

EECS 151/251A

Discussion 9

Zhaokai Liu

Agenda

- Adders

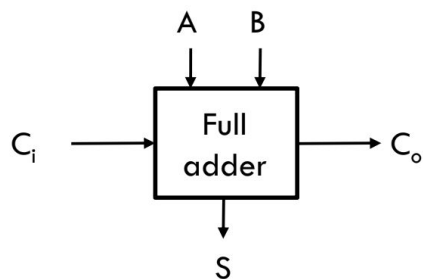
- Single-bit Full Adder
- Ripple-carry Adder
- Carry-bypass Adder
- Carry-lookahead Adder
- CLA Trees

- Multipliers

- Array Multiplier w/o and w/ CSA
- Wallace Tree
- Booth Recording
- Baugh-Wooley Multiplication

Adders

Single-bit Full Adder



$$S = A \oplus B \oplus C_i$$

$$S = A \bar{B} \bar{C}_i + \bar{A} B \bar{C}_i + \bar{A} \bar{B} C_i + A B C_i$$

$$C_o = A B + B C_i + A C_i$$

a	b	c_i	c_{i+1}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

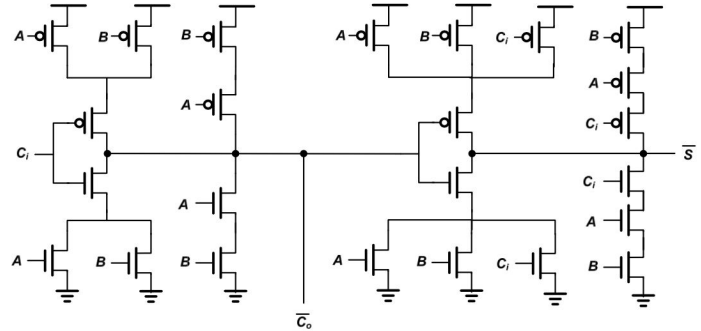
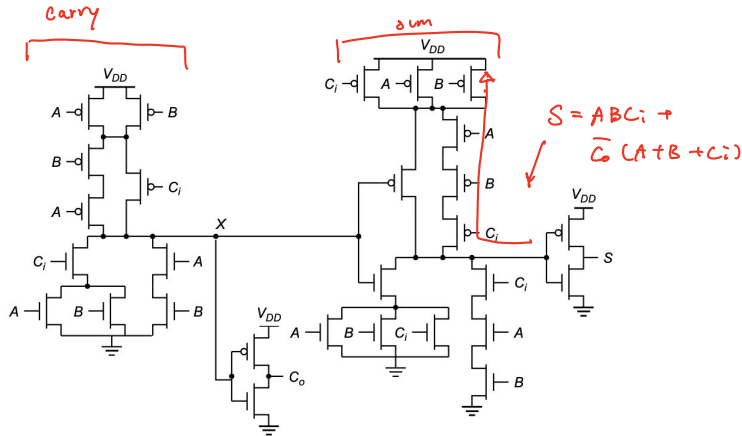
- A **full adder** implements a single-bit adder with carry in
- A **half adder** doesn't have a carry in, but still has a carry out
- The full adder is the primitive used in many adder topologies

Static CMOS Full Adder

Direct mapping of logic function

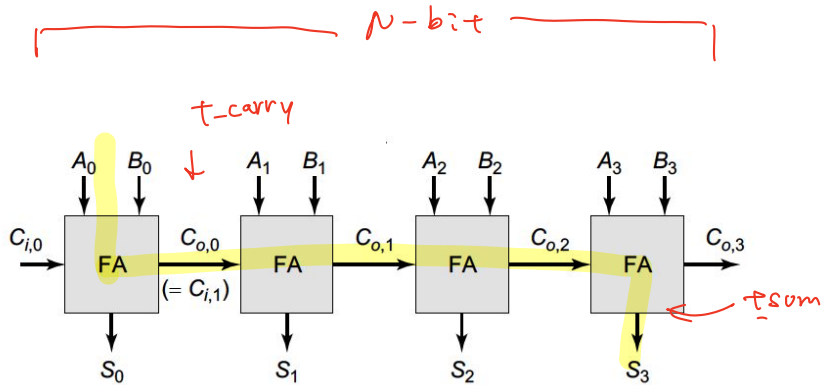


A better structure: The mirror adder



Ripple-carry Adder

For a 1-bit added, assume: $t_{\text{sum}} > t_{\text{carry}}$



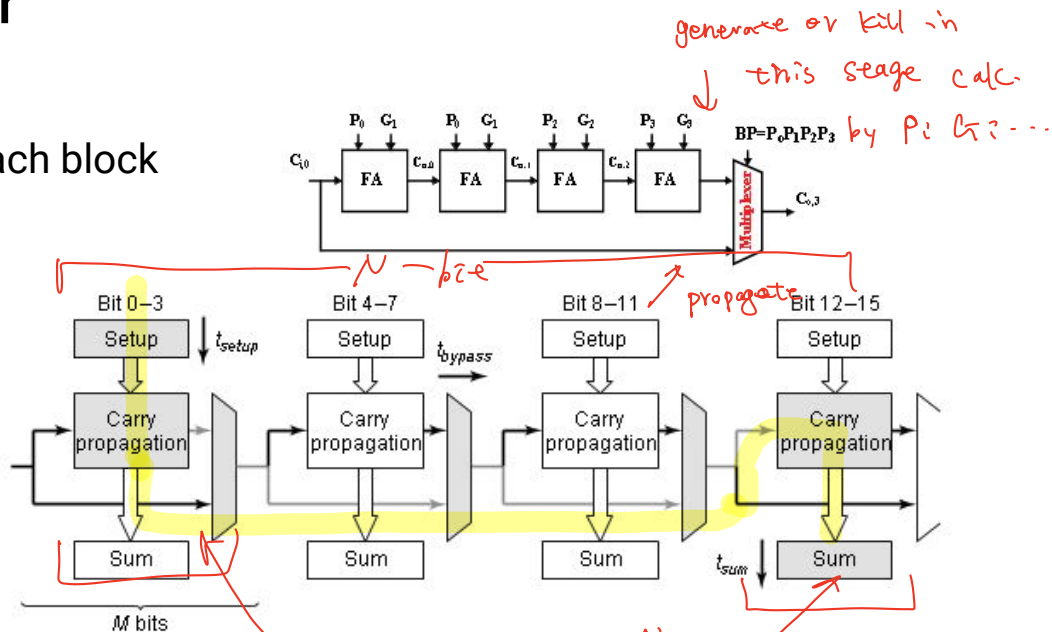
What's the critical path here?

$$(N-1) \cdot t_{\text{carry}} + t_{\text{sum}}$$

Carry-bypass Adder

- N inputs/M bits in each block

a	b	c_i	c_{i+1}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



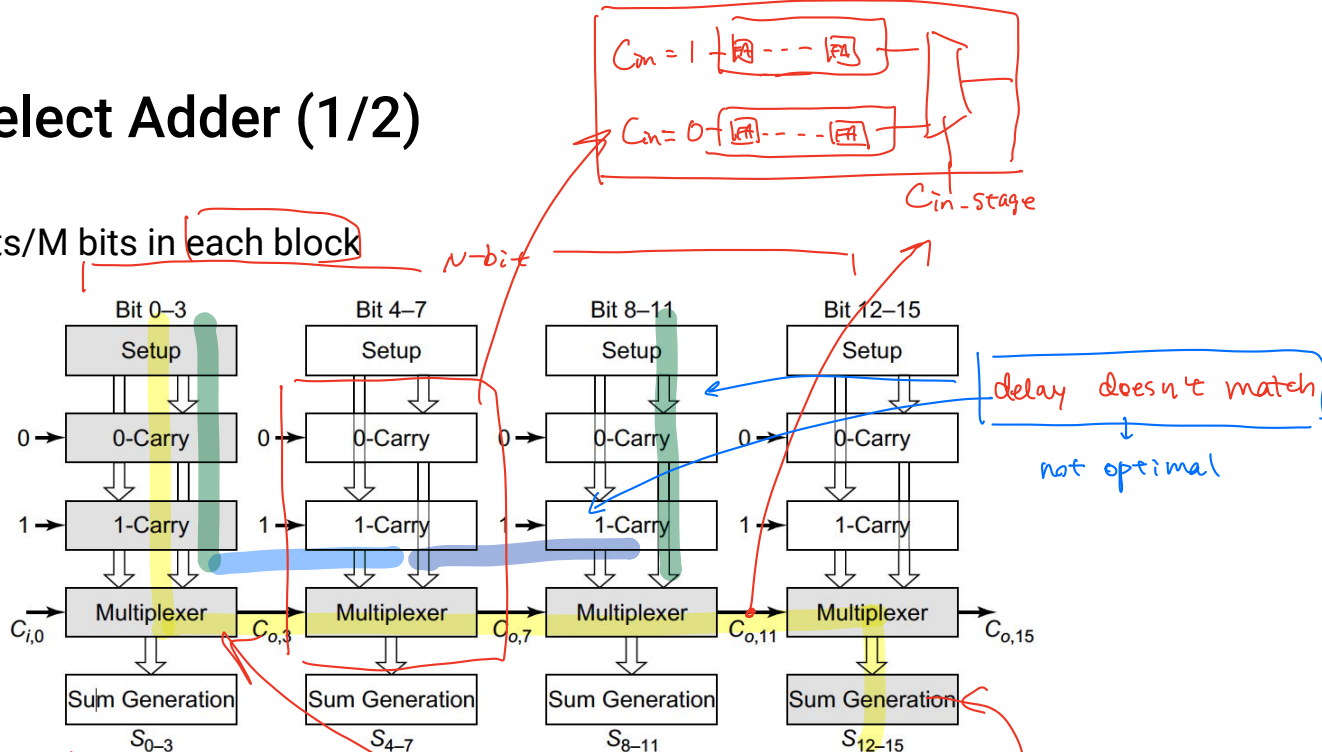
- **What's the critical path here?**

$$t_{\text{setup}} + M \cdot t_{\text{carry}} + \left(\frac{N}{M} - 1\right) \cdot t_{\text{mux}}$$

$$(M-1) t_{\text{carry}} + t_{\text{sum}}$$

Carry-select Adder (1/2)

- N inputs/M bits in each block



- What's the critical path here? $t_{setup} + M t_{carry} + \frac{N}{M} t_{mux} + t_{sum}$

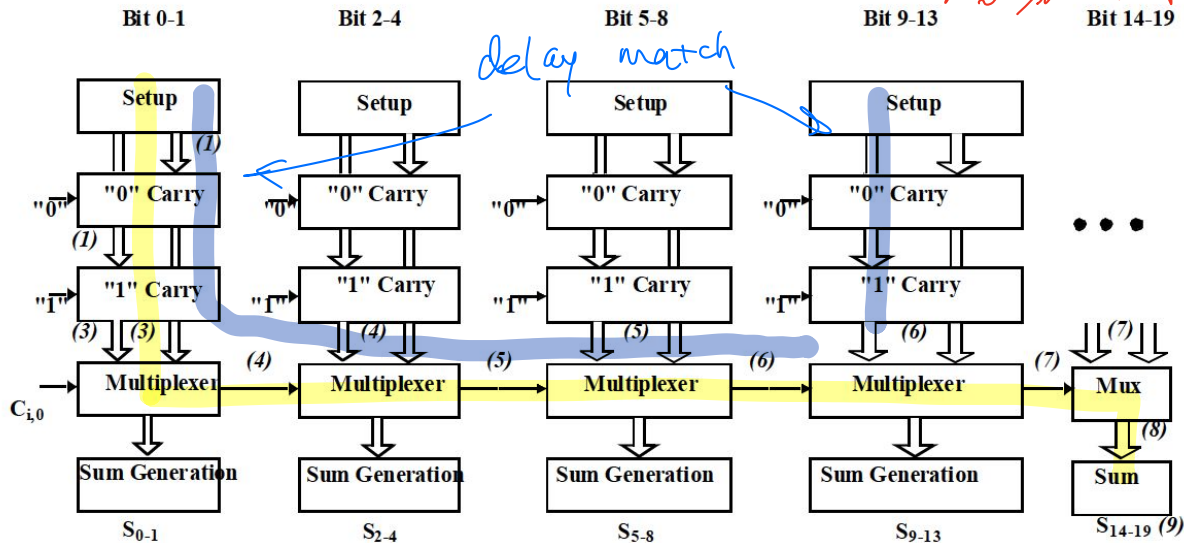
Carry-select Adder (1/2)

$\text{idx} \quad 1 \quad 2 \quad 3 \quad \dots \quad P$
 $\# \text{ bits} \quad M \quad M+1 \quad \dots \quad M+P-1$

$$N = M + (M+1) + \dots + (M+P-1)$$

when M is small N is large
 $N \approx \frac{P^2}{2} \Rightarrow P = \sqrt{2N}$

- N inputs/M bits in the first stage, P stages

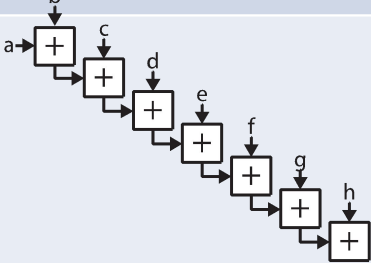
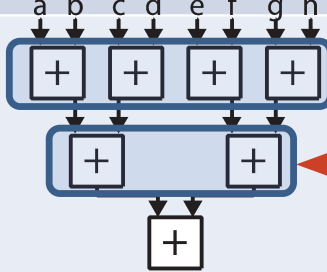


- What's the critical path here?

$t_{\text{setup}} + M t_{\text{carry}} + P t_{\text{mux}} + t_{\text{sum}}$
 ($M=2$ here)

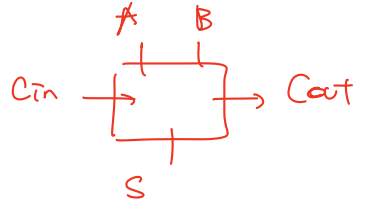
Quick Aside: Associativity

- An operator, #, is associative iff: $(a \# b) \# c = a \# (b \# c)$
- Addition*, multiplication, AND, OR, XOR, are associative
- Allows for tree computation
- Ex. $a + b + c + d + e + f + g + h$

$(((((a+b) + c) + d) + e) + f) + g) + h$	$((a+b) + (c+d)) + ((e+f) + (g+h))$
Delay: 7 Additions HW Requirements: 7 Adders	Delay: 3 Additions HW Requirements: 7 Adders
	 <p>Adders in same layer can be computed in parallel!</p>

Carry-Lookahead Adder: Redefining FAs

- **Problem: carry logic not associative -> linear FA chain**
- **Solution: re-define FAs to generate 2 new signals**
 - **g** (Generate): True if adder is guaranteed to **generate a carry**
$$g_i = a_i \cdot b_i$$
 - **p** (Propagate): True if carry-out equals carry-in (**propagate carry-in**)
$$p_i = a_i \wedge b_i$$
- Both **g** & **p** have no dependence on carry-in (**c_i**)



$$G = A \cdot B$$

$$P = A \wedge B$$

Carry-Lookahead Adder: Redefining FAs

- Sum & carry-out of FA defined in terms of these new signals
 - Sum is true if:
 - A single input is true, carry-in is false
 - Inputs are both 0 or 1, carry-in is true

$$s_i = p_i \oplus c_i$$

- Carry-out is true if:
 - Carry generate is true
 - Propagate is true and carry-in is true

$$c_{i+1} = g_i + p_i \cdot c_i$$

- However, sum and carry-out depend on carry-in

Carry-Lookahead Adder: Redefining FAs

- **Problem: carry logic not associative -> linear FA chain**
- **Solution: re-define FAs to generate 2 new signals**
 - **g** (Generate): True if adder is guaranteed to **generate a carry**
$$g_i = a_i \cdot b_i$$
 - **p** (Propagate): True if carry-out equals carry-in (**propagate carry-in**)
$$p_i = a_i \wedge b_i$$
- Both **g** & **p** have no dependence on carry-in (**c_i**)

Carry-Lookahead Adder: Redefining FAs

- Sum & carry-out of FA defined in terms of these new signals
 - Sum is true if:
 - A single input is true, carry-in is false
 - Inputs are both 0 or 1, carry-in is true

$$s_i = p_i \oplus c_i$$

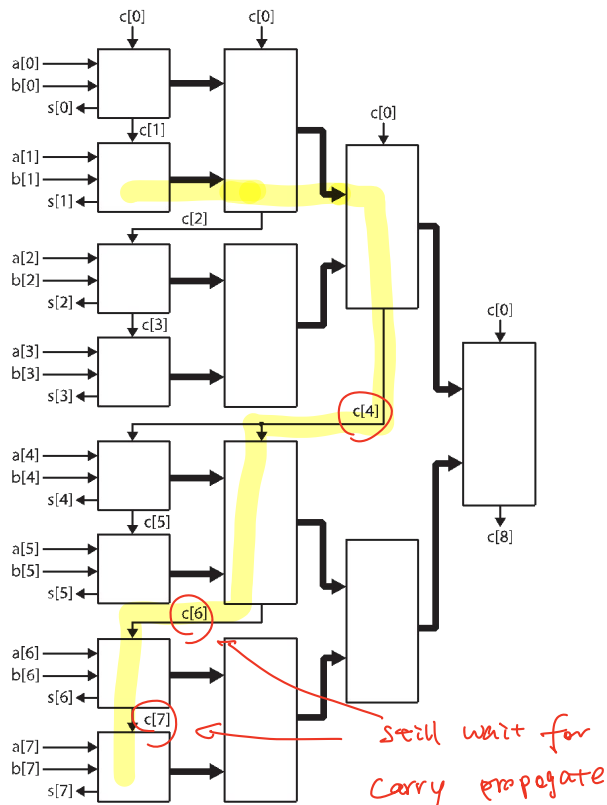
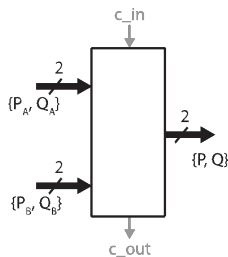
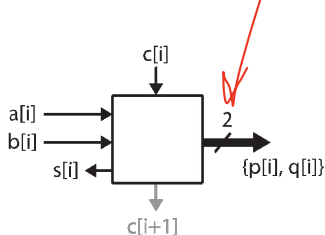
- Carry-out is true if:
 - Carry generate is true
 - Propagate is true and carry-in is true

$$c_{i+1} = g_i + p_i \cdot c_i$$

- However, sum and carry-out depend on carry-in

CLA: Tree Structure

- Smallest blocks are modified full adders
- Calculate g and p immediately
- Wait for carry-in to compute sum bit
- Some FAs are required to create carry-out



CLA: Grouping

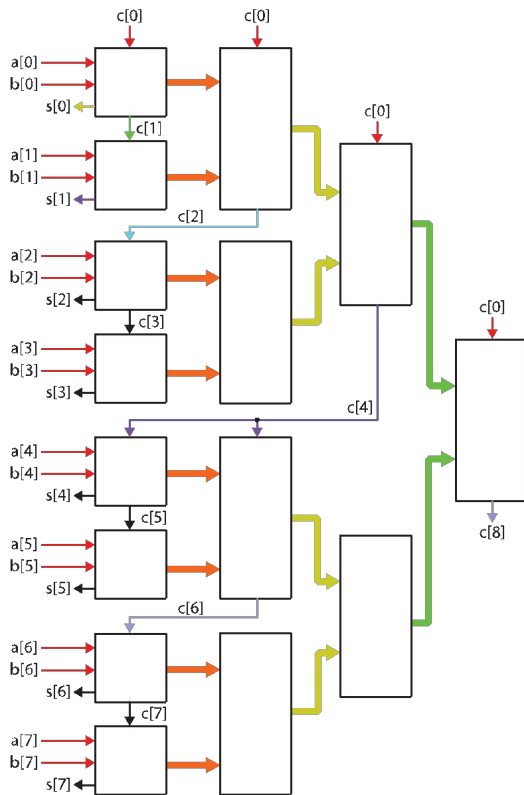
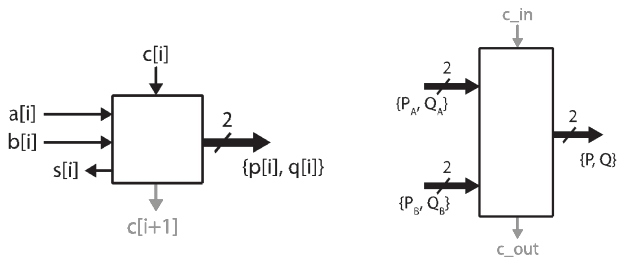
- Group together adders and create P & G for higher levels of the hierarchy.

- P = entire group propagates a carry
- G = entire group generates a carry
- P & G can be computed without carry-in

- Carry-in required to generate carry-out

- $P = P_A \cdot P_B$
- $G = G_B + G_A \cdot P_B$
- $C_{\{out\}} = G + C_{\{in\}} \cdot P$

group P_i and G_i



Parallel Prefix Adder

- Remaining problem: CLA as described still ripples carry through groups in first layer of tree
- Solution: unroll the expression for the carry bit

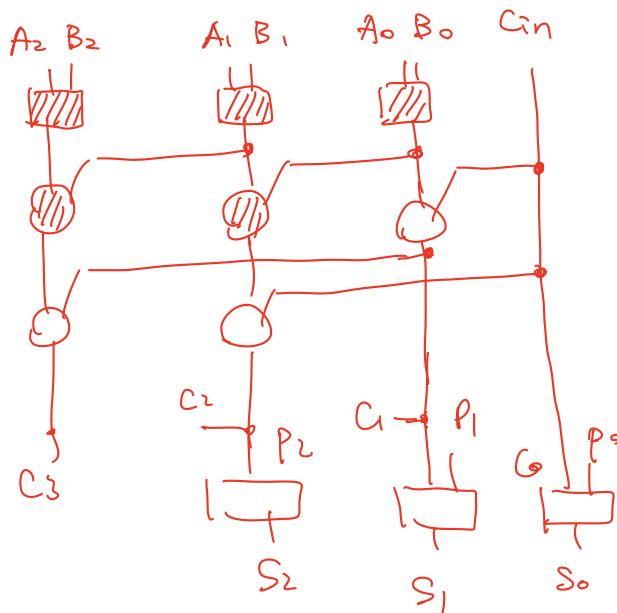
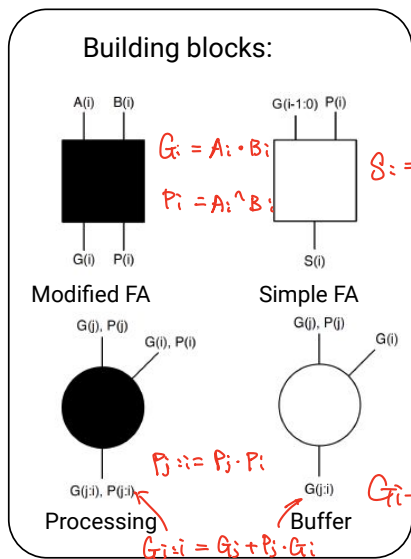
- o $c_0 = 0$ (unsigned) $c_0 = 1$
- o $c_1 = g_0 + p_0 \cdot c_0 = g_0$ $c_1 = g_0 + p_0$
- o $c_2 = g_1 + p_1 \cdot c_1 = g_1 + p_1 g_0$ $c_2 = g_1 + p_1 (g_0 + p_0) = g_1 + g_0 p_1 + p_0$
- o $c_3 = g_2 + p_2 \cdot c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0$ \vdots
- o $c_4 = g_3 + p_3 \cdot c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$ \vdots
- o Recall p's and g's can be computed in parallel (not dependent on carry-in)
- o These operations are associative \rightarrow "prefix tree" (parallel) computation!

- Overall flow:

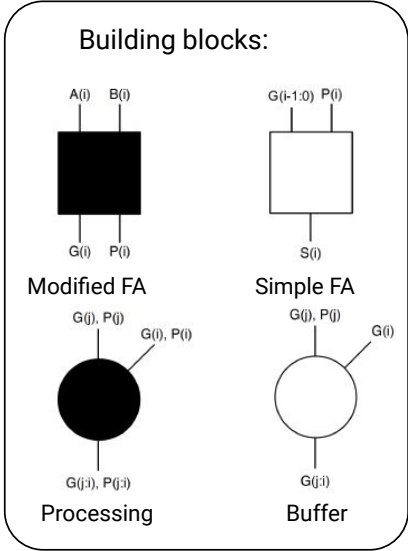
- o Break into group of bits \rightarrow Precalculate P_i, G_i in each group \rightarrow combine the groups in a tree structure \rightarrow calculate carries in parallel \rightarrow simple full adder to generate sum

generate p_i and g_i \longrightarrow group of $G_{j:i} \quad P_{j:i} \longrightarrow$ calc. S_i and C_i

Prefix Tree Adder Graphs: 3-bit Kogge-Stone adder



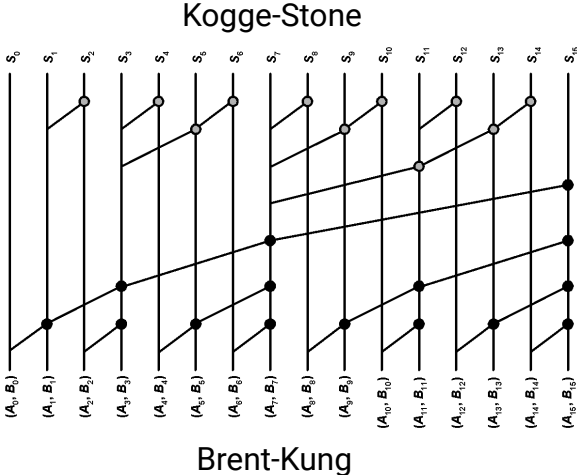
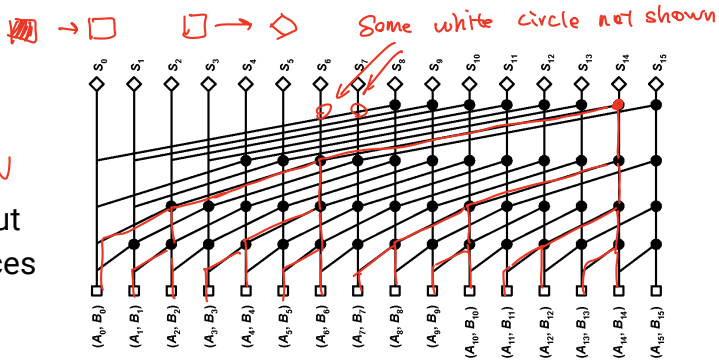
Prefix Tree Adder Graphs



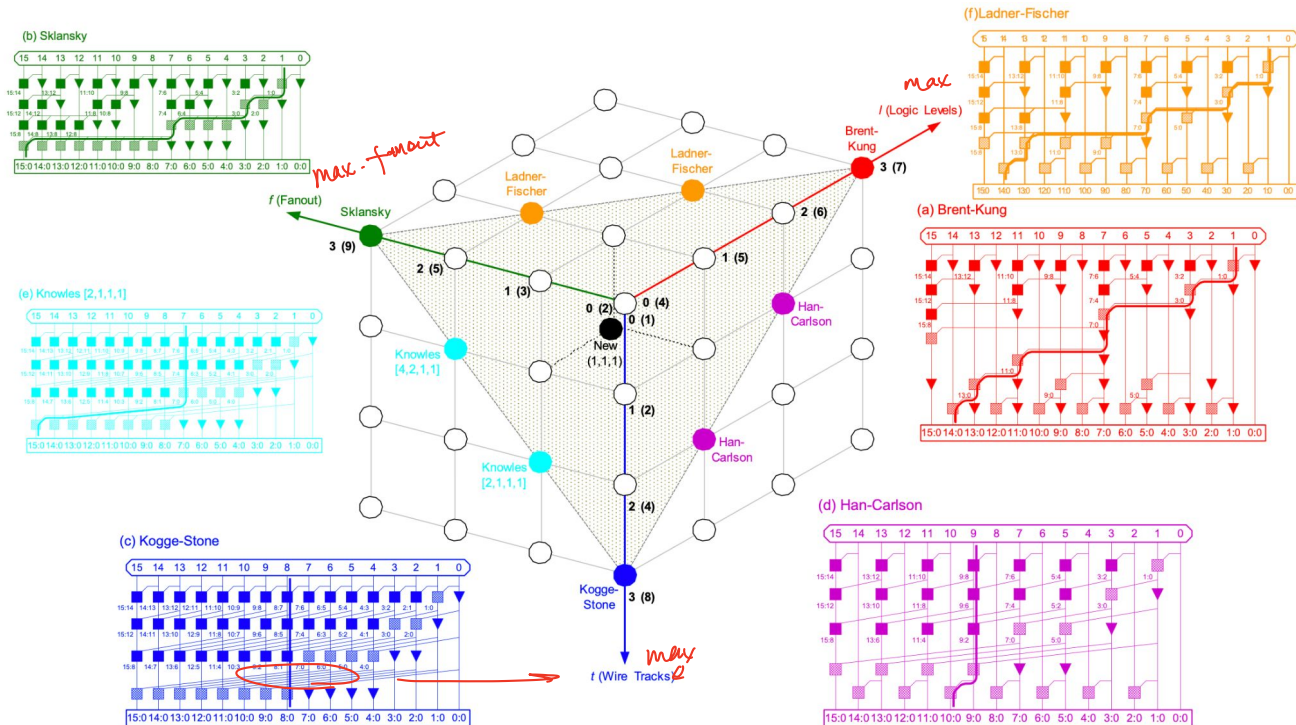
Min. critical path & fanout
Cost: more logic resources

Tradeoff logic reuse w/
critical path

Most reuse (min. area)
Cost: longest critical path



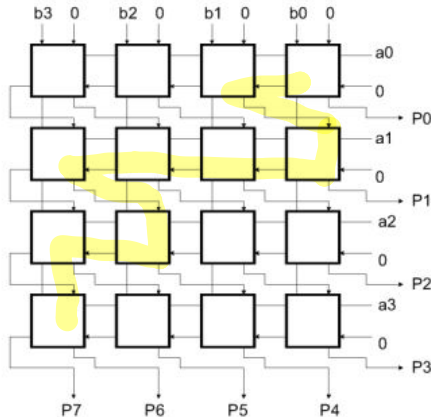
“A Taxonomy of Parallel Prefix Networks”, Harris [2003]



Multipliers

Unsigned Multiplication Example

- Partial Products can be generated in parallel
- Challenge: improve the addition of partial products



$$\begin{array}{r} 4'b0011 \quad (3) \\ * 4'b0110 \quad (6) \\ \hline \end{array}$$

$$\begin{array}{r} 0000 \\ 0011 \\ 0011 \\ + 0000 \\ \hline 00010010 \quad (18) \end{array}$$

Partial Products

What's the critical path here?

Carry-Save Addition

$$\begin{array}{r}
 C_i \quad 0000 \\
 a \quad 0110 \\
 b \quad 0101 \\
 \hline
 C_o \quad 1000 \\
 S \quad 0011
 \end{array}
 \quad a+b=6+5$$

- When we generate a carry in a given column, add it to the 2 values in the next column.

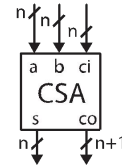
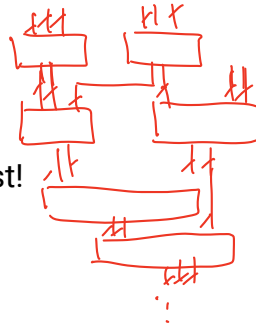
- $S_i = A_i \oplus B_i \oplus C_i$
- $C_{\{0, i+1\}} = A_i B_i + A_i C_i + B_i C_i$
- Delay adding carry bits until the end

- Basis of CSA:

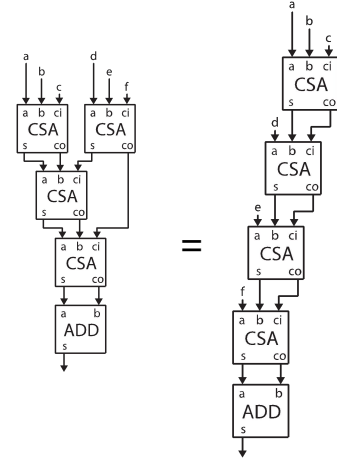
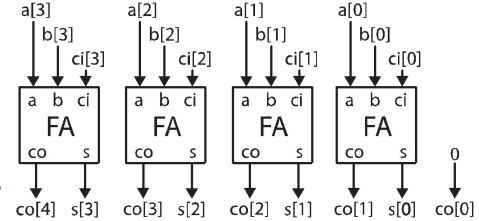
- Takes in a, b, c_{in} (multi-bit)
- Produces a sum and c_{out} (multi-bit)

- Benefits:

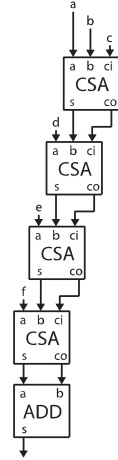
- CSAs have no carry ripple \Rightarrow small & fast!
- Only 1 standard CLA/PPA at end
- Addition is associative \Rightarrow trees!**



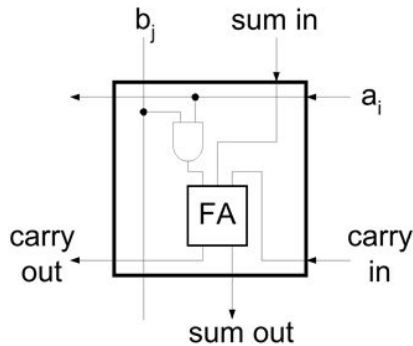
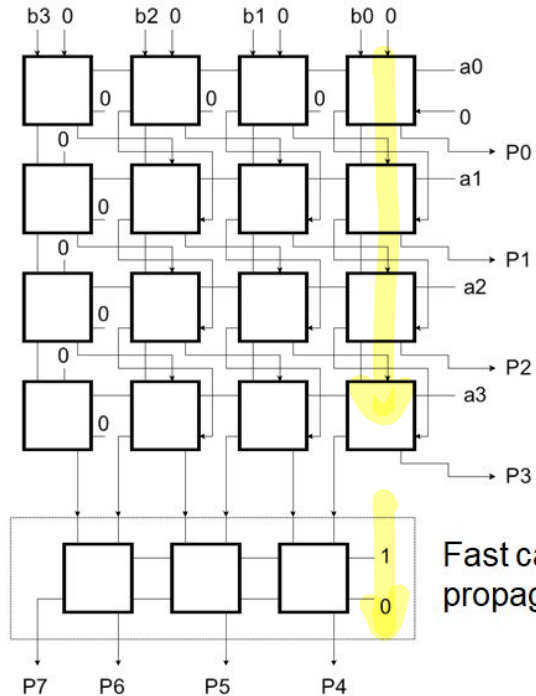
3 inputs
↓
2 outputs



=



Array Multiplier w/ CSA



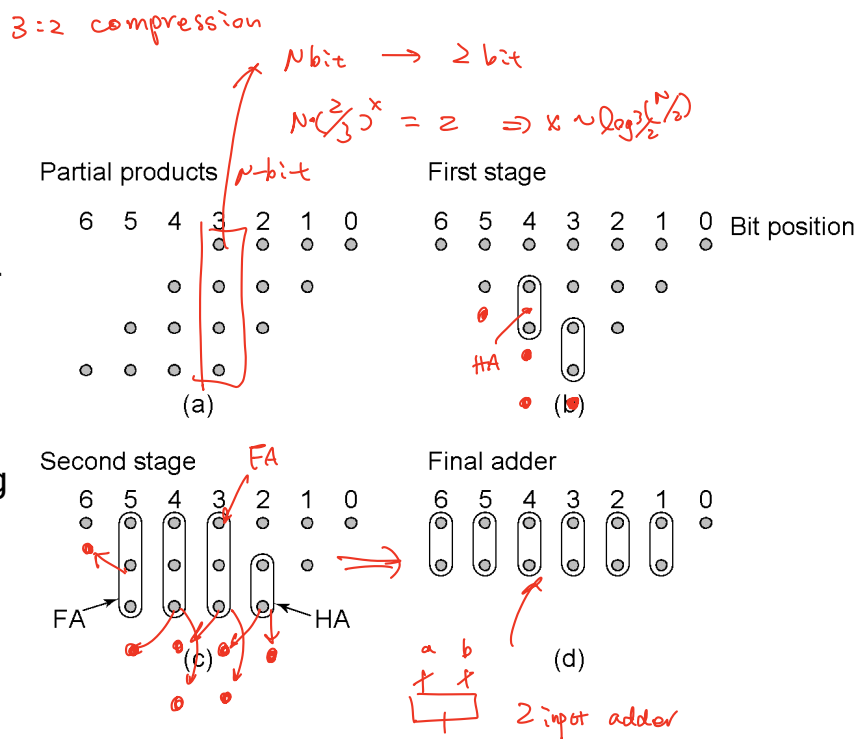
What's the critical path here?

$\sim \log_2 N$

Wallace Tree Multiplier

Method to construct Wallace Tree:

1. Draw a dot diagram where each column has as many dots as number of partial products
2. Group dots in the same column by 2 (half adder) or 3 (full adder)
3. Propagate carries and sum by adding one dot in the grouped column and one dot in the next column



Handwritten note: total delay $\sim \log_2(\frac{N}{2}) + \log_2 N$

Radix and Multiplication

- Binary multiplication \rightarrow N partial products! Can we reduce this?
 - Yes! Let's use a larger radix (think: base)
- E.g. 2 bits at a time (radix 4) \rightarrow halve number of partial products

B Digit	Partial Product	Partial Product (Rewritten)
0	$0 \cdot A$	0
1	$1 \cdot A$	A
2	$2 \cdot A$	$4 \cdot A - 2 \cdot A$
3	$3 \cdot A$	$4 \cdot A - A$

- Recall: Multiplications by powers of 2 are left shifts
 - Let's use this property!

Booth Recoding

- $4*A = A \ll 2$
- $2*A = A \ll 1$
- Recall: radix 4 multiplication \Rightarrow shift left by 2 positions for next partial product
- Therefore, any $4*A$ term can be handled in the next partial product!
 - Multiplier looks a 3 (rather than just 2) bits
 - Extra bit is MSB of the previous

B Digit	Partial Product	Partial Product (Rewritten)
0	$0*A$	0
1	$1*A$	A
2	$2*A$	$4*A - 2*A$
3	$3*A$	$4*A - A$

Booth Recoding

Carry in from last 2 bit

B_{i+1}	B_i	B_{i-1}	Action	Comment
0	0	0	Add 0	
0	0	1	Add A	Includes $+4*A$ from previous radix 4 digit = $+A$ in this position due to left shift by 2
0	1	0	Add A	
0	1	1	Add $2*A$	Includes $+4*A$ from previous round ($+A$ in this position). $*2$ is implemented as a left shift by 1
1	0	0	Sub $2*A$	$4*A$ will be added in when handling next radix 4 digit. $*2$ is implemented as a left shift by 1
1	0	1	Sub A	$4*A$ will be added in when handling next radix 4 digit. Includes $+4*A$ from previous radix 4 digit ($+A$ in this position) $2+1=3=4-1 \Rightarrow \text{Sub } A$
1	1	0	Sub A	$4*A$ will be added in when handling next radix 4 digit.
1	1	1	Add 0	$4*A$ will be added in when handling next radix 4 digit. Includes $+4*A$ from previous radix 4 digit ($+A$ in this position) $2+1+1=4 \Rightarrow \text{left to next 2-bit}$

weight 2 1 1

Booth Recoding Example (Unsigned)

- $6 * 7$
- $B_{-1} = 0$

$$\begin{array}{r}
 4'b0110 \quad (6) \\
 * 4'b0111 \quad (7) \\
 \hline
 - \quad \quad 0110 \quad (\text{Sub } A) \\
 + \quad 01100 \quad (\text{Add } 2A) \\
 + \quad \cancel{0000} \quad (\cancel{\text{Add } 0}) \\
 \hline
 + \quad 1111010 \quad (\text{Sub } A) \\
 + \quad 01100 \quad (\text{Add } 2A) \\
 + \quad \cancel{0000} \quad (\cancel{\text{Add } 0}) \\
 \hline
 \end{array}$$

B_{i+1}	B_i	B_{i-1}	Action
0	0	0	Add 0
0	0	1	Add A
0	1	0	Add A
0	1	1	Add 2*A
1	0	0	Sub 2*A
1	0	1	Sub A
1	1	0	Sub A
1	1	1	Add 0

Handwritten note: $\sim A + 1$ with an arrow pointing to the first row of the table.

Signed Multiplication: Baugh-Wooley

- Recall: 2's complement MSB has negative weight
- Nominally:
 - Subtract last partial product
 - Sign-extend the rest of the partial products
- Recall 2's complement negation: $-A = \sim A + 1$
 - Add $\sim A + 1$ instead! Result: basically same hardware as unsigned mult.
 - Implementation: invert some bits, insert a 1 left of the first & last partial products

				1	p0[3]	p0[2]	p0[1]	p0[0]
+			$\sim p1[3]$	p1[2]	p1[1]	p1[0]	0	
+		$\sim p2[3]$	p2[2]	p2[1]	p2[0]	0	0	
+	1	$\sim p3[3]$	$\sim p3[2]$	$\sim p3[1]$	$\sim p3[0]$	0	0	0

P[7]	P[6]	P[5]	P[4]	P[3]	P[2]	P[1]	P[0]	