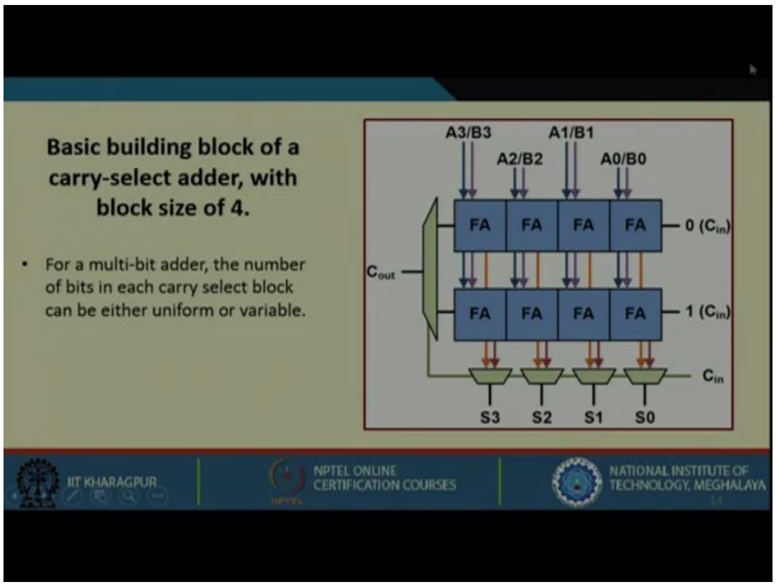
Carry Select Adder

- Basically consists of two parallel adders (say, ripple-carry adder) and a multiplexer.
- For two given numbers A and B, we carry out addition twice:
  - With carry-in as 0
  - With carry-in as 1
- Once the correct carry-in is known, the correct sum is selected by a multiplexer.

Now, let us look at an alternate kind of an adder called carry select adder; this is more like a philosophy that can make addition faster. Because the hardware is quite inexpensive today, here we are using 2 adders for every addition, let us say we use two ripple carry adders, and in addition we use a multiplexer.

So, why you are using 2 adders? Because the idea is that we do not know the input carry let us say. So, it will take some time for the input carry to be generated. So, I take two adders; the two adders are parallely adding the numbers, first one is assuming that the carry is 0, second one is assuming that the carry is 1. So, we are generating the two sums later on when the carry is known. So, one of the two sums can be selected by using a MUX only after a MUX delay that is quite fast; this is the basic idea.

(Refer Slide Time: 28:10)



Basic building block of a carry-select adder, with block size of 4.

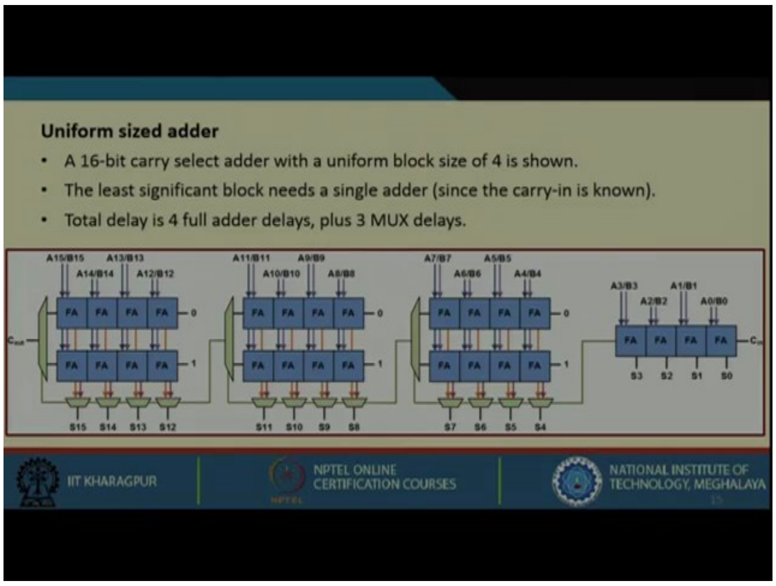
- For a multi-bit adder, the number of bits in each carry select block can be either uniform or variable.

So, I am showing a schematic diagram of a 4-bit can select adder. You can see, here I am assuming there is a ripple carry adder kind of a thing; 4 full adders, there is another adder; the first one is assuming carry in is 0, second 1 is assuming carry in is 1.

So, the two full adders are generating the sums; the first one is this orange one, and this is brown. So, the two additions are fed to a chain of 2 to 1 MUX. Later on when the actual value of carry in gets known, by that time the full adder operations are already done. So, you can simply select the MUX and take out one of the 2 addition sums that you have calculated; similarly the carry out.

So, either you take carry out from here or carry out from here depending on the value of C_{in}. So, this is what carry select adder means. Basically you are trying to save time by carrying out 2 additions both with a carry in of 0, and other with a carry in of 1, and then later on you select one of the 2 sums based on the actual carry which is generated.

(Refer Slide Time: 29:35)



Uniform sized adder

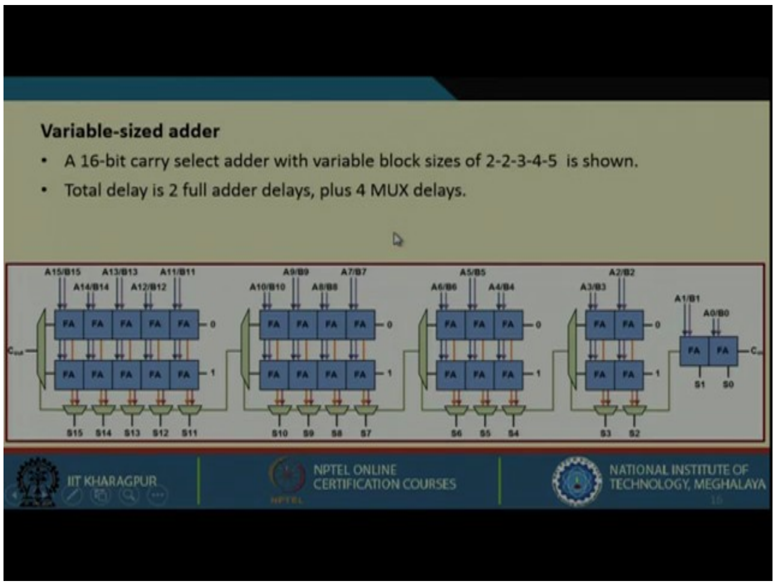
- A 16-bit carry select adder with a uniform block size of 4 is shown.
- The least significant block needs a single adder (since the carry-in is known).
- Total delay is 4 full adder delays, plus 3 MUX delays.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, let us take 2 example cascades. So, here we are showing how a 16-bit carry select adder can be generated by a cascade of 4 such 4-bit adders. For the least significant stage you do not need a carry select adder because you know the carry in already. So, there is no need of any speculation it can be 0 it can be 1. So, the last stage there will be a single ripple carry adder, but in the only other stages there will be a module like the one we have explained. Now you see so after the addition time of a 4-bit adder is done. So, all these adders are ready with their sum outputs. So, this adder is also ready with its carry output.

So, once it is done you select the MUX; take out the correct sum select this MUX, take out the correct sum, select the MUX take out the correct sum. So, it is just a chain of MUXs; it is quite fast --- MUX means may be 2 gate delays. So, the total time will be the addition time of a 4-bit adder plus the total delay of this MUX; after that time you will be getting all these some bits in parallel. Now there is an alternate way that can make the performance even better. You can make these adders of variable sizes like this.

(Refer Slide Time: 31:09)



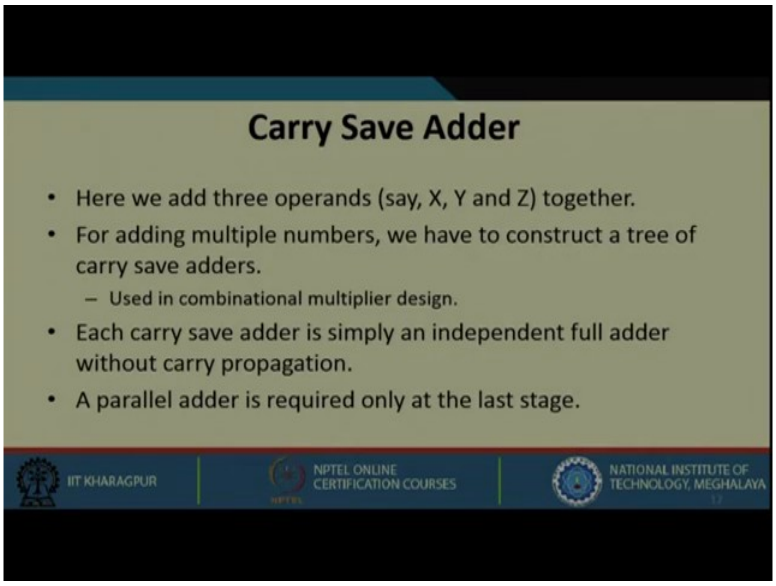
Variable-sized adder

- A 16-bit carry select adder with variable block sizes of 2-2-3-4-5 is shown.
- Total delay is 2 full adder delays, plus 4 MUX delays.

The idea is that there is no point in having all 4-bits; you see 2-bits is a minimum time you need for this carry to propagate to this MUX. Now by the time this MUX propagates the signal, this 3-bit addition is also done by the time this MUX propagates the signal this 4-bit addition is already done.

So, like this you have to make a calculation of the full adder delays and also the MUX delays, and come up with an optimum configuration this is a so called 2 2 3 4 5, 2 2 3 4 5 configuration. For a 16-bit adder by optimizing the sizes you can actually minimize the overall addition time.

(Refer Slide Time: 32:09)



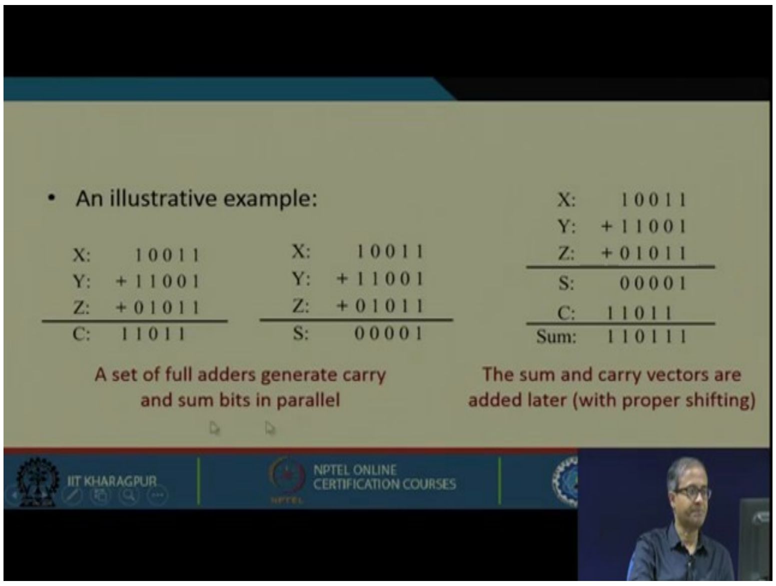
Carry Save Adder

- Here we add three operands (say, X, Y and Z) together.
- For adding multiple numbers, we have to construct a tree of carry save adders.
  - Used in combinational multiplier design.
- Each carry save adder is simply an independent full adder without carry propagation.
- A parallel adder is required only at the last stage.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, the last kind of adder that we look at today is carry save adder. The concept is very simple, in a carry save adder we are adding 3 numbers together. If there are more than 3 numbers, to add then you have to construct a tree of such adders. The carry save adder is basically an independent array of full adders, and only in the last stage we need a parallel adder.

(Refer Slide Time: 32:44)



- An illustrative example:

```
X:     10011        X:     10011           X:      10011
Y:   + 11001        Y:   + 11001           Y:    + 11001
Z:   + 01011        Z:   + 01011           Z:    + 01011
-------------       -------------          --------------
C:     11011        S:     00001           S:      00001
                                           C:     11011
                                           --------------
                                           Sum:   110111
```

A set of full adders generate carry and sum bits in parallel

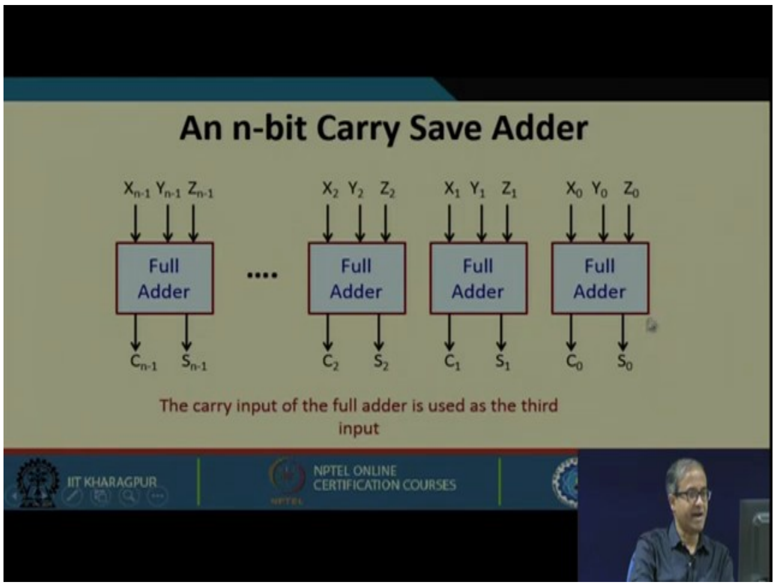
The sum and carry vectors are added later (with proper shifting)

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Let us take 2 examples; that we want to add 2 numbers X and X. Let us say we add it twice, here we are generating only the sum without any carry propagation independently like the 3 numbers X, Y, Z we are adding together.

Later on what we do this sum and carry which you have generated like this. We do a shift of this carry and add them up together this sum and carry, and will be getting the final correct sum. So, we are saying is that we are not unnecessarily carrying out carry propagation, we are having carry and sum as 2 separate words, and later on we are adding them together after proper shifting like 1-bit shift carry will be shifted by 1 place you can add them together.

(Refer Slide Time: 34:03)

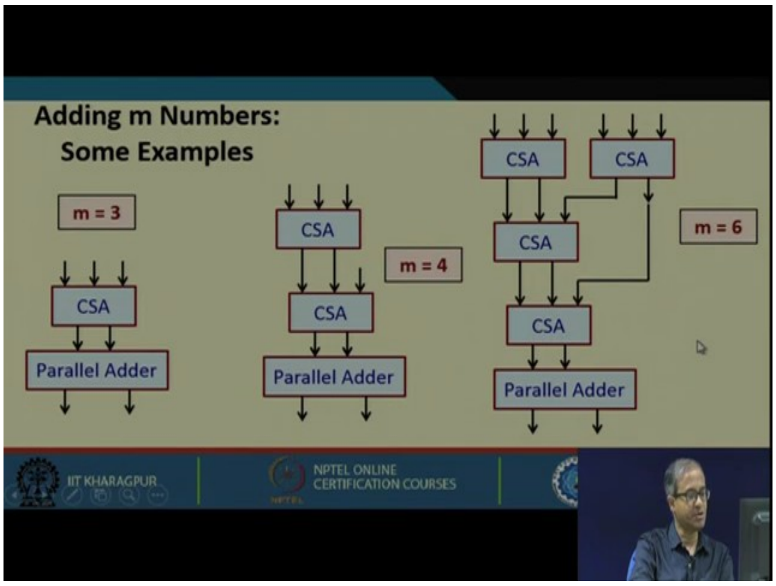


An n-bit Carry Save Adder

The carry input of the full adder is used as the third input

So, a carry save adder technically is just a set of independent full adders without an interconnection because there are 3 inputs A, B and a carry in. So, technically a full adder is able to add 3 numbers X, Y, Z and is generating sum and carry as the outputs.

(Refer Slide Time: 34:24)



So, some examples I am showing here for m = 3; you can use a single carry save adder to add the 3 numbers generate this sum and carry, the last stage use a parallel adder to add them up to get the final sum. Let us say m = 4; you use 1 carry save adder to generate 2, to add 3 numbers sum carry, and the fourth number add it with this sum.

So, you can have a tree of carry save adders; the last stage you need a parallel adder, the advantage is that the total addition time can be fast and you can add many such numbers together in parallel.

So, with this we come to the end of our discussion on adders. In our next lecture we shall be starting some discussion on the design of multipliers.

Thank you.