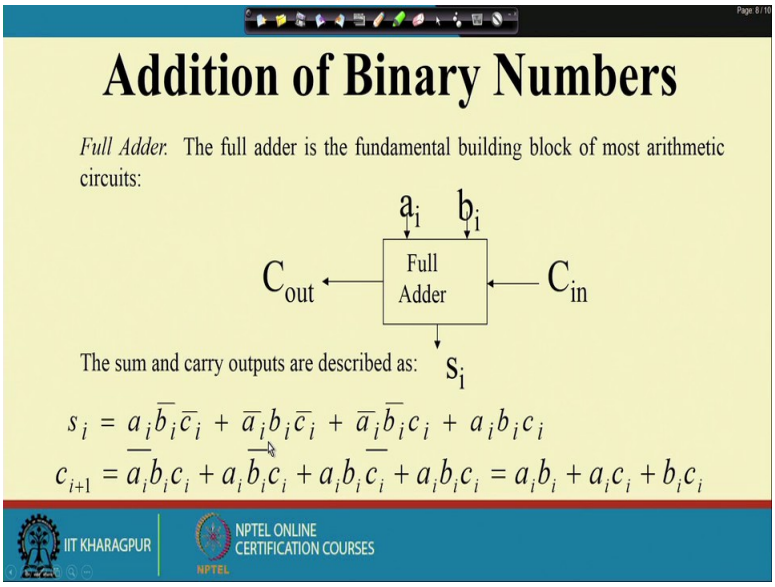
Architectural Design of Digital Integrated Circuits
Prof. Indranil Hatai
School of VLSI Technology
Indian Institute of Engineering Science and Technology, Shibpur, Howrah

Lecture – 16
Efficient Adder Architecture (Contd.)

Welcome back, to the course on Architectural Design of ICs. So, here we are seeing different adder architecture. So, we have started with the that means, basic addition operation which are this full adder and half adder. So, after that we will just continue with this.

(Refer Slide Time: 00:33)



So, this is the addition of binary numbers, here you see if I am having this a i b i C in and this is the full adder circuit; that means, which will produce this C out and sum output, ok. So, that we have already discussed how it works what is the; that means, working principle of this particular full adder cell. So, whenever this working principle of full adder cell we have seen. So, at the time from that particular truth table; that means, or this operation table of this full adder cell.

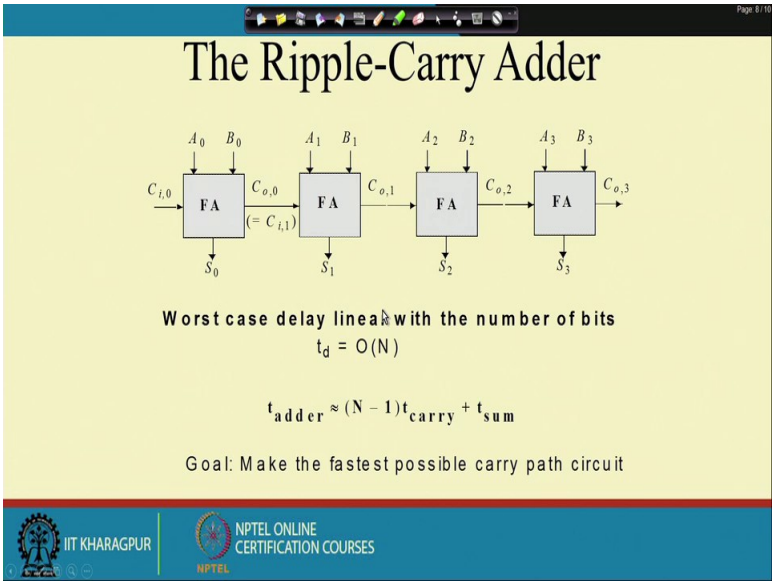
We have come out to this logical expression for sum and this logical expression for carry out, ok. So, sum for sum that is nothing, but a XOR b XOR c i; that means, if I just simplify this logical expressions at the time I will get nothing, but a XOR b XOR c i a i XOR b i XOR c i and the c i c i plus 1 or that is the C out. So, that is nothing, but this a i

b i plus a i c i plus b i c i ok. So, that means, ab plus bc plus ca nothing, but that I will get.

So, after that that means, I need that means, there is this ripple carry adder that architecture comes. So, why I need this ripple carry adder? Ripple carry adder why I need whenever we are considering n bit numbers, ok. So, n bit numbers means whenever I am the having; that means, here we are doing only single bit addition operation, right, so, we are having multiple bit operation as you might be knowing that I will; that means, present days computer that has become 64 bit, ok.

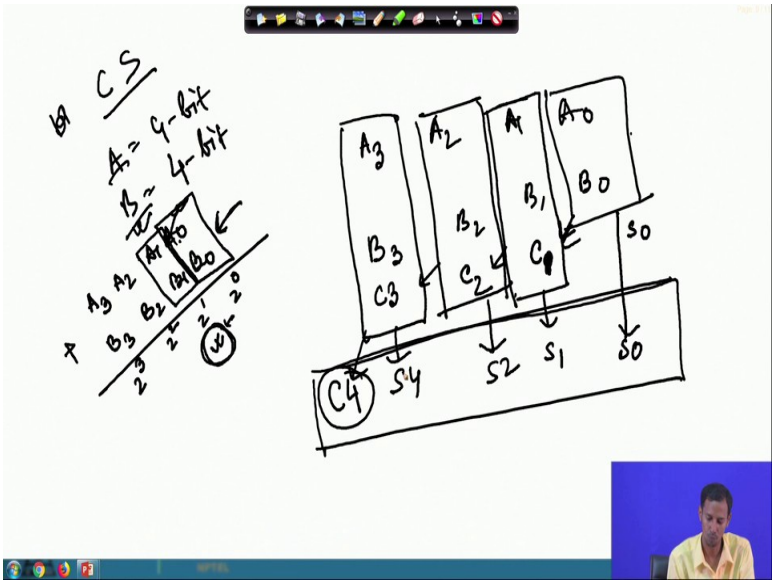
Some of the cryptographic core that is 128 bit long, so, that means, the addition operation and; obviously, for this cryptographic algorithm or for any algorithm which is implemented on this particular computer architecture. So, at the time I need to support this addition operation of 64 bit or 128 bit, so; that means, I need to design one addition or adder operator or adder circuit which can work with the 64 bit as the input or as the word length of the inputs. So, whenever we are considering this 64 bit addition operation then at the time how we can design them that we will see.

(Refer Slide Time: 03:16)



So, this ripple carry adder says that the carry is basically rippling. So, why carry is a rippling? Ok. So, if I consider suppose this is the 4-bit; that means, ripple carry adder or I need to design one 4-bit adder. So, 4-bit adder means I am having this is A and B of each of 4-bit.

(Refer Slide Time: 03:46)



So, each of 4-bit means for if I consider A of 4-bit and B of again 4-bit right. So, 4-bit means that is A 3 A 2 A 1 and A 0 now, B 3 B 2 B 1 and B 0. I will have to add these two numbers ok. So, now, whenever if I just add this two particular bits. So, if I am considering only these two LSB bit positions so, at that time what will happen this is nothing, but a full adder, right. So, nothing, but a full adder cell means there is no other input which are effecting basically this particular logic. So, that is why there is A 0 and B 0. So, what I said that initially that is this is binary weighted values. So, that means, this is 2 to the power this is 2 to the power 0, this is 2 to the power 1, this is 2 to the power 2 and this is 2 to the power 3.

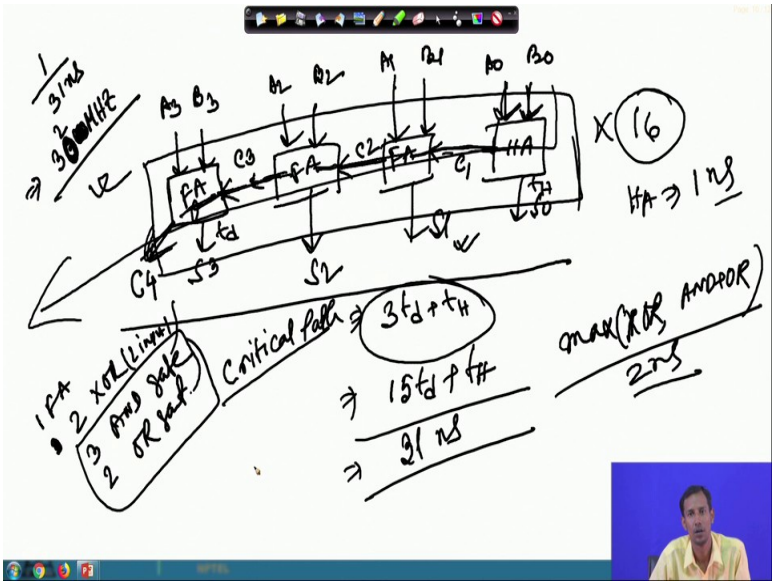
So, whenever I will add this two particular bit position at the time the carry out which is generated for this two addition of this two bit that will come to this 2 to the power 1, ok. So, that means, now for this two particular case or whenever I consider this, but bit position so, at the time it is not that only A 1 and B 1 they are basically evolved or involved for specifying this, the sum value at this particular point. Why? Because these carry which is generated from this the previous level that will also effect the corresponding sum and carry values.

So, that is why at that time it will not be one half addition or half adder operation. So, at the time what it will be? It will be like A 3 A 2 A one A 0 and if I just add B 3 B 2 B 1 and B 0. So, at the time for this two this is half addition operation and for these there will

be one if I consider this is sum 0 and this is the C 0 or if I just say this as a C 1 let us considering named as C 1.

So, that means, this is the carry which is generated from the this A 0 and B 0. So, now, I can add this particular things and again I will get what 1 and 1; that means, 1 and 0 1 means the carry and sum so; that means, from this particular block this will be S 1 and this will produce the carry as C 2, then again for this do the same; that means, it will produce the corresponding S 2 and this will produce the corresponding C 3 then for here S 4 and this is the final carry out. So, that means, if I just want to add this A and B are of 4 bit. So, at the time I will get the output sum as 4 at the sum and one for the carry output bit.

(Refer Slide Time: 08:07)



So, here the point is that now if I just draw this particular circuit for this addition operation then what I need? If I just draw the block diagram of this. So, at that time how it will look like this is A 0 and B 0 this is one half adder then the carry C 1 is basically comes over here and A 1 B 1, this is S 0 this is S 1 then this will produce the C 2 where this will added with A 2 and B 2 then this will produce S 2 and this C 3 which is nothing but it will add with A 3 B 3 it will produce S 3 and it will produce a C 4. These are the 4 adder cells.

So, now, if I just; that means, if I just what I said if this is my total this 4-bit adder circuit so, at the time what will be the operating frequency of this particular circuit? So, to find

out the operating frequency what I need that I need to calculate the corresponding critical path. So, here what is the critical path? The critical path basically where from it is starting from any of these two LSB bit position then that basically logic is transferring from this to this to this to this and up to here this is basically this carry is basically traversing. So, that is the longest path in this particular circuit.

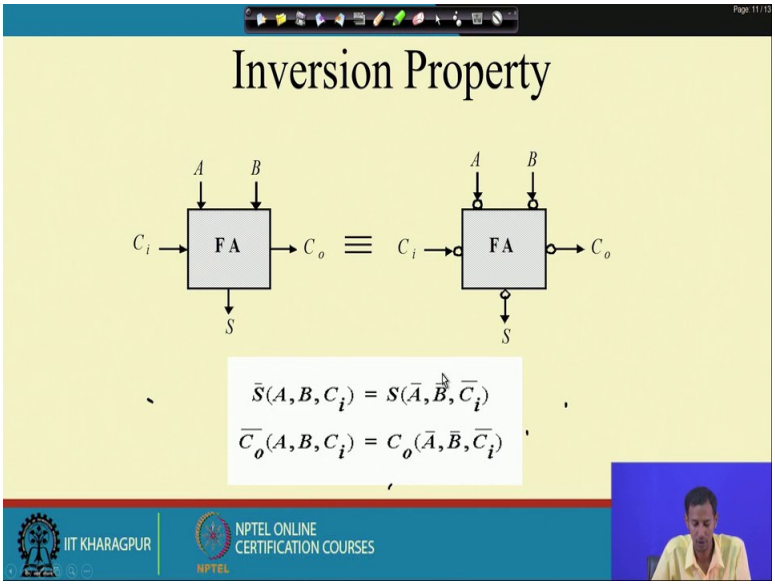
So, the longest path means that is the longest path means which is passes through several maximum number of gates in the corresponding circuit. So, that is the critical path. So, now, this critical path, if I consider this delay for this is for full adder is t d and for this is t H. So, then the critical path is 3 of t d plus t of H, ok. So, if I increase this number as 32 or 16 or 64 so, at the time this will also increase. So, if I just do it if I just for 16 if I just need I make it so, at the time what it will be for 15 t d plus t H, ok. So, one full adder cell one full adder cell what it consist it consist of sorry 2 XOR gate 2 input and 1, 2, 3, 3 AND gate and 2 OR gate for implementing this for implementing the carry logic, ok.

So, now; that means, for whenever we are doing this so, at the time the max of XOR gate or this AND plus OR gate delay that will be the critical path for or the sorry that will be the delay for one particular full adder cell. So, if that is if I consider that as an 2 nanosecond and for half adder for half adder the delay if I consider as 1 nanosecond so, at the time this will be of 31 nanosecond. So, that means, the operating frequency of this particular circuit that is 1 by 31 nanosecond that is near about I think 330 megahertz not 330, I think 30. Let us considering 30 or 32 or something like this 32 or 30 megahertz, ok.

So, this if I increase this number on the more; that means, at the time the rippling time other this, the carry rippling time from LSB to the MSB that becomes more. So, that is the disadvantage of this particular circuit; that means, whenever we are considering for more number of bits at the time you can get a larger number of delay. So, for that reason if I if you just see here; that means, this is these are the that means, four full adder cell and the worst case delay linear with the number of bits. What I said, that means, if I increase the number of bits instead of 4-bit if I consider 8 bit so, the corresponding delay worst case delay that will be increased linearly, if I increase the number of bits the delay will also increase. So, that we have all ready seen.

And, then after seeing this particular problem or seeing this challenge what will be my target? My target will be how I can make this addition operation as fast as possible and what is the this carry path is the; that means, critical path. So, I how can I break this particular critical path, so that I can improve the operating frequency of this particular circuit, ok. So, that is our intension.

(Refer Slide Time: 14:55)

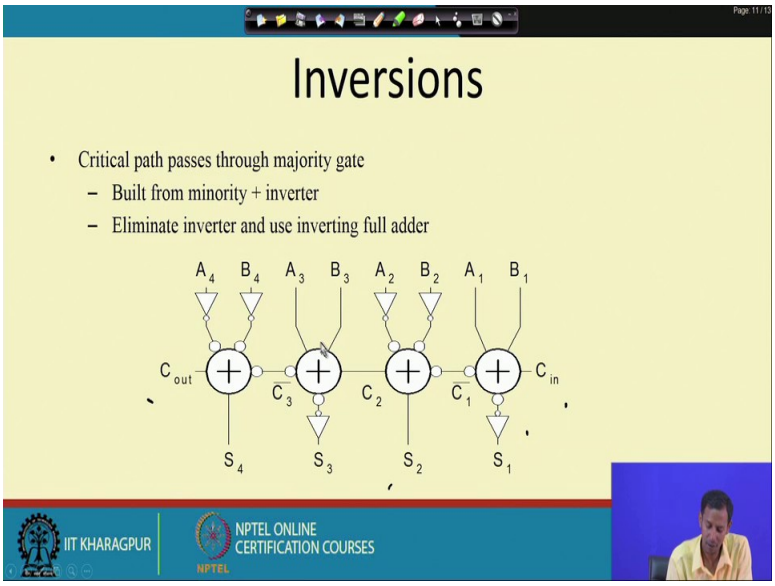


So, actually for that what people are; that means, done the inversion property. Why I need the invention property? See if you see that that for implementing this A XOR B XOR C. So, for implementation of this I need the inverter circuit at the output of that. So, it is not that it is that means, it is not it will not give you the corresponding that expressions for AB plus BC plus CA.

So, you can get it and then if you just see; that means, if you just see that transistor; that means, NAND gate level implementation so, at that time you have to; that means, put one inverter at the output as it is not get coming as here I need that is or the operation is coming as NAND operation. So, I need AND operation. So, I need one inverter at the output final output for making when NAND to the AND, ok.

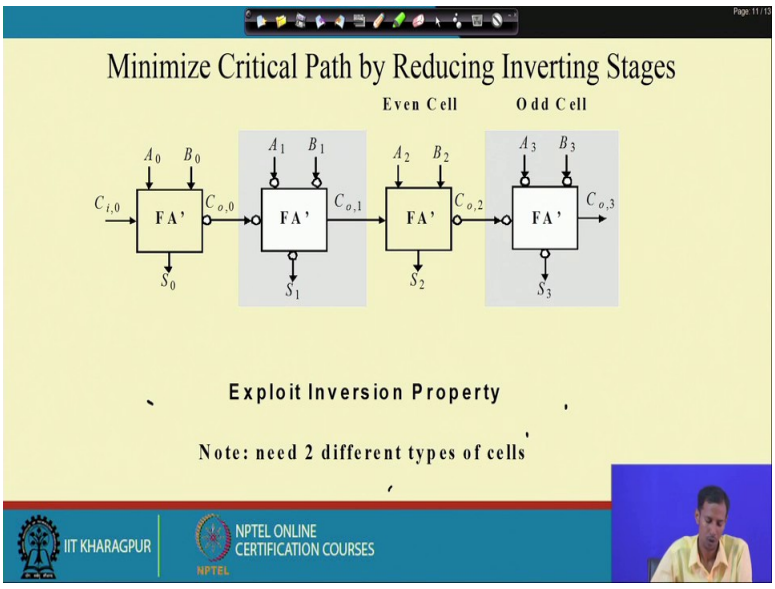
So, for that reason to reduce the number of transistors so, what people have done a from the beginning they have put that instead of A, B and C they have taken; that means, this corresponding values invertedly so; that means, the complement of A, B and C that are being used for in this particular full adder cell, ok.

(Refer Slide Time: 16:53)



So, and here you see that this is basically the that means, addition operation or implementation or using this transistors where I am doing in initially we have just inverted and then we have just put this into the this corresponding full adder cell of for this sum and that means, carry out that circuit, ok.

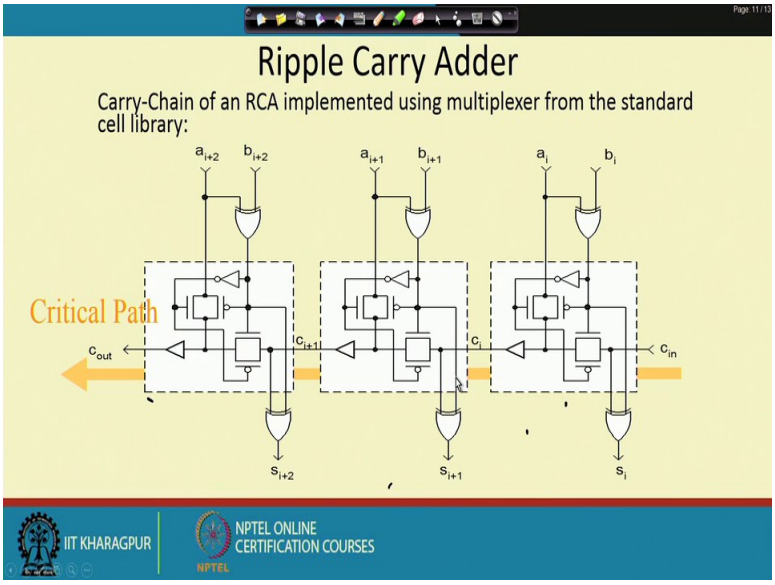
(Refer Slide Time: 17:40)



But, here you see only for this inversion is done on the corresponding this even position. For odd position, for odd position it is not done anything ok. So, by doing this kind of architecture you can reduce the delay by inverting the corresponding ah; that means, this

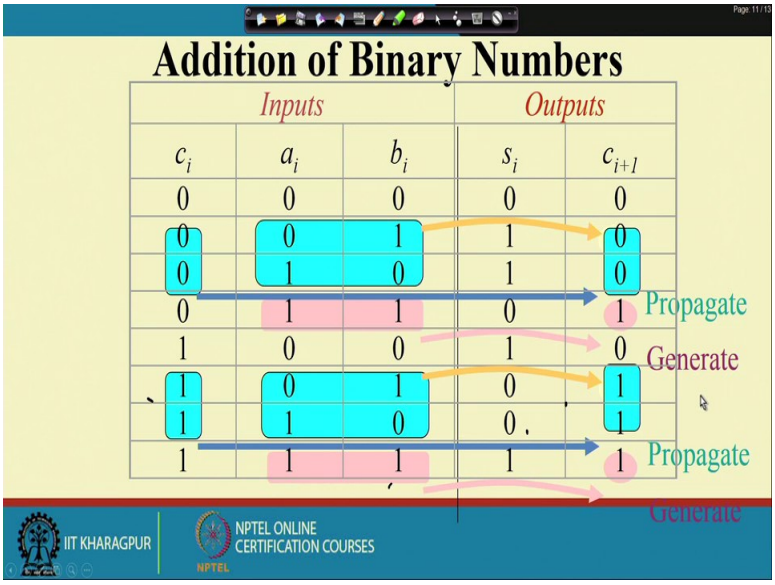
the even stage I can reduce the critical path or I can reduce the or; that means, this critical path that can be reduced though it follows the same which is nothing, but the carry ripple adder.

(Refer Slide Time: 18:30)



So, and this is the; that means, this transistor level implementation of this and here you see the critical path is this basically which is following or which is passes through each of this bit position or each of this full adder or half adder cell. So, this is the that means, this ripple carry adder implementation using the standard cell library.

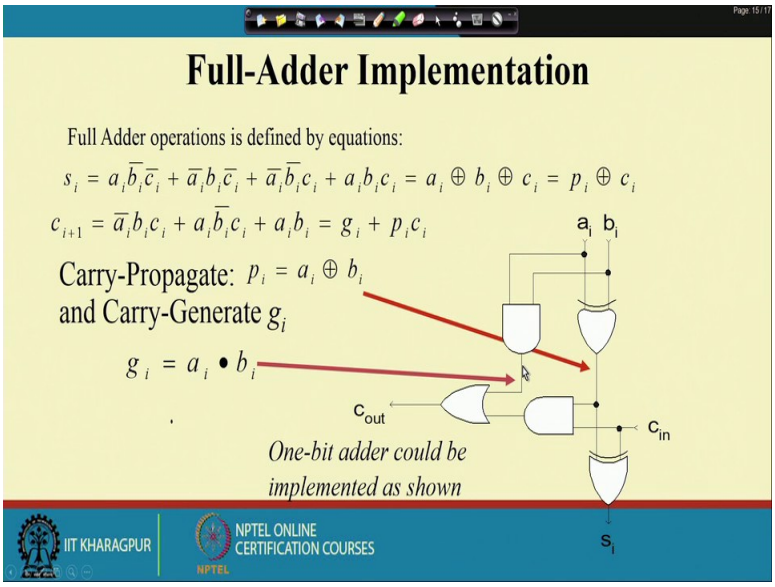
(Refer Slide Time: 19:03)



So, now again come back to the corresponding operation table of the truth table there is sorry the full adder. So, here you see whenever this for this 0 1 and 1 0 of input a and b whenever this carry bit is 0 ok. So, at the time this carry output bit is 0 only the sum is 1, but whenever this a and b they both of them are 1 1 and carry is 0, so at that time the carry out is 1.

And the same thing for this last one if you see for 1 1 if the carry is input is 1, so, at the time sum is 1, but the carry is already 1. So that means, now for this one and for 1 1 of a and b it does not matter whether the carry input bit is 0 or 1 it the carry output bit will be always set to 1 for carry in bit is 0 or 1 only it is effecting what this sum value whether that is 0 or 1, ok. So, here; that means, this is this I can say the carry is basically propagating, sorry so, so the carry is being generated ok.

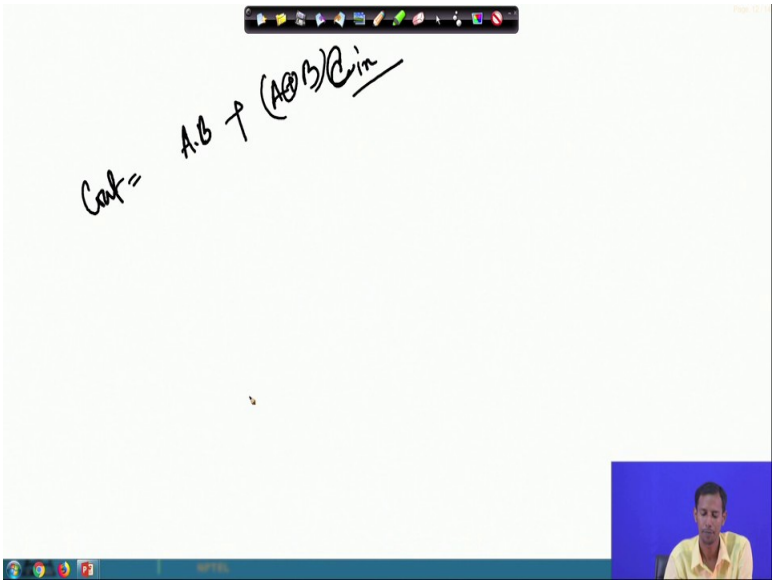
(Refer Slide Time: 20:52)



So, then how this carry is being propagated if you just go back to the previous slide whenever this 0 1 and 1 0 whether that carry bit input bit is 0. So, at the time this carry output bit is 0. Whenever this 0 1 and 1 0 if carry input bit is 1. So, at the time carry output bit is sorry carry output bit is 1 so; that means, here the carry is being basically for this particular case the carry is being propagated. So, propagated means based on this if this particular combination happen so, at the time based on this carry input bit the carry output bit will be selected whether that is 0 or that is 1.

So, then from this if I just write this as carry propagated if you just go back for 1 1 sorry for 1 1 and for 1 1 over here this is a. So, carry for carry generation the logic is that that is a and b a i into b i. For propagation what is that for 0 1 and 1 0 what is that that is the; that means, the logic is that a XOR b and that is ANDed with 0, that means, for both the cases the sum is 1. So, now, if some that carry is 0 then carry in is 0. So, that is 0 if carry in is 1. So, at the time carry out is becoming 1 so, that means, that logic for propagation logic I need; that means, for sum that is which is 1, sorry, for carry propagation the logic is that a i XOR with b i, ok.

(Refer Slide Time: 23:38)



So, now, for carry output bit I said that the one logic for carry output is AB plus BC plus CA, the another logic for this carry output bit is this the another logic for this is that is A B plus A XOR B into C or C in. So, if I just come to this particular that particular logic so, at the time if I consider this carry for carry generation I am considering this a i and b i. For carry propagation p i equals to a i XOR b i. So, now, I can write this C out as the function of g i plus p i into c i, where c i is the input.

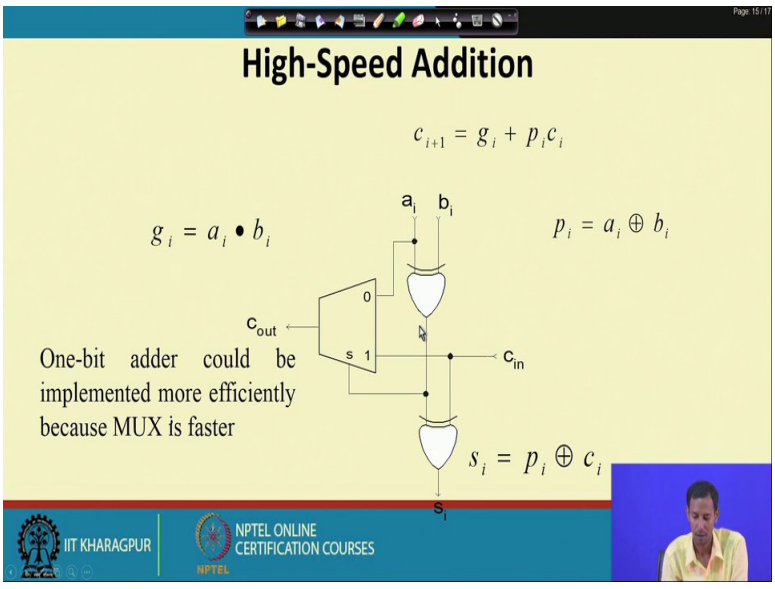
So, now, this is the; that means, corresponding logic for or the corresponding gate level implementation of this particular logic or this particular function ok. So, here you see that this particular is for this propagation and for generation what I need the corresponding AND gate ok. So, after that this is the that means, this p i into c i is done

and then that is or width is g i and this is the final p i plus c i for doing the sum and this is for the C out, ok.

So, what is the problem in ripple carry adder is that I have to wait if I just go back to the slide if I just go back to the slide. So, at the time if you see here at this particular point at this particular point unless and until this C 0 is basically computed, I cannot start the operation of this A 1 B 1 then again this unless and until this carry is being generated here I cannot start the operation of this particular full adder cell. Again for this again I have I cannot start that this for; that means, that is why I have to wait for this carry has to be available at this particular time ok. So, the and that is basically the bottleneck in ripple carry adder as the carry is rippling or the carry is basically passes through this all this particular phases or stages.

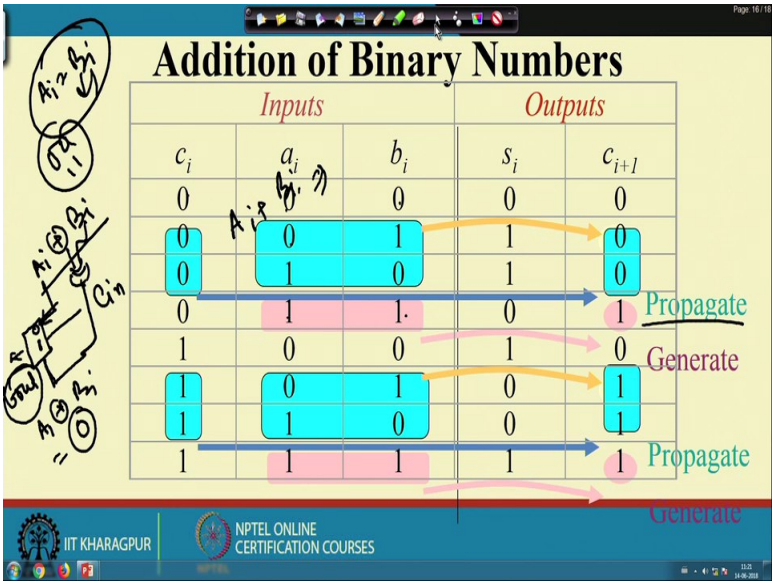
So, unless and until this computation is over it cannot start the next computation ok. So, that means, it is the serial operation, but whenever we are doing this whenever we are doing this; that means, we are generating propagating the carry that is means first and then generating the carry too and then I can; that means, in each of the stages I am basically producing the carry generation and carry propagation parallelly, ok. So, how we can; that means, this is for one; that means, single bit operation. So, then again for here actually what I am doing I am just propagating and using this I am just generating.

(Refer Slide Time: 28:14)



So, instead of that if I just use one mux over here for calculating the c out why c out is calculated. So, you see c i plus 1 that is equals to g i plus pc i. So, g i is the carry generated p i is the carry propagated, right. So, for this particular case it is producing the sum. At this particular point if you just see what is the logic for mux here you see this a i b i that is the select line and then this two input is one of the input is a i another input is c in ok; So, now if you just implement this logic; that means, whenever this if you just if I just go back to this.

(Refer Slide Time: 30:04)



So, what it says whenever this is basically 0, that means, this is the this A i sorry. If I consider this A i plus B i so, if I just here at this if this particular point this is what A i XOR with B i correct for this propagation, but if you just see depending on the carry input bit I am getting the carry output bit, how? What is the logic? If my carry input bit is; that means, if this is the; that means, this is this is fixed right for this two particular case wherever this carry input bit is 0 0 sorry 0 0 so, at the time carry output bit is 0 0 whenever this bit is 1 1. So, at the time carry output bit is 1 1, ok.

So, that means, now if I just consider this; that means, this is the c in which is at 1 and this is at 0 so; that means, for XOR of this a i and b i for 0 1 case for 0 1 case what is happening? For 0 1 case or 1 0 case in both the case this particular circuit is giving me 1. So, that means, as c in is the as same as the c out, so, at that time I can get the results at

the output of c out. What will happen for this 1 and 1? For 1 and 1 and for 0 and 0 both are 0, right.

So, at the time if I see that for 0 0 this case and for 0 0 this case the c i is changing from 0 1, but here does it basically happening as 0 1? No. For 0 and 0 for and for 1 and 1 this A i XOR B i that will be 0 in both the case. So, at the time I need to select this 0 line and at the time what is the output of c i that is 0 at this particular point and one here for this and same for 0 0 for 1 1 for 1. So that means, that means any of the input bit whether that is a i or b i that will be connected to because at the time what is happening a i equals to bi for 0 0 and 1 1 case.

So, any of the input bit that will be connected at this 0-th position and that that will be the c out at that particular point. So, the using if I just so, using this mux now I can implement the same function. So, where this is the select line input and this is the a i and b i. So, this is this you used for high speed addition operation. So, high speed addition operation means I can improve or I can; that means, from the beginning what will be the values of carry input bit sorry the carry output if I can predict and I can select the corresponding value. So, by that I can reduce the computation time of carry whenever we are considering a long chain of addition operation ok.

So, thank you for today's class again we will discuss it on the new classes.

Thank you.