# Digital Building Blocks: Adders

Acknowledgment: Slides are adapted from Harris and Harris and Hennessy and Patterson textbooks

## Introduction

- **Digital building blocks:**
  - Gates, multiplexers, decoders, registers, arithmetic circuits, counters, memory arrays, logic arrays
- **Building blocks demonstrate hierarchy, modularity, and regularity:**
  - Hierarchy of simpler components
  - Well-defined interfaces and functions
  - Regular structure easily extends to different sizes

## 1-Bit Adder

**Any fan-in gates:**

$t_{\mathrm{Cout}} = t_{\mathrm{AND2}} + t_{\mathrm{OR3}}$

$t_{\mathrm{S}} = t_{\mathrm{XOR3}}$

$A_{\mathrm{FA}} = 3A_{\mathrm{AND2}} + A_{\mathrm{OR3}} + A_{\mathrm{XOR3}}$

**Only fan-in two gates:**

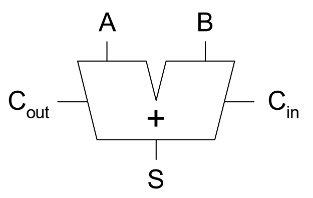
$t_{\mathrm{Cout}} = t_{\mathrm{AND2}} + 2t_{\mathrm{OR2}}$

$t_{\mathrm{S}} = 2t_{\mathrm{XOR2}}$

$A_{\mathrm{FA}} = 3A_{\mathrm{AND2}} + 2A_{\mathrm{OR2}} + 2A_{\mathrm{XOR2}}$

**Full Adder**

**Cin = 0 for "half adder"**



| $C_{in}$ | A | B | $C_{out}$ | 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 |

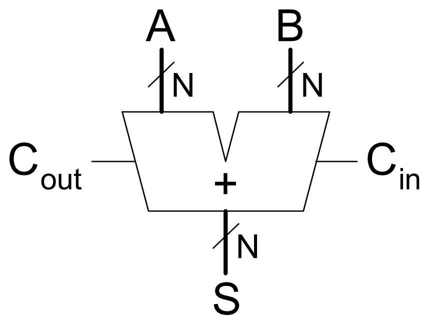
$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + AC_{in} + BC_{in}$

# Multibit Adders (CPAs)

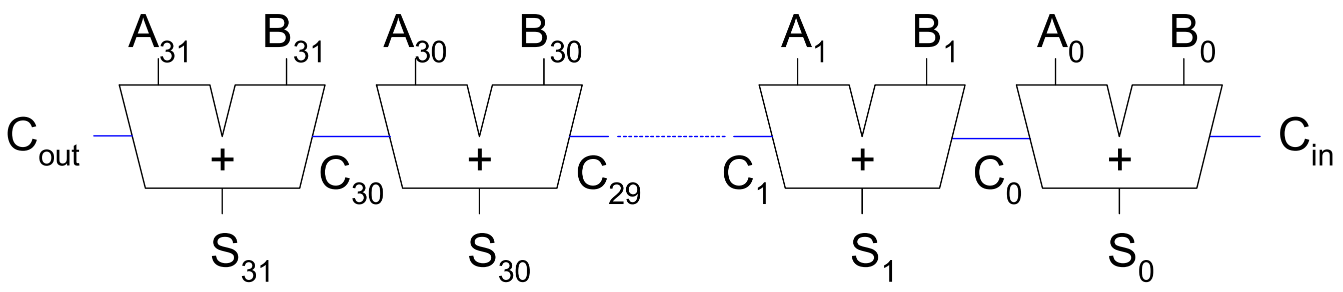
- Types of carry propagate adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (fast)
  - Prefix (faster)
- Carry-lookahead & prefix adders faster for large adders but require more hardware

**Symbol**



## Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



## Ripple-Carry Adder Delay and Complexity

$$A_{\text{ripple}} = NA_{FA}$$

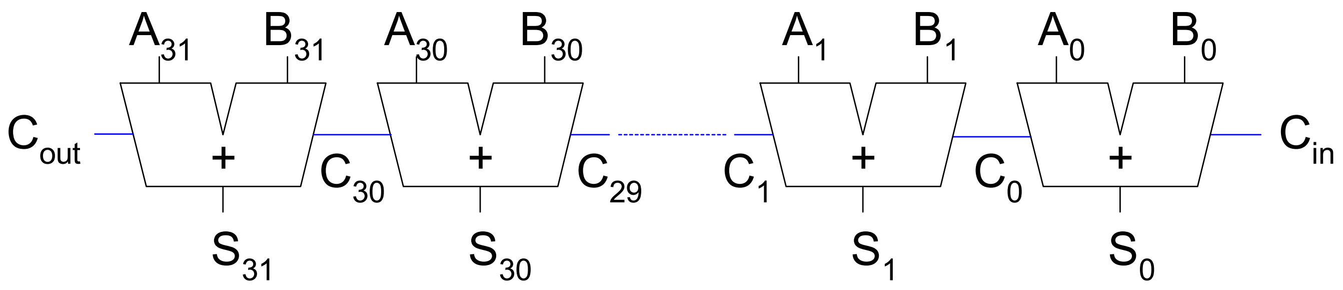
where $A_{FA}$ is the area of a 1-bit full adder

$$t_{\text{ripple}} = Nt_{FA}$$

where $t_{FA}$ is the delay of a 1-bit full adder

- This is a crude estimation in terms of delays and complexity of subblocks. Sometimes (e.g. for clock cycle calculation) more accurate estimates are needed.

## Ripple-Carry Adder (More Accurate) Delay



**$t_{AND2} = t_{OR2} = 1$ ns, $t_{XOR2}$ =2 ns only fan-in-two gates**

**$C_0 = A_0 B_0 + A_0 C_{in} + B_0 C_{in}$ takes 3 ns**

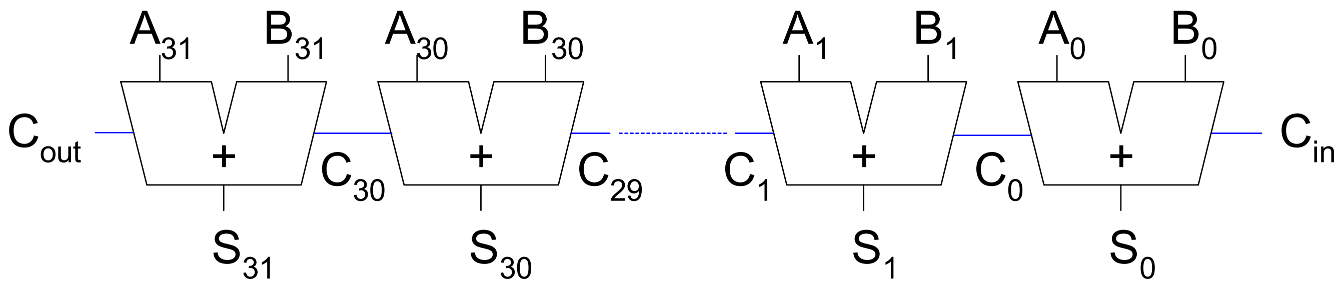
**$C_1 = A_1 B_1 + A_1 C_0 + B_1 C_0$ takes 6 ns**

**$S_{31} = 31*3$ ns + 4 ns = 97 ns (not accurate)**

**$S_{31} = 31*3$ ns + <u>2 ns</u> = <u>95 ns</u> (considering overlap)**

**$C_{out} = 31*3$ ns = 96 ns $\rightarrow$ critical path**

# A Better (FA) Design for Ripple-Carry Adder



$t_{AND2} = t_{OR2} = 1$ **ns,** $t_{XOR2}$ **=2 ns only fan-in-two gates**

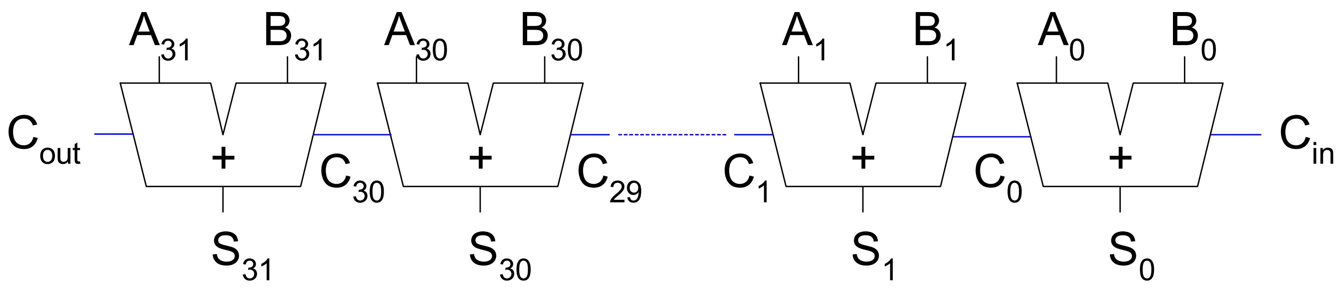
$C_0 = A_0 B_0 + (A_0 + B_0)\, C_{in}$ **takes 3 ns**

$C_1 = A_1 B_1 + (A_1 + B_1)\, C_0$ **takes 5 ns**

$S_{31} = C_{out} = 3 \text{ ns} + 30 {*} 2 \text{ ns} + 2 \text{ ns} = 65 \text{ ns}$

- Big idea here is to manipulate Boolean formulas and compute subexpressions as soon as Boolean variables are available. The same idea will be used for faster adders

# A Hint on What Can be Improved Further



**$t_{AND2} = t_{OR2} = 1$ ns, $t_{XOR2} = 2$ ns only fan-in-two gates**

**$C_0 = A_0 B_0 + (A_0 + B_0)\, C_{in}$ takes 3 ns**

**$C_1 = A_1 B_1 + (A_1 + B_1)\, C_0$ takes 5 ns**

**Plug $C_0$ into the expression for $C_1$**

$$C_1 = A_1 B_1 + (A_1 + B_1)\,(A_0 B_0 + (A_0 + B_0)\, C_{in}) =$$

$$= A_1 B_1 + (A_1 + B_1)\,(A_0 B_0) + (A_1 + B_1)\,(A_0 + B_0) C_{in}$$

**now takes 1 ns + 1 ns + 1 ns + 1 ns= 4 ns**

## Carry-Lookahead Adder

- Compute carry out ($C_{\text{out}}$) for $k$-bit blocks using *generate* and *propagate* signals
- **Some definitions:**
  - Column $i$ produces a carry out by either *generating* a carry out or *propagating* a carry in to the carry out
  - Generate ($G_i$) and propagate ($P_i$) signals for each column:
    - Column $i$ will generate a carry out if $A_i$ AND $B_i$ are both 1.

$$G_i = A_i B_i$$

    - Column $i$ will propagate a carry in to the carry out if $A_i$ OR $B_i$ is 1.

$$P_i = A_i + B_i$$

    - The carry out of column $i$ ($C_i$) is:

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$

## Carry-Lookahead Addition

- **Step 1:** Compute $G_i$ and $P_i$ for all columns
- **Step 2:** Compute $G$ and $P$ for $k$-bit blocks
- **Step 3:** $C_{in}$ propagates through each $k$-bit propagate/generate block

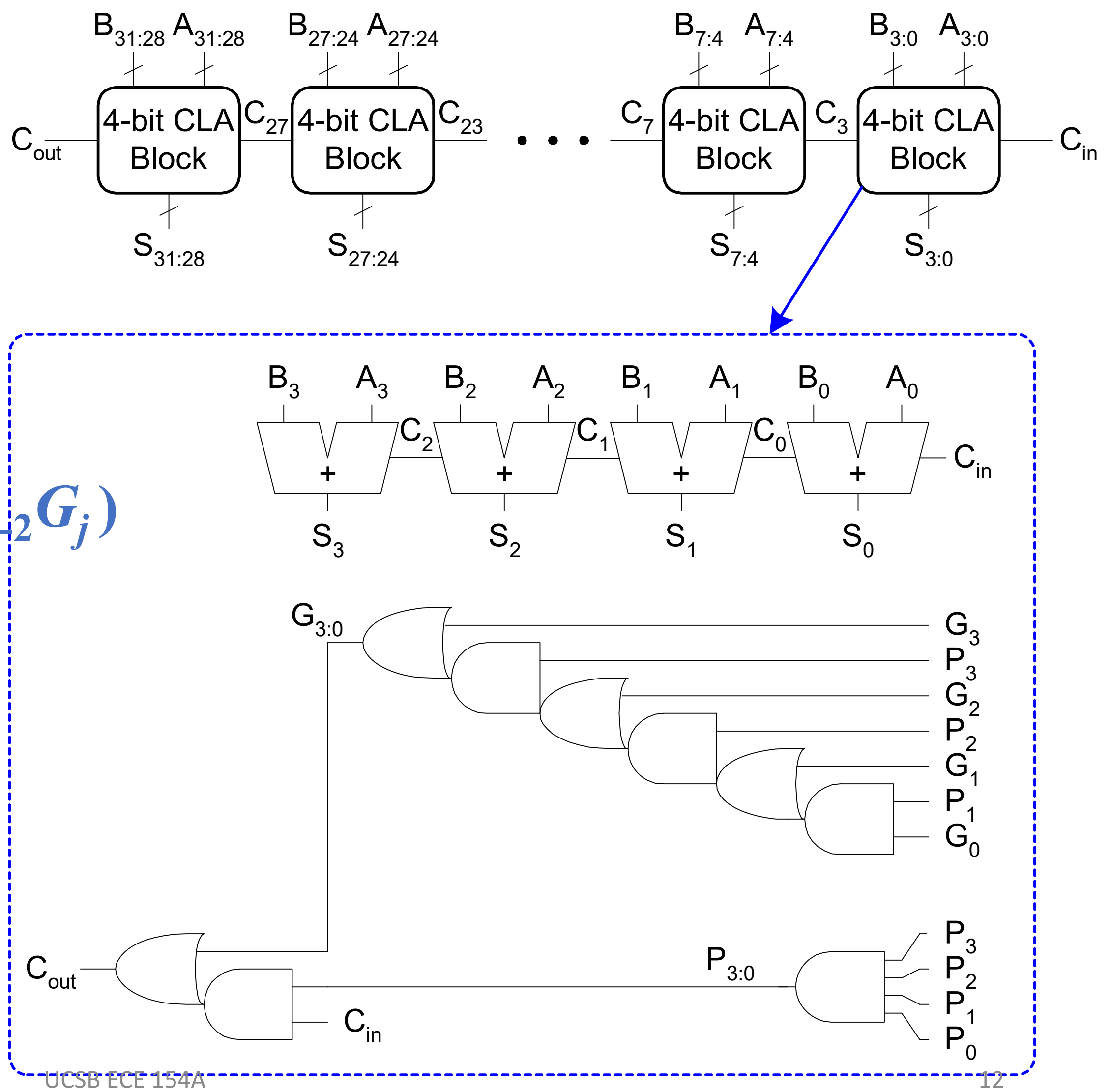
- **Example:** 4-bit blocks ($G_{3:0}$ and $P_{3:0}$) :

$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0 )$$
$$P_{3:0} = P_3 P_2 P_1 P_0$$

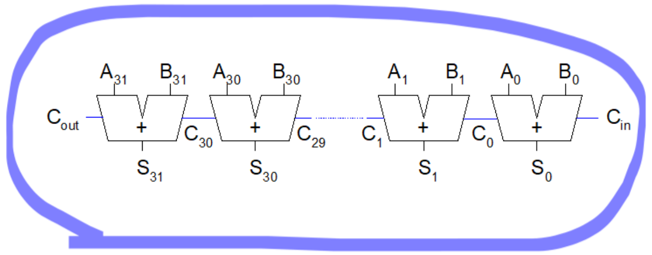
- **Generally,**

$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} G_j )$$
$$P_{i:j} = P_i P_{i-1} P_{i-2} P_j$$
$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

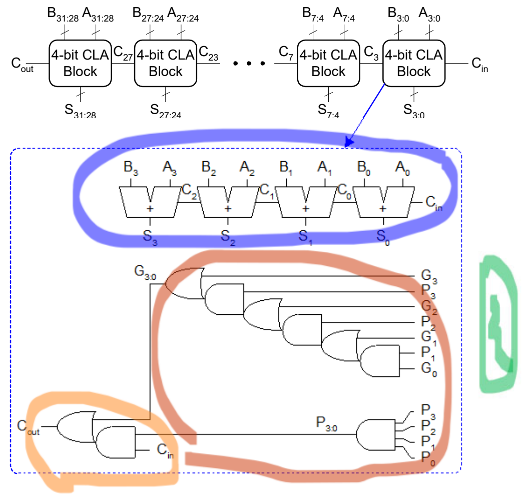
# 32-bit CLA with 4-bit Blocks

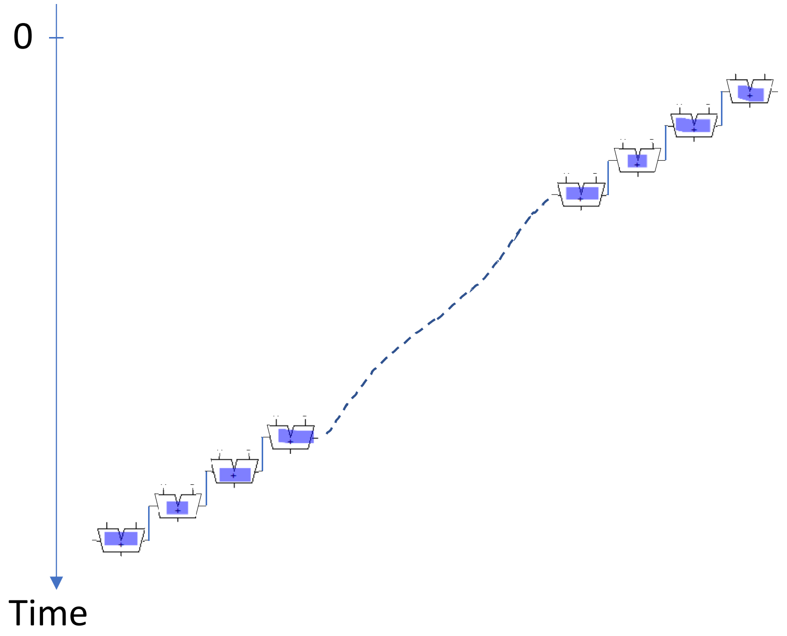


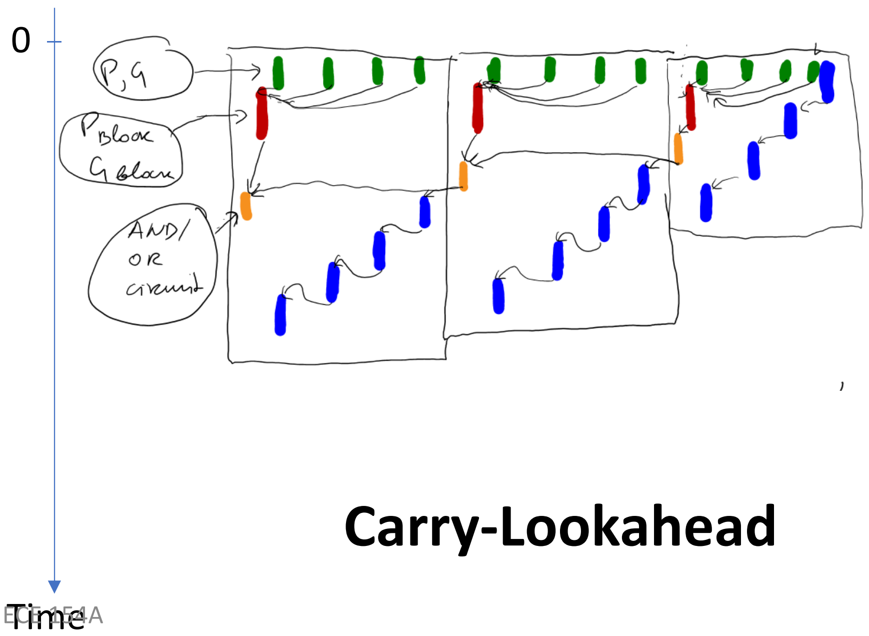
$$G_{i:j} = G_i + P_i\,(G_{i-1} + P_{i-1}\,(G_{i-2} + P_{i-2}G_j\,)$$
$$P_{i:j} = P_iP_{i-1}\,P_{i-2}P_j$$
$$C_i\ = G_{i:j}\ + P_{i:j}\,C_{j-1}$$



## Ripple-Carry







## Carry-Lookahead

# Carry-Lookahead Adder (Crude) Delay

For $N$-bit CLA with $k$-bit blocks:

$$t_{CLA} = t_{pg} + t_{pg\_\text{block}} + (N/k - 1)t_{\text{AND\_OR}} + kt_{FA}$$

- $t_{pg}$ : delay to generate all $P_i$, $G_i$
- $t_{pg\_\text{block}}$ : delay to generate all $P_{i:j}$, $G_{i:j}$
- $t_{\text{AND\_OR}}$ : delay from $C_{\text{in}}$ to $C_{\text{out}}$ of final AND/OR gate in $k$-bit CLA block

An $N$-bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$

## Carry-Lookahead Adder (Crude) Complexity

For $N$-bit CLA with $k$-bit blocks:

$$A_{CLA} \approx (N/k - 1)(A_{\text{AND\_OR}} + A_{pg} + A_{pg\_\text{block}}) + NA_{FA}$$

- $A_{pg}$ : Area of circuit generating all $P_i$, $G_i$
- $A_{pg\_\text{block}}$ : Area of circuit generating all $P_{i:j}$, $G_{i:j}$
- $A_{\text{AND\_OR}}$ : Area of AND/OR gates for carry computation in $k$-bit CLA block
- Approximate because some FA does not have to generate $C_{\text{out}}$

A carry-lookahead adder is higher complexity compared to ripple carry adder (scaling is still linear)

## Main Idea of Parallel Prefix Adder

- Computes carry in ($C_{i\text{-}1}$) for each column, then computes sum:

$$S_i = (A_i \oplus B_i) \oplus C_{i\text{-}1}$$

- Computes $G$ and $P$ for 1-, 2-, 4-, 8-bit blocks, etc. until all $G_i$ (carry in) known
- $\log_2 N$ stages

## Parallel Prefix Adder: (Notation) Trick #1

- Carry in either *generated* in a column or *propagated* from a previous column
- Assume that column -1 holds $C_{in}$, so

  $$\boldsymbol{G_{-1} = C_{in}, P_{-1} = 0}$$
- Carry in to column $i$ = carry out of column $i$-$1$:

  $$\boldsymbol{C_{i-1} = G_{i-1:-1}}$$

  $G_{i\text{-}1:\text{-}1}$: generate signal spanning columns $i$-1 to -1
- Sum equation:

  $$\boldsymbol{S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}}$$
- This trick allows to replace all Cs with corresponding Gs
- **Goal:** Quickly compute $G_{0:-1}$, $G_{1:-1}$, $G_{2:-1}$, $G_{3:-1}$, $G_{4:-1}$, $G_{5:-1}$, … (called ***prefixes***)

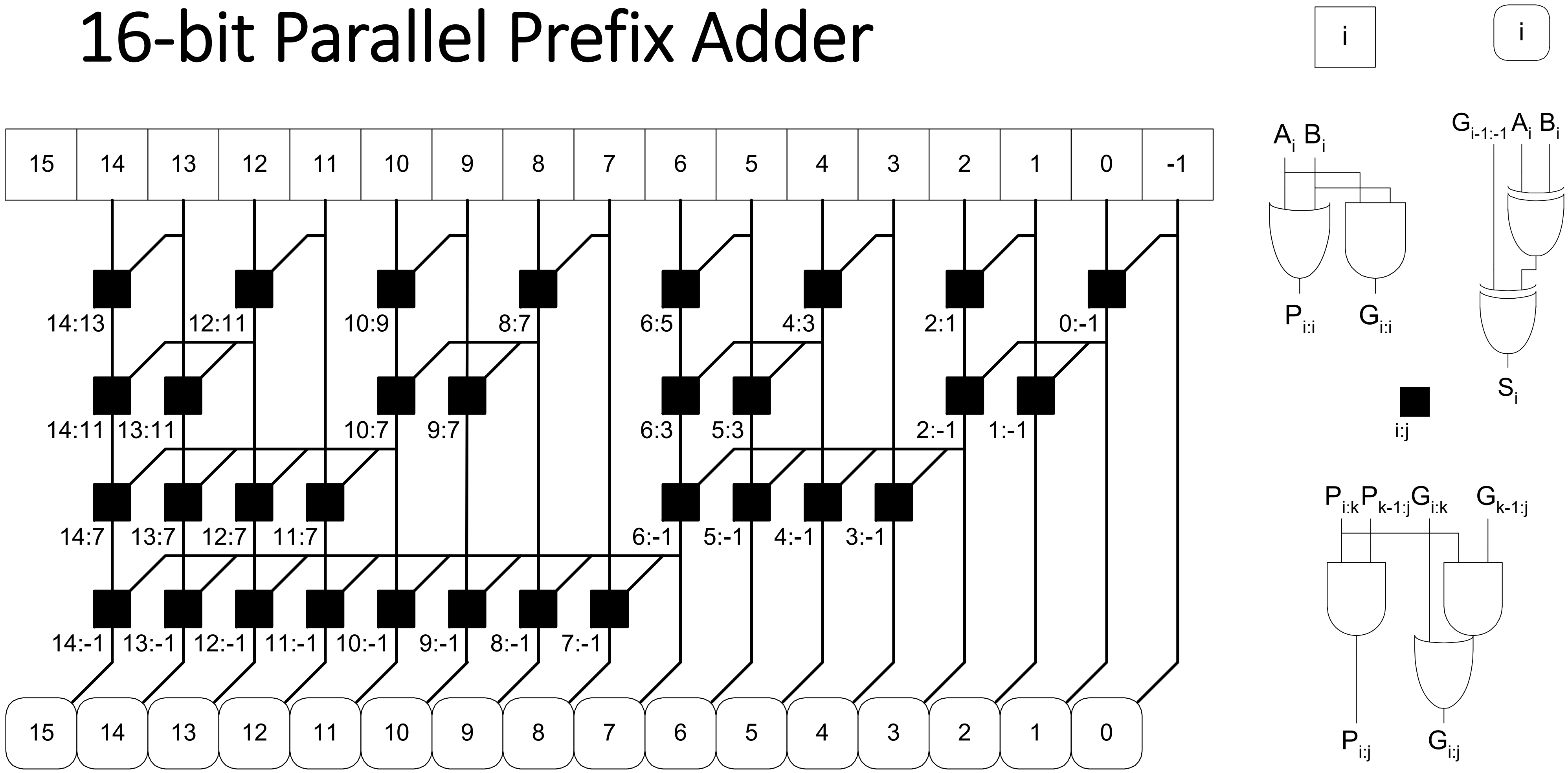
# Parallel Prefix Adder: Trick #2

- Generate and propagate signals for a block spanning bits *i*:*j*:

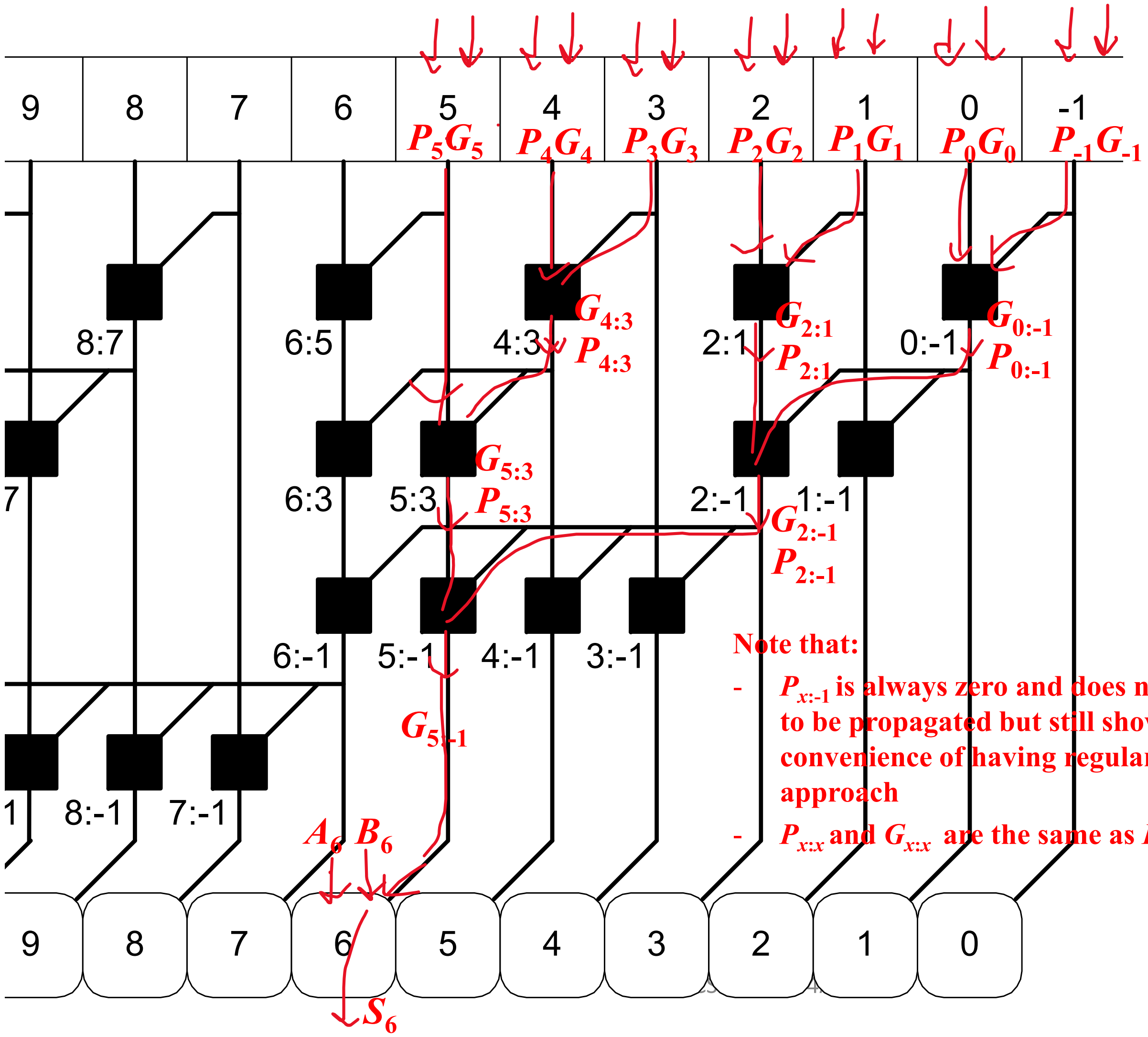
  $$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k\text{-}1:j}$$
  $$P_{i:j} = P_{i:k} P_{k\text{-}1:\mathrm{j}}$$

- In words:
  - **Generate:** block *i*:*j* will generate a carry if:
    - upper part (*i*:*k*) generates a carry or
    - upper part propagates a carry generated in lower part (*k*-1:*j*)
  - **Propagate:** block *i*:*j* will propagate a carry if *both* the upper and lower parts propagate the carry

# 16-bit Parallel Prefix Adder

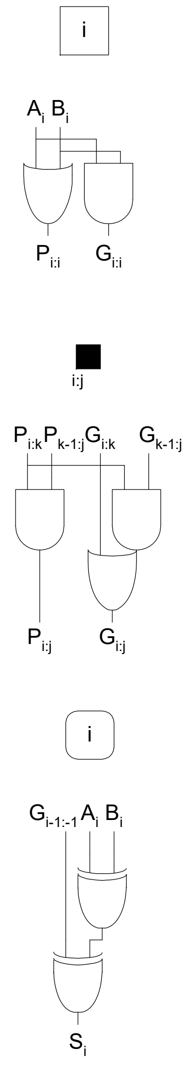


# Example for $S_6 = (A_6 \oplus B_6) \oplus G_{5:-1}$



Note that:

- $P_{x:-1}$ is always zero and does not have to be propagated but still shown for convenience of having regular approach
- $P_{x:x}$ and $G_{x:x}$ are the same as $P_x$ and $G_x$



# Prefix Adder (Crude) Delay and Complexity

$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_\text{prefix}}) + t_{\text{XOR}}$$

- $t_{pg}$: delay to produce $P_i$ $G_i$ (AND or OR gate)
- $t_{pg\_\text{prefix}}$: delay of black prefix cell (AND-OR gate)

$$A_{PA} = NA_{pg} + N/2\log_2 N\,(A_{pg\_\text{prefix}}) + NA_{\text{XOR}}$$

- $A_{pg}$: area of circuit to produce $P_i$ $G_i$ (AND or OR gate)
- $A_{pg\_\text{prefix}}$: area of black prefix cell (AND-OR gate)
- $A_{XOR}$: area of 2 XOR gates

# Adder (Crude) Delay Comparisons

Compare delay of: 32-bit ripple-carry, carry-lookahead, and prefix adders

- CLA has 4-bit blocks
- 2-input gate delay = 100 ps; full adder delay = 300 ps

$$t_{\text{ripple}} = Nt_{FA} = 32(300\text{ ps})$$
$$= \mathbf{9.6\ ns}$$

$$t_{CLA} = t_{pg} + t_{pg\_\text{block}} + (N/k - 1)t_{\text{AND\_OR}} + kt_{FA}$$
$$= [100 + 600 + (7)200 + 4(300)]\text{ ps}$$
$$= \mathbf{3.3\ ns}$$

$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_\text{prefix}}) + t_{\text{XOR}}$$
$$= [100 + \log_2 32(200) + 100]\text{ ps}$$
$$= \mathbf{1.2\ ns}$$

## Adder (Crude) Area Comparisons

Compare areas of: 32-bit ripple-carry, carry-lookahead, and prefix adders

- CLA has 4-bit blocks
- All logic gates have the same area $A$

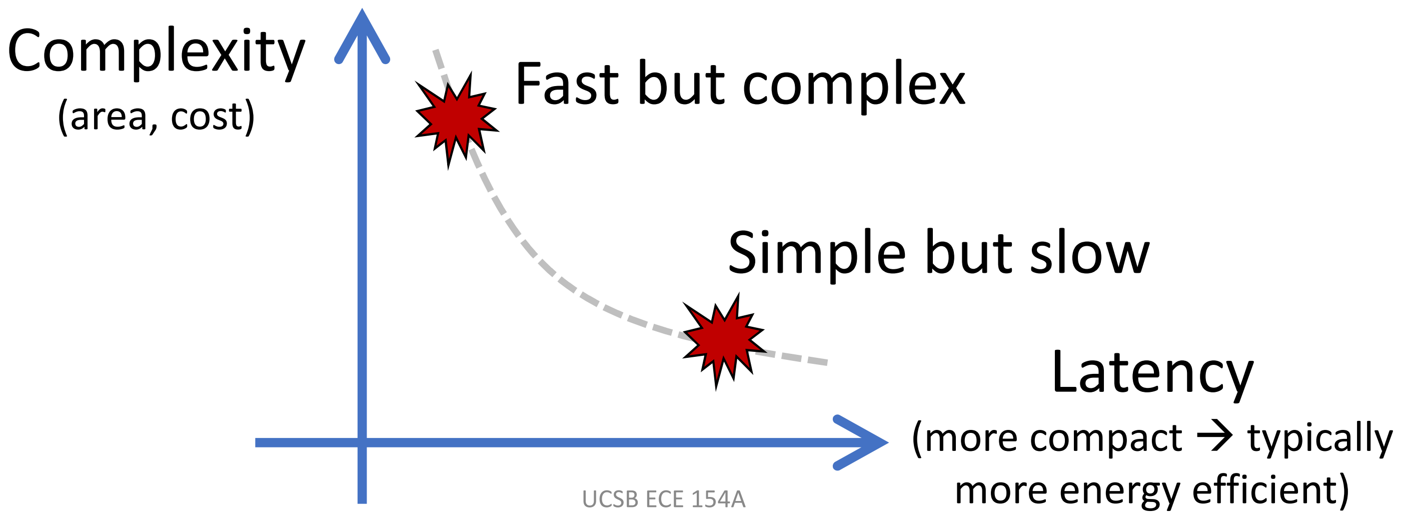
$$A_{\text{ripple}} = NA_{FA} = 32\times 6A$$
$$= \mathbf{192A}$$

$$A_{CLA} = (N/k-1)(A_{\text{AND\_OR}} + A_{pg} + A_{pg\_\text{block}}) + NA_{FA}$$
$$= (7)(2+2+7)A + 192A$$
$$= \mathbf{269A}$$

$$A_{PA} = NA_{pg} + N/2\log_2 N\,(A_{pg\_\text{prefix}}) + NA_{\text{XOR}} =$$
$$32\times 2A + 16\log_2 32\times 3A + 32\times 2A$$
$$= \mathbf{368A}$$

# Important Concepts to Remember

- Three types of adders: ripple-carry, carry-look ahead and prefix adder
- Different adders come with unique area-latency metrics
    - highlights typical area-latency tradeoffs in digital circuits
    - simple adders used in todays high-performance processors
        - FO4 delay: ~100 ps in 180 nm vs. ~ 1 ps in 5 nm


Complexity
(area, cost)
Fast but complex
Simple but slow
Latency
(more compact → typically more energy efficient)