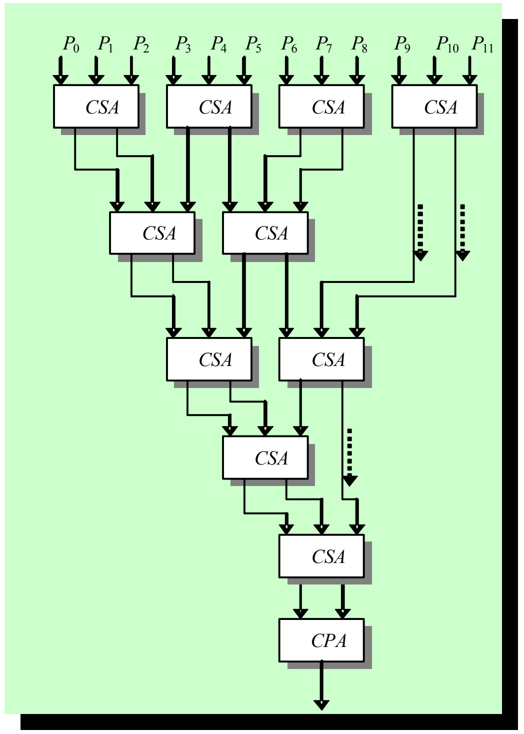
Digital Integrated Circuit Design

Lecture 12 – Building Blocks (Adders)



P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10} P_{11}
CSA
CPA

Adib Abrishamifar
EE Department
IUST

Contents

Outline

- Types of logic circuit
  - Combinational
  - Sequential
- Design methods
  - Gate level design
    - Example: Full adder
  - Block level design
    - Example: Parallel adder, ALU
- Propagation Delay

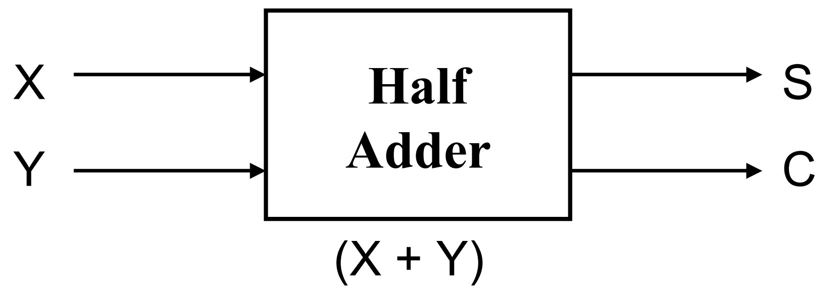
Outline

- Building Blocks for Digital Architectures
  - Arithmetic unit
    - Bit-sliced datapath (Adders, Multipliers, Shifters, Comparators, etc.)
  - Memory
    - RAM, ROM, Buffers, Shift registers
  - Control
    - Finite state machine (PLA, random logic)
    - Counters
  - Interconnect
    - Switches
    - Arbiters
    - Bus

Contents

Adders

- **Half Adder (Gate-level Design)**

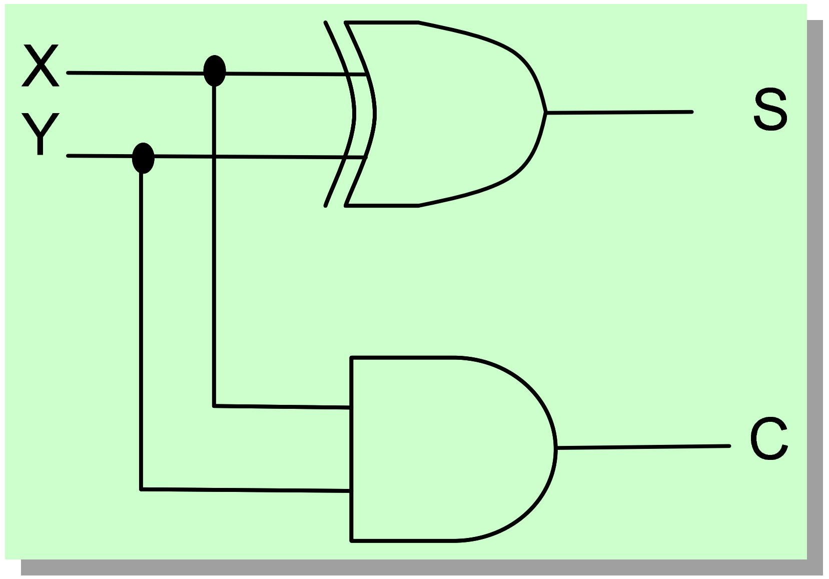


X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- C = XY
- S = X'Y + XY' = $X \oplus Y$

Adders

- **Half Adder (Gate-level Design)**

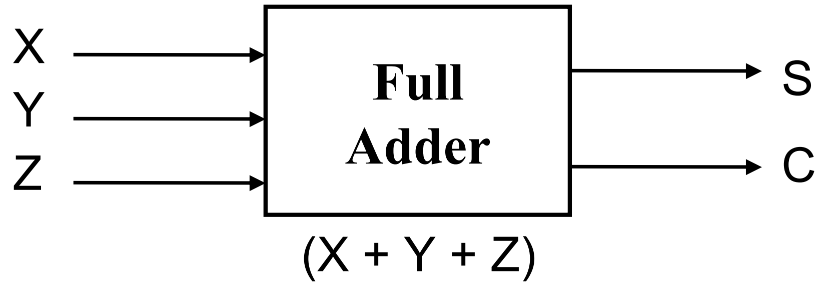


Adders

- **Half Adder**
  - **To add two binary numbers, we need to add 3 bits (including the carry)**

	1	1	1		carry
	0	0	1	1	X
+	0	1	1	1	Y
	1	0	1	0	S

- **Need Full Adder (so called as it can be made from two half-adders)**



Contents

Adders

- **Full Adder (Gate-level Design)**

X	Y	Z	C	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

C

X \ YZ	00	01	11	10
0			1	
1		1	1	1

S

X \ YZ	00	01	11	10
0		1		1
1	1		1	

  - **C = XY + XZ + YZ**
  - **S = X'Y'Z + X'YZ'+XY'Z'+XYZ**

Adders

- Full Adder
  - Alternative formula using algebraic manipulation
    - $C = XY + XZ + YZ$
    - $\quad = XY + (X + Y)Z$
    - $\quad = XY + ((X \oplus Y) + XY)Z$
    - $\quad = XY + (X \oplus Y)Z + XYZ$
    - $\quad = XY + (X \oplus Y)Z$

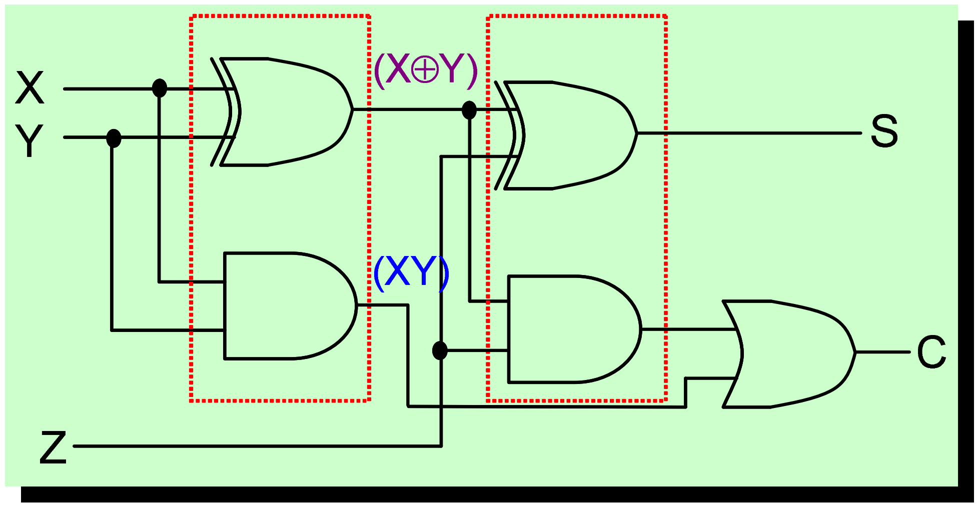
    - $S = X'Y'Z + X'YZ' + XY'Z' + XYZ$
    - $\quad = X'(Y'Z + YZ') + X(Y'Z' + YZ)$
    - $\quad = X'(Y \oplus Z) + X(Y \oplus Z)'$
    - $\quad = X \oplus (Y \oplus Z)$ or $(X \oplus Y) \oplus Z$

Adders

- **Full Adder**

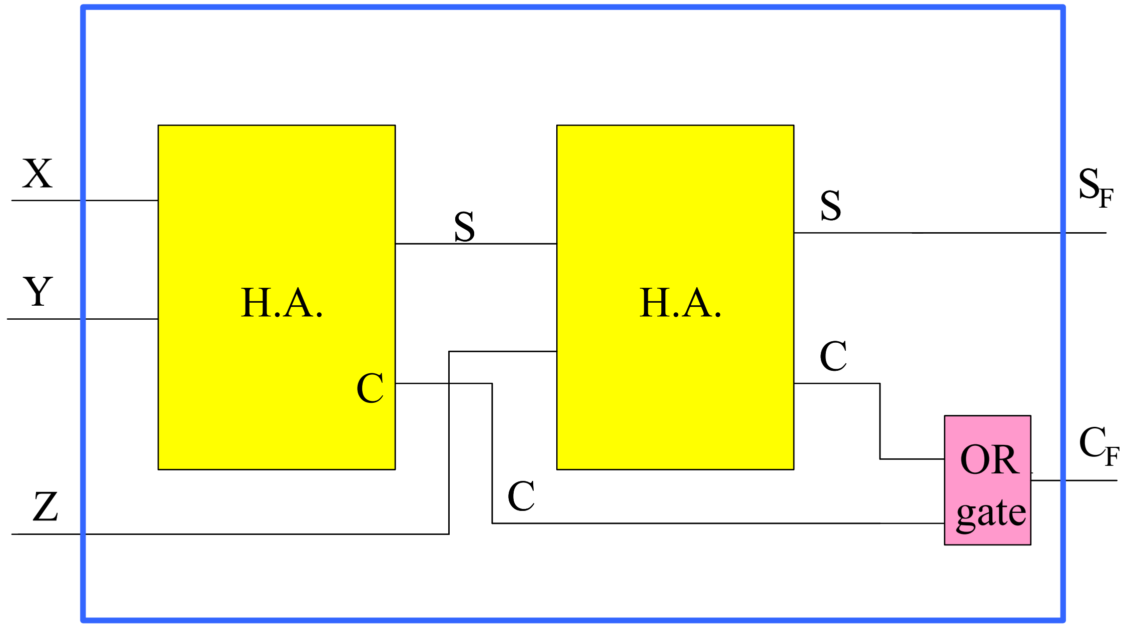
  $C = XY + (X \oplus Y)Z$

  $S = (X \oplus Y) \oplus Z$



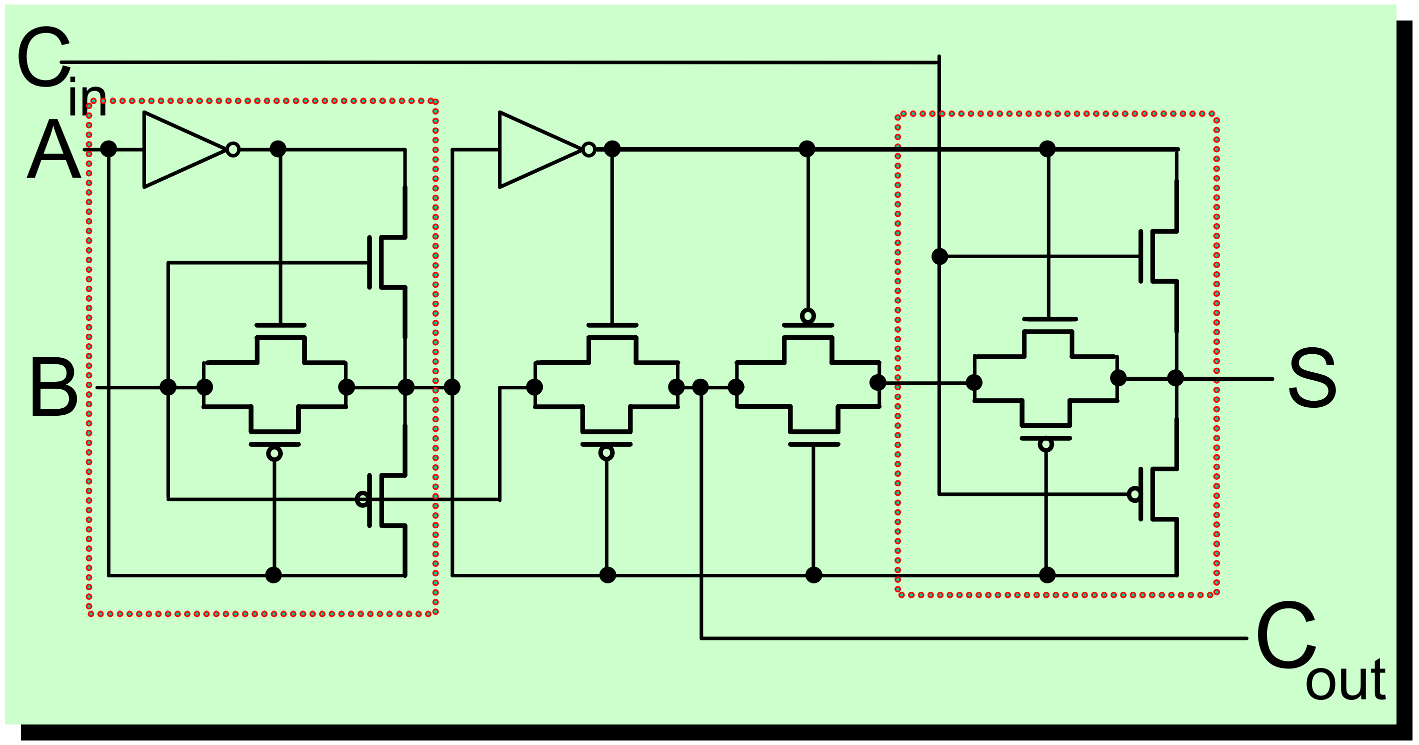
Adders

- **Full Adder made from two Half-Adders (+ OR gate)**



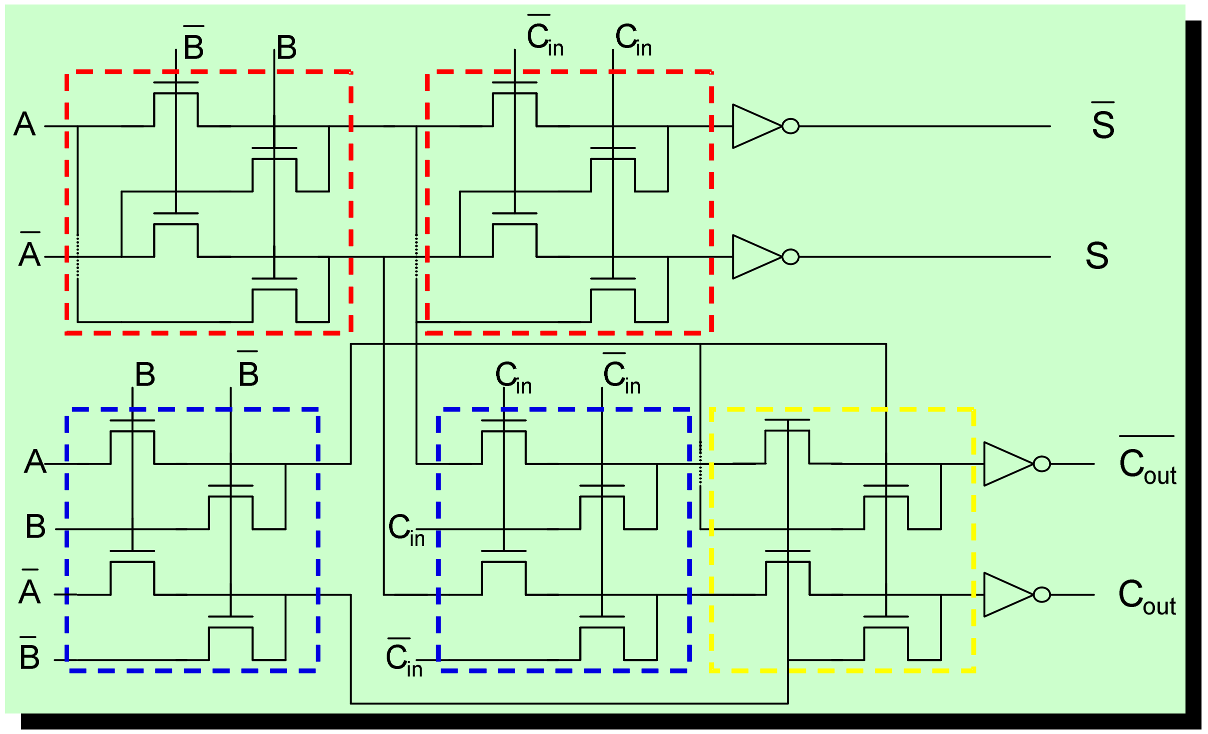
Adders

- FA Gate Level (XOR FA, 16 Transistors)



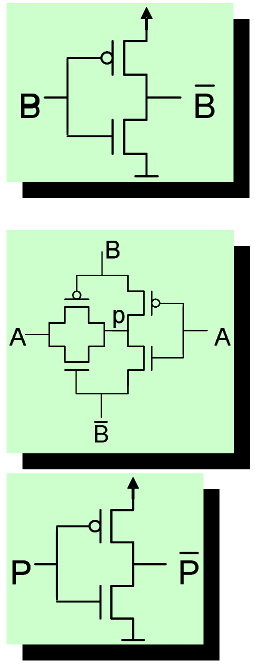
Adders

- CPL FA (20 + 8 transistors)

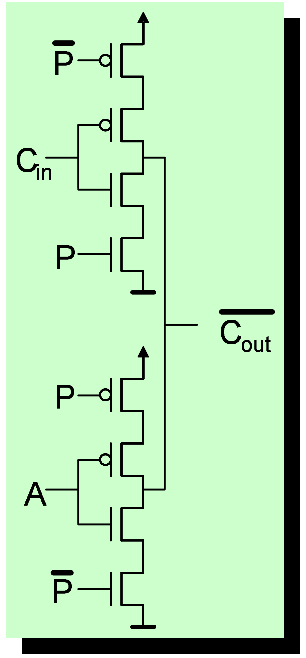


Adders

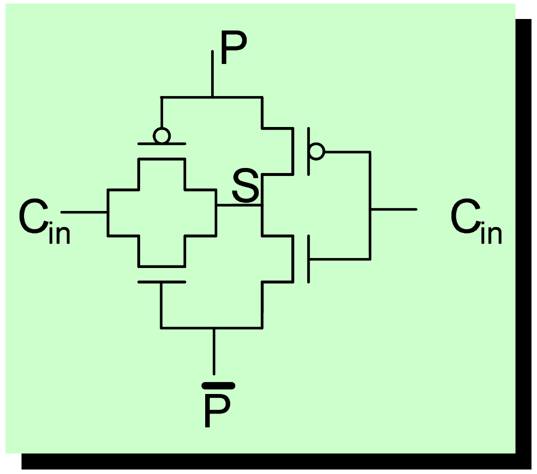
- **Delay Balanced FA (Identical Delays for Carry and Sum, 20 +2 transistors)**



Signal set-up



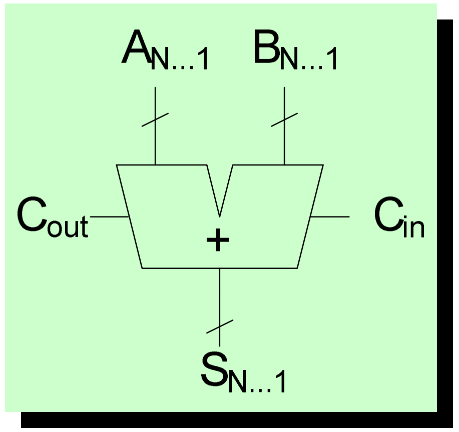
Carry generation

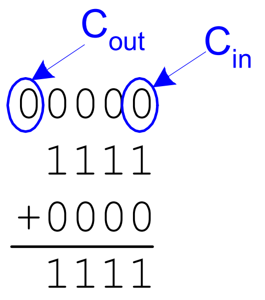


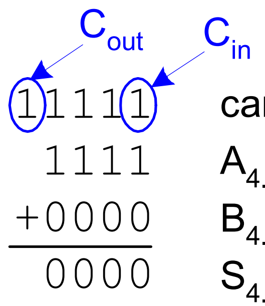
Sum generation

Adders

- **N-bit adder called CPA (Carry Propagate Adder)**
  - **Each sum bit depends on all previous carries**
  - **How do we compute all these carries quickly?**





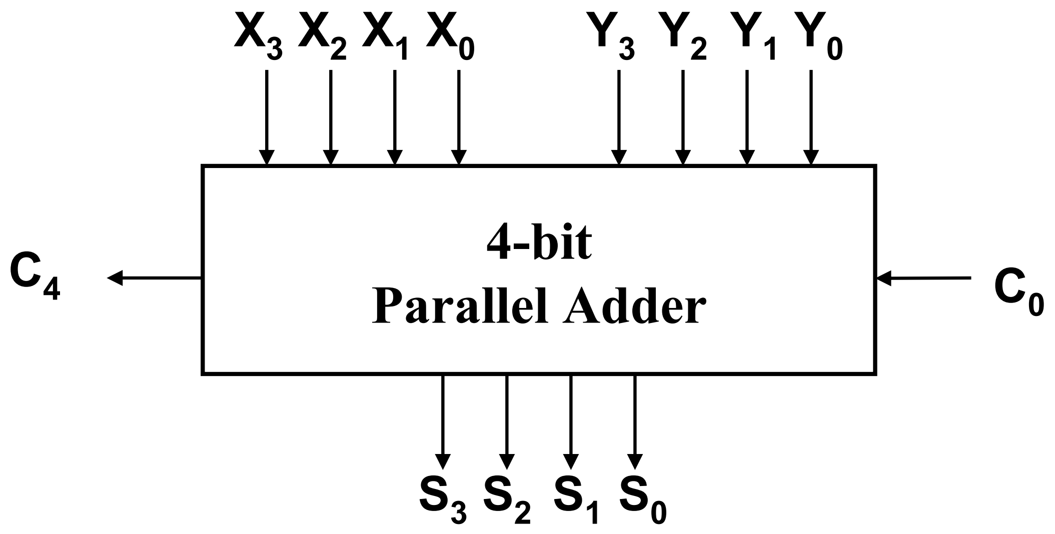


00000	11111	carries
1111	1111	$A_{4...1}$
+0000	+0000	$B_{4...1}$
1111	0000	$S_{4...1}$

Contents

Adders

- Ripple-Carry Adder (Parallel Adder)
  - Consider a circuit to add two 4-bit numbers together and a carry-in, to produce a 5-bit result

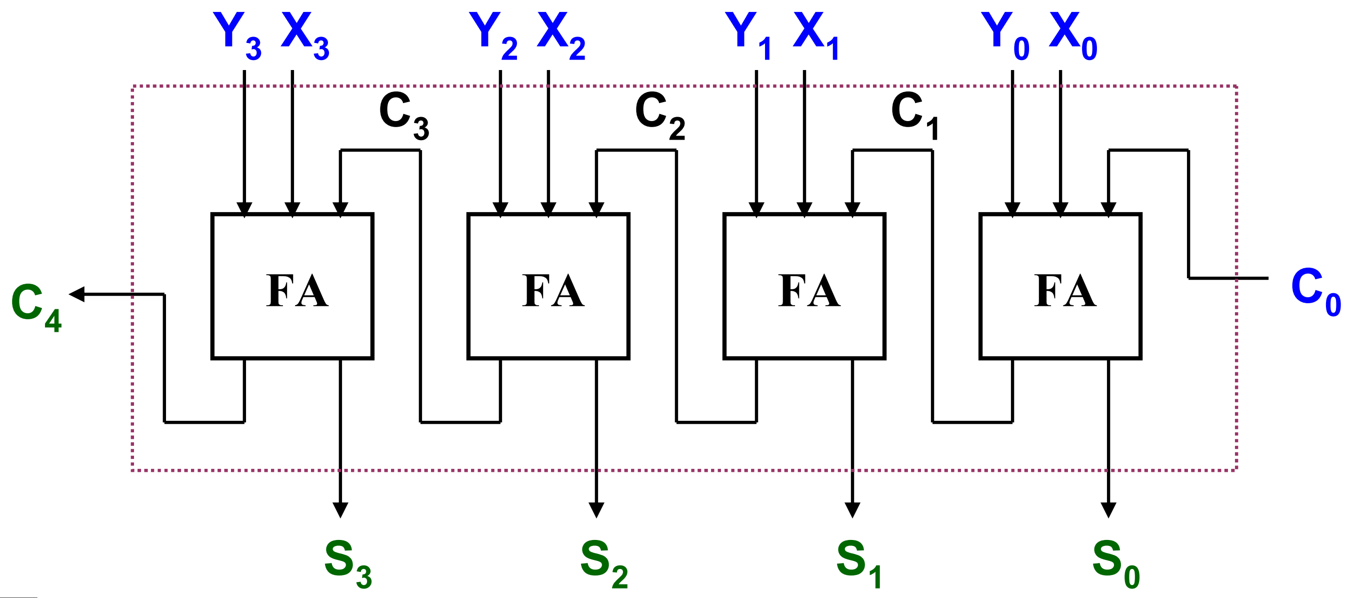


Adders

- Ripple-Carry Adder (Parallel Adder)
  - Simplest design: cascade full adders
    - Critical path goes from C_{in} to C_{out}
    - Design full adder to have fast carry delay
    - Addition formulae for each pair of bits (with carry in), has the same function as a full adder
    - $C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$
    - $S_i = X_i \oplus Y_i \oplus C_i$

Adders

- Ripple-Carry Adder (Parallel Adder)
  - Cascading 4 full adders via their carries

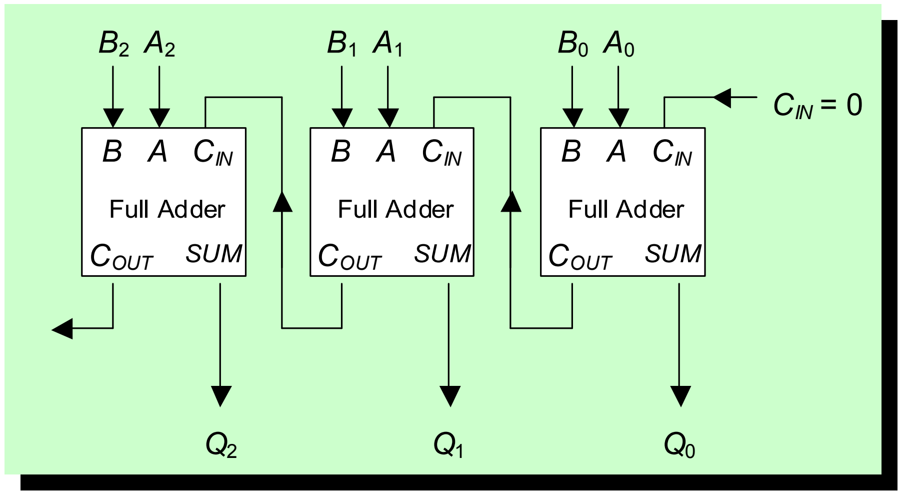


🟦 Input

🟩 Output

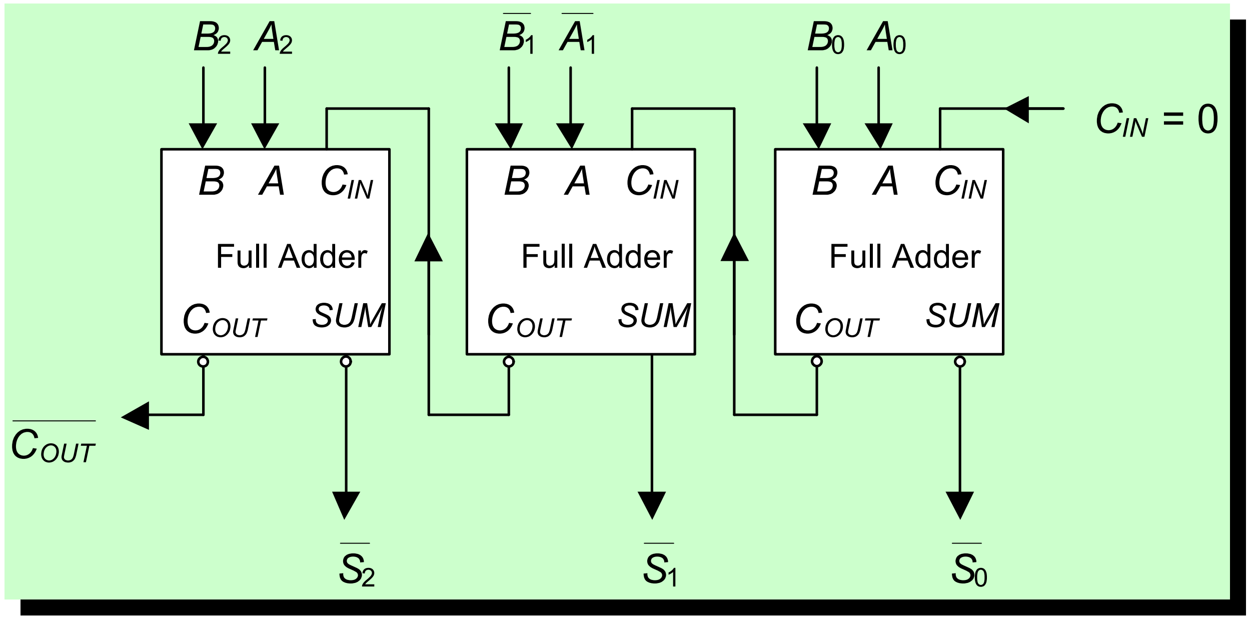
Adders

- **Ripple-Carry Adder (Parallel Adder)**
  - **Note that carry propagated by cascading the carry from one full adder to the next**
  - **Called Parallel Adder because inputs are presented simultaneously (in parallel)**



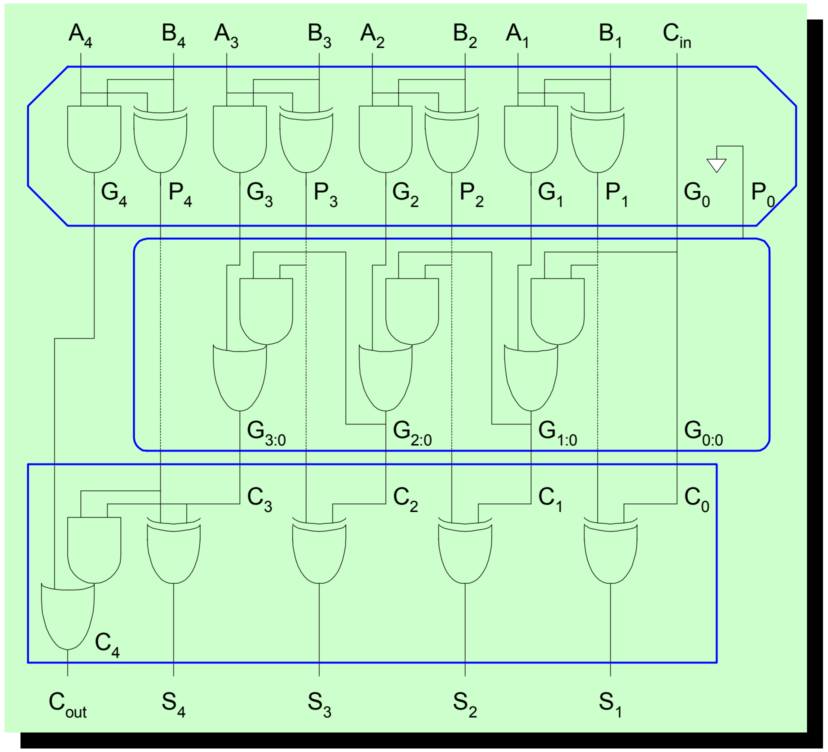
Adders

- **Ripple-Carry Adder (Parallel Adder)**
  - **Output Inverting (Carry-Sum)**



Adders

- **Ripple-Carry Revisited**
  - **$G_{i:0} = G_i + P_i \cdot G_{i-1:0}$**

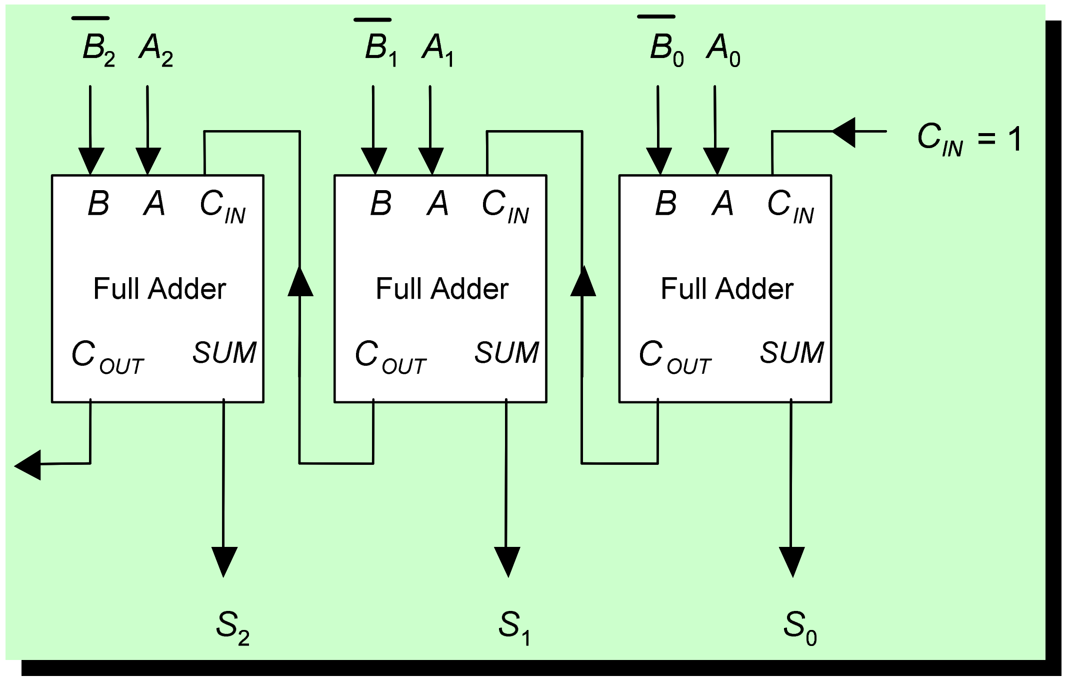


Adders

- How To Subtract ?
  - Suppose added input to the adder that gave the one's complement of B
  - What happens if set C_0 to 1 in the parallel adder?
  - Sum = A + B + 1
  - Then if select inverted B', Sum is A + B' + 1 = A + (B' + 1) = A + (-B) = A - B
  - Therefore can do subtract with the parallel adder if we add inverters and set C_0 to 1

Adders

- **Subtraction**

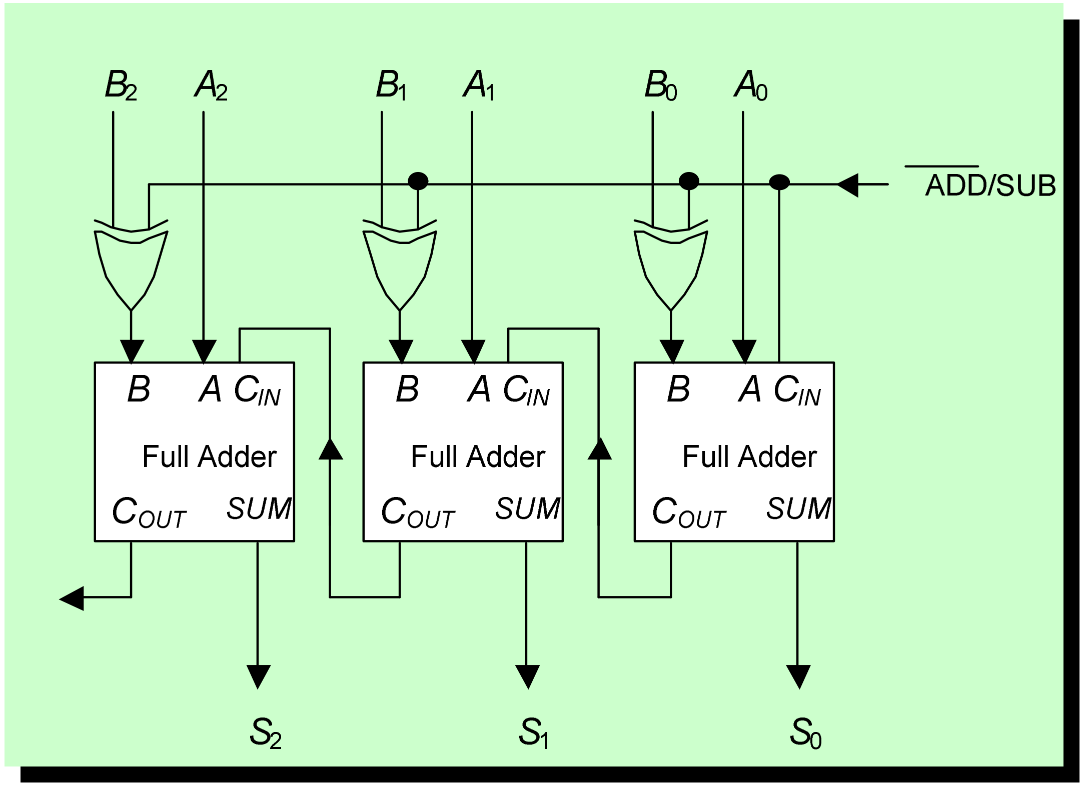


Adders

- **Add / Sub Circuits Comparison**
  - **Both the addition and subtraction circuits are based around the parallel adder**
  - **For addition:**
    - **A and B are inputted directly to the adder**
    - **$C_{IN} = 0$**
  - **For subtraction:**
    - **A is inputted directly**
    - **All the bits of B are complemented**
    - **$C_{IN} = 1$**

Adders

- **An Adder / Subtraction Circuit**



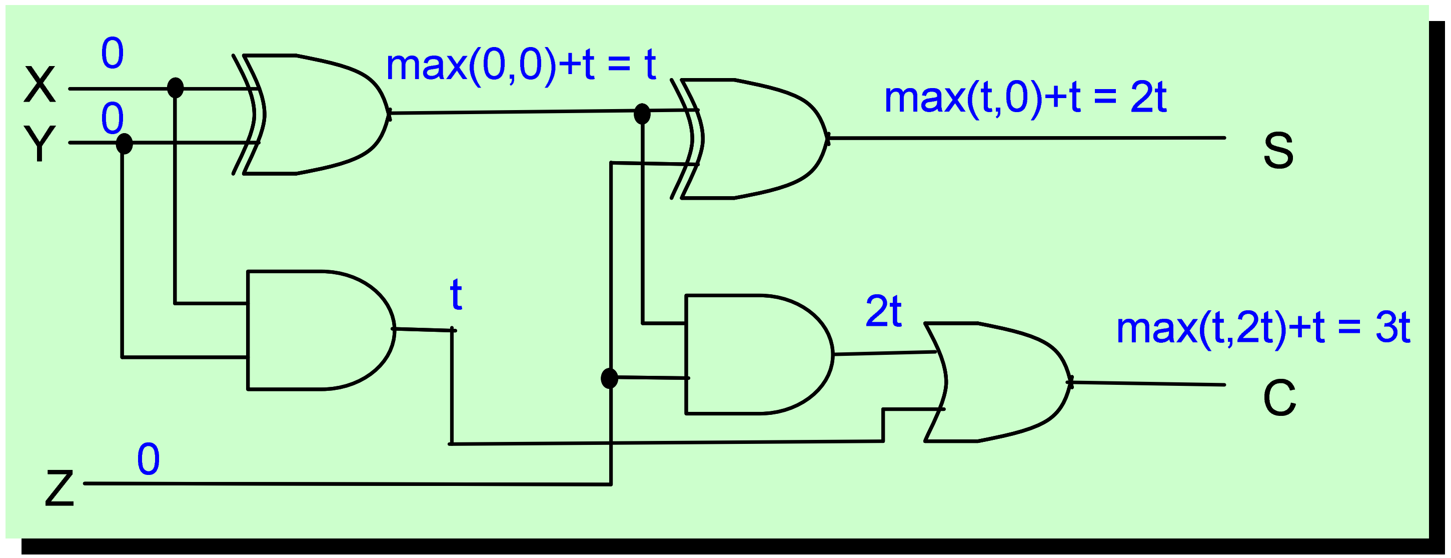
Adders

- **Ripple-Carry Adder (Parallel Adder)**
  - **Advantage:**
    - **Simple logic, so small (low cost)**
  - **Disadvantage:**
    - **Slow (O(N) for N bits) and lots of glitching (so lots of energy consumption)**
    - **Propagation delay of ripple-carry parallel adders is proportional to the number of bits it handles**
    - **Maximum Delay: $((n-1)\times 2+3)t$ (it is shown in near future!)**

Contents

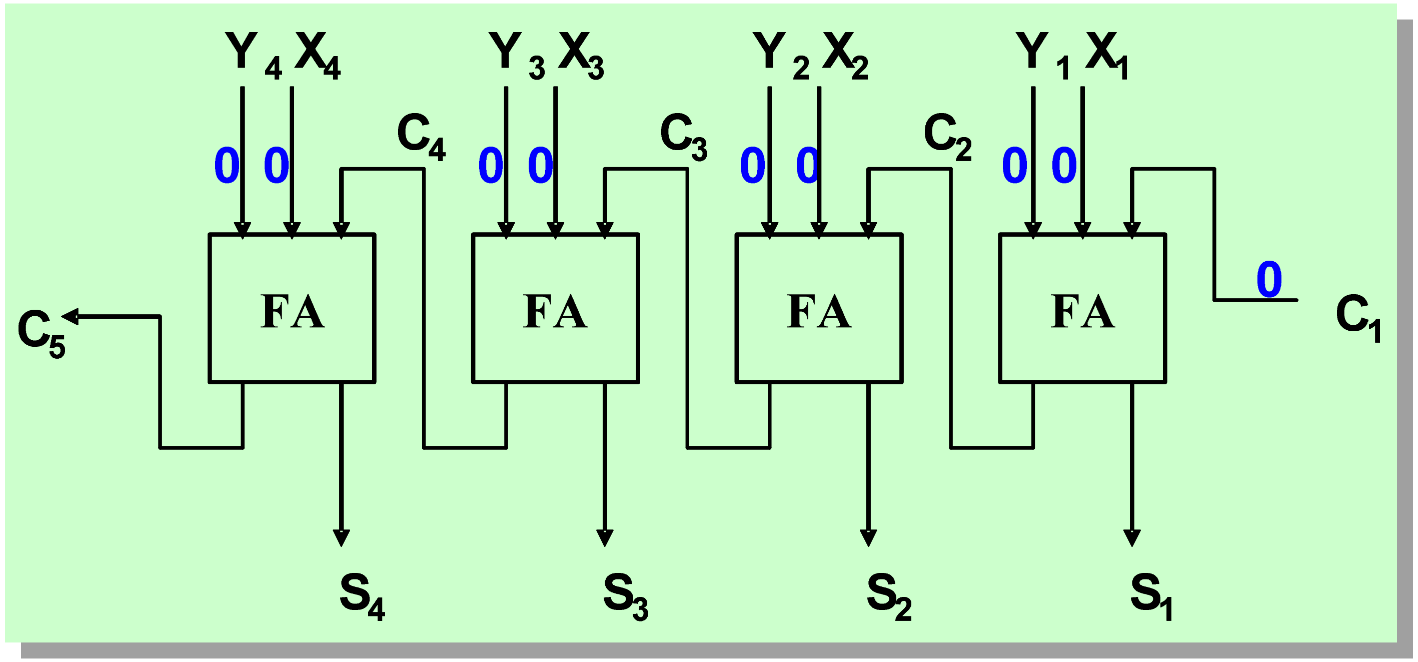
Adders

- **Calculation of Circuit Delays**
  - **As a simple example, consider the full adder circuit where all inputs are available at time 0. (Assume each gate has delay t)**



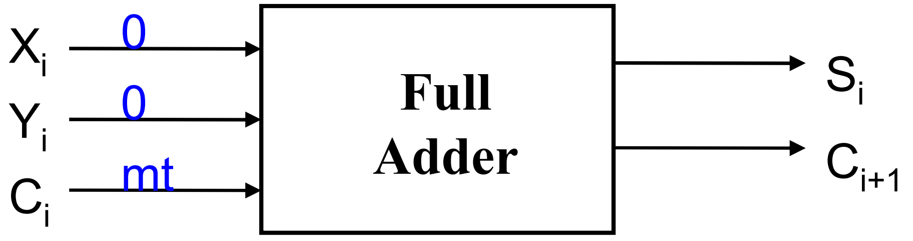
Adders

- **Calculation of Circuit Delays**
  - **More complex example: 4-bits parallel adder**



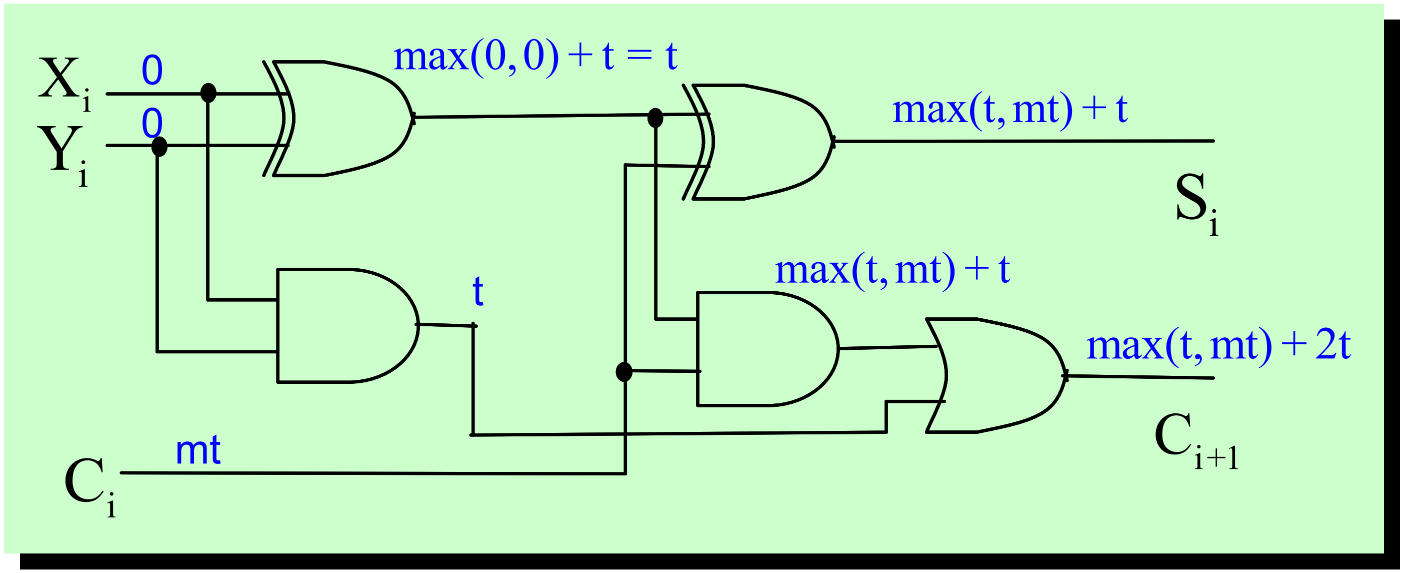
Adders

- **Calculation of Circuit Delays**
  - **Analyze the delay for the repeated block**
  - **where X_i, Y_i are stable at 0t, while C_i is assumed to be stable at mt**



Adders

- **Calculation of Circuit Delays**



Adders

- Calculation of Circuit Delays
  - When i = 1, m = 0: S1 = 2t and C2 = 3t
  - When i = 2, m = 3: S2 = 4t and C3 = 5t
  - When i = 3, m = 5: S3 = 6t and C4 = 7t
  - When i = 4, m = 7: S4 = 8t and C5 = 9t

  - In general for an n-bit ripple-carry parallel adder
  - $S_n = ((n-1)\times 2+2)t$
  - $C_{n+1} = ((n-1)\times 2+3)t$

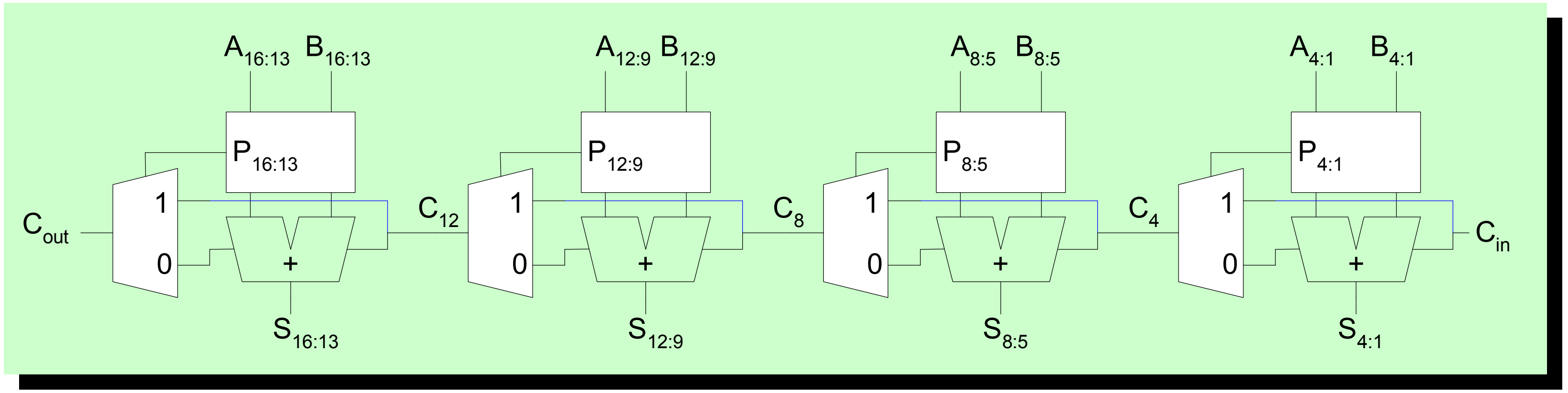
Adders

- Ways of improving the speed
  - Use better technology (e.g. ECL faster than TTL gates)
    - Faster technology is more expensive, needs more power, lower-level of integrations
    - Physical limits (e.g. speed of light, size of atom)
  - Use gate-level designs to two-level circuits! (use sum-of-products/product-of-sums)
    - Complicated designs for large circuits
    - Product/sum terms need MANY inputs!
  - Use clever (other) techniques

Contents

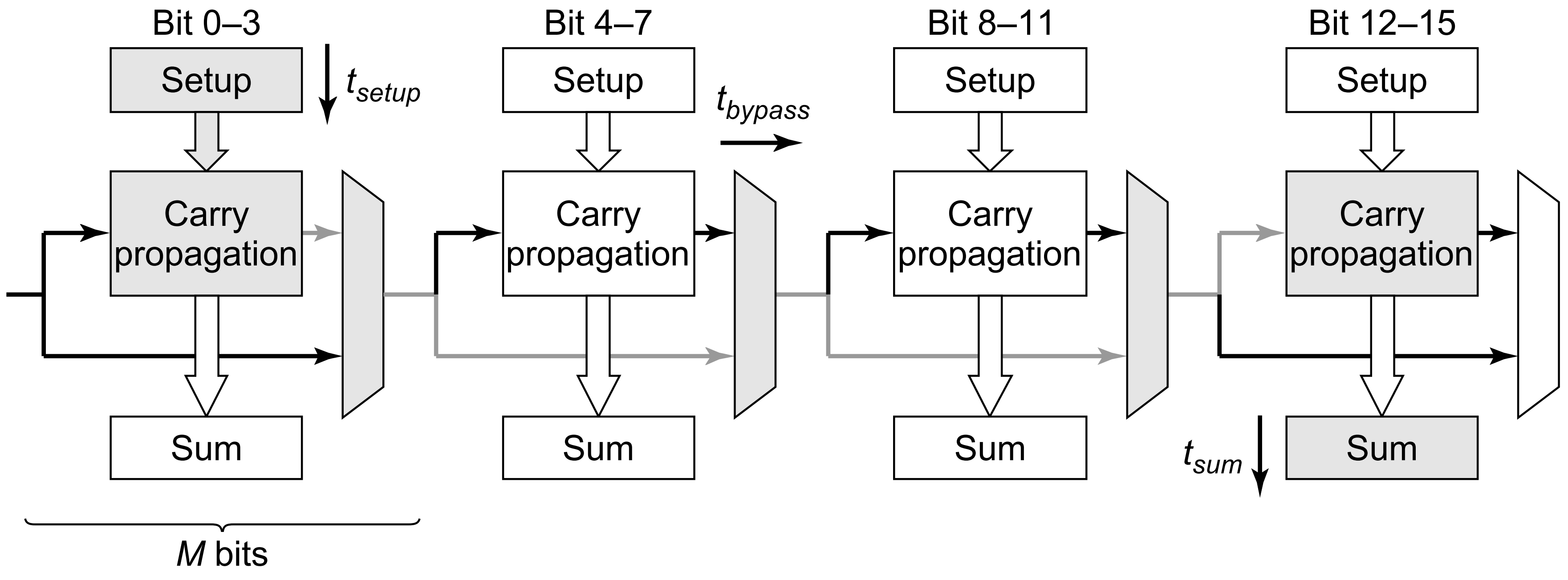
Adders

- **Carry-Bypass Adder (Carry-Skip Adder)**
  - **Carry-ripple is slow through all N stages**
  - **Carry-Bypass Adder allows carry to skip over groups of n bits**
    - **Decision based on n-bit propagate signal**



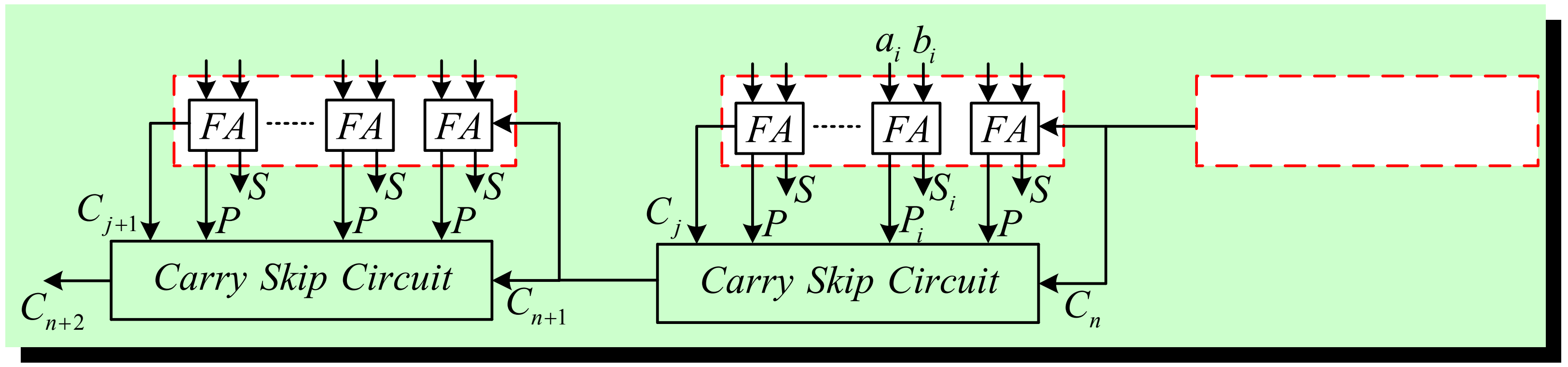
Adders

- **Carry-Bypass Adder (Carry-Skip Adder)**
  - **Carry-skip allows carry to skip over groups of m bits**



Adders

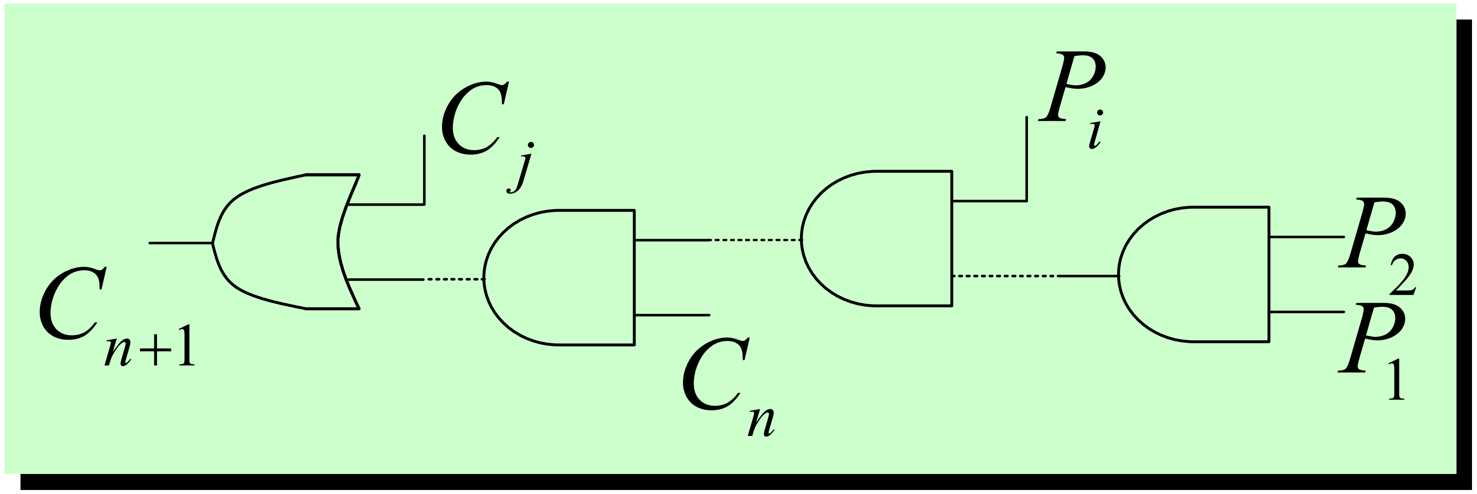
- **Carry-Bypass Adder (Carry-Skip Adder)**



- **P_i = XOR (a_i , b_i)**

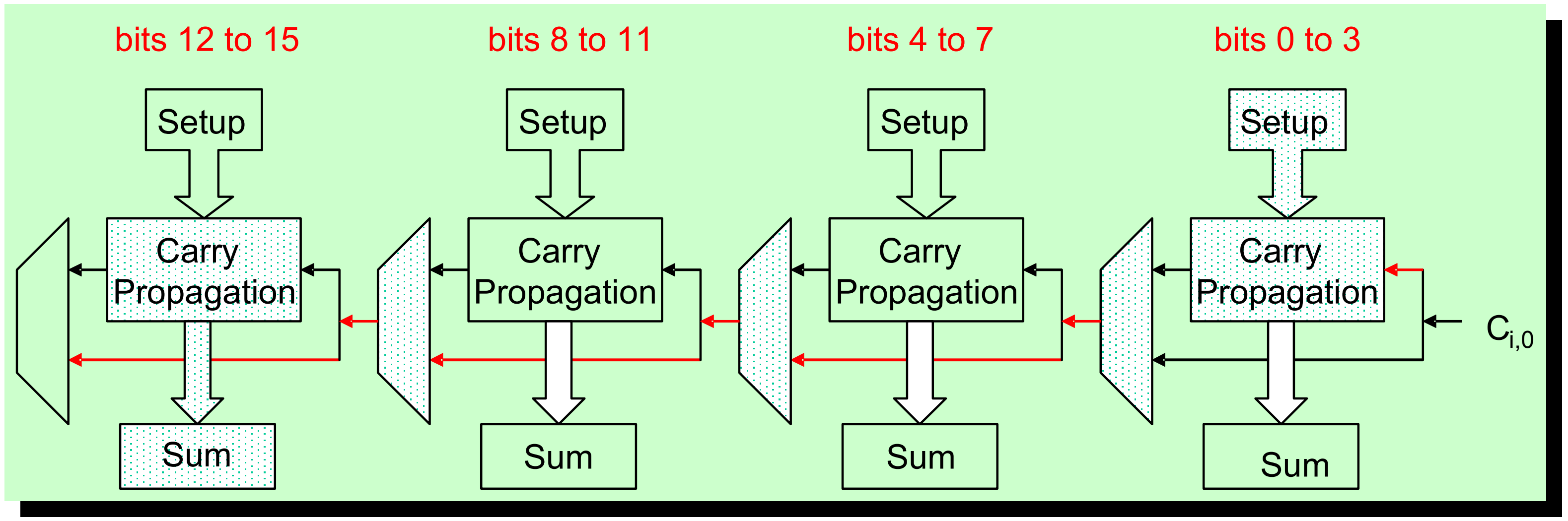
Adders

- Carry-Bypass Adder (Carry-Skip Adder)
  - Carry Skip Circuit
  - This Circuit calculate the carry for the next stage rapidly



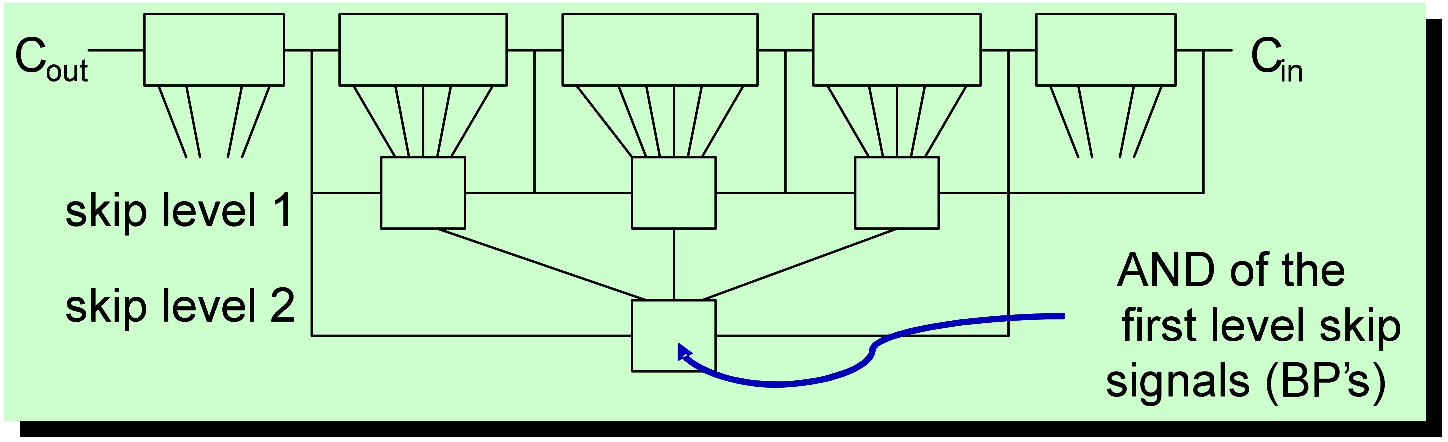
Adders

- **Carry-Bypass Adder (Carry-Skip Adder)**
  - **4 (B)-bit Block**
  - **Worst-case delay → carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups (B is the group size in bits), ripples in the last group from bit 12 to bit 15**
  - **$T_{add} = t_{setup} + B\, t_{carry} + ((N/B) - 1)\, t_{skip} + B\, t_{carry} + t_{sum}$**



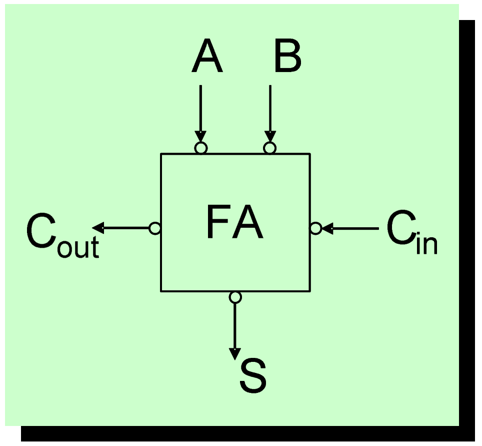
Adders

- **Carry-Bypass Adder (Carry-Skip Adder)**
  - **Variable block sizes**
    - **A carry that is generated in, or absorbed by, one of the inner blocks travels a shorter distance through the skip blocks, so can have bigger blocks for the inner carries without increasing the overall delay**

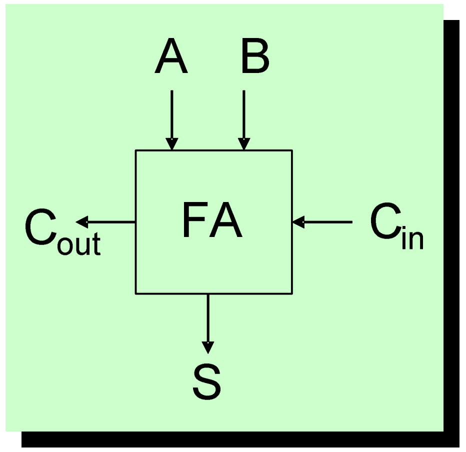


Adders

- **Inversion Property**
  - **Inverting all inputs to a FA results in inverted values for all outputs**



≡

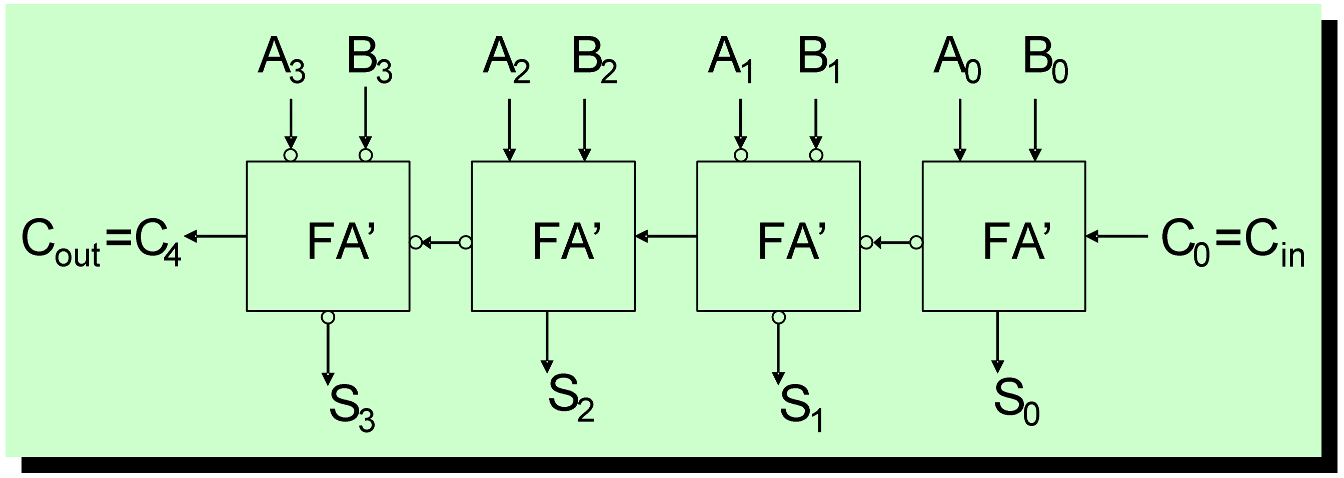


$$\overline{S}(A,B,C_{in})=S(\overline{A},\overline{B},\overline{C}_{in})$$

$$\overline{C}_{out}(A,B,C_{in})=C_{out}(\overline{A},\overline{B},\overline{C}_{in})$$

Adders

- **Exploiting the Inversion Property**
  - **Minimizes the critical path (the carry chain) by eliminating inverters between the FAs**



Adders

- Fast Carry Chain Design
  - The key to fast addition is a low latency carry network
  - What matters is whether in a given position a carry is
  - generated $G_i = A_i \& B_i = A_iB_i$
  - propagated $P_i = A_i \oplus B_i$ (sometimes $A_i \mid B_i$)
  - annihilated (killed) $K_i = \overline{A_i} \& \overline{B_i}$
  - Giving a carry recurrence of $C_{i+1} = G_i \mid P_iC_i$

$C_1 = G_0 \mid P_0C_0$

$C_2 = G_1 \mid P_1G_0 \mid P_1P_0 C_0$

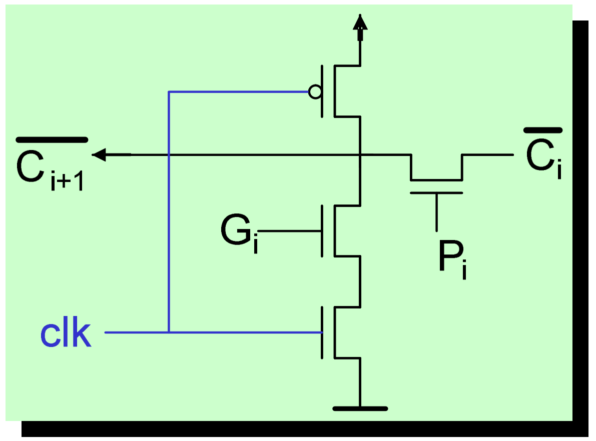
$C_3 = G_2 \mid P_2G_1 \mid P_2P_1G_0 \mid P_2P_1P_0 C_0$

$C_4 = G_3 \mid P_3G_2 \mid P_3P_2G_1 \mid P_3P_2P_1G_0 \mid P_3P_2P_1P_0 C_0$

Contents

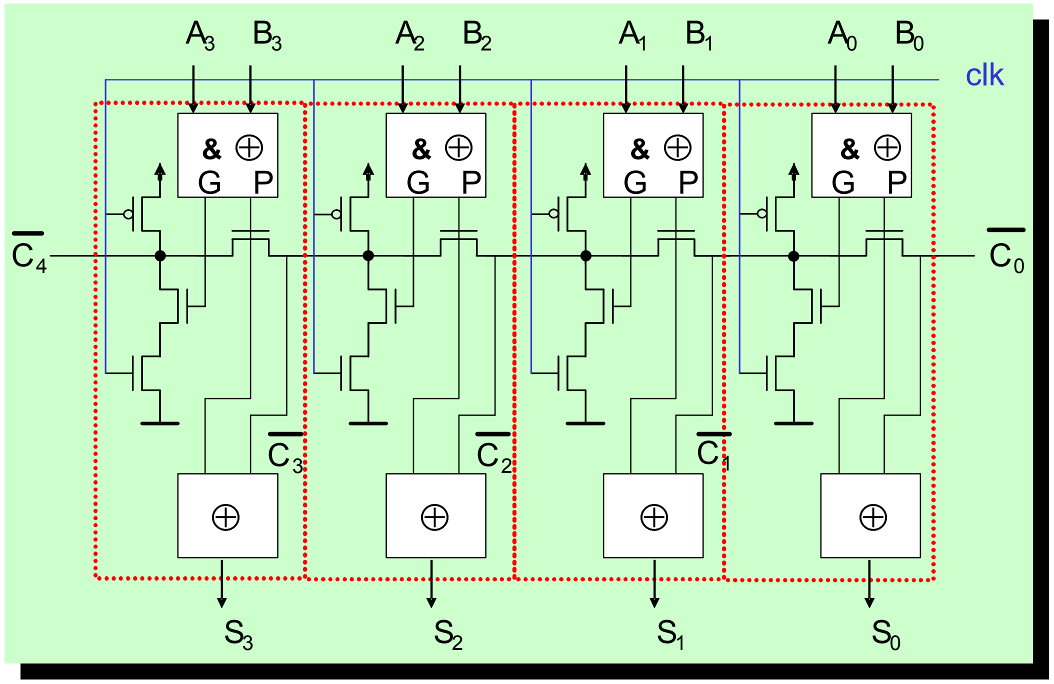
Adders

- Manchester Carry Chain (MCC)
  - Switches controlled by G_i and P_i
  - Total delay of
    - time to form the switch control signals G_i and P_i
    - setup time for the switches
    - signal propagation delay through N switches in the worst case



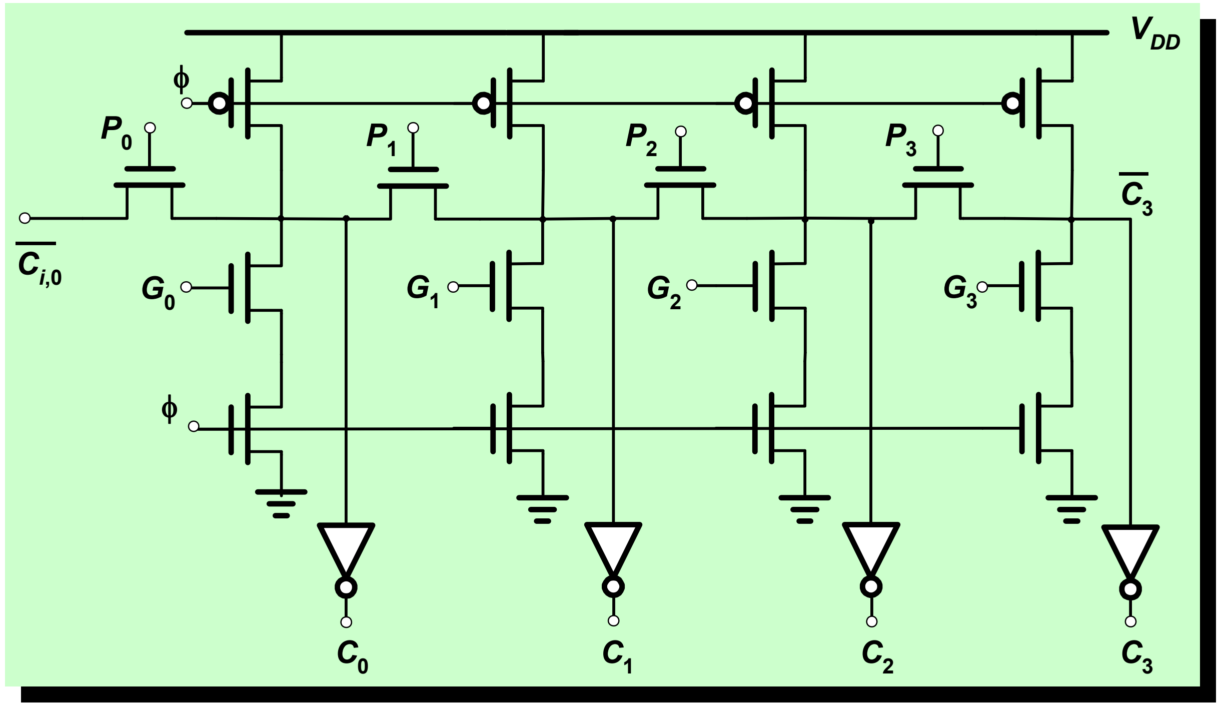
Adders

- **Manchester Carry Chain (MCC)**
  - **4-bit Sliced MCC Adder**



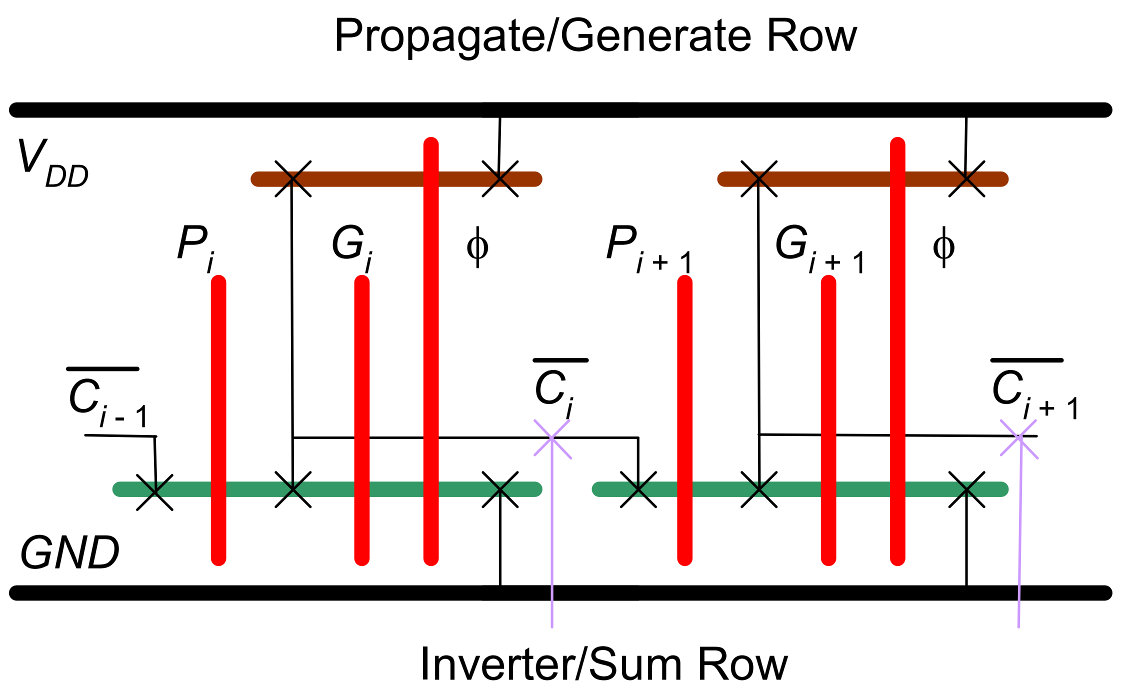
Adders

- **Manchester Carry Chain (MCC)**



Adders

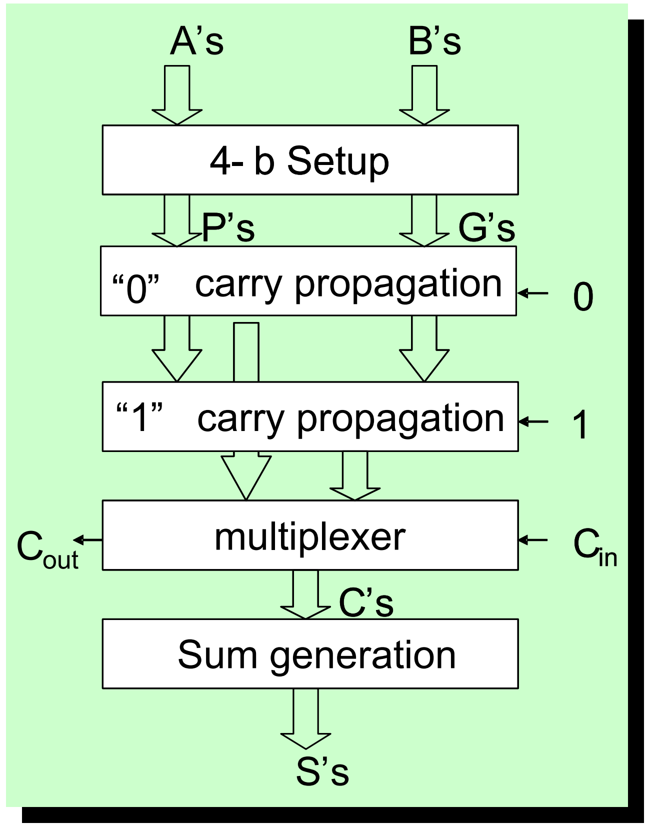
- **Manchester Carry Chain (MCC)**
  - **Stick Diagram**



Contents

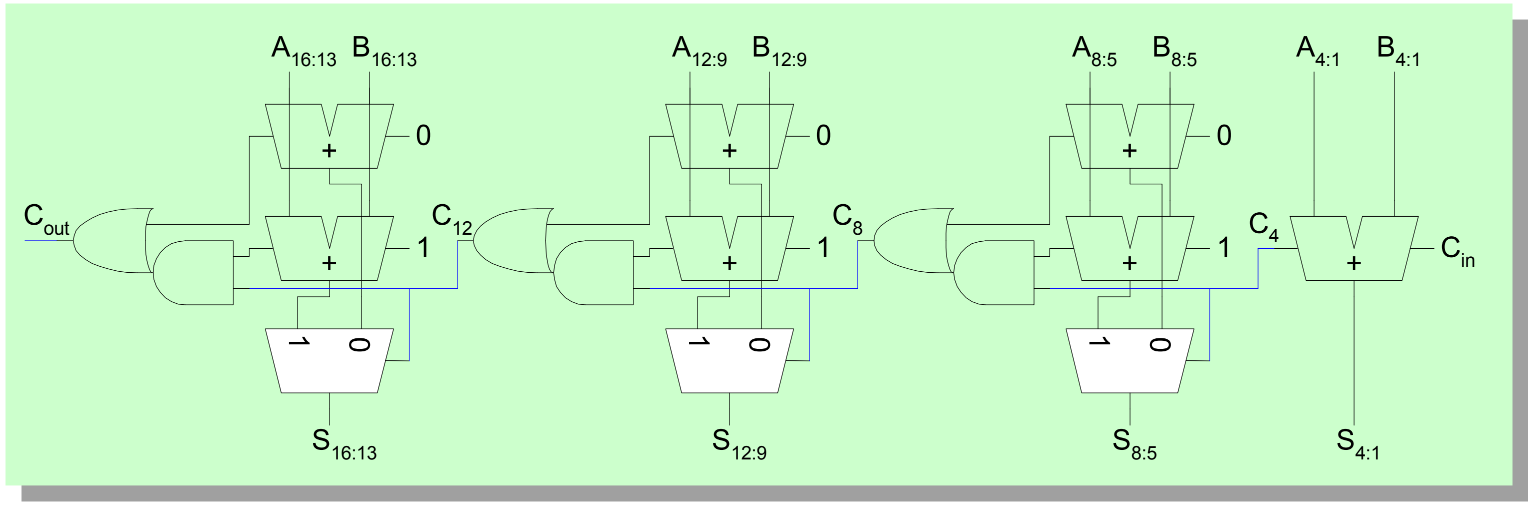
Adders

- **Carry Select Adder**
  - **Trick for critical paths dependent on late input X**
  - **Precompute the carry out of each block for both carry_in = 0 and carry_in = 1 (can be done for all blocks in parallel) and then select the correct one by Mux**



Adders

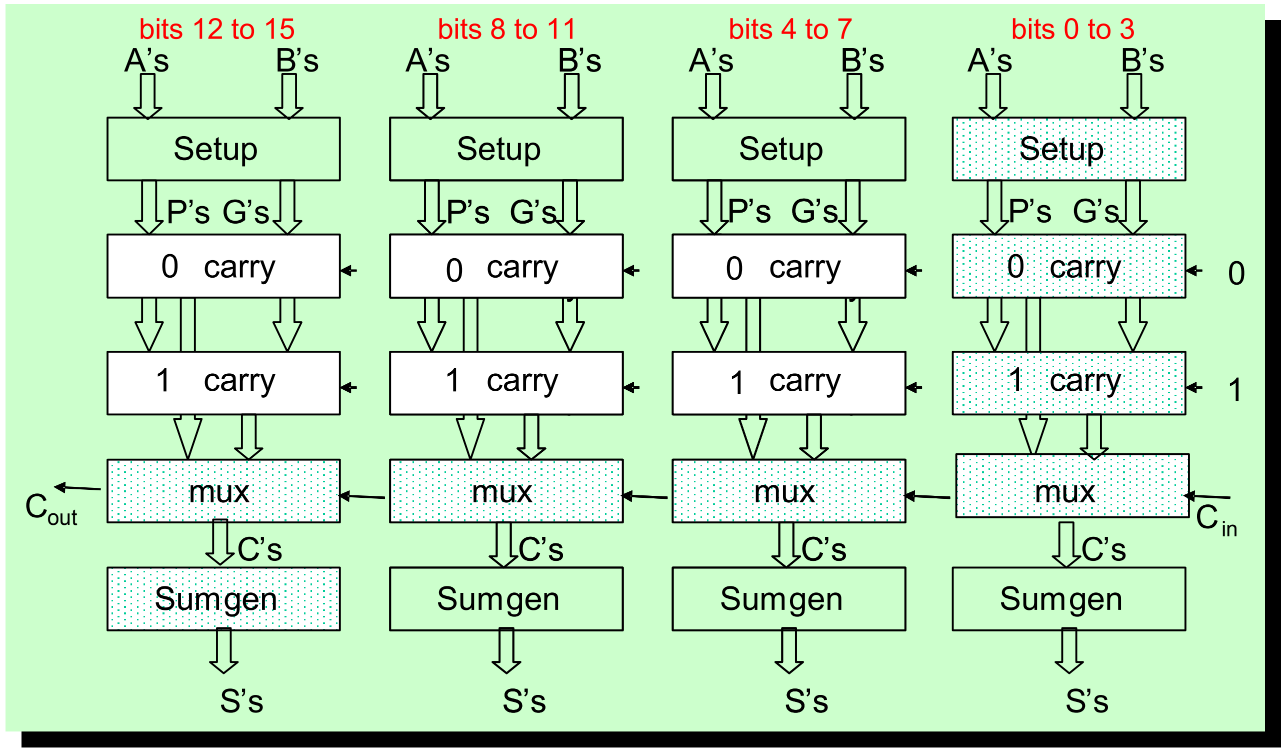
- **Carry Select Adder**



Contents

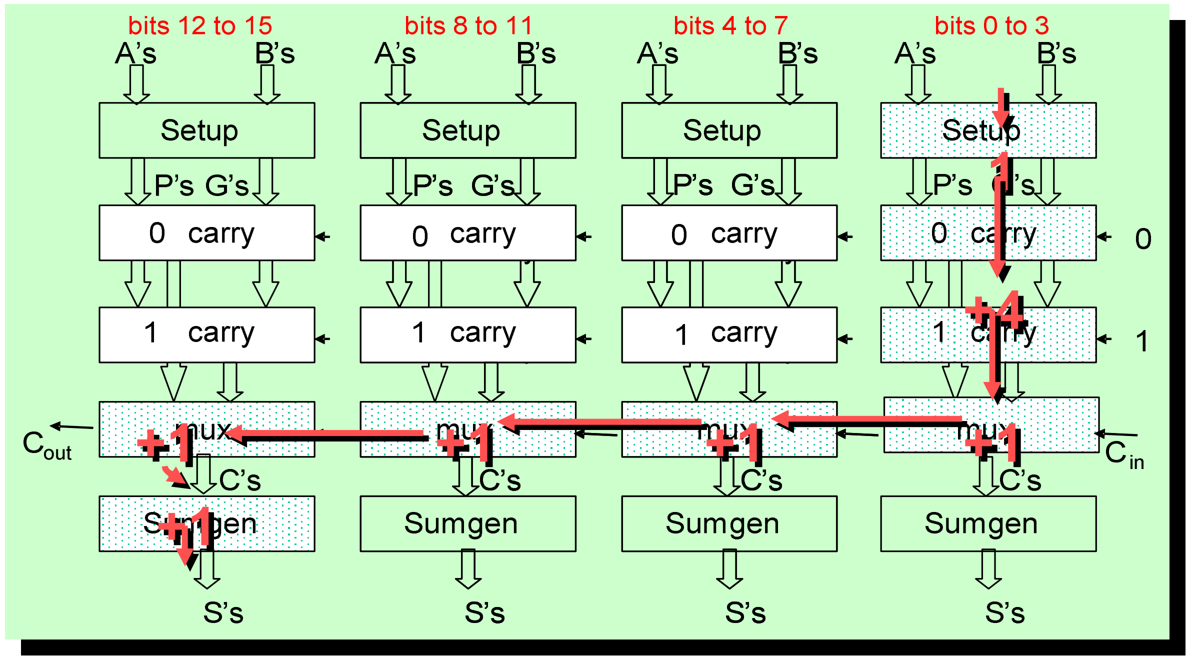
Adders

- **Carry Select Adder (Linear)**



Adders

- **Carry Select Adder (Linear): Critical Path**

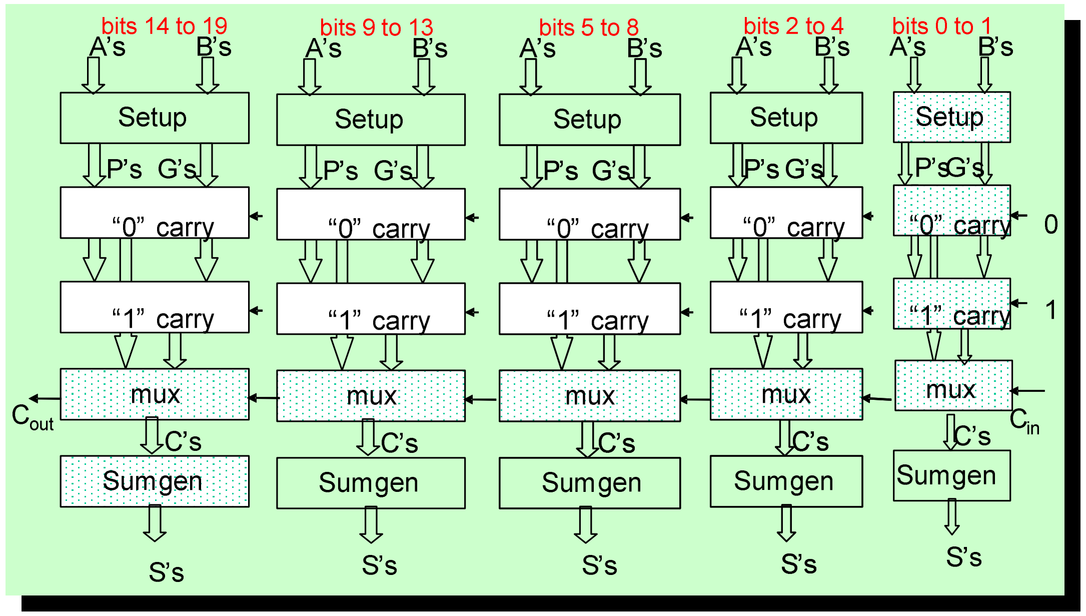


$$T_{add} = t_{setup} + Bt_{carry} + N/Bt_{mux} + t_{sum}$$

Contents

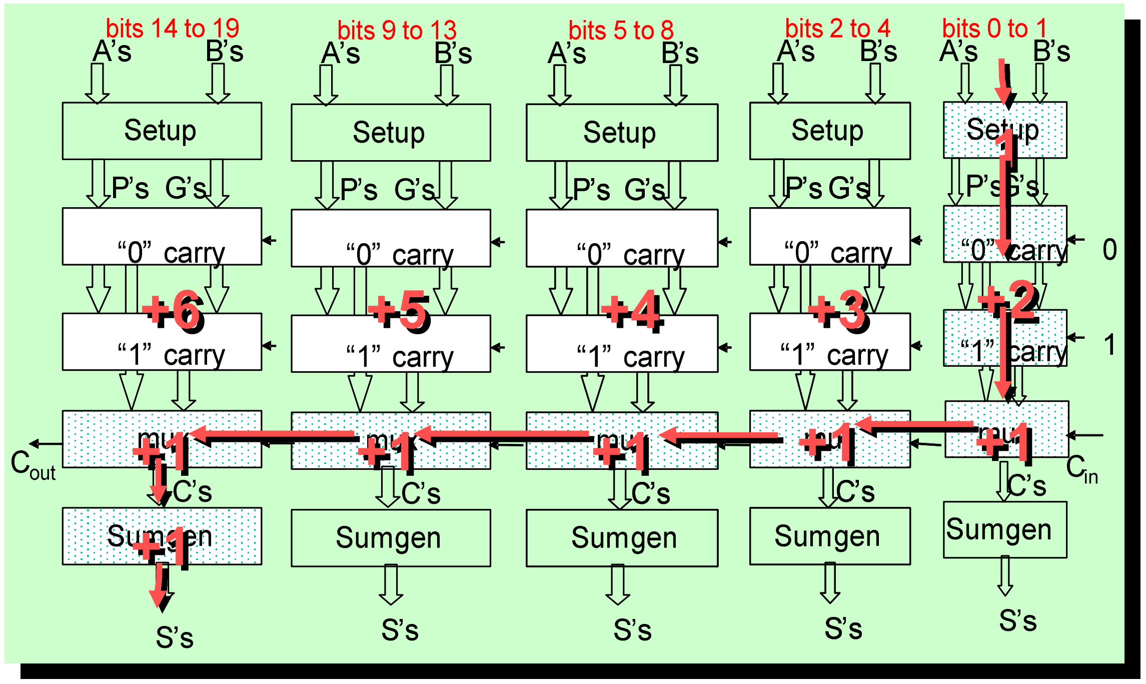
Adders

- **Carry Select Adder (Square Root)**
  - **This idea can be carried on in a logarithmic tree fashion to obtain the addition in log2(N) time**



Adders

- **Carry Select Adder (Square Root)**

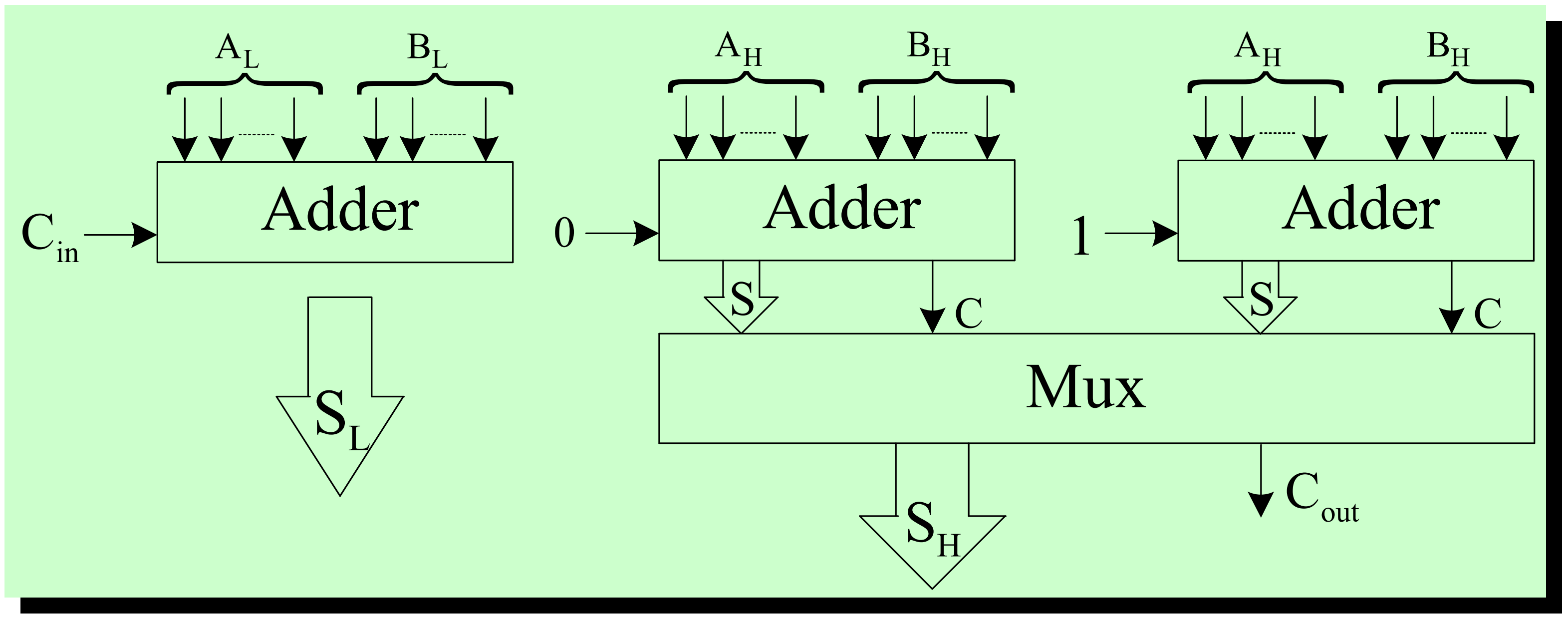


$$T_{add} = t_{setup} + 2t_{carry} + \sqrt{N}\, t_{mux} + t_{sum}$$

Contents

Adders

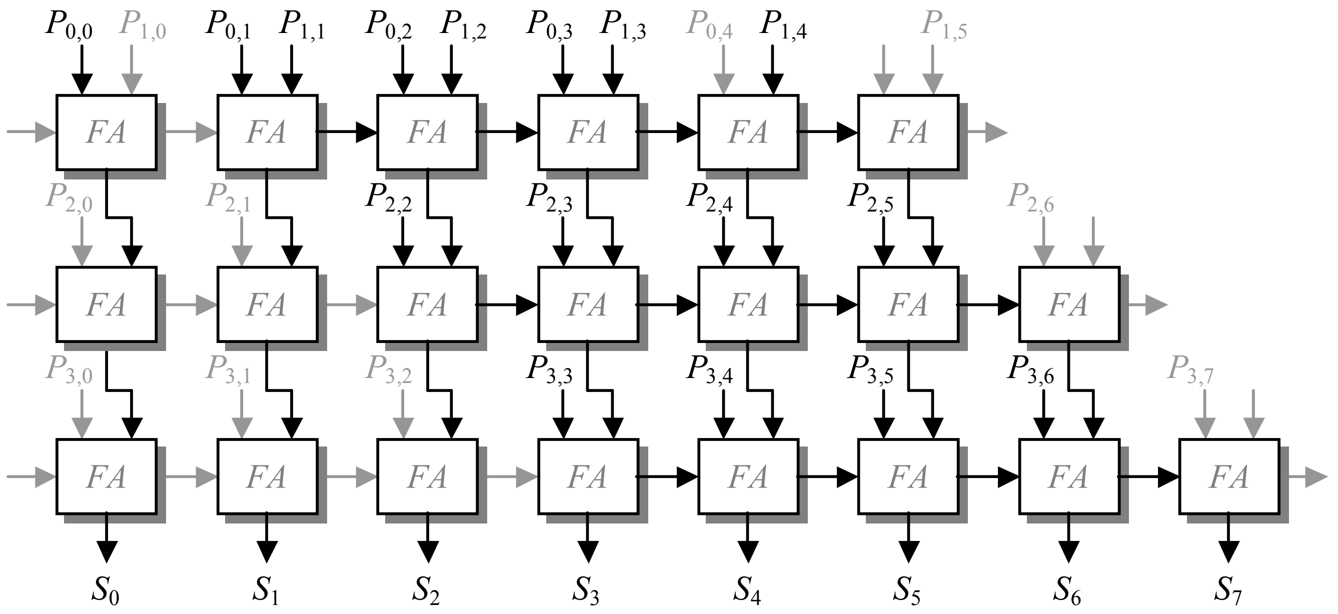
- **Conditional Carry Adder**
  - **Increase speed twice!**

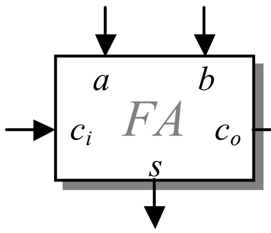


Contents

Adders

- **Carry Save Adder**
  - **When adding k n-bit numbers**
    - **Linear Array (or Tree) Summation**



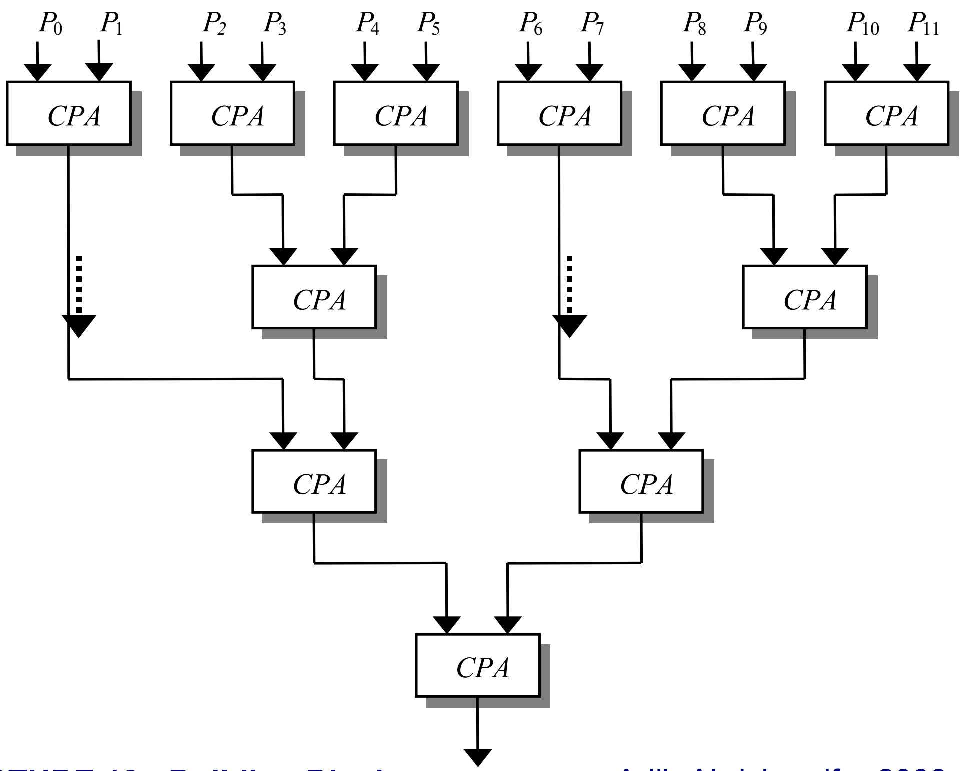


Adders

- Carry Save Adder
  - When adding k n-bit numbers
    - Tree Summation

CPA

(Carry Propagate Adder)

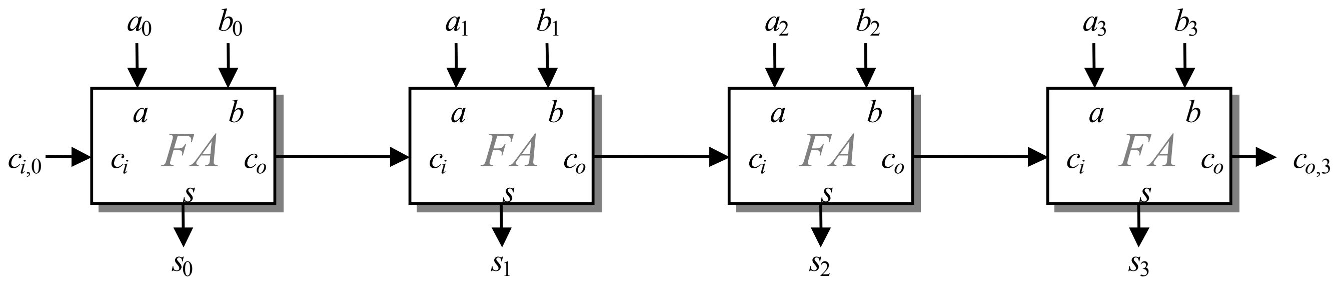


P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10} P_{11}
CPA

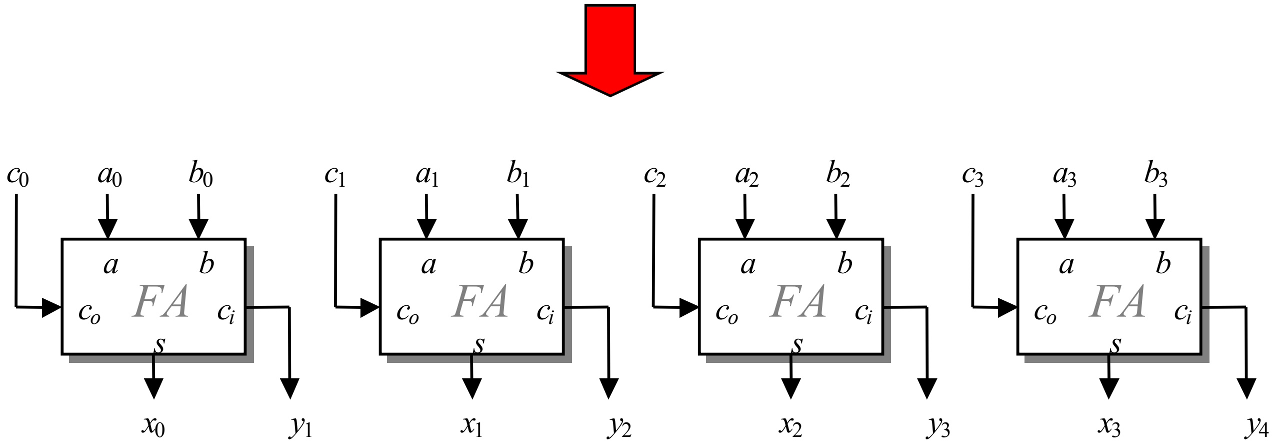
Adders

- **Carry Save Adder**
  - **When adding k n-bit numbers**

CPA:
$A + B = S$

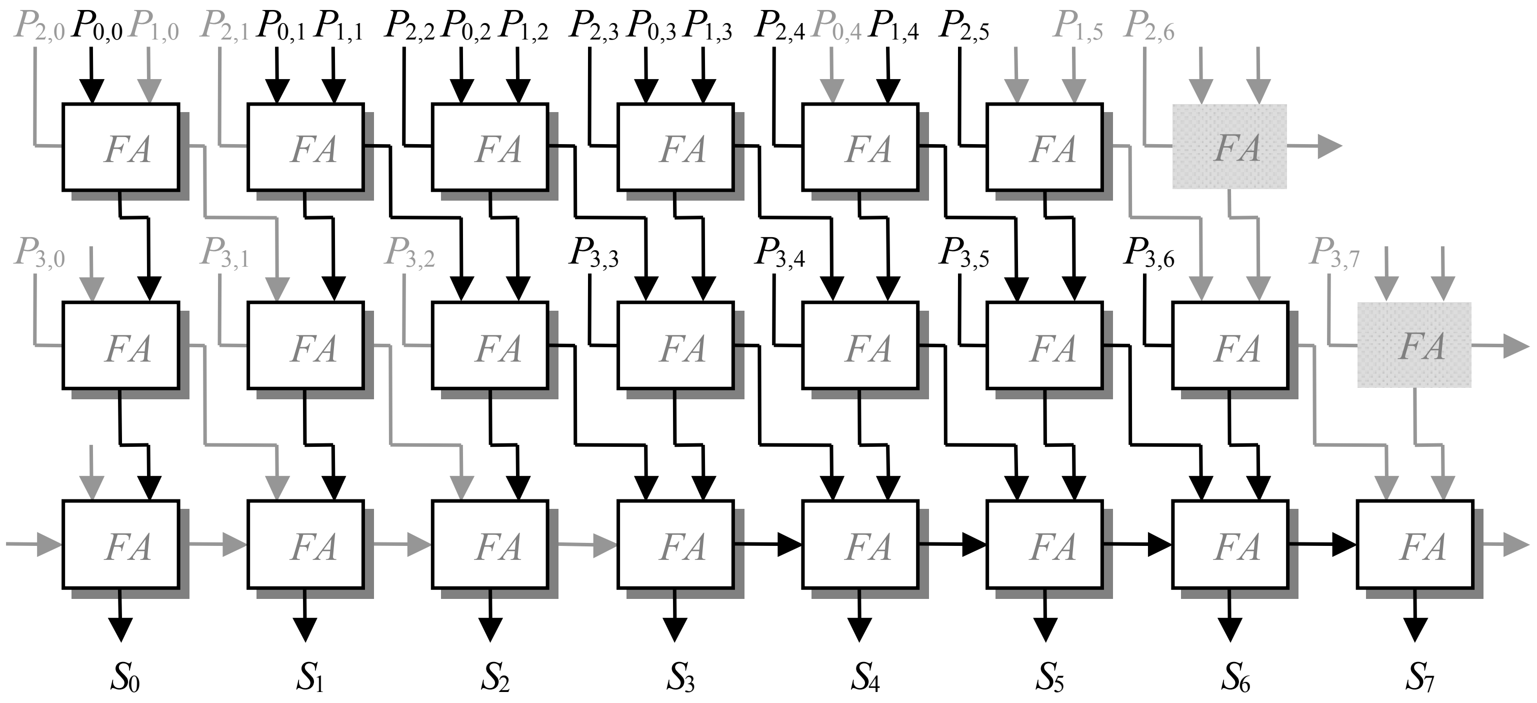


CSA:
$A + B + C = X + Y$


c_0 a_0 b_0 c_1 a_1 b_1 c_2 a_2 b_2 c_3 a_3 b_3
a b FA c_o c_i s
x_0 y_1 x_1 y_2 x_2 y_3 x_3 y_4

Adders

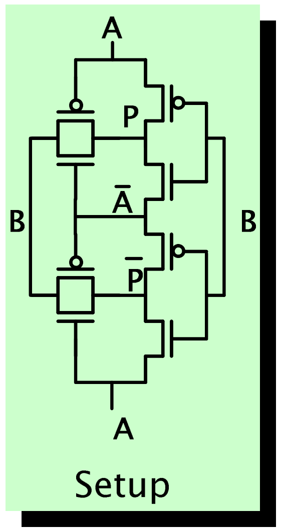
- **Carry-Save Adder**

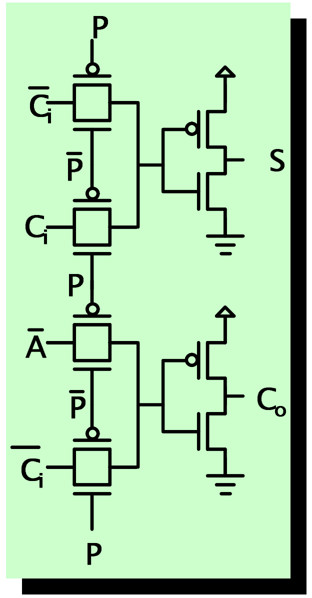


$P_{2,0}$ $P_{0,0}$ $P_{1,0}$ $P_{2,1}$ $P_{0,1}$ $P_{1,1}$ $P_{2,2}$ $P_{0,2}$ $P_{1,2}$ $P_{2,3}$ $P_{0,3}$ $P_{1,3}$ $P_{2,4}$ $P_{0,4}$ $P_{1,4}$ $P_{2,5}$ $P_{1,5}$ $P_{2,6}$
$P_{3,0}$ $P_{3,1}$ $P_{3,2}$ $P_{3,3}$ $P_{3,4}$ $P_{3,5}$ $P_{3,6}$ $P_{3,7}$
FA
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7

Adders

- Carry Save Adder
  - Intermediate FA Cells
    - Better to have the same sum and carry delays (both contribute to critical path)



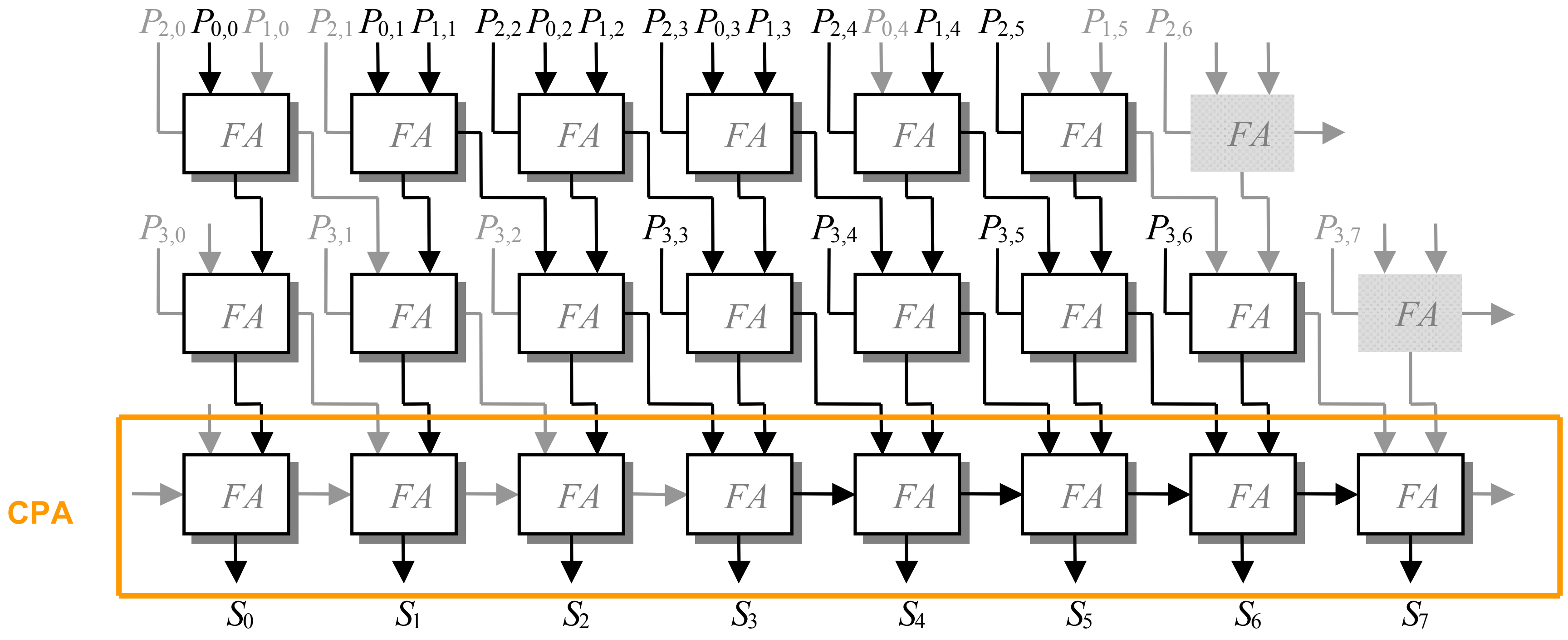


Adders

- Speed up summation with faster adders (Logarithmic)
  - Linear (Tree) array has several equal-length critical paths
    $\Rightarrow$ All adders need to be replaced
  - The carry-save array has only ONE critical path
    $\Rightarrow$ Replace only the final CPA (see next slide)

Adders

- **Speed up summation with faster adders**
  - **Replace only the final CPA**

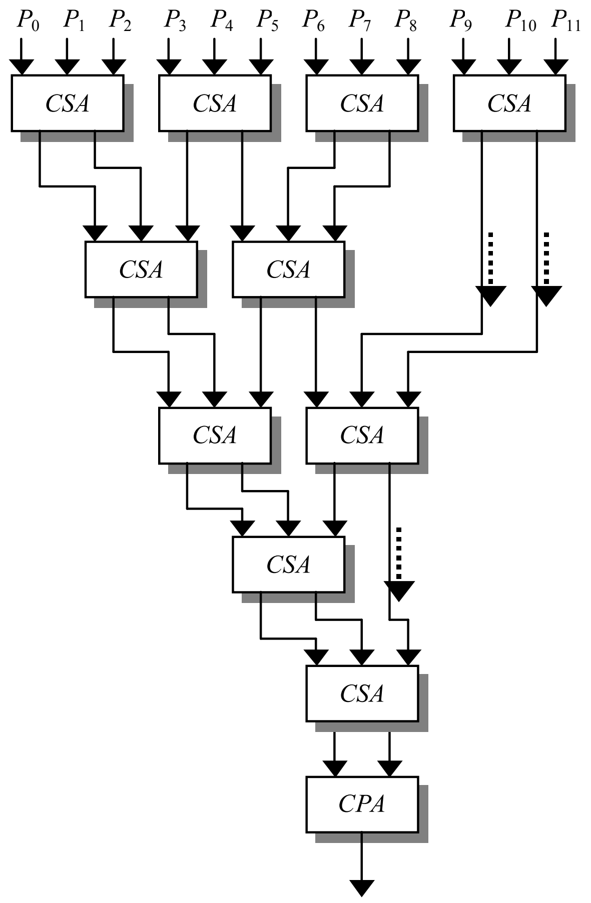


$P_{2,0}$ $P_{0,0}$ $P_{1,0}$ $P_{2,1}$ $P_{0,1}$ $P_{1,1}$ $P_{2,2}$ $P_{0,2}$ $P_{1,2}$ $P_{2,3}$ $P_{0,3}$ $P_{1,3}$ $P_{2,4}$ $P_{0,4}$ $P_{1,4}$ $P_{2,5}$ $P_{1,5}$ $P_{2,6}$
$P_{3,0}$ $P_{3,1}$ $P_{3,2}$ $P_{3,3}$ $P_{3,4}$ $P_{3,5}$ $P_{3,6}$ $P_{3,7}$
FA
CPA
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7

Contents

Adders

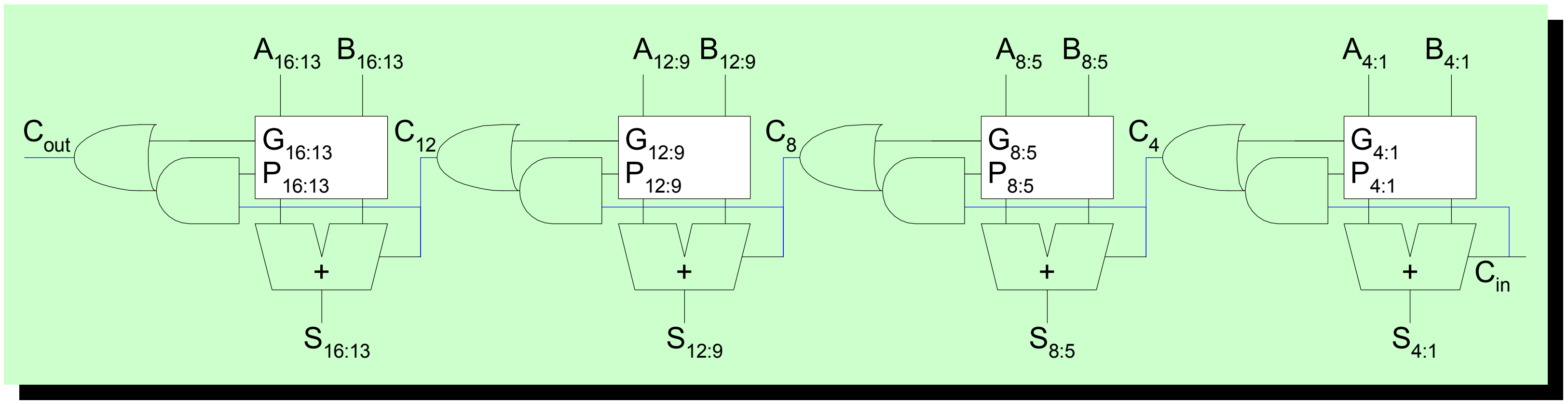
- **Wallace Tree**
  - **Sums in *O*(log *m*) steps**


P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10} P_{11}
CSA
CPA

Contents

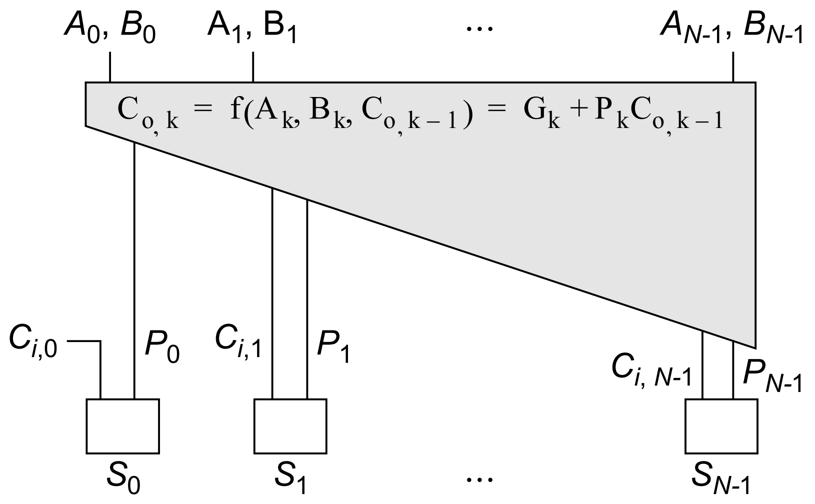
Adders

- **LookAhead - Basic Idea**
  - **Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel**
  - **Uses higher-valency cells with more than two inputs**



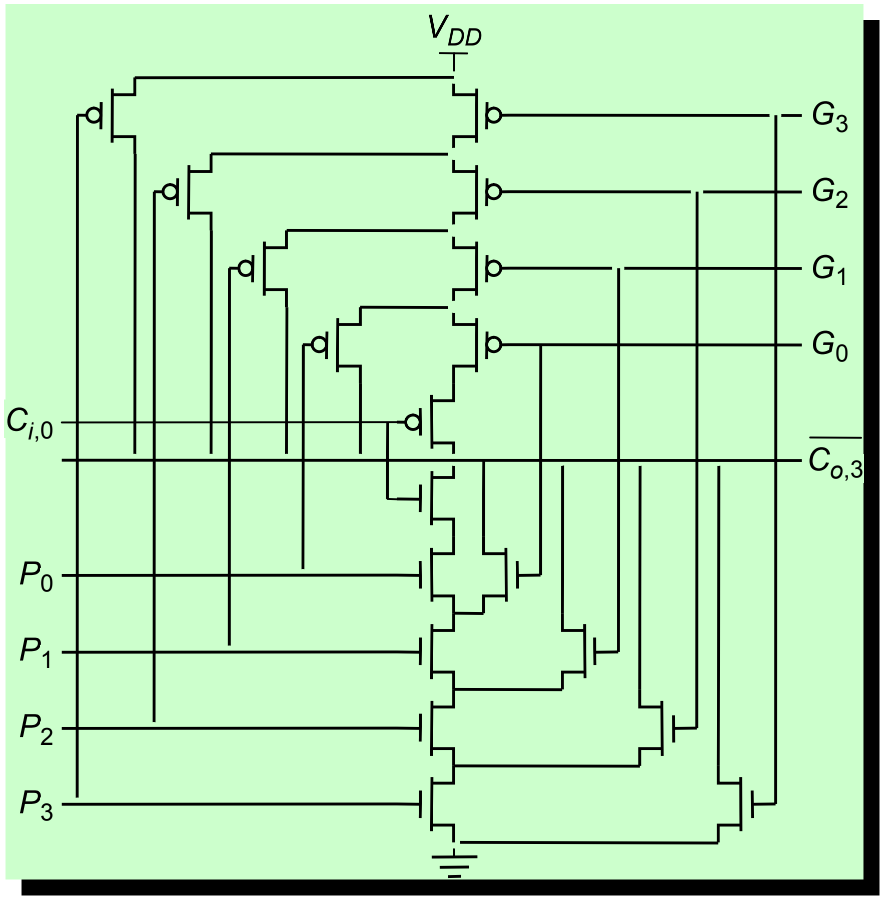
Adders

- **LookAhead - Basic Idea**
  - $C_1 = G_0 + P_0C_0$
  - $C_2 = G_1 + P_1G_0 + P_1P_0\ C_0$
  - $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0\ C_0$
  - $C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0\ C_0$



Adders

- **Look-Ahead: Topology**

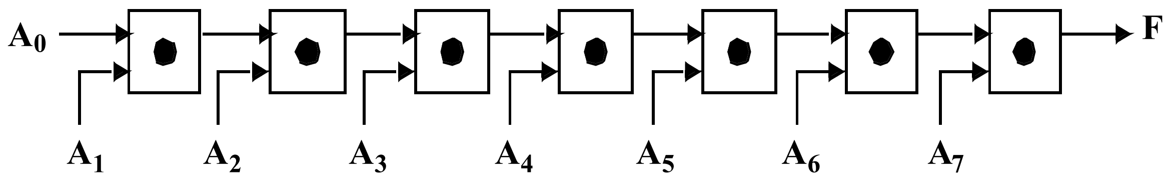


Adders

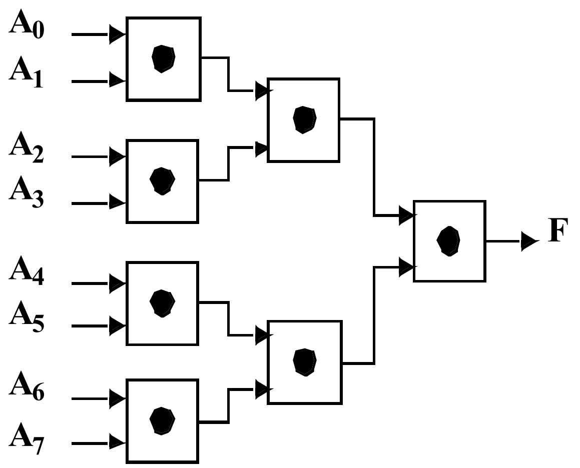
- Look-Ahead
  - Expanding Lookahead equations
    - $C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$
  - All the way
    - $C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_o + P_oC_{i,o})))$

Adders

- **Logarithmic Look-Ahead Adder**



$t_p \sim N$



$t_p \sim \log_2(N)$

Adders

- **Carry Lookahead Trees**

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

$$C_{o,0} = G_0 + P_0 C_{i,0}$$

$$C_{o,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$$

$$C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$$

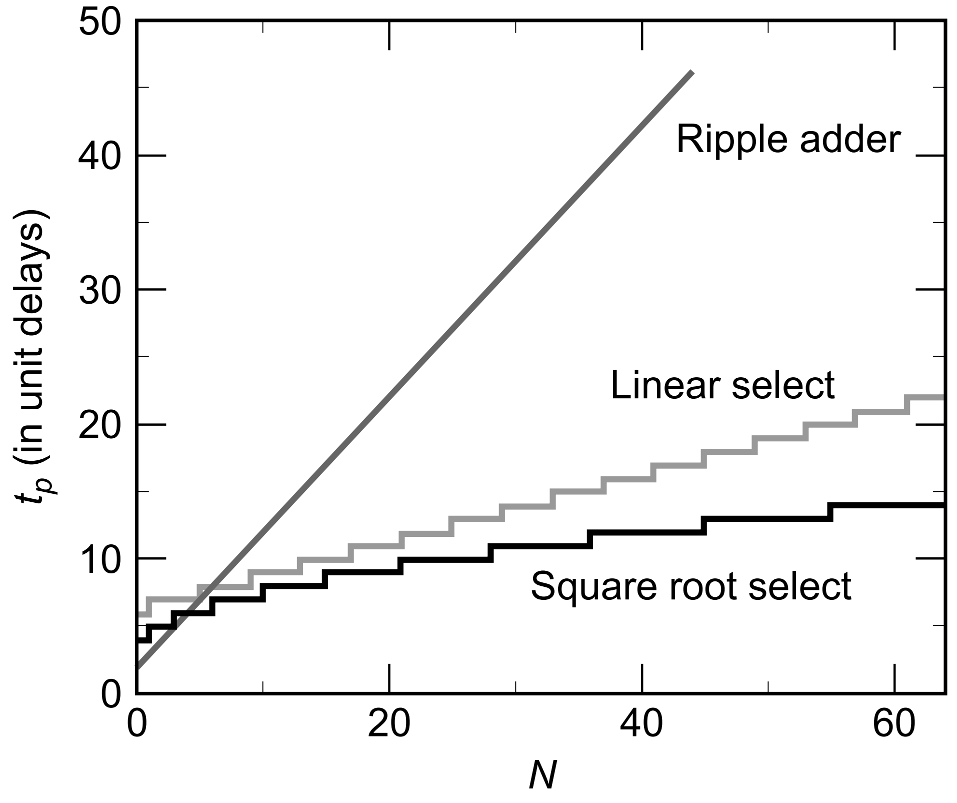
$$= (G_2 + P_2 G_1) + (P_2 P_1)(G_0 + P_0 C_{i,0}) = G_{2:1} + P_{2:1} C_{o,0}$$

- **Can continue building the tree hierarchically**

Contents

Adders

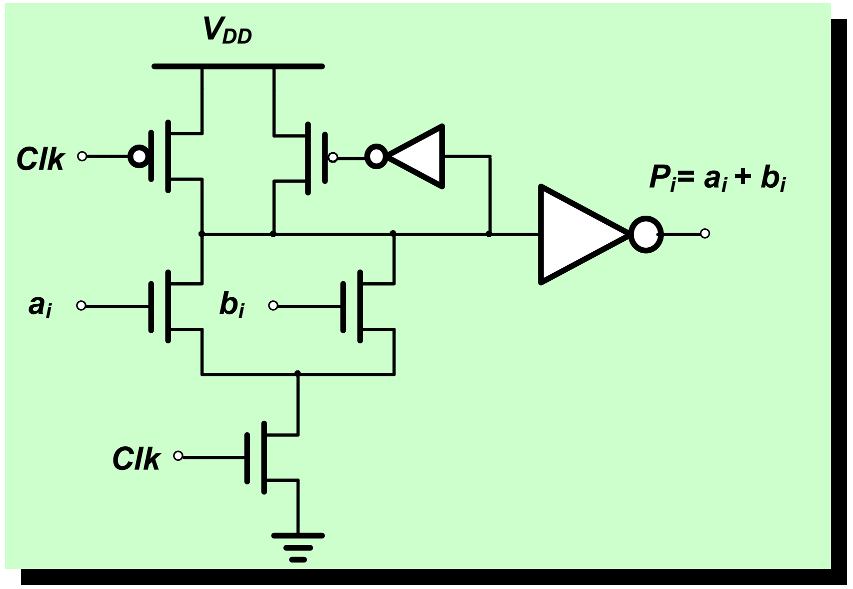
Adder Delays - Comparison

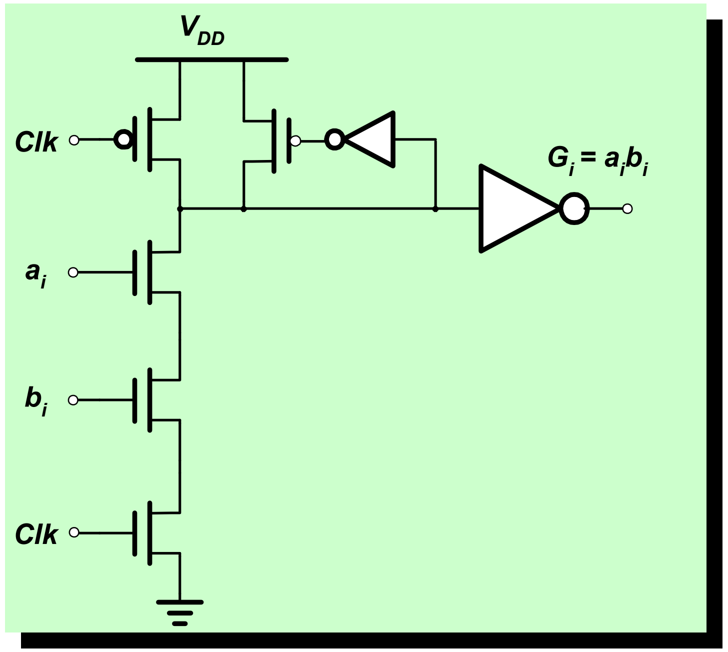


Contents

Adders

- **Example: Domino Adder**



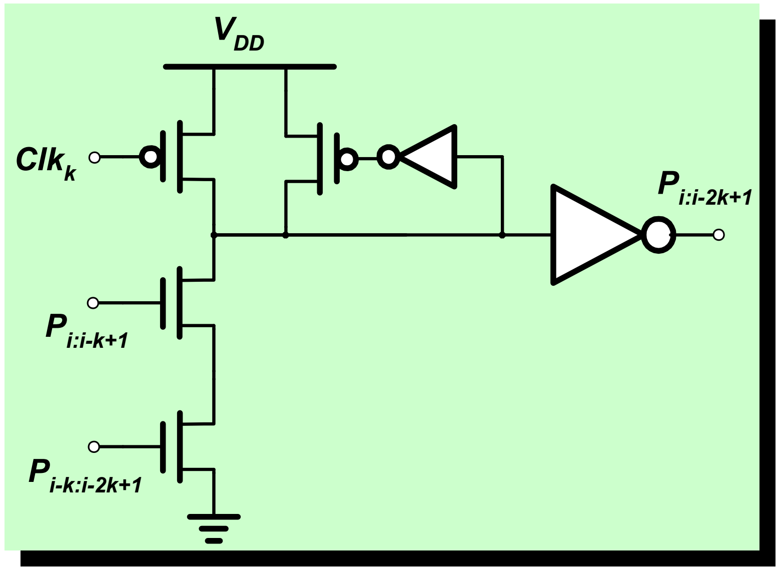


Propagate (P) = $A \oplus B$

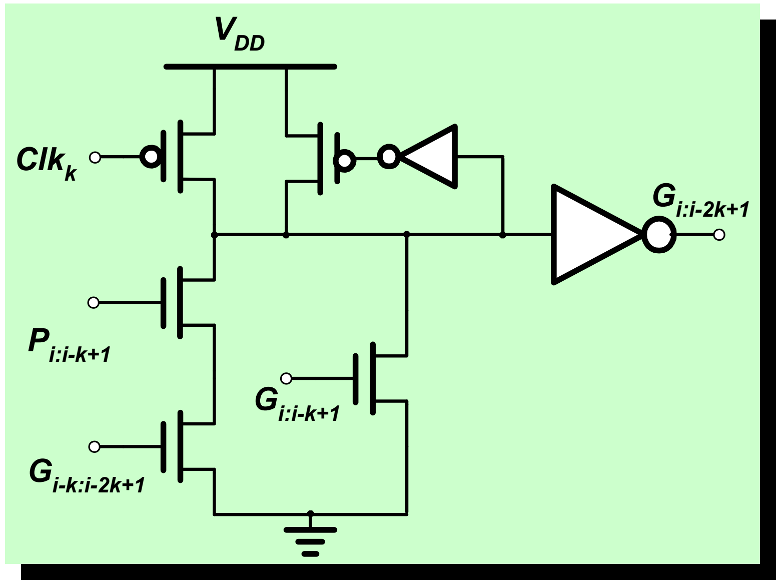
Generate (G) = AB

Adders

- **Example: Domino Adder**



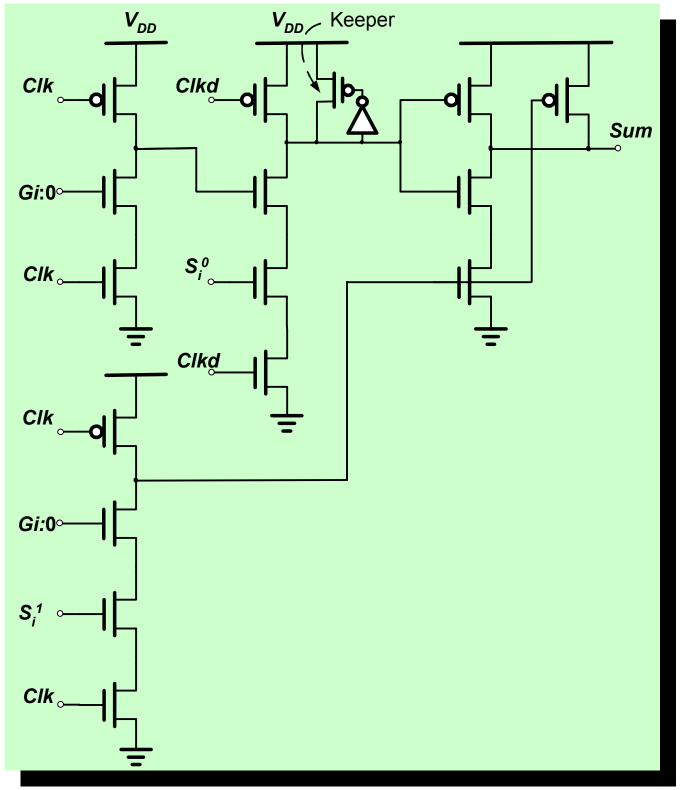
Propagate



Generate

Adders

- **Example: Domino Sum**



Contents

Summary

- This lecture describes one of the basic building blocks (Adders) and also implementation of them in transistor level
- Also noted how to choose an adder and designing fast ones