

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

**Lecture – 22**  
**Efficient Adder Architecture (Contd.)**

Welcome back to the course on Architectural Design of ICS. So, in the last class, we have seen that how this conditional sum adder has been that means, what is the architecture for that, how we can; that means, develop the architecture of conditional sum adder and the basic starting; that means, the idea has started from the this carry select adder operation ok.

So, we are doing what? We are putting a redundant computation to reduce the number of delay by putting extra hardware to the corresponding circuit ok. So, that we have already seen in the last class. And we have seen that why we can we say that this is the most possible the fastest adder among all the adders what we have seen till now because that only the delay level is only one full adder cell plus the multiplexer the number of multiplexer delay is that is  $\log_2 n$  ok.

So, where  $n$  is if I consider 16, at that time 4 multiplexer delay; if I consider, that means, consider 8, so, at that time 3 multiplexer delay. So, within 8 to 16. So, any of this value for  $n$  value, it will be 4. So, from; that means 4 to as the 8 ok. So, 4 to 8 any value for that or 4 to 7 if I consider. So, for 4 to 7 any value if I consider. So, at that time this corresponding multiplexer number of multiplexer will be this 3 number of multiplexer, for within 4 b i t, I need 2 multiplexers something like this I need to.

So, this is, but the thing is that whenever we have to design for the disadvantage of conditional sum adder is that, as we are increasing the number of this multiplexer and the circuit becomes much more complex whenever we consider more number of bits ok. So, these as though it follows one regular structure, but if we consider; that means, more number of bits, at that time the connection becomes very much; that means, complicated as there are so much, so, many that means, components which you need to wire, so, that becomes very much complex whenever we are we consider more number of conditional sum adder bit ok.

So, for that reason, there is another adder are actually not for the that reason; there is a to reduce actually we know that that carry look ahead adder, we generate and then propagate the carry and based on that we calculate the sum and carry for each of the stage ok. So, to reduce the, that means, and in the expression what we have seen that this  $C_{i+1}$  that is equals to  $g_i + p_i \cdot C_i$  ok.

So, yeah actually  $C_{i+1}$ , that is equals to  $g_i + p_i \cdot C_i$ . So, if I consider that I equals to 7, so that means,  $C_i$  that you has to consider that has to for the last term, it has to consider this  $p_7, p_6, p_5, p_4, p_3, p_2, p_1, p_0$  and then the input carry which is  $c_0$ . So that means the levels that will increase if my I restrict the fan in. So, at that time, the levels is increased because of this particular ANDing operation which is which I required for this  $g$  and multiplication of  $g$  and  $p$ .

That means, the generation of the sorry this propagation of the carry ok. So, to reduce that there is one that means, that adder has evolved which is known as Ling Adders ok. So, then how it reduces this a number of that means, terms in carry look ahead adder, that we will see.


(Refer Slide Time: 04:36)

Page 6/8


## Ling Adder

<p>Variation of CLA:</p> $p_i = a_i \oplus b_i$ $g_i = a_i \cdot b_i$ $C_{i+1} = g_i + p_i \cdot C_i$ $S_i = p_i \oplus C_i$	<p>Ling's equations:</p> $t_i = a_i + b_i$ $g_i = a_i \cdot b_i$ $H_{i+1} = g_i + t_{i-1} \cdot H_i$ $S_i = t_i \oplus H_{i+1} + g_i t_{i-1} H_i$
--	---

Ling, IBM J. Res. Dev, 5/81



IIT KHARAGPUR  
Kharagpur 751014



NPTEL ONLINE  
CERTIFICATION COURSES

Computer Arithmetic

68

So, this is the CLA; that means basic CLA operation. So, the  $p_i$  signal that is  $a_i \oplus b_i$  and  $g_i$  is  $a_i \cdot b_i$ . Whereas,  $C_{i+1}$  that has been generated based on this  $g_i + p_i \cdot C_i$  and  $s_i$  is generated from  $p_i \oplus C_i$  but according to lings equation, they have modified the equation in such a way, so that it can reduce the number of terms

which are related which are coming for these particular things. So, it says that this  $t_i$  equals to  $a_i \oplus b_i$  and  $g_i$  equals to  $a_i \cdot b_i$  and this  $H_{i+1}$  equals to  $g_i \oplus t_i$  minus 1 dot  $H_i$  ok.

So, initially what was there; that means, that was  $p_i$  was  $a_i \oplus b_i$ , but here this is OR and that is this  $t_i$  minus 1 and this  $s$  has been calculated based on this  $t_i \oplus H_{i+1}$  plus  $g_i$  into  $t_i$  minus 1 into  $H_i$ .

So, this is the modified equation. If you see if you just that means, compare this two equation, it is same like the generation of the; that means, propagation of this generation and propagation of the carry where this is basically modified with  $h$ , this  $c$  is modified that means, changed with or renamed as  $h$  here and  $s$  has been this is not XOR operation this is with  $C_i$  only; this is XOR with this  $H_{i+1}$  along with OR with this  $g_i \oplus t_i$  minus 1 in and  $H_i$  ok.

(Refer Slide Time: 06:40)

Page 8 of 14

## Ling Adder

<p>Variation of CLA:</p> $C_{i+1} = g_i + g_i C_i + p_i \cdot C_i$ $= g_i + (g_i + p_i) \cdot C_i$ $C_{i+1} = g_i + t_i \cdot C_i$	<p>Ling's equation:</p> $H_i = g_i + t_{i-1} \cdot H_{i-1}$ <p>Ling uses different transfer function. Four of those functions have desired properties (Ling's is one of them)</p>
---	--

IIT KHARAGPUR  
Knowledge for All

 NPTEL ONLINE  
 CERTIFICATION COURSES  
Computer Arithmetic
69

Then what is that; that means the function of this. So, variation of CLA is that  $C_{i+1}$  equals to  $g_i \oplus g_i C_i \oplus p_i \cdot C_i$  ok.

So, if I just rewrite it in this term. So, at that time  $g_i$  will be  $g_i \oplus g_i \oplus p_i \cdot C_i$ . So, this  $g_i \oplus p_i$ , that is as this  $t_i$  ok. So, and this according to this ling equation, so this  $H_i$  can be written as this  $g_i \oplus t_{i-1} \cdot H_{i-1}$ . So, here you see, ling

uses different transfer function; four of this function have desired properties, ling is one of them.

(Refer Slide Time: 07:35)

**Ling Adder**

Conventional:

$$C_4 = g_3 + t_3g_2 + t_3t_2g_1 + t_3t_2t_1g_0 + t_3t_2t_1t_0C_{in}$$

Fan-in of 5

Ling:

$$H_4 = g_3 + t_2g_2 + t_2t_1g_1 + t_2t_1t_0g_0 + t_2t_1t_0C_{in}$$

$$H_4 = g_3 + g_2 + t_2g_1 + t_2t_1g_0 + t_2t_1t_0C_{in}$$

Fan-in of 4

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 70  
 NPTEL | Digital Arithmetic

So, in the conventional method, if I just for C 4, if I consider 4 bit. So, at that time if you see at the last term what I said that at the last term what I need I need this here if I instead of that p, if I just replace that with t so; that means, I need t 3, t 2, t 1, t 0 along with this C in which is the first carry input bit. But according to the ling equation, if I just write this, so at that time H 4 will be written as t 2, t 1, t 0 and C in. So that means, these 5 requirement in conventional carry look ahead adder that has been reduced in ling equation where this is basically; that means, here if the fan in is 5 here, the fan in is 4.

So, if the fan in is 4 here and the fan in of 5 here, that means, we can reduce the delay.

(Refer Slide Time: 08:43)

The slide is titled "Advantages of Ling's Adder" and features a yellow background. At the top, there is a navigation bar with various icons and the text "Page 8/8". The main content consists of three bullet points and a note. The bullet points are: "Uniform loading in fan-in and fan-out", " $H_{16}$  contains 8 terms as compared to  $G_{16}$  that contains 15.", and " $H_{16}$  can be implemented with one level of logic (in ECL), while  $G_{16}$  can not." Below the bullet points is a note: "(Ling's adder takes full advantage of wired-OR, of special importance when ECL technology is used)". At the bottom of the slide, there is a blue footer containing the logos of IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES Computer Arithmetic" and the page number "71".

## Advantages of Ling's Adder

- Uniform loading in fan-in and fan-out
- $H_{16}$  contains 8 terms as compared to  $G_{16}$  that contains 15.
- $H_{16}$  can be implemented with one level of logic (in ECL), while  $G_{16}$  can not.

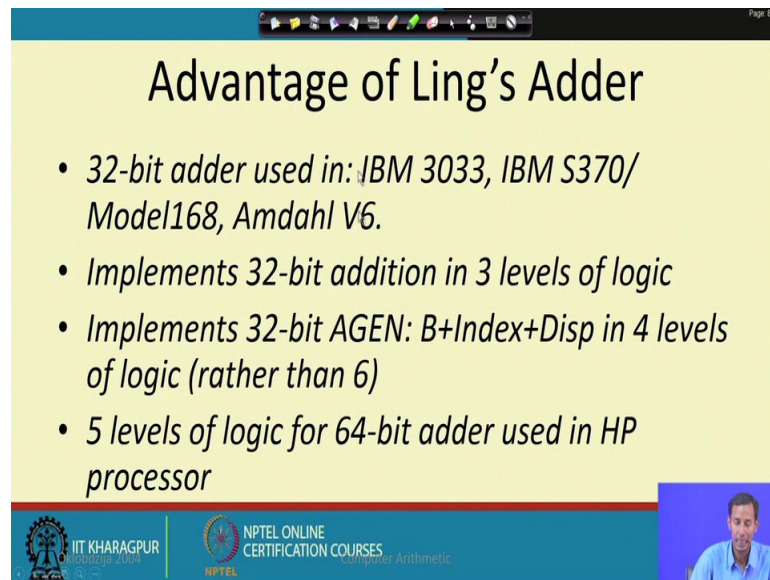
(Ling's adder takes full advantage of wired-OR, of special importance when ECL technology is used)

IIT KHARAGPUR  
NPTEL ONLINE CERTIFICATION COURSES  
Computer Arithmetic  
71

So that means, then the advantage of Ling's Adder? Uniform loading in fan in and fan out this  $H_{16}$  contains 8 terms as compared to  $G_{16}$  that contains 15, ok. So that means, if I consider this  $G_{16}$  sorry, this  $C_{16}$  at that time I need according to the CLA, I need the term as 15. But here in according to this new equation of  $H$  using this 8 number of terms, I can compute this  $H_{16}$ . Then,  $H_{16}$  can be implemented with one level of logic while  $G_{16}$  cannot.

So that means, then obviously, as I am I can increase or sorry, I can decrease that number of logic level that means, I can improve the corresponding speed or the operating frequency of the adder design using this lings equation.

(Refer Slide Time: 09:46)



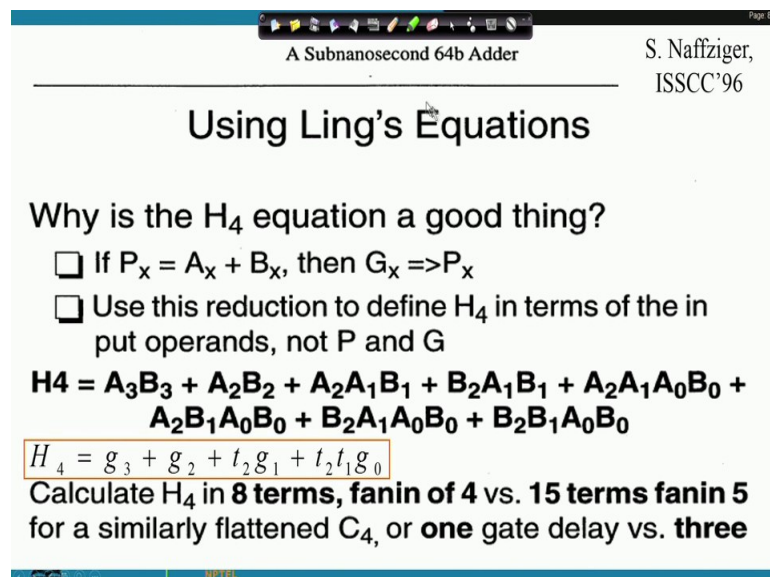
**Advantage of Ling's Adder**

- 32-bit adder used in: IBM 3033, IBM S370/Model168, Amdahl V6.
- Implements 32-bit addition in 3 levels of logic
- Implements 32-bit AGEN: B+Index+Disp in 4 levels of logic (rather than 6)
- 5 levels of logic for 64-bit adder used in HP processor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Computer Arithmetic

And this is basically, this 32 bit adder, this Ling's Adder that has been used in IBM 3033 model. And then; that means 3 bit addition in 3 levels of logic and then 5 level of logic for 64 bit adder used in HP processor. So that means Ling's Adder is very much useful in the earlier days.

(Refer Slide Time: 10:30)



A Subnanosecond 64b Adder | S. Naffziger, ISSCC'96

**Using Ling's Equations**

Why is the  $H_4$  equation a good thing?

- If  $P_x = A_x + B_x$ , then  $G_x \Rightarrow P_x$
- Use this reduction to define  $H_4$  in terms of the input operands, not P and G

$$H_4 = A_3B_3 + A_2B_2 + A_2A_1B_1 + B_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0 + B_2A_1A_0B_0 + B_2B_1A_0B_0$$
$$H_4 = g_3 + g_2 + t_2g_1 + t_2t_1g_0$$

Calculate  $H_4$  in **8 terms, fanin of 4** vs. **15 terms fanin 5** for a similarly flattened  $C_4$ , or **one gate delay** vs. **three**

So, earlier days, that processors accomplished this computer microprocessor ok. So, then this is this one work which is basically if you just want to that means, know more on this

Ling's Adder architecture. So, you follow the work on this; that means, which is approved; that means, published in international solid state conference in 1996 ok.

So, there you will get how that means, the equation has been developed and how the, that means, from this particular equation, how the corresponding architecture has been developed, that you will find that means, in details in this particular paper ok.

(Refer Slide Time: 11:09)

A Subnanosecond 64b Adder ISSCC96

### Using Ling's Equations

$$H_4 = A_3B_3 + A_2B_2 + A_2A_1B_1 + B_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0 + B_2A_1A_0B_0 + B_2B_1A_0B_0$$

$$H_4 = g_3 + g_2 + t_2g_1 + t_2t_1g_0$$

S. Naffziger, ISSCC'96

So, that, this is the transistor level implementation of lings equation, ok.

(Refer Slide Time: 11:13)

A Subnanosecond 64b Adder ISSCC96

### Using Ling's Equations

A fast group of 4 propagate is also necessary to combine for the next level (16) of carry generate:

$$I_4 = P_0P_1P_2P_3$$

If  $P = A + B$ , this can be done in one gate delay using "wired-or" techniques also.

S. Naffziger, ISSCC'96

75

And we all this, you will get from this particular paper. So, I am not going into the details of that then. So, that is the advantage of that means, Ling's Adder. And what is that? That starting point of that is the carry look ahead adder that then the present days basically, we use this Prefix Adders or The Parallel Prefix Adders; that means, here what we do; that means, suppose I need to add two numbers a and b.

So, at that time in the in the that means, conditional sum also what we do we pre compute those values and then we add or actually in condition of sum, what we do we pre compute those values considering 0 and 1 and then based on the multiplexer we basically select with signal I have to pass at the output level.

And in carry look ahead adder what we do, we basically; that means, initially from the bit; that means, input bit we basically generate and then propagate the carry we have two logic, one general for generation of the carry; one for propagation of the carry then based on the equation now we are trying to calculate the sum and carry output.

So, here also the same thing ok, so it has; that means, this parallel prefix adders it has that means, this pre processing and post processing that means, scheme.

(Refer Slide Time: 12:55)

**Method 1:**

Step 1: Pre-processing  $g_i = a_i \bullet b_i$      $p_i = a_i \oplus b_i$

Step 2: Prefix Computation

$$G_{i+1} = \begin{cases} g_i, & \text{if } i = k \\ G_{i+1} + P_{i+1} G_{i+1}, & \text{otherwise} \end{cases}$$

$$P_{i+1} = \begin{cases} p_i, & \text{if } i = k \\ P_{i+1} P_{i+1}, & \text{otherwise} \end{cases}$$

Step 3: Post-processing

$$c_i = G_{i0}$$

$$S_i = p_i \oplus c_{i-1}$$

**Method 2:**

Step 1: Pre-processing  $g_i = a_i b_i$      $p_i = a_i \oplus b_i$      $k_i = a_i + b_i$

Step 2: Prefix Computation

$$G_{i+1} = \begin{cases} g_i, & \text{if } i = k \\ G_{i+1} + K_{i+1} G_{i+1}, & \text{otherwise} \end{cases}$$

$$\bar{K}_{i+1} = \begin{cases} \bar{k}_i, & \text{if } i = k \\ \bar{K}_{i+1} \bar{K}_{i+1}, & \text{otherwise} \end{cases}$$

Step 3: Post-processing  $c_i = G_{i0}$      $S_i = p_i \oplus c_{i-1}$

So, it has follows two method ok. So, in one method, there are the follows three step; one step one is Pre-processing. So, pre processing in pre processing generate the carry using this particular equation and propagate the carry using this particular equation and then



you compute the prefix. So, how you can compute the prefix? The prefix can be computed using  $g_i$  and  $k_i$ . So that means, this is nothing but a actually we will see this the corresponding graph of this different adder architecture parallel prefix architecture.

So, here actually we will get one node or this will follow one regular structure ok. So, considering we have to generate this  $g_i$  and  $k_i$  based on this particular equation. If  $i = n - k$  both the values are same. So, at that time  $g_i$  will be the  $g_i$  and  $k_i$  value and if it is not if  $i$  and  $k$ , they are not same. So, at that time I have to do  $g_{i-1} + p_{i-1}$  in dot  $g_{i-1}$  and  $k_{i-1}$  ok. So that means, this is the coordinate of  $g$  and  $p$  which are basically presented within these parentheses.

So, then again for  $p$ , this is for  $g$  and this is for  $p$ , again we have to follow another equation. So, for  $i = k$  that will be  $p_i$  or otherwise it will be  $p_{i-1} + p_{i-2}$  in dot  $p_{i-1}$  and  $k_{i-1}$ . So, then again I need, so once this pre prefix computation is done based on this post processing of  $g_i$  and  $p_i$ , then we can go for this post processing for calculating the corresponding final carry in and sum bit. So, the  $C_i$  value is  $G_i$  and this sum is  $p_i$  XOR with  $C_{i-1}$  ok.

So, this finally, will generate, so that means, this is a simple logic which will generate the corresponding sum and carry for each of this stage. So that means, now if I consider 7 bits, so, at that time this will varies from 0 to 7. Then again, there is another method where in this method one we are taking  $g_i$  and  $p_i$ . So, in another method, we have to; that means, take or we can take 3, 3 variables that is  $g_i$ ,  $p_i$  and  $k_i$  where  $k_i$  is nothing but this XNOR operation sorry NOR operation of  $a_i$  and  $b_i$ .

And then, for prefix computation we will use this  $k_i$  instead of  $p_i$  in this case we will use this  $k_i$  as by following this particular equation that is  $g_i$  and  $k_i$  that is  $g_i$  and  $g_{i-1} + \bar{k}_{i-1}$  in dot  $g_{i-1}$  and  $k_{i-1}$ . If for  $i = k$  that is  $g_i$  for  $i$  not equals to  $k$ , then this equation and for  $k_i$  that is for  $i = k$  that means, the bar of or the complement of  $k_i$  otherwise  $k_{i-1} + k_{i-2}$  in dot  $k_{i-1}$  and  $k_{i-2}$  otherwise. And then again finally, follow the same post processing for final computation of the for each of this position or the each of the stage; that means, again same if the carry that means,  $i$  is vary varying from 0 to 7 so; that means, then for 7 this will be this sum and carry that will be computed ok.

(Refer Slide Time: 17:15)

Associative property and idempotent property

Associative property

$$(G, P)_{h[i,j]} * (G, P)_{j[i,k]} = (G, P)_{h[i]} * (G, P)_{j[i,k]}$$

Associativity allows pre-computation of sub-terms of the prefix equations

$$(G, \bar{K})_{h[i,j]} * (G, \bar{K})_{j[i,k]} = (G, \bar{K})_{h[i]} * (G, \bar{K})_{j[i,k]}$$

Idempotent property

$$(G, P)_{h[i,j]} * (G, P)_{h[i,j]} = (G, P)_{h[i,j]}$$

Idempotency allows these sub-terms to overlap, which provides some useful flexibility in the parallelization

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, then whenever we are computing these parallel prefix adders, so at that time it has two follow two; that means, property it has to follow. So, once or; that means, there are basically two properties which are related to this parallel prefix adders. So, one of the property is Associativity and another that means, of the property is Idempotent property ok. So, the Associative allows pre-computation of sub terms of the prefix equation. Whereas, this idempotency allows these sub terms to overlap, which provides some useful flexibility in the parallelization of the corresponding circuit ok. So that means, how we can that means, define this associativity property. So, if I am having this  $g, p$ , this is the dot operation with this  $g, p$  of  $j, k$ .

So, here you see that this is the within this parenthesis that that means, corresponding term this is  $h$  to  $j$  and this is  $j$  to  $k$ , then I can write  $h$  to  $i$  to  $i$  to  $k$ ; that means, this  $j$  can be replaced as with  $i$ . And for this  $k$  dash that means, this is for method 1 and this is for method 2 where this I am, for method one I am using  $p$  for method 2, I am using  $k$ .

So, for method 2, this is  $g, k$  bar  $h, j$ . So,  $j$  to  $k$ , so, again it can be written  $h$  to  $i$  then  $i$  to  $k$  ok. Then for idempotent that means, idempotent this property, I can write this as the same; that means, where this here for this method 1, it will consider  $p$ ; for method 2, it will consider  $k$  with the equation is only different ok. So, these two properties are related to parallel prefix adder computation.

(Refer Slide Time: 19:41)

To simplify the representation of  $G$  and  $K$  or  $P$ , an operator called as dot operator represented by '\*' is introduced to create group generate and group kill bar

$$(G, P)_{[2:1]} = (G, P)_{[2:1]} * (G, P)_{[1:4:1]}$$

**Method 1:**

$$(G, K)_{[2:1]} = (G, K)_{[2:1]} * (G, K)_{[1:4:1]}$$

**Method 2:**

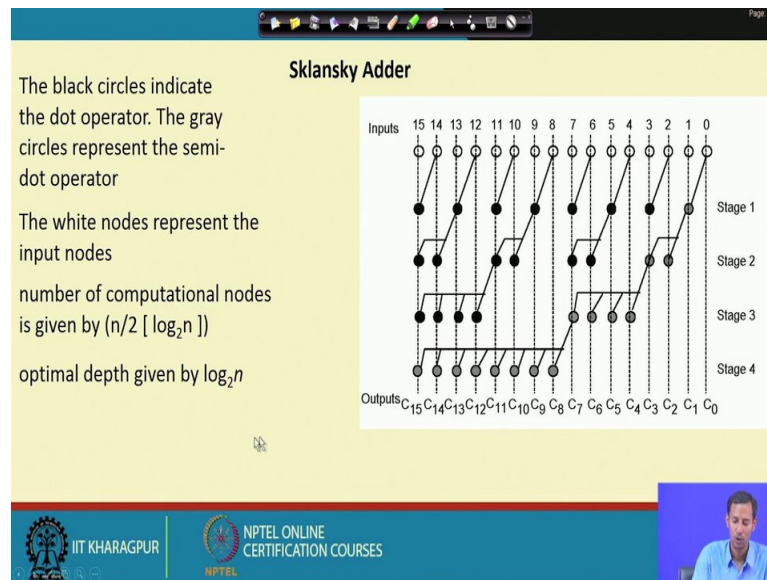
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, whenever we are that means, telling this dot operation, so, what does it means? This simplifies the representation of  $g$  along with  $k$  or  $p$  for method 1,  $k$  for method 2, an operator called as dot operator represented by star is introduced to create group generate or group kill bar.

So, how we can that means, in method 1, in method 1, this is the dot operation; in method 2 this is the dot operation, ok. Here you see this is the; that means, in method 2, this is the dot operation in method 1 this is the dot operation ok. So, what I need is basically, so based on this we have to; that means, what we are doing this prefix we are basically pre computing and that is basically work or that is represented by one node.

Now, we will follow the; that means, one regular structure to compute the corresponding; that means, the bit or the corresponding; that means, the sum and carry value for each of this bit position ok.

(Refer Slide Time: 21:02)



So, how we can do that? Suppose I am having this 16 bit inputs or 16 bit that means, addition I require. So, this blank box, this white color box, they represent or this is nothing but this is the DFG of these parallel prefix adders which is known as Sklansky Adder ok.

So, here this basically represent these input nodes, this black boxes this black boxes they are represented basically the dot operator. So, dot operator we have already seen that g and k p for method 1 and g and k for method 2, ok. So, these are the dot operators and these are the semi dot operators ok. And based on that, we can get the final output ok.

So, I need this four stage if I am considering this kind of this; that means, whenever this will; that means, the carry from here that will effect this, then carry from there that will effect this. So, then again this carry can be come to here. So, this is one of the architecture if you follow. So, what we have to do? Here, we have to compute the prefix based on method 1 or method 2. So that means, this dot operators or this particular semi dot operators based on the equation what we have seen in the earlier section.

So; that means, now what does it means, this basically needs two that means, the coordinates of 2 as if I represent this as a 2 D graph. So, at that time, so, in this direction and in this direction I need to; that means, represent this particular point or this particular node ok. So, that is why, based on this from which node to which node will be

connected, so that we will get from the corresponding information which we have already mentioned earlier, ok.

So, here in this case if we consider this particular Sklansky adder method or this Sklansky, this is a DFG of this Sklansky adder, if we follow that, so the number of computational node that will be  $n \log_2 n$  if  $n$  is 16, so then here 8 and for this is 4, so that means, total number of computational node will be 32 and the optimal depth will be  $\log_2 n$ ; that means, there is 16 means 4.

(Refer Slide Time: 24:20)

The black circles indicate the dot operator. The gray circles represent the semi-dot operator

The white nodes represent the input nodes

number of computational nodes is given by  $(n \lceil \log_2 n \rceil - n + 1)$

optimal depth given by  $\log_2 n$

**Kogge-Stone Adder**

Inputs: 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Outputs:  $C_{15}, C_{14}, C_{13}, C_{12}, C_{11}, C_{10}, C_9, C_8, C_7, C_6, C_5, C_4, C_3, C_2, C_1, C_0$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

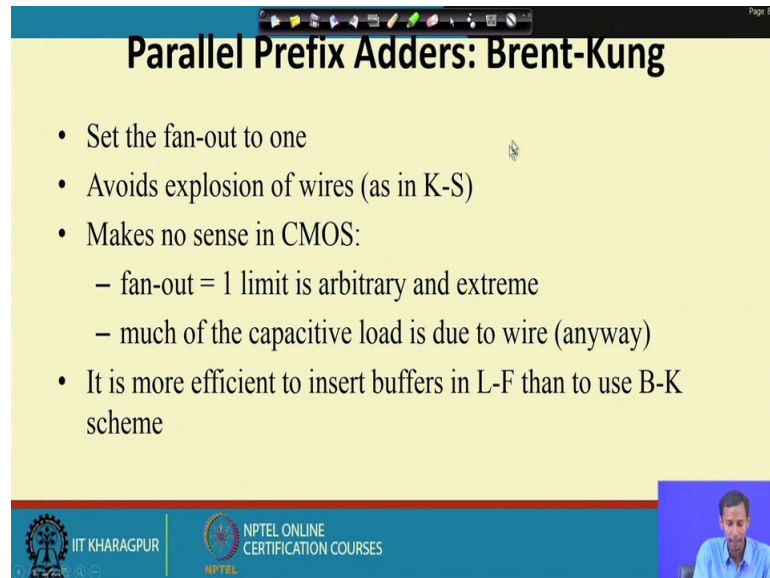
Then there is another method which has been proposed by this two particular scientist this; that means, mister Kogge and Mister Stone. So, they have named this that means, adder architecture as Kogge-Stone adder.

So, there are that means, you see more of the that means, this corresponding; that means, operators are overlapped to each other ok. So, whenever here that means, overlap to each other means, we are increasing the corresponding number of this computational node. Though the that means, optimal depth, if I consider the previous case the optimal depth remains same ok, so the number of these black dots and these gray dots that increases along; that means, in this particular Kogge-Stone adder.

So, the here the number of computational node is  $n \log_2 n - n + 1$ . Then again to improve the corresponding performance of this Kogge-Stone adder, we have

another architecture which is this Brent-Kung Adder. So, here what is that means,, how it has been generated, by setting the fan out to 1 and another that means, problem with this is that here more wiring is wrings are there.

(Refer Slide Time: 25:55)



### Parallel Prefix Adders: Brent-Kung

- Set the fan-out to one
- Avoids explosion of wires (as in K-S)
- Makes no sense in CMOS:
  - fan-out = 1 limit is arbitrary and extreme
  - much of the capacitive load is due to wire (anyway)
- It is more efficient to insert buffers in L-F than to use B-K scheme

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, to reduce the number of wire, we have that means, come to this adder architecture which is Brent-Kung adder ok.

So, this basically avoids the explosion of wires which basically happens in that means, Kogge-Stone adder and it is more efficient to insert buffers in; that means, in compared to this ; that means, according to this Brent-Kung adder scheme we can insert the buffer more efficiently ok.

(Refer Slide Time: 26:33)

The black circles indicate the dot operator. The gray circles represent the semi-dot operator

The white nodes represent the input nodes

number of computational nodes is given by  $[2\log_2 n - 2]$

optimal depth given by  $(2n - 2 - \lceil \log_2 n \rceil)$

### Brent-Kung Adder

Inputs: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Outputs: C<sub>15</sub> C<sub>14</sub> C<sub>13</sub> C<sub>12</sub> C<sub>11</sub> C<sub>10</sub> C<sub>9</sub> C<sub>8</sub> C<sub>7</sub> C<sub>6</sub> C<sub>5</sub> C<sub>4</sub> C<sub>3</sub> C<sub>2</sub> C<sub>1</sub> C<sub>0</sub>

Stage 1  
Stage 2  
Stage 3  
Stage 4  
Stage 5  
Stage 6

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, then again if we just follow the circuit that means, the architecture for this Brent-Kung adder. So, here you see, this is the architecture for this Brent-Kung adder.

Here the number of that means, the these nodes that are decreasing, but here you see at the num that means, with this; that means, as the number of nodes are increasing, that means, decreasing the number of stage that are basically increase, sorry this as the number of nodes are decreasing, but the number of stages are increasing for this Brent-Kung Adder ok. So, this is the that means, the computational node is required this number and the optimal depth which is can be derived from this particular equation and n is the that means, the bit consider for the adders or the addition operation.

(Refer Slide Time: 27:33)

### Parallel Prefix Adders: Han-Carlson

- Is a hybrid synthesis of L-F and K-S
- Trades increase in logic depth for a reduction in fan-out:
  - effectively a higher-radix variant of K-S.
  - others do it similarly by serializing the prefix computation at the higher fan-out nodes.
- Others, similarly trade the logical depth for reduction of fan-out and wire.



Then again whether actually we have this Han-Carlson that means, this algorithm ok. So, this basically is one this is one like hybrid architecture.

(Refer Slide Time: 27:51)



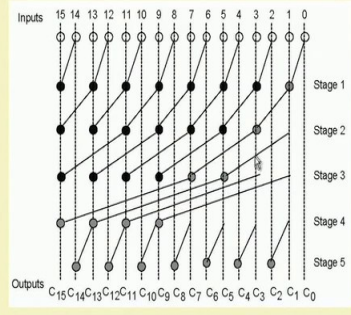
### Han-Carlson Adder

The black circles indicate the dot operator. The gray circles represent the semi-dot operator

The white nodes represent the input nodes

number of computational nodes is given by  $(n/2 \lceil \log_2 n \rceil)$

optimal depth given by  $\lceil \log_2 n + 1 \rceil$



And here what we do we basically do, to reduce the number of stages we use at the very beginning, we use at the very that means, in the first we use a one different; that means, this Kogge-Stone architecture and that in the in the that means, lower portion we use different method to reduce the number of states which are the that means, this disadvantage of Brent-Kung adder. And here you see, by this particular Han-Carlson



adder, so we can reduce the number of computational node by this and the optimal depth by this ok.

(Refer Slide Time: 28:36)

Page 8/8

### Ladner-Fischer Adder

The black circles indicate the dot operator. The gray circles represent the semi-dot operator

The white nodes represent the input nodes

number of computational nodes is given by  $(n/2 \lceil \log_2 n \rceil)$

optimal depth given by  $\lceil \log_2 n + 1 \rceil$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, then we will actually this Ladner that means, land Ladner and Fisher Adder, this basically L and F is Ladner and Fisher Adder; that means, that is the hybrid of this Han-Carlson that means, adder is the say hybrid of this L S and K S these two ok. So, what is there in this Ladner-Fisher Adder?

So, in Ladner-Fisher adder we it has follows one different topology where the number of computational node is  $n/2 \log_2 n$  plus  $n$  and optimal depth is  $\log_2 n$  plus 1. So, what this; that means, the number of state; that means, remains same that is 5.

(Refer Slide Time: 29:19)

### Parallel Prefix Adders: variety of possibilities

from: Erce

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are again that means, this we have already that means seen this.

(Refer Slide Time: 29:23)

Pyramid Adder

M. Lehman, "A Comparative Study of Propagation Speed-up Circuits in Binary Arithmetic Units", IFIP Congress, Munich, Germany, 1962.

Fig. 2. Pyramid carry (eight bits) (Ex. 9).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 29:25)

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	4	7	4
Kogge-Stone	10	7	3
Han-Carlson	5	7	4
Ladner-Fischer	5	7	3
Sklansky	5	7	4

So, if we just that means, see that; that means, what are the that means, the difference in this particular adder design if you consider this 8 bit parallel prefix adder considering different type of adder architecture.

The number of dot, dot means that black box or the black circle and where semi dot means this gray circle the number of requirement along with this logic depth for 8 bit parallel prefix adder for Brent-Kung the dot are 4 semi, dot are 7 and the logic depth of 4 for Kogge-Stone, the dots are more and this similar to as there are more overlap ok. So, that dot are 10, semi dots are 7 and the logic depths that has been reduced that is 3 by putting that means, parallel computation then Han-Carlson that is 5, 7, 4; for Ladner-Fisher that is 5, 7, 3 and this Sklansky that is 5, 7 with 3 sorry 4.

(Refer Slide Time: 30:32)

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	11	15	6
Kogge-Stone	34	15	4
Han-Carlson	17	15	5
Ladner-Fischer	17	15	4
Sklansky	17	15	5

So, this is for 8 bit and then for this is for 16 bit, if you see that which one is the best one from this, if I just that means, try to calculate this is a minimum number which I am getting for that means, that is the Brent-Kung adder, but the logic depth that has been increased. That means, whenever I need area as my major constraint. So, at that time I will follow Brent-Kung adder; that means, I have the liberty or I can; that means, I can that means, leave this speed operation.

That means, the speed is not that much; that means, major constraint at that time where area is my major constraint. So, at that time I can follow this Brent-Kung adder otherwise, if we if area is my that sorry not that major constraint, but speed is my major constraint. So, at that time we have to follow this Kogge-Stone architecture.

(Refer Slide Time: 31:29)

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	26	31	8
Kogge-Stone	98	31	5
Han-Carlson	33	31	6
Ladner-Fischer	33	31	6
Sklansky	33	31	5

So, this is for 16 bit and this is for 32 bit.

(Refer Slide Time: 31:34)

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	57	63	10
Kogge-Stone	316	63	6
Han-Carlson	129	63	7
Ladner-Fischer	129	63	7
Sklansky	129	63	6

So, here you see that is 5 for 64 bit that is 16 because that is  $\log_2 n$ . So,  $n = 64$  means, this will be 6 for 128, this will be 7, but the number of dot operation that are on the higher side in compared to this particular 5.

(Refer Slide Time: 31:52)

Performance comparison of 8-bit Parallel Prefix Adders  
in using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10 <sup>15</sup> Joules)
Brent-Kung	51.74536	0.53	27.4250408
Kogge-Stone	64.43130	0.35	22.550955
Han-Carlson	54.65521	0.53	28.9672613
Sklansky	54.04176	0.35	18.914646
Ladner-Fischer	51.74552	0.52	26.9076704

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, then if we take this power and delay if we take this power and delay of the conjunction of this particular 5 adder architecture and at that time you see this Sklansky Adder, that gives you the lower number lower number of; that means, a power delay product whenever we are that means, developing 8 bit parallel prefix adder using 180 nanometer technology.

(Refer Slide Time: 32:19)

Performance comparison of 16-bit Parallel Prefix Adders  
in using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10 <sup>15</sup> Joules)
Brent-Kung	102.2749	0.88	90.001912
Kogge-Stone	140.9152	0.53	74.685056
Han-Carlson	111.5113	0.84	93.669492
Sklansky	110.0999	0.50	55.04995
Ladner-Fischer	101.8946	0.84	85.591464

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then this is for 16 bit parallel prefix adder using 180 nanometer technology again, you just see that the Sklansky the delay is basically. So, based on this average power

consumption is 110 and the corresponding this power delay product that is 55, which is the minimal 1 in compared to the other 4 adder architecture ok.

(Refer Slide Time: 32:53)

### Parallel Prefix Adders: variety of possibilities

Knowles 1999

	Structure	Buffering	Delay (ref invs)	Length (μm)	Transverse wire flux	
					By level	Total
Ladner-Fischer (fig 2)	[16,8,4,2,1]	[2,1,1,0,0]	13.7	38	[1,2,2,2,2]	9
-	[16,4,2,2,1]	[2,1,1,0,0]	13.2	38	[1,4,4,2,2]	13
-	[16,2,2,2,1]	[2,1,1,0,0]	13.0	41	[1,8,4,2,2]	17
(fig 6)	[4,4,2,2,1]	[1,1,0,0,0]	13.2	35	[4,4,4,2,2]	16
-	[4,4,2,2,1]	[1,1,1,0,0]	12.7	39	[4,4,4,2,2]	16
-	[2,2,2,1,1]	[1,1,1,0,0]	12.1	46	[8,8,4,4,2]	26
Kogge-Stone	[1,1,1,1,1]	[1,1,0,0,0]	12.1	63	[16,16,8,4,2]	42
	[1,1,1,1,1]	[1,1,1,0,0]	11.8	63	[16,16,8,4,2]	42

- Delay is given in terms of FO4 inverter delay: w.c. (nominal case is 40-50% faster)
- K-S is the fastest
- K-S adders are wire limited (requiring 80% more area)
- The difference is less than 15% between examined schemes

And, then, so, these are the; that means, the structure how we can select.

(Refer Slide Time: 32:57)

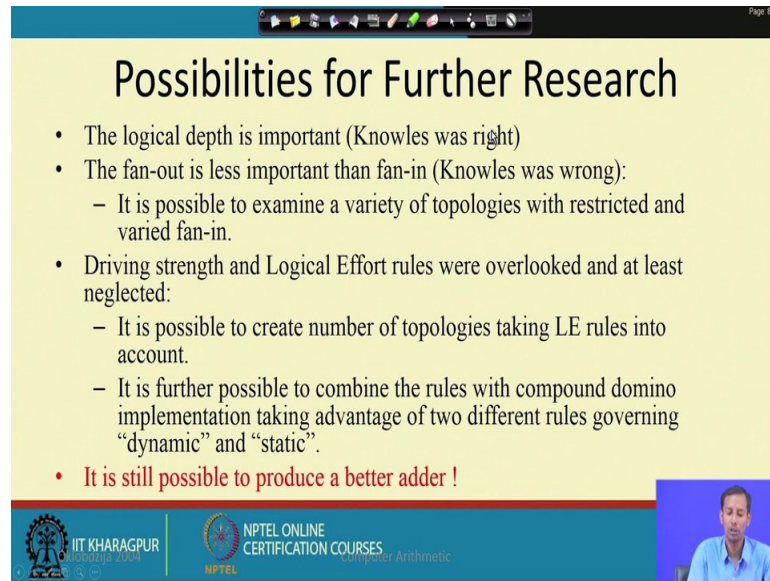
### Pyramid Adder:

M. Lehman, "A Comparative Study of Propagation Speed-up Circuits in Binary Arithmetic Units", IFIP Congress, Munich, Germany, 1962.

Fig. 2. Pyramid carry (eight bits) (Ex. 9).

So, this is the pyramid adder, another adder, how we basically do and we have already seen this.

(Refer Slide Time: 33:08)



The slide is titled "Possibilities for Further Research" and contains the following text:

- The logical depth is important (Knowles was right)
- The fan-out is less important than fan-in (Knowles was wrong):
  - It is possible to examine a variety of topologies with restricted and varied fan-in.
- Driving strength and Logical Effort rules were overlooked and at least neglected:
  - It is possible to create number of topologies taking LE rules into account.
  - It is further possible to combine the rules with compound domino implementation taking advantage of two different rules governing “dynamic” and “static”.
- **It is still possible to produce a better adder !**

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the text "Computer Arithmetic". A small video inset in the bottom right corner shows a man speaking.

So, the possibilities for further research is that the logic depth is very much important. So, how we can reduce or whether I can reduce the number of logic depth more that is one that means is very much; that means important aspect. So, you can that means, further you can just look into that how we can do that. Then fan out is less important than fan in because if we restrict the fan in so, at that time the levels will increase. So, it is possible to examine a variety of topologies with restricted and varied fan in. So, that is also another that means way direction of further research.

Then driving strength and the logical effort rules were overlooked and at least neglected. It is possible to create number of topologies like this logical rules into account and it is further possible to combine the rules with compound domino implementation taking advantage of two different rules governing dynamic and static. It is still possible to produce a better adder. So that means, there is a, so, considering this particular that means, adders things we can take; that means, different different combination of this prefix addition ok.

So, different what will be the combination, that means, what will be the optimized combination of so that I can get in terms of minimized number of this dot and semi dot operation along with the logical depth will be on the reduce of on the lower side ok. So, both the things if we can optimize at that time obviously, I will get one very good adder architectures ok. So, along with this, this is the end of this adder architecture. In the



future, that means, we will again start with a new chapter in the next class, which is this different.

We will see now this multiplier architecture, different multiplier architecture that we will see so.

Thank you for today.