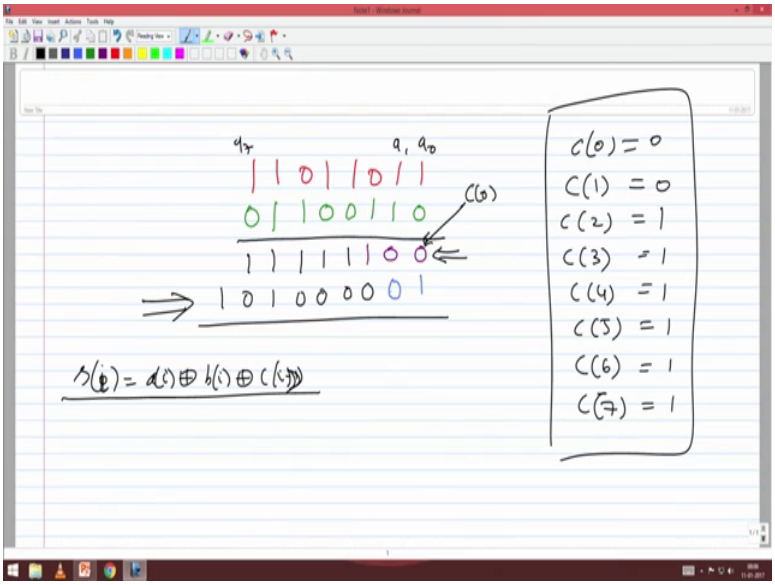
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 04
Fast Adder Circuits (Contd)

So, we will start with where we left yesterday.

(Refer Slide Time: 00:27)

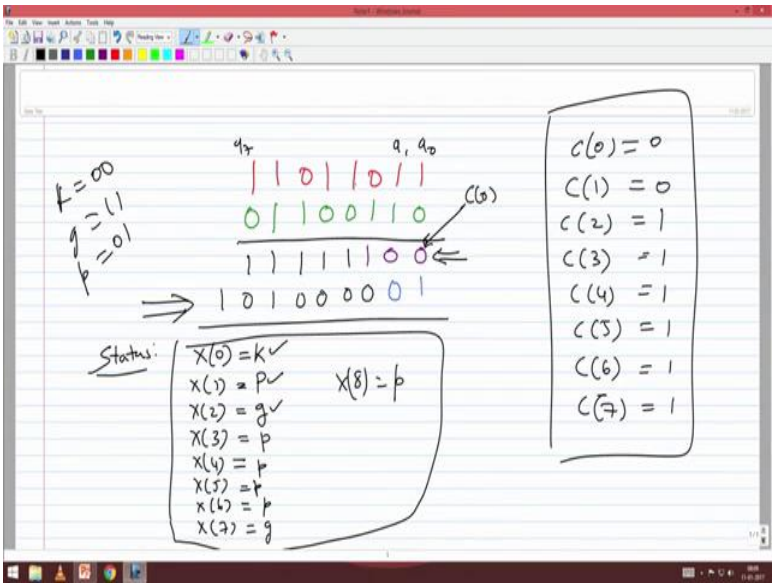


So, suppose I want to add. So, the sum bit here is 1, I am writing that carries the carry that came from the previous stage is 0. The sum bits 1 and the carry that goes to the next stage is 0. So, here again the sum bits 0 and the carry that went to the previous next stage is 1, here again sum bits 0 the carry is 1, sum bits 0 carry is 1 sum bits 0 carry is 1 sum bits 0. So, this is the answer correct if I add 1 1 0 1 1 0 1 1 with 0 1 1 0 0 1 1 0. I get this answer and these are all the carry bits. This is the carry in bit; this is a carry in bit that came from the previous stage.

So, let us write the carry bits. So, 0 1 2 3 4 5 6 7, c 0, c 1, c 2, c 3, c 4, c 5, c 6, c 7. This is 0 this is 0 this is 1 the remaining all are ones. So, this is the carries. If I know the carry at the earliest immediately I can compute the sum bit. Because your s 0 is nothing, but sorry s i nothing, but a i exclusive or bi exclusive or c i minus 1 correct or c i in this because in our terminology we say this is 0, this is c 0 and this is a naught a one and so on till a 7 and this is b naught b 1 to b 7. So, if I know c i I can get s i immediately. So,

how quick we calculate c i is the important problem. If I am going to calculate c i very fast, then I will be getting sum bits. So, what happens here is first let us see 1 0 means what is this status of. So, we first calculate the status of the carry carries.

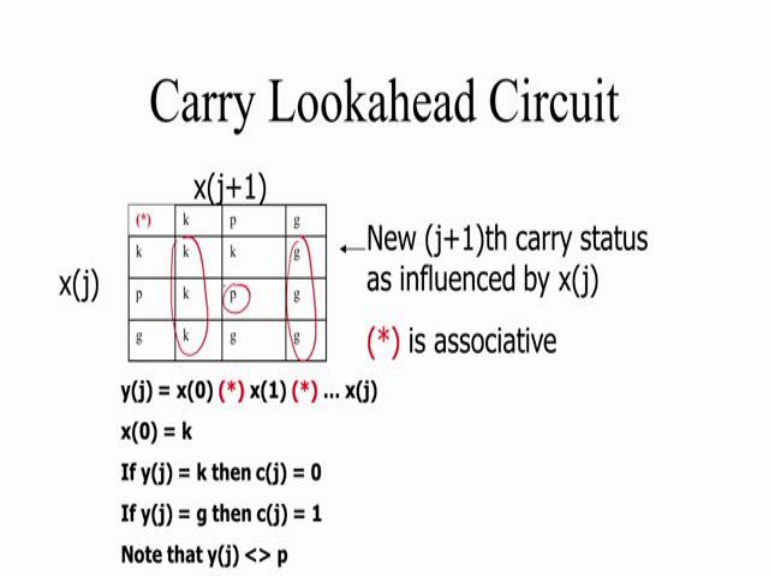
(Refer Slide Time: 04:06)



It is called the x vector. So, we will calculate x 0 x 1 x 2 x 3 x 4 x 5 x 6 x 7. What is x 0? X 0 is it is starts with k right. This is the k because c 0 is 0. So, what is x 1, x 1 is propagate right 1 and 0. What is x 2 this is generate. X 3 is propagates, x 4 is propagate. So, x 1 x 2 x 3 x 4 x 5 is also propagate. X 6 is also x 6 is generate. X 7 is propagate - 1 2. So, kill propagate generate propagate, propagate, propagate, propagate. So, kill next is propagate next is generate followed by 1 2 3 4. 4 propagates and x 7 is generate and have x 8 is propagate. So, then what I do right this is clear. So, what is the time required to basically generate this x values. It is it is constant time right. We just used one and or gate for each and will give you. So, kill I call it as 0, 0 generate I call it as 1 1 and the propagate as 0 1 then basically I need to 1 and 1 or gate to generate these 2 bits. So, in unit time, I am able to find out these x values right.

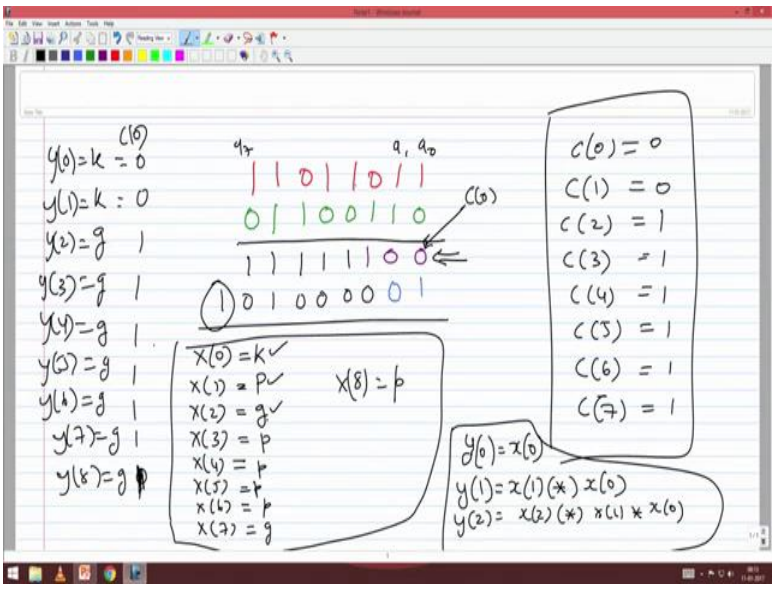
Now, I do what we call as the thing. So, I do this operation let us go back to this operation.

(Refer Slide Time: 06:45)



I do this operation on the x, kill with anything is kill. So, kill with anything is kill, generate with anything is generate. And of course, propagate with propagate is propagate. So, this yesterday we saw this table right. I hope you have copied it we will now go back to that.

(Refer Slide Time: 07:13)



Now, what we do here is let us compute this y values. So, y 0 is x 0 y 1 is equal to x 1 that star x 0, y 2 is nothing, but x 2 star x 1 star x 0 and so on right. So, if you start doing

this, you will find that y 0 is kill, y 1 is propagate with kill right. So, it is a kill right propagates start kill right.

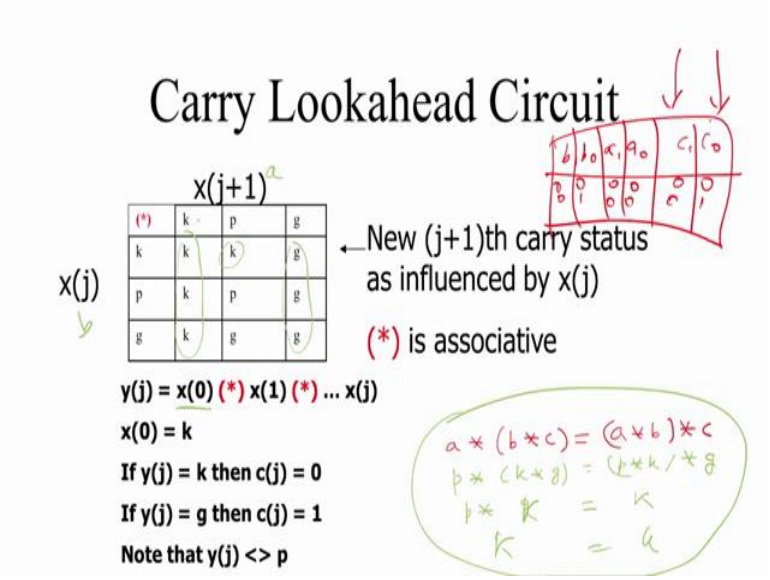
Now, x 2 is y 2 is generate with propagate with kill. Just look at here generate with propagate with kill. So, generate star propagate star kill with be generate right, y 3 will be p with g with p with k. So, it will be g y 4 will be again p with p with g with p with k again it will be g, y 5 will be g, y 6 will be again g y 7 is g y 8 is g correct, are you able to follow is it is it clear and what have you done here if I am seeing a kill my contribution to the next stage is 0. If I see a g my contribution to the next stage is 1 immediately I can decide it is independent of this. If it is a propagate then I have to go back to your right hand side and I first if I encounter a g, then this is 1 if first I encounter a k it is going to be 0 right. This is as simple as this and that is what this computation mimics correct do you understand this.

Now, we see after doing this prefix computation, if I see a kill then it should be 0 the carry is 0. So, c 0 is 0, if I see a kill. So, c 1 is what 0 c 2 is 1 1 1 1 1 1 0. Sorry 1 1 1 1. So, now, you see that there is a 0, 0 followed by 6 ones 1 2 3 4 5 6 and this is the last carry this, ok you are able to, yes.

Student: Sir, is in the calculation why sequential again.

I am going to come to that right, now are you able to follow my procedure you are able to follow the procedure and you are appreciate the correctness of this procedure. Now if I am able to calculate these y values, very fast then immediately I will get that carry values the moment I get the carry value immediately I can generate the sum bits. So, my objective is the moment I get these x values, how will how quickly can I compute these y values. Now what is this computation? This computation of y what sort of computation it is, x 0 x 1 star x 0 x 2 star x 1 star x 0 first class, it is a prefix computation correct. It is a prefix computation right.

(Refer Slide Time: 11:16)



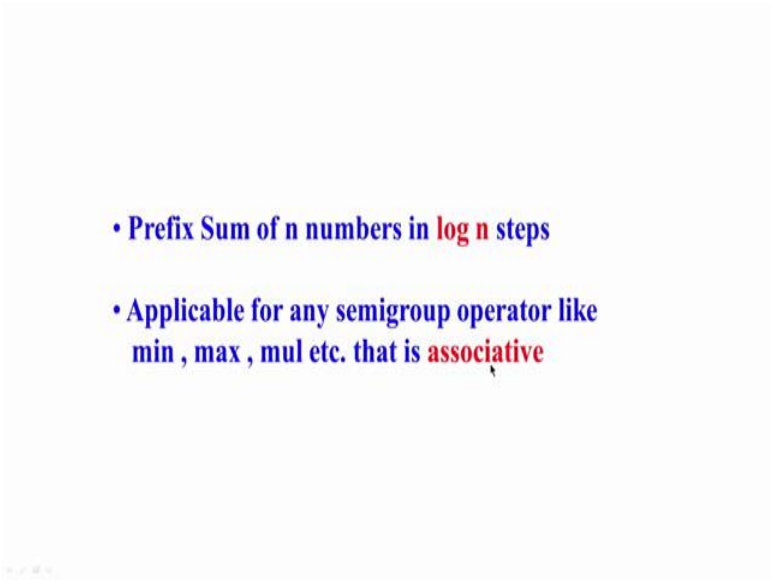
The next thing is please note that this star is an associative operator; it is a semi group operator star is a associative operator semi. For example, if take this thing let us take 3 let us take let us say a star b star c is equal to a star b star c correct.

Let us put arbitrary values for a b and c just tell me some random values p k, k. So, p star k star let us say g, is equal to p star k star g. Now what is p star k p k star g, k star g is g k star g is g p star g is g k, k star oh k star. Now we are doing it x j plus 1 which is anyway what all be. So, k star k star g is k and p star k is.

Student: p star g p star g.

Whatever way you want you want this which should be left operator and right operator. So, a star b means a is here and b is here. Let us assume right a star b means a is here, and b is here. So, if I want to do k star g k is here and j is here. So, k star g is k and if I want to do p star k is going to be k. Now p star k is k p star k is k star k is g, like this how many combinations we will have 27 computations. So, there will be 3 possible values of a b and c. So, 3 into 3 into 3, so 27 combinations we can do and we can verify that this is a associative the moment I want to do a prefix sum on an associative operator for n bits, how fast can I do log n time you have already seen that.

(Refer Slide Time: 14:05)



So, in our previous slide right a prefix computation of n numbers can be done in or n numbers or n elements can be done in log n step provided the operator is associative. Please note that this star operator is associative. And hence we can basically do this entire thing in log n time right. So, the moment I get these 2 bits or the a a and b in constant time I can generate the status namely the x values. And once I have the x values in log n time I can basically generate the y values, the moment I have the y value then I can generate in constant time the carry right. So, please note that the y j will be only either k or g. Each of the ys why because c 0 or whatever x 0 is k and here with anything else will either land with k or g right; the moment I have one k for sure in this.

So, the other y values because I am doing a prefix computation. This x 0 will be involved in all the computations, and the x 0 will surely give you a k. So, surely p will not come it will be either k or intermediately if you see at g, it will be g. So, the ys that you calculate at the end will be either g or k and it cannot be p. That is very important. Here the answer for that is that x 0 is k and x 0 is involved in all your computations right. Please note that x 0 is involved in all your computations. And so the y values will be surely not p as seen from this table. If you see one k, then all the things would be in k right.

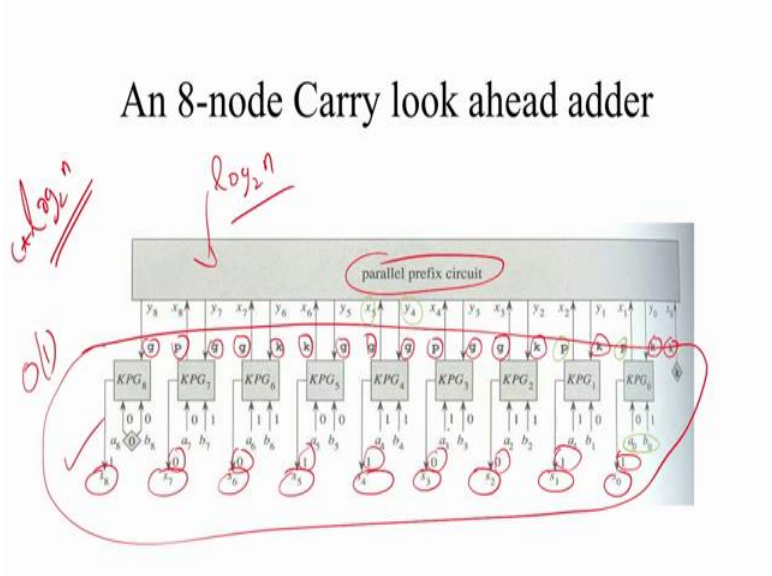
So, unless you see an intermediate g, that we have also seen in the example that we have worked out. So, that is another important point that we need to take right. So, I will give you the circuitry for a carry look head adder in the next slide, but now you understand

how I can do carry look head addition in log n time correct. So, while a carry ripple adder takes n time right.

We will now give you the; we will now look at the circuit for the carry look head adder is it clear or you able to follow right. So, much funda is there in building such adders right. Many of the books say that you can do know the constant the carry will be computed in constant time, and they assume an n input gate, n input gate beyond is not possible right. I cannot have a suppose I have 128 adder bit adder I cannot have a 128 bit gate right. It is practically impossible right. So, this is this is the way we analyze a circuit, this is something different and so I just want to tell that point right. There is lot of. So, you also would be dreaming why did I do this you know discrete mathematics. So, this is where it helps in designing circuit is.

One simple point that I could give you are able to follow this.

(Refer Slide Time: 17:45)



So, this is a computation. So, I do y 0 y 1 y 2 till y n. And it is this star is associative and hence I can do prefix computation, that is what I am trying to tell you in this slide. So, the way we will look at is the first thing when the a s come here a 0 to a 8 right, when the a comes here what we do is that we generate the kill propagate or we first get the kill propagate or generate bit, that takes constant time. And that we give it into the parallel prefix circuit right and the parallel prefix circuit will give you the y values; it will give you the y values. So, I give the x values to the bits come in, I give the x values the

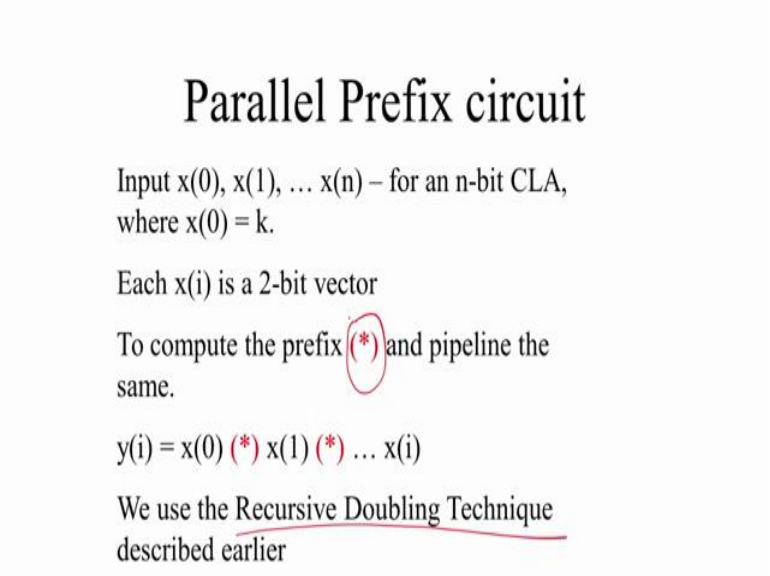
parallel prefix circuit will give you the y values. So, here also there is an example 0 1 means it is propagate, 1 0 is propagate 1 1 is 1 1 is generate, you know 1 0 is propagate 1 1 is generate 0 0 is kill 1 1 is generate 0 1 is propagate right and nothing more here and of course, the first c 0 is kill.

Now, I give it to the circuit. So, kill propagate will give me kill. So, kill will give me kill, kill propagate will give me kill, kill propagate gives me kill, kill generate gives me generate, generate propagate will give me a generate, generate gives me generate kill gives me kill, kill generate gives me generate, generate propagate gives me generate. So, the y values are basically output from this parallel prefix circuit from the y values I immediately get the carry as one and basically generate the sum bits. Immediately I could generate the sum bits. So, in this case I am getting you know kill, kill as the carry right. So, y 0 is kill, kill means the carry is 0 0 0 1 0 XOR 0 XOR 1 is 1, here my carry is kill again. So, 0 XOR 0 XOR 1 is 1. So, this is a sum bits if you see here.

Now, here I am getting kill. So, 0 XOR 1 XOR 1 is 0 here I am getting generate. So, 1 XOR 1 XOR 0 is 0 here I am getting generate 1 XOR 1 XOR 1 as one here I am getting generate 1 XOR 0 XOR 0 is 1 and getting kill 0 1 1 is 0 and getting generate here. So, 1 XOR 0 XOR 1 is 0 and getting generate here 1 XOR 0 XOR x 1 1 right. So, so this entire stuff this part of the circuit takes constant time order one time that is constant time. This parallel prefix as we have seen earlier takes log n time because we have n processing units. So, we will consider. So, so the total overall complexity of addition is going to be log n plus 1 or we can say some constant times log n. I think I have done it twice. I hope you for you all remember this.

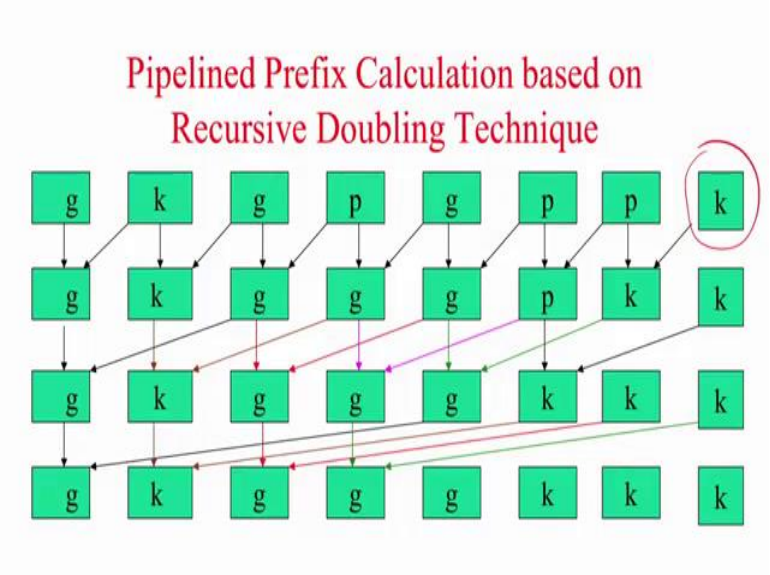
Now, let us go and construct the circuit for this.

(Refer Slide Time: 21:24)



So, we used a recursive doubling technique that we saw in the class number one, because this star is a associative operator.

(Refer Slide Time: 21:38)



So, what we do here. So, we construct this entire unit right we do not. So, what each of this unit will do is that it will compute that star operation right. We can very quickly construct you know a circuit for this truth table right. So, we can have what would be the truth table here. So, I will have a truth table for x j plus 1 star x j right. So, I can say. So, a star b is. So, there is 2 bits for that. So, I will have b 1 b naught a 1 a naught and

whatever the output c 1 c naught. Let us a star b equal to c then I can construct a circuit very quickly for this. So, kill means 0 0 0 0 0 0 0 kill star kill should give me kill right 0, propagate star kill should give me kill. So, propagate is what 0 1 0 0 should give me 0 1 right. I have all the 16 combinations and the 2 out 16 combinations and the 2 outputs here.

So, I can easily construct a circuit which will compute this star. It a truth table right, I leave it as a very simple exercise it will it will turn out to be a simple and or combination. So, I can write all the 16 combinations here do Karnaugh map and get the circuit for this. So, what we see in each of this block is a circuit, which will take 2 2 2 status 2 2 status like kill propagate or kill generate, and then it will compute the new value the star operation.

So, what happens here? So, this is the first level right. In the first level what we do in recursive doubling, we compute with the nearest neighbors. So, propagates star kill will happen here right. So, propagate star kill is kill propagate star propagate is propagate star generate is generate, generate star propagate is generate propagates star generate is generate or sorry yeah correct. And generate star kill is kill, kill star generate is generate in the next stage I have to go 2 away right. Here I am one away I have to go 2 away. So, so what I do kill star propagate is kill, kill star generate is generate. Propagate star generate is generate, generate star generate is generate star then generate star kill is kill and so on.

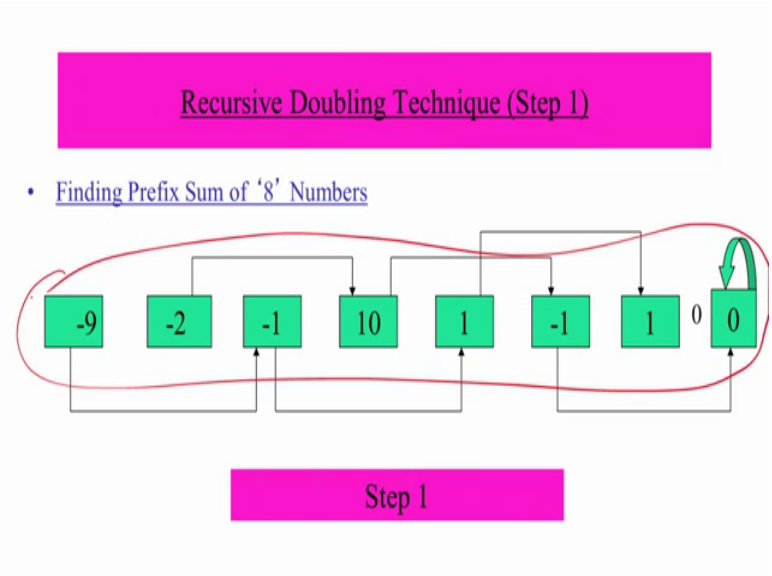
And next time I am going 4 away. So, the end I get this whole prefix computation right. In the case of the addition we saw we were reusing the same unit here again here right still we will do it of in log n time, but I do not want to reuse this I am replicating this unit for. So, since I want to do an 8 bit addition the at least I need 3 stages log of 8 is 3 right. So, I am replicating this 3 times and showing, but I can reuse the same unit again for the star right, but then I need multiplexers to keep rooting it. So, this is very easy for me to do, but there is another advantage of doing this also.

But please understand that if I have such an organization, this is this is basically not a computer program right we have to do it in hardware. So, I just realize it as just array of those star computing units and each computing unit, I connect it in such a way by the end I am getting the prefixes. The moment I get this prefixes from this I know that the carry

is 0 0 0 1 1 1 0 1 which I will again XOR with the respective sum bits, respective input bits to get the sum bit should be right. So, the construction of this circuit essentially goes with the way we have described the recursive doubling in class number one are able to follow. Yes or no.

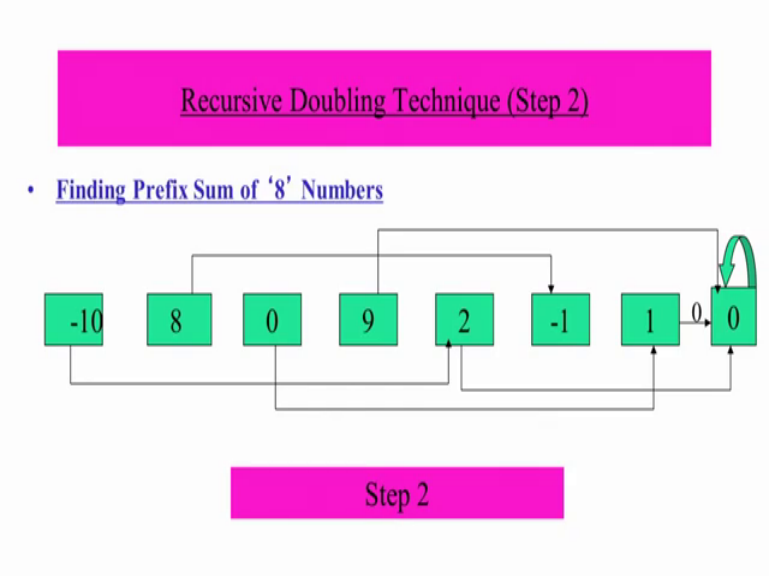
Now, what is the advantage? See in the case of the recursive doubling that we saw earlier.

(Refer Slide Time: 26:19)



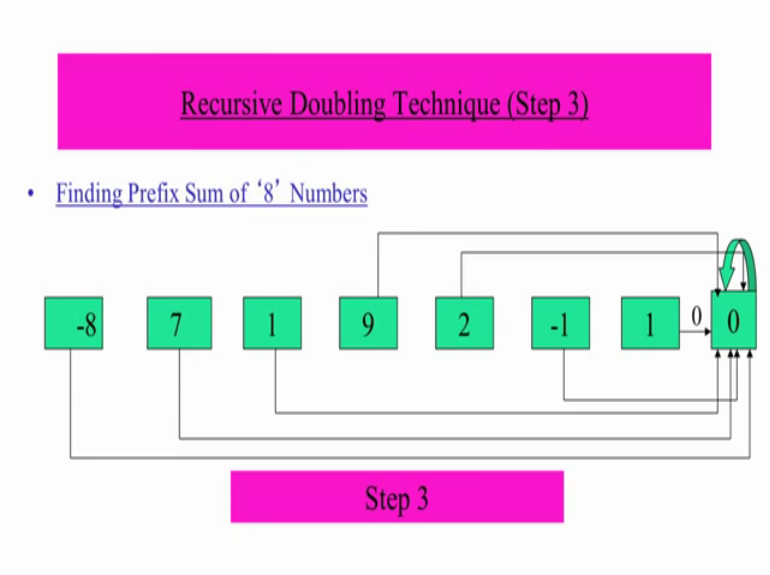
We were using this same unit, we were using the same unit again and again to compute next stages right, but now in this case.

(Refer Slide Time: 26:38)



Are you able to follow what I said just now, in this case I am using the same unit the same unit was used this is the start the same unit was used in step the same units are used in step 2 and step 3 and so on.

(Refer Slide Time: 26:47)



But in this case I am not using the same units. I am just replicating the units I have lot of real estate in my chip right. Today a transistor size is if I assume roughly a transistor is a square thus edge of the transistor is 14 nanometer, 14 into 10 power minus 9 meter. So, I

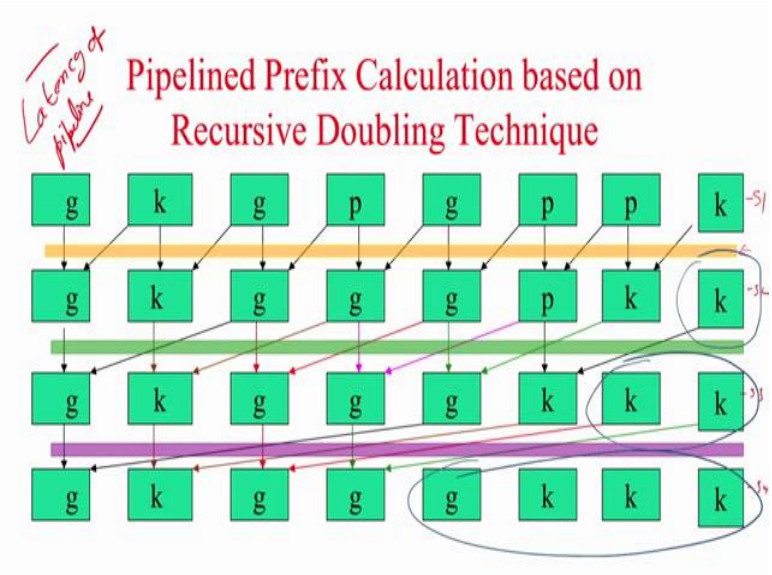
could put billions of transistors. So, I do not mind putting some edges. So, what is a big advantage of this?

Student: Less process.

Less processing power I am doing that processing the big. So, I have improved the area I have increased the area corresponding to you know a carry ripple ladder please note that the area is no plotted up and since the area is plotted up; that means, I have put more transistors. So, more transistors will be toggling 0 to 1 and 1 to 0; that means, more power is also consumed. So, I have increased my power consumption I have also increased my area consumption, but I am getting a good timing performance I have decreased the execution time, but what is specific advantage of having something like this. So, suppose I want to add a 128 bit number it takes 7 units of time here right, 7 plus 1 unit 7 plus 2 9 units of time correct do you get this.

Now, what will happen here is that I can do something called pipelining. So, I am introducing pipelining at a very early stage right this is we will go we will do pipelining in the case of processors also CPU how are we going to pipelining we will go and discuss that as we proceed in this course, but please note that I have a done ai can do a pipelining.

(Refer Slide Time: 28:29)



So, what we see as this you know this red this yellow green and pink, these are storage these are all registers. So, the first set of data will come and you calculate this first stage and you store the results.

When does let us call this as stage 1 stage 2 stage 3 and stage 4. First set of data comes to stage one you compute that and store the results here. And you also take along with it your a's and b's are also you just keep propagating. So, this not only stores the result of this computation right understand, but it will also take along with it a's and b's.

Now, when in the next stage when this stage 2 is computing the results of stage one, I can bring in another 2 set of new numbers. When the first 2 first set of first pair of new first pair of numbers are in stage 3, second pair of numbers could be in stage 2 and the third pair of numbers could be in stage 1. This reg this register or the storage output will isolate these stages, so that some computation happening there will not actually come to the next stage, because the registers get updated only in the clock pulse right. So, I hope you remember what is a register flip flop and registers right. So, the register basically gets updated only when there is a clock pulse.

So, first set of data gets processed and stored here then the second stage will start working on the first set of the first result. Now the second set of data can be processed by stage one, while the first set of data is processed by stage 2 and the processing that happens here will not change the inputs to the second stage, because I have registers here and the registers will not change until it sees a clock pulse. So, the results are stored for computation based stage 2. And that result will gets saved in the green register. So, when the first set of data is processed by stage 3, the second set of data can be processed by it is stage 2, and the third set of data can enter stage one. First set and the result will be stored in the respective pink green and yellow are the next clock pulse. And again when the first set of data processed by s 4 second set can come to s 3 third set can come to s 2 and 4th set can be. By this I can keep adding numbers one after another very fast.

So, if want to add 8 different numbers suppose I have a program in which there are 8 number additions happening immediately one after another. The first number will come after first answer will come after 4 cycles, the second answer will come up in the fifth cycle itself. The third answer will come up in the 6 cycle and so on. So, the in the first answer it will take 4 cycles. So, that we call as the initial latency, the word that we use is

latency, is the latency of the pipeline. The second answer will come up immediately in the next cycle, third answer will come up in the next cycle and the 4th answer will come up in it and so on right. So, what is the cycle duration? So, if I do not have this registers then one that entire computation should happen in one cycle. Assume that each takes one unit of time the cycle duration would be 4.

Since now I have put this thing. So, in each cycle I am expected to do only one operation right, I am appre ah. So, my cycle now time which we required 4 in the absence of these storage now it will become one right correct, are you able to follow if I am going to do the entire operations in one cycle then that circuit depth here is 4, remember this is a good topological sorting here. I can remove all the first stage second stage third stage and 4th stage. So, the circuit depth will be 4 if I do not have the storage. Circuit depth is 4 means my cycle time would be 4; that means, my frequency will be 1 by 4 0.25 whatever.

Now, what is my cycle time it becomes one right. So, my cycle time my frequency also becomes one. So, the frequency actually gets multiplied by 4. So, if I would have run with 100 you know 500 megahertz I could now run with 4 times it 2 gigahertz, are able to appreciate this right. So, by pipelining what else what is happening my frequency actually increases multiple frequency of operation of the circuit can improve multi 4 and what happens I am not only working at 2 gigahertz, but at the end of every cycle I am going to give you one answer except for the first fellow. First fellow will take 4 cycles, but after that every cycle provided I have numbers to add suppose I have 1 million numbers to add in the fa. So, what would have taken 1 million into 4 cycles 4 million cycles would have be the total time requirement in the absence of this pipeline right.

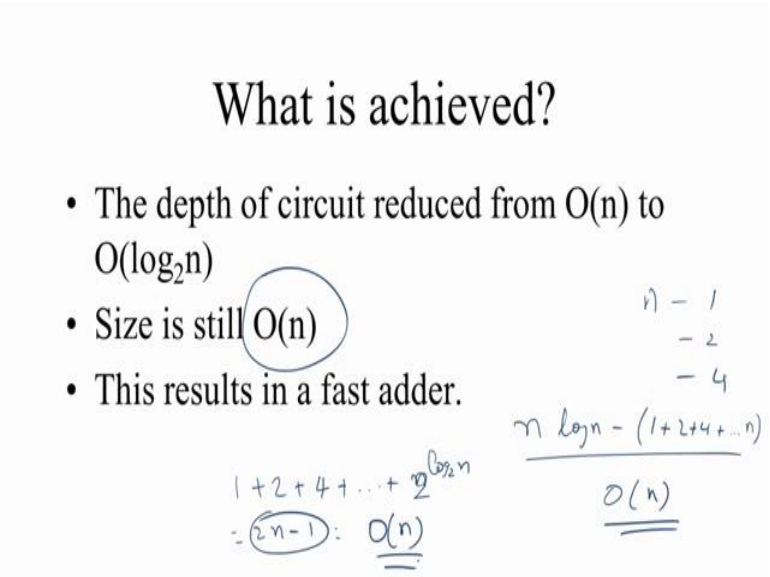
In the presence of this pipeline first fellow will come at 4, second fellow will come at 5 and so on. So, 1 million plus 4th cycle I would get all the answers. So, it is not that my frequency have become 4 times, my execution time also the or what we call as the throughput number of numbers that I could add within some unit time that is also become 4. So, these are very big advantage that we get by pipeline you understand this right.

So, the basics of pipeline the intuition for pipelining comes because when I do not have this storage stages, after this first stage finishes processing the fa a data it wait is for the

remaining 3 cycles right, for the entire computation to finish right the unnecessarily it is holding the data correct. And similarly when the first third and 4th are processing right, the second one actually is idle. When the when the data enters here assume there is no storage no pipelining, when the data enters this part the second fellow is simply sleeping, it need not do anything and then it process then when the third and 4th are doing something the second fellow is just keeping quite.

So, every stage that we look here or every level that we look here is not doing any useful computation when the other levels are doing some computation. So, when a data enters all the 4 level do some computation and when the level k is performing the computation all the remaining 3 levels keep quite or they do not do anything sensible, there either they are maintaining the value or they are just idle. So, that is something. So, that is something we have exploited to see that we get this pipeline behavior, fair enough. Did you follow any doubts? So, the depth of this circuit as reduced from order n order to order log n correct.

(Refer Slide Time: 36:46)



But the size is still order n not exactly order n, but some constant times n right. And this results actually in a very fast order. This results in a very fast adder. Why is this why is the circuit still order n, but see I am having log n stages each have n, n hardware. So, it should become n log n right, no, I have this is log n stages each stage I have n. So, let us say it was 8 right to start with, this is log n stages now. So, it should be 8 into 3 right 24

units you have 32 units you have right. So, it should be n log and in some sense average what you mean by average, I am I am realizing this circuit.

So, I go and say that this circuit is still order n and I am not wrong. Why? Can just carefully look into this and tell me. So, in stage number one please note somebody got the answer see what pipelining.

Student: Effectively becomes the only one layer at the time being.

No, but I am saying the size of the hardware is n I am going to put the transistors here right. So, as please look at this slide right you have 24 yeah.

Student: for log n numbers with n log n systems, so in n size.

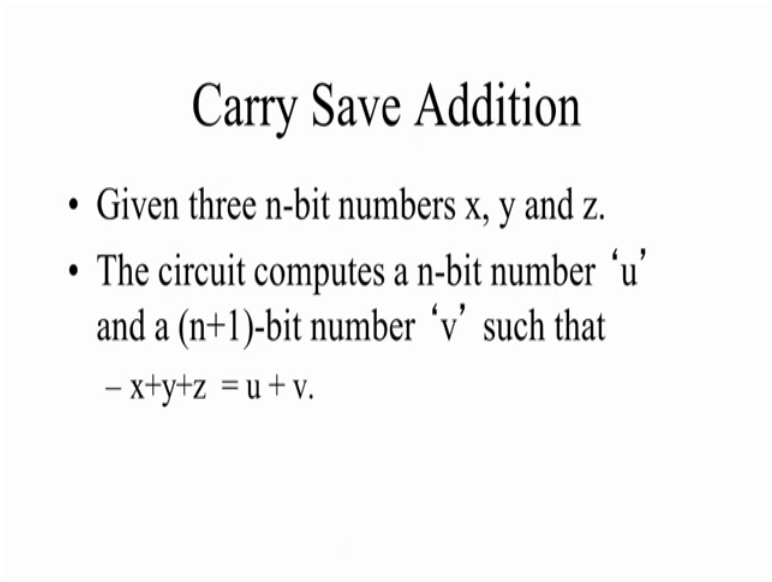
No the size is n log n. Do you appreciate this or not I have log n stages each I have put n fellows here yeah tell me?

Student: There are some minor.

Exactly, so please, so here please see here that this block is not used at all. This is just storage. So, at the first stage I can remove of one, the second stage I am I can remove of 2 the third stage I can remove of 4 these are all unused right I can is just storage element I just have to push this correct. So, this essentially means that. So, first stage I have n right I remove one second stage I remove 2 stage third stage I remove 4 right. So, now if we could compute this sigma right, so this will be n into log n minus 1 2 plus 4 plus till n right. So, when you compute this, this we can show this is a order n right. So, we can still get this to very close other right.

So, if you calculate from the back right this will be one plus 2 plus 4 plus till n. So, what is this what is 1 plus 2 plus 4 plus till n 2 log n is same thing. So, what is 1 plus 2 plus 4 plus? 2 n 2 n minus 1 correct. Are you absolutely sure hey what is this correct right? So, that is something that we need to be extremely careful while designing circuit is right.

(Refer Slide Time: 40:51)



So, we will meet again on Monday. So, kindly revise these things. So, Monday morning 10 o'clock, we will continue with carry save addition.

Thank you.