Computer Organization and Architecture

Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 02
High-Speed Circuit Design - Fast Adder Circuits

So, if you had done, you had done discrete mathematics.

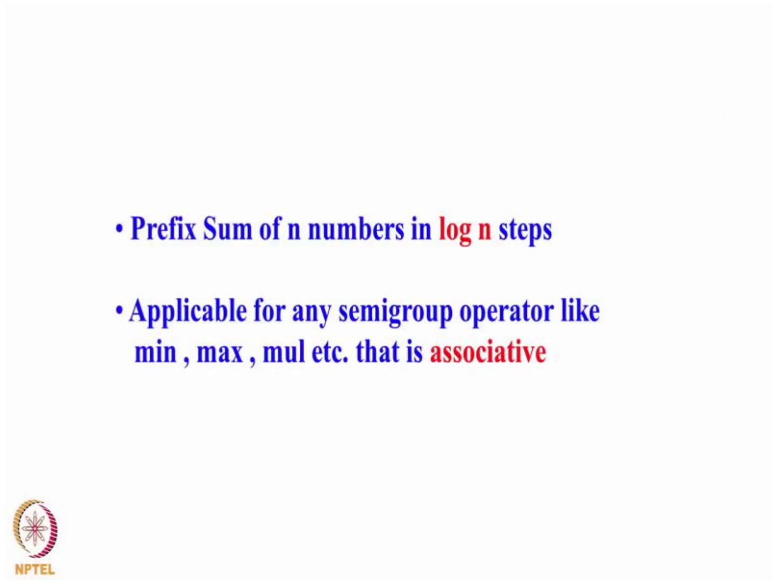
Student: Yes, sir.

Right. So, you know what is a semi group. So, semi group is you know what is a algebra.

Student: yes, sir algebra.

How to define a algebra and discrete mathematics? It has a set of elements which we call as a carrier set and some operators. If that operator is associative on that set of elements, because operator basically defines an operation on that set of elements. So, it is associative on the set of elements then that algebra is called as semi group, plus there are other properties. So, that is why we call it as a semi group operator.

(Refer Slide Time: 00:57)



- Prefix Sum of n numbers in log n steps
- Applicable for any semigroup operator like min , max , mul etc. that is associative

In the sense that if you say it is a semi group operator automatically it means that it is an associative operator. You all follow recursive doubling any doubts in that yes or no man something.

Student: Sir no.

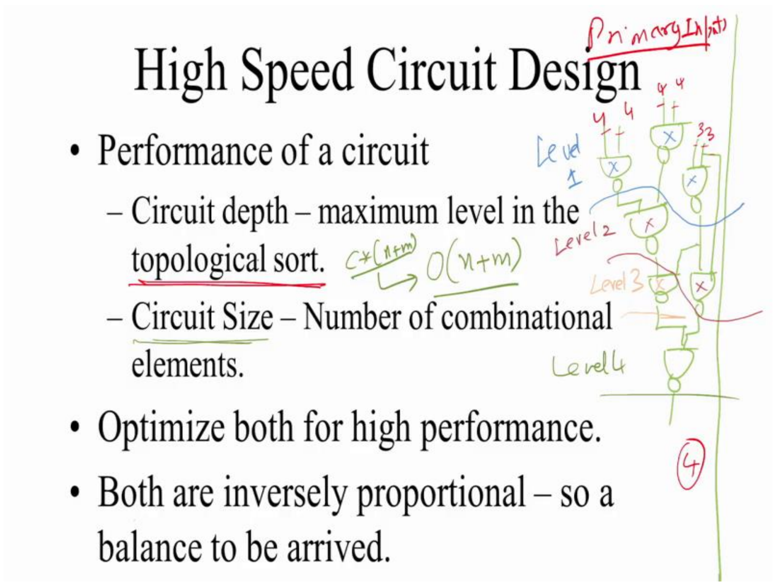
No doubts fine. So, in this part of what today and tomorrow we will learn about carry ripple carry look ahead adders and then another very interesting multiplier called Wallace tree multiplier. See there are many multiplication algorithms that are there in standard books like (Refer Time: 01:31) all these things. Today, when we look at very fast computations I think much more parallel multiplier like Wallace tree makes sense. So, let us not go and read all the multiplication algorithms ever discovered by mankind. So, we will just do this Wallace tree multiplier. So, I will teach you this multiplier plus these 2 adders. I think we have done the carry ripple adder, even para guide you have we should have done the carry ripple as a part of additional course, but I will quickly revise that and then go into carry look ahead adder construction.

Now what we need is speed. I need operations that work very fast. Now when we want speed, the speed is determined by if I take a circuit. How will you define the speed of the circuit, the speed is determined by the delay incurred in the circuit. So, how do you measure the delay in the circuit? When I change an input any arbitrary input what is the time required for the output to change for the corresponding change in the input. The maximum delay is what we call as the delay incurred by the circuit. So, for some input changes it could happen say in one unit of time. For some other input it can take 5 units of time. For some other input may take 10 units of time.

Now, the delay incurred by the circuit will be 10 is a maximum of all possible such delays and that would be called as the delay of the circuit. And this delay is directly proportional to what we call as the circuit depth. Circuit depth is that assume that I have identical components in the circuit, say I have nand gates I can make a entire circuit using nand gates yes right. Now from the input from any of the input I will go to the output. There are different ways different paths in which I could go. And let me keep counting the number of nand gates that I encounter when I move from n input to n output.

The total number of nand gates the maximum number of nand gates between an input and an output is what we call as the depth of the circuit right.

(Refer Slide Time: 03:57)



High Speed Circuit Design

- Performance of a circuit
  - Circuit depth – maximum level in the topological sort.
  - Circuit Size – Number of combinational elements.
- Optimize both for high performance.
- Both are inversely proportional – so a balance to be arrived.

Primary Inputs
C*(n+m)
O(n+m)
Level 1
Level 2
Level 3
Level 4
4

So, let us take this very this simple example. So, this is one nand gate. So, let us take this input let us take this input. For this input let me have one more I could reach from this input to the output with just 1 2 2 2 gates, but again there is another path from this input to this output which involves 3 gates. So, the depth of this input is 3 the depth of this input is 3 the depth of this input is 4 this is 4 and this is 3.

So, the entire depth of this circuit is now 4. So, this is a circuit. So, the depth of this input is. So, there is one path which is as you see here which takes 2 while there is another path which takes 3. So, the depth of this is 3. The depth of this this is also 3 the depth of this is 4, 4, 4 and 4. So, the depth of this entire circuit is 4. Now a very quick way of doing it is what we call as a topological sort, what is a topological sort, we take. So, there are note that in a circuit there are these 6 inputs, which I have marked in red here these are basically called as primary inputs primary inputs. They are called primary inputs because they are note driven by any other circuit element right.

Now, let us take all the primary inputs let us first remove all the gates that are driven only by the primary inputs. So, this is a gate these 3 gates for example, are driven only by primary inputs. So, this is basically level one. Now we remove all the level one gates now and find out what are all the gates that are now, input less if I remove these 3 gates in level one note that these 2 gates actually become input less, if I remove off these 3 gates now this brown gates become input less. So, those are the level 2 gates. When I remove all the level 2 gates, when I remove all the level 2 gates note that this gate becomes input less, only this gate still this gate has one input here. So, this becomes level 3 and finally, this becomes level 4 and I remove the level 3 gates.

So, I remove first I remove all the gates which are driven only by the primary inputs and that become level one. I remove all the level one gates and find out who are all saved completely and I get 2 gates which are level 2, I remove the level 2 gates I get one more here level 3 I remove that and I get level 4 right. This is called actually this type of what we do here this algorithm is called topological sorting. So, what will the topological sorting give you, it will give you gates at the same level, and it will also give you number of levels right 2 outputs right it will for every level it will give you the gates that belong to that level and also the number of levels.

So, we have 4 levels in this circuit. So, by that the depth of the circuit could be calculated by doing what we call as a topological sorting. So, what is the time complexity of this algorithm? I visit every edge ones and I visit every node ones. So, the time complexity if n is the number of edges and m is the number of n is the number of vertices and m is the edges then this the complexity is order n plus m right. So there is actually some constant into n plus m, which we call in computer science you put it by this notation the big o notation which is order n plus m. So, what is order n plus m in terms of complexities it is a polynomial algorithm and what is it is complexity.

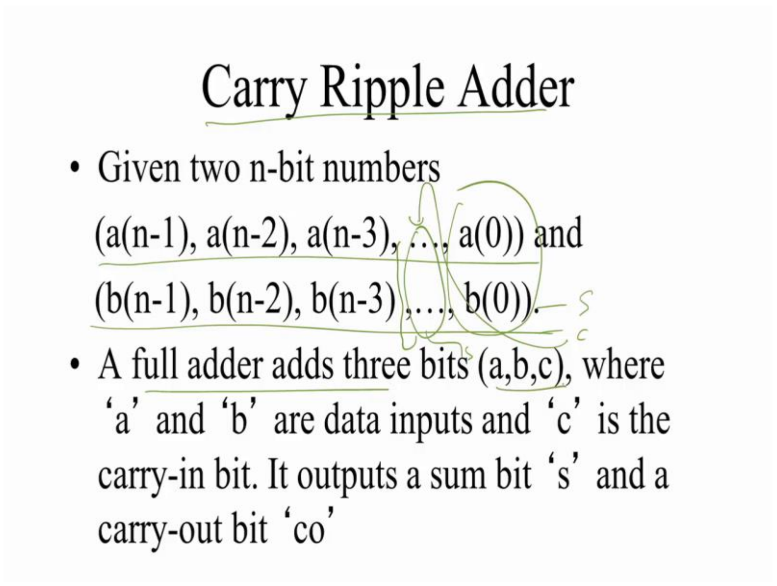
Student: Linear.

It is linear complexity right linear, why it is linear because it is linear in the number of inputs. So, what is a size of the input of this graph it is equal to the number of vertices plus number of edges right, that is the size of the input. So, the algorithm also takes order n plus m. So, this is this is a linear complexity algorithm right. So, this topological sort is proportional to the size of the circuit right.

Now, what we need to do when we try to do an high speed circuit design, is that we optimise both for the size and for high performance. I need to have the size of the circuit to be less and I should have the depth of the circuit to be less right. Either I should have small size and the depth also should be small then I get the best circuit because today we are talking of performance in the notion of computer architecture is not just the timing, it also timing plus the power that you are consuming and the area that you are consuming right. If I make a big one acre chip nobody is going to buy because I have been repeatedly telling you in the previous class correct in the even in the previous course.

So, we need to we need to make a chip that is marketable what it means I need cost I need power and area all should be within a within our board. So, what in in essentially; that means, that my area should be less and I should get performance. Unfortunately, both are contradictory right. So, both are inversely proportional. If I go and say improve my increase my area I could get very good performance, but if I want to decrease my area then my performance basically increases my performance also decreases; that means, the time actually increases right. So, when I start designing circuit is I should optimise both on the circuit size and also on the circuit depth. And we will try and see how these 2 are you know what is the trade off between these 2, with the example of a carry ripple adder and a carry look ahead adder.

(Refer Slide Time: 12:32)



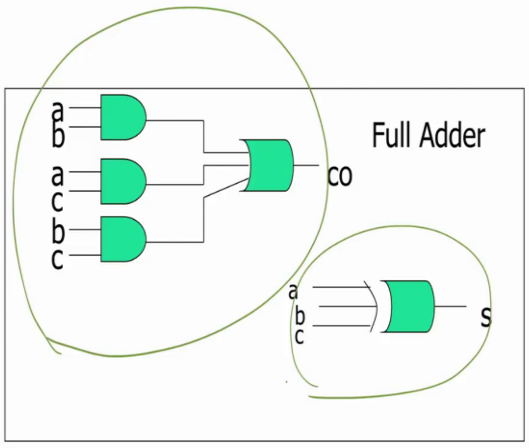
Carry Ripple Adder

- Given two n-bit numbers (a(n-1), a(n-2), a(n-3), …, a(0)) and (b(n-1), b(n-2), b(n-3) ,…, b(0)).
- A full adder adds three bits (a,b,c), where 'a' and 'b' are data inputs and 'c' is the carry-in bit. It outputs a sum bit 's' and a carry-out bit 'co'

Now, what is addition step more simplest operation I am giving you 2 n bit numbers let us call it a n minus 1 to a 0 a n minus 1 to a 0, and another b n minus 1 to b 0. And now we want to construct an addition circuit for this. The first what we will do is to construct carry ripple adder, what is a carry ripple adder in few minutes I will finish that because you have learnt in the previous course, a carry ripple adder is mimics the way we do actually addition. First we will add a 0 and b 0 we will get a sum and a carry bit that carry bit we will take to a 1 e will add that carry bit a 1 and b 1 that will give me another sum bit another carry bit that I will take on and so forth.

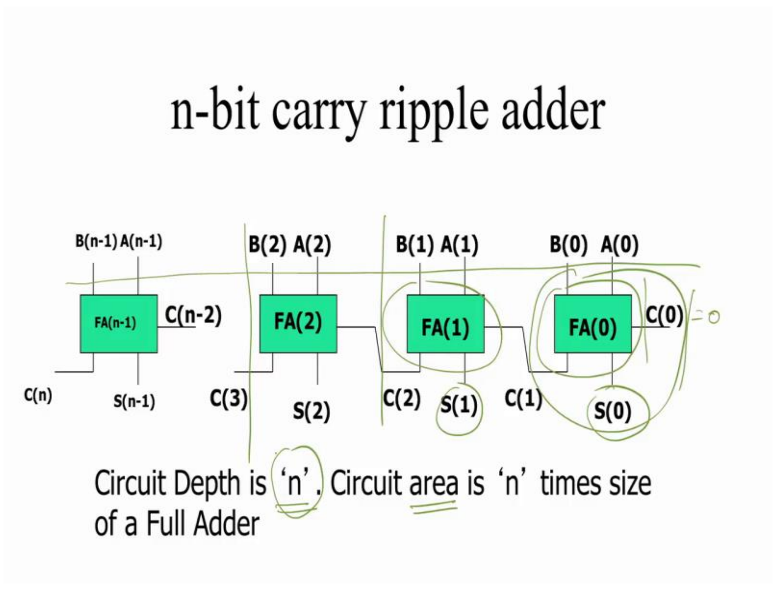
So, this is carry ripple addition. So, the carry ripple addition the basic module for a carry ripple adder is what we call as a full adder which actually adds 3 bit is a b c, where a and b corresponds to the input bit is and c is the carry bit that is coming from the previous stage right. And it outputs 2 bit is one is the sum bit and another is the carry out bit which is passed on to the next stage right. We have seen this in detail we have also implemented this in the previous course.

(Refer Slide Time: 13:57)



So, there are 3 bit is a b c, this is how we calculate this carry, carry out which is a b plus a c plus b c and we also have a sum bit which is a x r b x r c right. We also saw how we came out with these equations right, by just using a simple corn of map correct or truth table.

(Refer Slide Time: 14:27)



n-bit carry ripple adder

Circuit Depth is 'n'. Circuit area is 'n' times size of a Full Adder

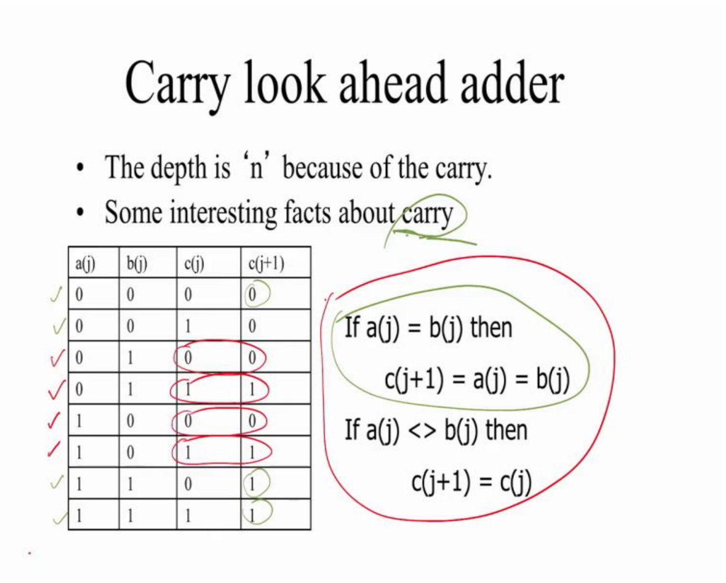
So, this is n bit carry ripple adder. So, we have we have several full adders as you see here full. So, full adder 0 to full adder n minus 1, will give a 0 and b 0 and c 0 normally this c 0 will be 0, to the first full adder which will give us the sum bit s 0 and a carry bit that we will pass it onto the next stage. The next stage will take a 1 b 1 and the carry bit from the previous stage it will give us s 1 then and also c 2 which is pushed onto the next stage and so on correct.

Now, when I do a topological sorting of this circuit at a very high level first these are all what are all the primary inputs to this circuit, is all these are primary inputs to the circuit. A a 0 b 0 till a n minus 1 b n minus 1 and c 0. So, let us remove all the primary inputs right.

Now, the only element that will become you now completely shared is this full adder 0 right. Now you remove all the elements of full adder 0, then only full adder 1 will become this because it is there is a carry line that is coming here. Then full adder one will come up, then when I remove of the elements of full adder one then full adder will come and so and so forth. So, the depth of this circuit is n if you do a topological sorting you will see that it is n or some constant times n right. And what is the area or the size of the circuit, it is n times the size of a full adder because I am putting n full adders. So, it s approximately n times the size of the full adder. Why approximately because there are some lines we need to connect and that will consume some area.

Now, I have a circuit whose depth is n and the area is also n right. For me this n is unacceptable that circuit depth of n is unacceptable because I need to work very fast. So, suppose tomorrow I want to do some cartography algorithm, I want to do an 128 bit addition if I take I will take 128 units of time to do this and say I am trying to do a voice codec I am trying to send my thing through what is app and I do not want somebody to listen I am doing an inscription of my voice right. I cannot afford to do this type of thing because the latency will become too high. So, I need to have very fast additions you know inside my circuit. So, I am just giving you an example of practical example where this n is actually unacceptable to me for say a 128 bit or so.

(Refer Slide Time: 17:11)



Carry look ahead adder

- The depth is 'n' because of the carry.
- Some interesting facts about carry

a(j)	b(j)	c(j)	c(j+1)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

If a(j) = b(j) then

c(j+1) = a(j) = b(j)

If a(j) <> b(j) then

c(j+1) = c(j)

Now, we will go and improve the speed. And how do we improve this speed. So, from now do not sleep for till this (Refer Time: 17:25) of last semester stuff. Now who created the problem in the previous slide, why did the circuit depth became n, because of the carry. If that damn carry was not there you would have been extremely happy. So, if I want to go and improve the speed then what should I do I should go and handle the carry properly. I should do something about that carry. So, then what we need to do.

Let us take the truth table of carry and start looking at it with great you know interest right. Let us take this truth table this is a truth table of the carry right. So, a j b j and c j and this is c j plus 1. Now let us see what is going to happen here there is something very interesting about this. When a j is equal to b j here c j plus 1 is equal to either a j or b j.

For example, when is a j equal to b j let us take these 4 entries. When a j is equal to b j your c j plus 1 is equal to a j is equal to b j right.

So, the green matches with the green, now let us take some other. Now when a j is not equal to b j then c j is equal to c j plus 1. These are the 4 cases where a j is not equal to c j note that c j is equal to c j plus 1, when a j equal to b j nearest, but your c j plus 1 becomes independent of c j, when a j is not equal to b j your c j plus 1 is equal to c j, so the dependency of a carry of a particular stage right, on the carry of the previous stage. Why that circuit depth of n came because the carry that I calculated was dependent on the previous stage right and that is why right. So, my sum bit and the carry bit of the current stage was dependent on the carry of the previous stage.

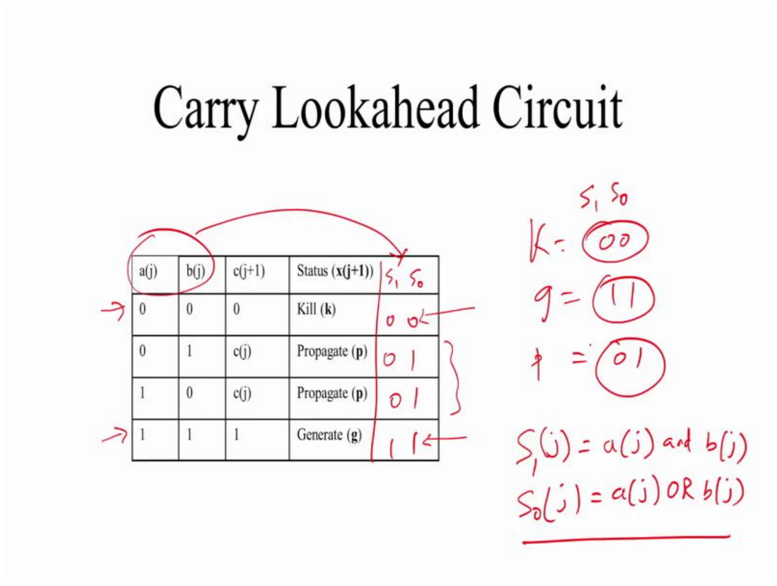
If I am going to compute the carry of the current stage work carry input to the current stage independent of the carry of the previous stages right, then I am going to be in a better shape right. Now note that that independence come for 50 percent of the cases right. If my a j is equal to b j the carry output is not dependent of the carry input right, only when my a j becomes So, for the next stage I know what the carry is without even computing the carry of the previous stages. Are you able to follow? The carry output for me is not dependent upon the carry input of my carry input, or I need to bother about the previous stages to my right if you keep adding from say right to left I need not bother about the carry inputs of the previous stage or the carries of the previous stages for me to generate a carry for the next stage.

I I can just look at my a j and b j and if they are equal then I give a damn for the carries onto my right, but if a j is not equal to b j then I am dependent on that carry. So, immediately just by looking at the truth table, please understand that I have got a 50 percent relief, from the carry of the previous stages. Are you able to follow? Any doubts in this right. So, this is one very interesting I use I used to tell this commentate obviousness is the enemy of correctness. So, it is a very obvious thing, but that has missed for say at least one decades or 2 decades before carry look ahead adder came into place, but this is something very interesting to see in this truth table fine.

So, the objective of doing the carry look ahead adder is to reduce the dependency of a carry to be generated in a particular stage or carry or this sum bit to be generated, to reduce the dependency of that on the carries of the previous stages. And one relief I get just by looking at this truth table is for 50 percent, of the cases the carry that I generate in a particular stage is independent of the carries that come from the previous stage are you able to follow and the remaining 50 percent I need to dependent upon this fine.

Now, please note this down. Please are these truth table with you, please right down this truth table in your note book. And also this statement this if then condition.

(Refer Slide Time: 22:45)



Carry Lookahead Circuit

a(j)	b(j)	c(j+1)	Status (**x(j+1)**)	s_1 s_0
0	0	0	Kill (**k**)	0 0
0	1	c(j)	Propagate (**p**)	0 1
1	0	c(j)	Propagate (**p**)	0 1
1	1	1	Generate (**g**)	1 1

s_1 s_0

$k = 00$

$g = 11$

$p = 01$

$S_1(j) = a(j)$ and $b(j)$

$S_0(j) = a(j)$ OR $b(j)$

So,. So, let us do this simple table here I my a j equal to b j, then I have 2 cases. Case 1 and case 4, and a j equal to b j and a j is 0, b j is 0 then what would be the carry. So, it will be 0 for the next stage. So, a j, b j comes in this stage right. What will be my carry for the next stage it is 0 right; that means, I call it as kill I kill the carry then I do not allow the carry to go if one goes then it says I am propagate. So, I say I kill. When a j equal to b j irrespective of the carry of my stages to the right the carry I generate for the next stage is 0.

So, I call that status as kill right. If my a j equal to b j, then I do 1 the carry that I propagate to the next stage is 1. So, I generate a carry for the next stage when my a j is not equal to b j I just take the carry from the previous stage that comes from the previous stage and pass it onto the next stage. I call these 2 cases as propagate right. So, the moment I get a js and b js immediately, I will know what is the status of the carry that I am going to pass onto my each stage, will know what is the status of the carry that it is going to pass onto the next stage right.

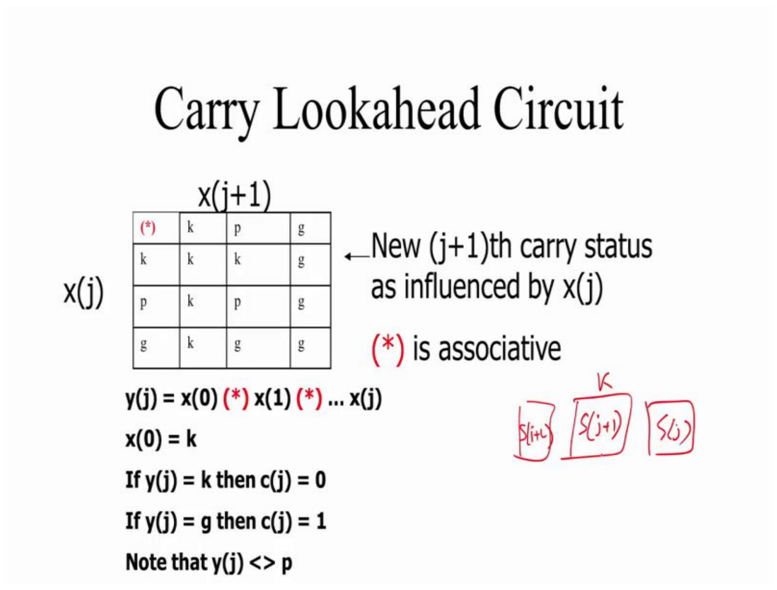
The moment I get my inputs, that n bit is right a 0 to a n minus 1 and b 0 to b n minus 1, each stage can immediately understand right, what would be the status of the carry that it will pass onto the next stage right. If it sees both are 0 then the status is kill it is going to give a 0 as a carry to the next stage. If the both the inputs are 1 then it is going to give a 1 to the next stage. So, I call it as generate if the both the inputs are not equal then it is going to take the carry from the previous stage, and give it to the next stage I know that it is going to be a propagate. So, the moment these 2 n bit is arrive to the circuit immediately every stage can determine the status of the carry that it is going to pass onto the next stage able to able to appreciate this point.

So, just note down this particular table. So, 0 0 means that status, status of what status of the carry that I am going to pass onto the next stage right. It can be kill it can be it can be generate or it can be propagate right. So, what is a simple circuit that can realise this? So, let us say kill I can encode it as 0 0 because I have 3 different status. So, I at least I need 2 bit is to encode it. Kill will be 0, 0 generate let it be 1 1 and let us say propagate is 0 1. So, these are the status bit s 1 is 0.

So, let us draw that here. So, this will be s 1 is 0 0 0 0 1 0 1 1 1. So, a j b j and this is just status please note that s 1 will be. So, s 1 of j will be a j and b j correct and s 0 of j will be a j or correct. Where this is the encoding kill means 0 0 generate means 1 1 propagate means 0 1. I need 2 bit is to encode this status because I have 3 status correct. So, the moment the a’s and b’s comes the n, n which come immediately I can go and generate the status of the carries that I want to pass on to the next stage in one stroke.

So, I need for I need n nand gates and n or gates which I will provide and in one shot immediately it will give me the status of the carries. So, this is my 2 bit output which will essentially tell me, whether it is kill generate or propagate and that just takes 1 1 gate delay right as you see here. So, fine now this is very interesting.

(Refer Slide Time: 27:42)



Carry Lookahead Circuit

x(j+1)

x(j) \ (*)	k	p	g
k	k	k	g
p	k	p	g
g	k	g	g

← New (j+1)th carry status as influenced by x(j)

(*) is associative

y(j) = x(0) (*) x(1) (*) … x(j)

x(0) = k

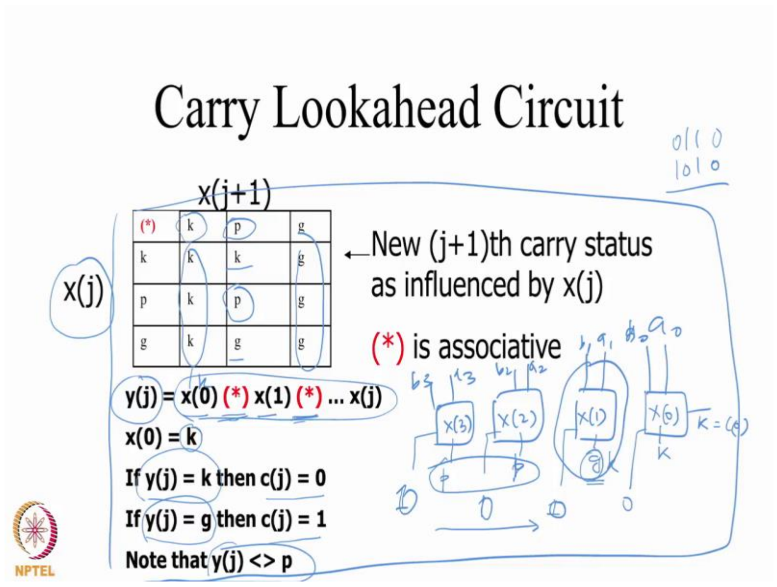
If y(j) = k then c(j) = 0

If y(j) = g then c(j) = 1

Note that y(j) <> p

Now this is stage j, this is stage j plus 1. This is stage j plus 2. Let us say. So, if stage j plus 1 is kill what will be the carry. If the stage, so let us see this what will be the carry of. So, let us draw this entire thing.

(Refer Slide Time: 28:36)



Carry Lookahead Circuit

x(j+1)

x(j) \ (*)	k	p	g
k	k	k	g
p	k	p	g
g	k	g	g

← New (j+1)th carry status as influenced by x(j)

(*) is associative

y(j) = x(0) (*) x(1) (*) … x(j)

x(0) = k

If y(j) = k then c(j) = 0

If y(j) = g then c(j) = 1

Note that y(j) <> p

0110
1010
b3 a3
b2 a2
b1 a1
b0 a0
x(3)
x(2)
x(1)
x(0)
K = c(0)
k
p
p
g
0
0
0
0
NPTEL

So, let us draw say 4 stages. Now the a’s and b’s have come here a 0 b 0 a 1 b 1 a 2 b 2 a 3 b 3 right. So, there is always 0. So, what you get here is kill right. And you generate this is what we call it as c 0. Now there is a status that we have generated for each. So, I have a status x 0 x one x 2 and x 3. Right I have x 0 x 1 x 2 and x 3. Now suppose let us say this is 0 0. So, let us take the 4 bit is here give me some random 4 bit is give me some random 4 bit is.

Student: 0 1 1 0.

0 1.

Student: 1 0.

And another.

Student: 1 0 1 1.

Right. So, a 0 and b 0 are 0. So, what is the status of this this is kill, this is kill and what is the status of this this is generate and this is propagate and this is what.

Student: Propagate.

This is also propagate fine now if I do. So, what is the carry here since this is kill this carry is going to be 0. So, this since this is kill this carry is going to be 0. Since this is generate the carry is going to be 1. Since this is propagate this carry is going to be 1. And since this is propagate the carry is going to be 1. So, now I can go and say that this this is 1 by looking at this g. So, from every propagate I can go to the last, but if this was kill say for example, if this is kill then this would have been 0 this would have been 0 right.

So, at any point when any of the stage is kill from there it start sending 0 if any of the point is g then it starts sending one right. So, I have a sequence of propagates like this for me to actually find the carry I need not go till the end, but I need to just go to the point where I see a generate or the first thing that I encounter. So, from this propagate I go back you know towards my right hand side and if I find the first one is a generate then I know all these are ones are you able to follow. So, so that looking back operation is what we call as this operator called star within brackets.

So, now, we will see this if my stage j plus 1 as kill right, then irrespective of what is happening in my right hand side I am going to have kill completely right. If my j plus 1 is kill that is any stage is kill, then I do not allow anything to propagate from the previous set. So whatever be my status of x j this operator will just kill. So, all that is going to propagate from this point is going to be zeros right. If my j plus 1 is generate right, if my j plus 1 is generate all the I do not care about what is happening to my right all that I am going to propagate from this end is going to be g, but my j plus 1 is propagate please note that and my previous stage was kill, then I know that it is going to be kill if my previous stage was generate I know it is generate, but if my previous stage was propagate then I go and propagate right you understand this. So just I will repeat this if my if my the status of my j plus 1th stage is kill irrespective of what is going to be on my right hand side I am going to just say it is going to be kill right. So, I am independent of this.

If my previous stage was generate if my current stage is generate then I do not care on the right hand side I am going to push generate, but if my previous stage was kill and I am propagate, then I make myself as kill if the previous stage was generate. I make myself as generate. If my previous stage was propagate then I remain to be in a propagate stage right. So, now, what I do is, I get the status of every carry that I contribute to the next stage, just by seeing this as and bs. Then what I do is I do a prefix computation on these carries on the status right.

So, what I do is y j for every stage j I compute y j is equal to this x operator x 0 operator x 1 operate x 2 till x j right. So if I do this operation and if I get y j is equal to k, then I know that the carry for my stage is 0. If I get y j is equal to g then my carry is going to be 1, and note that your I y j after this computation will never be propagate. Why because we know that x 0 is kill. So, this is always kill. So, the kill with anything else when I go will land up with kill unless I see a g there. So, at the end your y j will never be a propagate.

So, I just stop with this particular slide today. I want you to think about this right that I get the carry status namely x js and from the carry status I want to go and find out what would be the carry users. And for that I go and do this associative operation right, I go and so this operation and later. I will show that it is going to be associative right. So, I will continue with this tomorrows in tomorrows class, where I just want you to think about this particular slide kindly note it down and think about this slide. So, that you know we can continue from this slide tomorrow.