Advanced VLSI

Intro to Computer Arithmetic

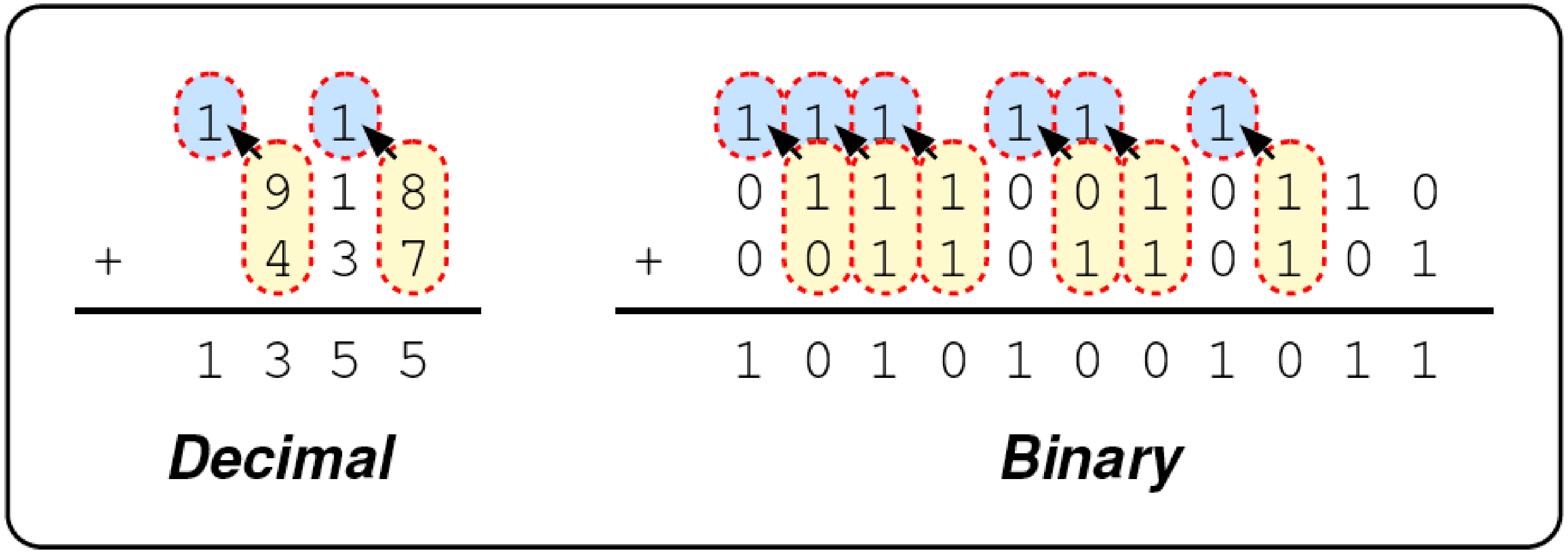
2025

Motivation

Arithmetic Circuits:

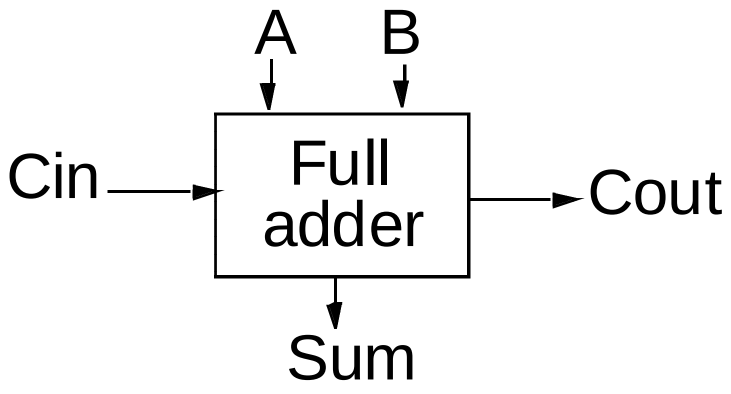
- **Core of every digital chip**
  Arithmetic circuits are the heart of digital chips.
- **Determine the performance of the system**
  The performance of digital chips is dictated by the arithmetic circuits. If the arithmetic circuit is optimized, performance improves.
- **Opportunities for improvement**
  New algorithms require novel combinations of arithmetic circuits. There is always room for improvement.

Adding Multiple Digits



Similar to Decimal Addition

- Starting from the Least Significant Digit, each digit is added.
- The carry of one digit is added to the digits on the right



The full adder is THE essential building block for almost all critical datapath operations!

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

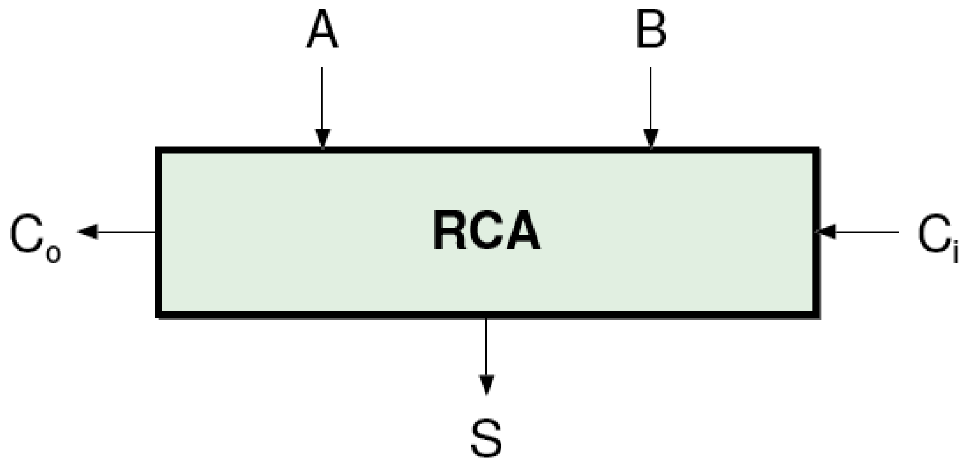
$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

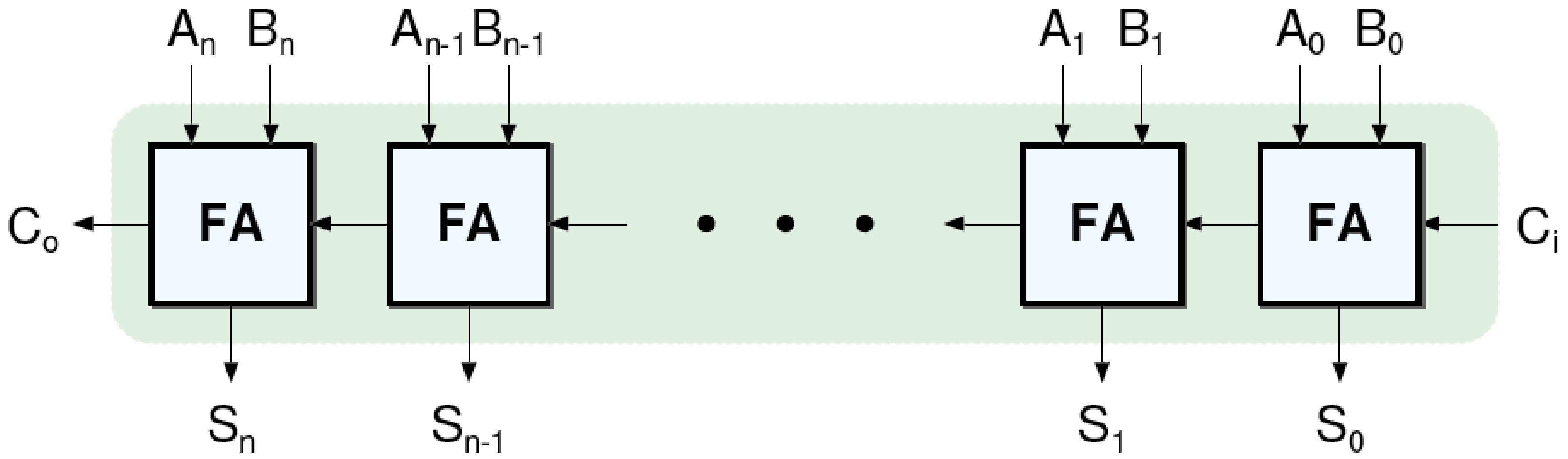
$$C_o = AB + BC_i + AC_i$$

$$\boxed{S = ABC + (A+B+C)\,\overline{C_o}}$$

useful for reducing circuit complexity !

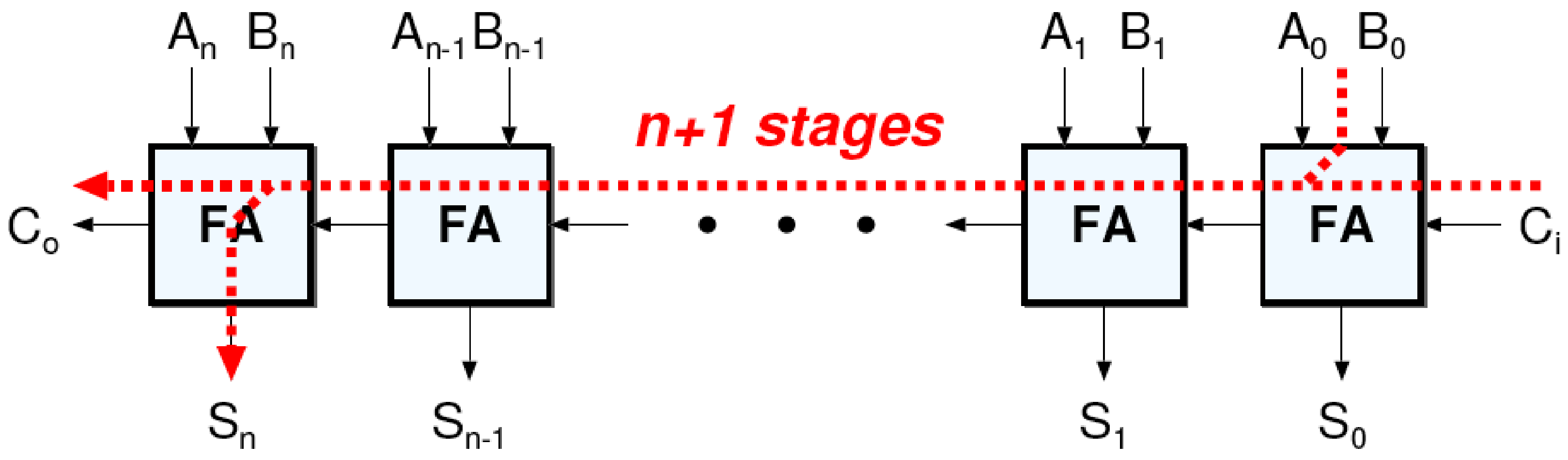
Ripple Carry Adder





Curse Of The Carry

The most significant output of the adder depends on the least significant inputs.



Carry Propagation

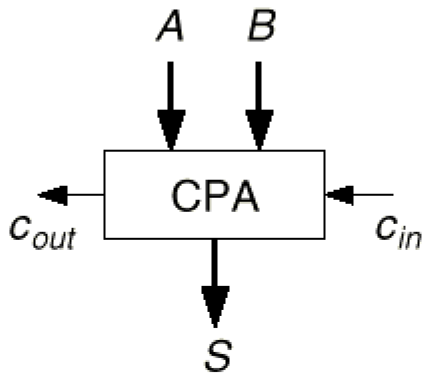
- The carry signal has to propagate from LSB to MSB
- **Fortunately, this case is rare**

Carry-Propagate Adders

- *Add* two n-bit operands A and B and an optional carry-in c_{in} by performing **carry-propagation** [1, 2, 11]
- *Sum* (c_{out}, S) is *irredundant* $(n+1)$-bit number

$$\boxed{(c_{out}, S) = c_{out}2^n + S = A + B + c_{in}}$$

$$\boxed{\begin{array}{r} 2c_{i+1} + s_i = a_i + b_i + c_i \ ; \\ i = 0, 1, \ldots, n-1 \\ c_0 = c_{in} \ , \ c_{out} = c_n \ \text{(r.m.a.)} \end{array}}$$

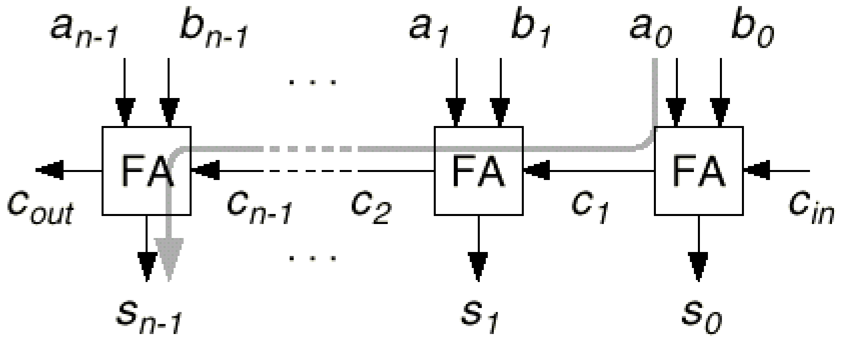


Carry-Propagate Adders

Ripple-carry adder (RCA)

- *Serial arrangement* of n full-adders
- *Simplest*, *smallest*, and *slowest* CPA structure

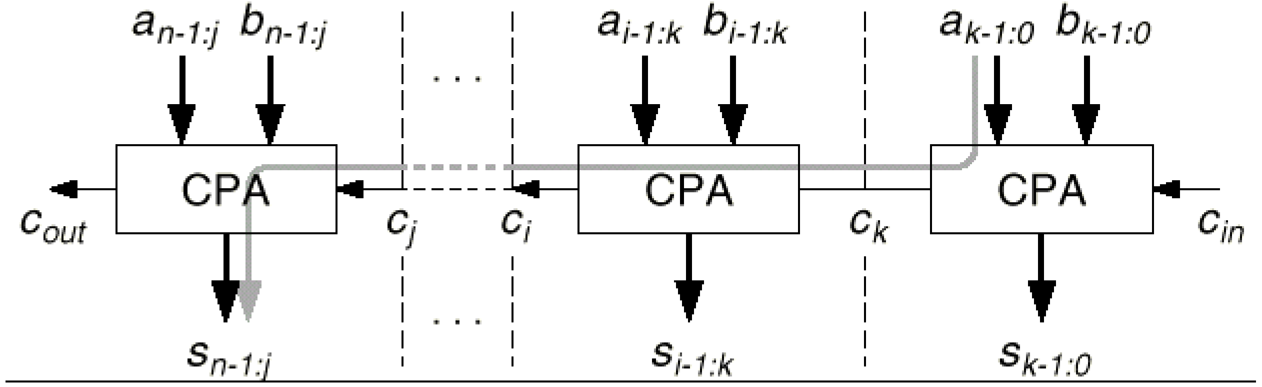
$$\boxed{A = 7n\ ,\ T = 2n\ ,\ AT = 14n^2}$$



Carry-Propagate Adders

Carry-propagation speed-up techniques

a) Concatenation of *partial CPAs* with fast $c_{in} \rightarrow c_{out}$



Basic idea: try to do the carry propagation quickly, independently of the result for each bit and then enter the fast carry into partial adders which operate otherwise independently on parts of the sum

Full Adder Revisited

Carry Propagation

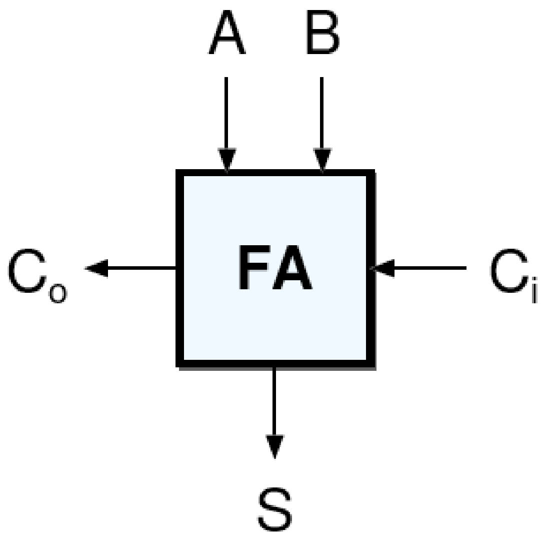
Consider the path $\mathrm{C_i} \rightarrow \mathrm{C_o}$:

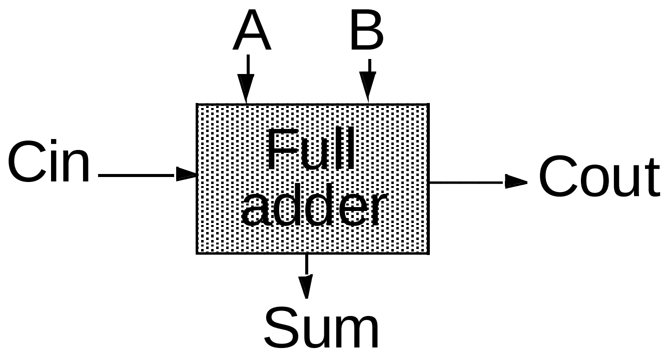
- If **A=1** and **B=1**
  A carry is **generated** no matter what the value of carry input $(\mathrm{C_o})$ is.

$$Generate = G = A \cdot B$$

- If only **A=1** or **B=1**
  The value of the carry input $(\mathrm{C_o})$ is **propagated** to the carry output.

$$Propagate = P = A \oplus B$$





A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

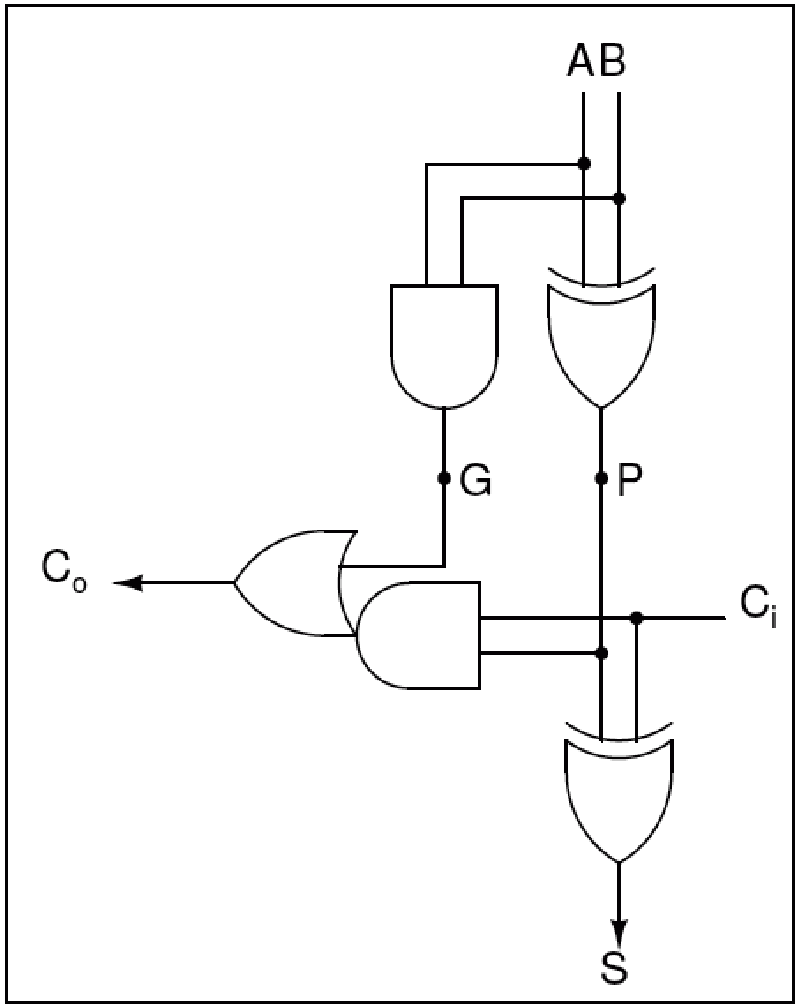
Full Adder with Propagate and Generate

Alternative Formulation

The Full Adder functionality can be expressed by using the Propagate (P) and Generate (G) signals:

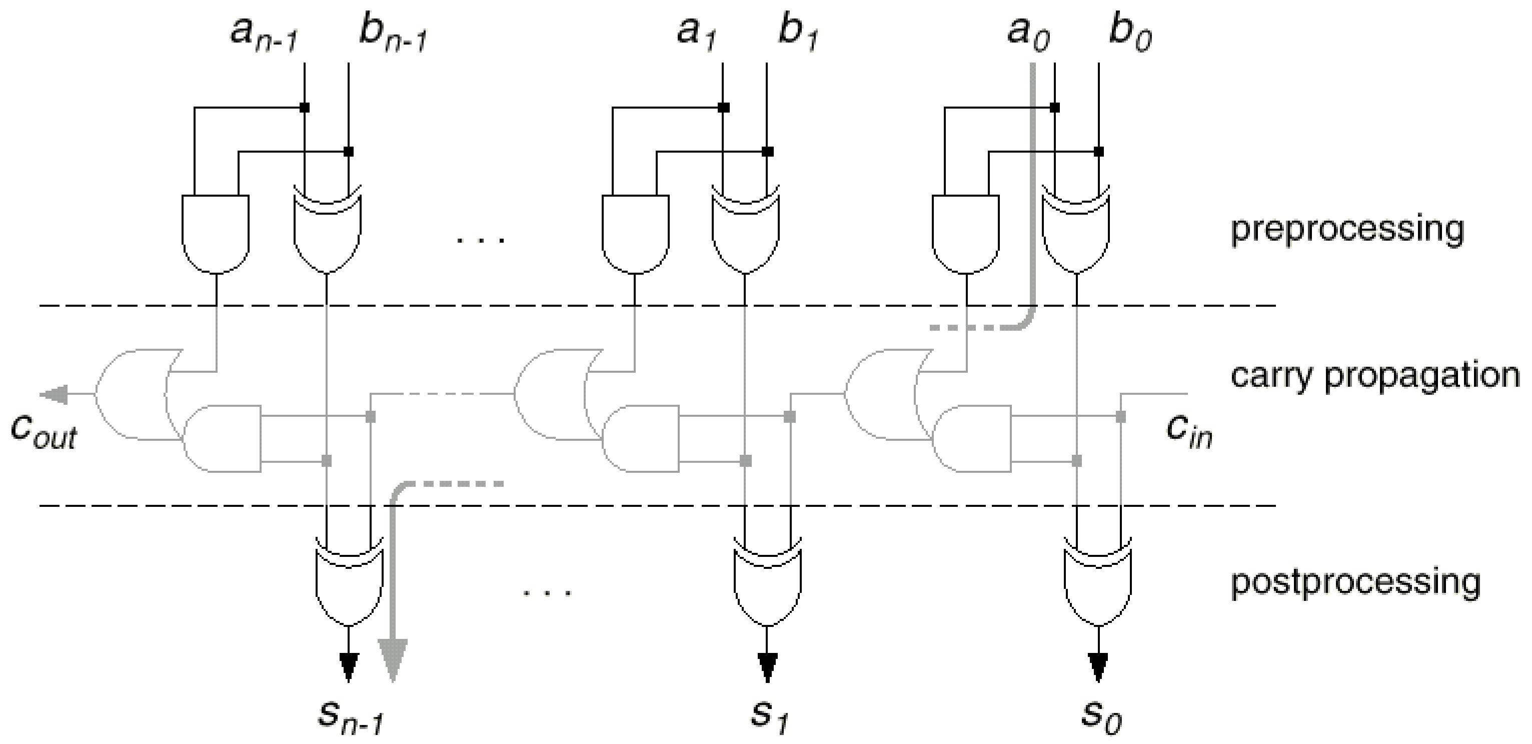
$$\begin{aligned} G &= A \cdot B \\ P &= A \oplus B \end{aligned}$$

$$\begin{aligned} S &= P \oplus C_i \\ C_o &= G + (P \cdot C_i) \end{aligned}$$



Area model:
3 simple gates (1x) and 2 complex (XOR) gates (2x) => 7
Delay model: 2 simple gates AB->CO => 2

Carry-Propagate Adders



Analysis of carry propagation

Sum digit in radix r	s_i	=	$(x_i + y_i + c_i) \bmod r$
Special case of radix 2	s_i	=	$x_i \oplus y_i \oplus c_i$

Computing the carries is thus our central problem

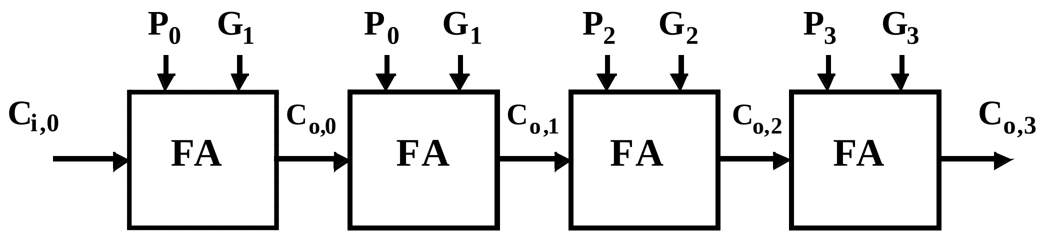
For this, the actual operand digits are not important

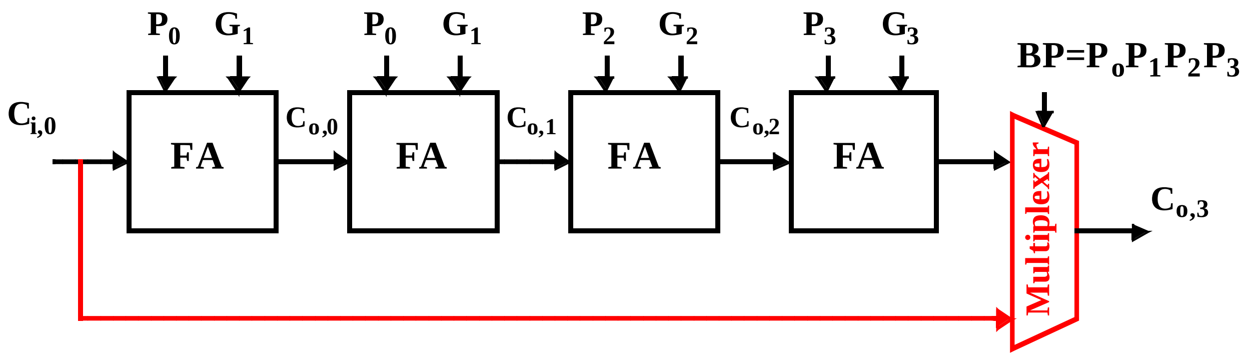
What matters is whether in a given position a carry is

$\boxed{\text{generated,}}$ $\boxed{\text{propagated,}}$ or $\boxed{\text{annihilated}}$ (absorbed)

$\boxed{g_i = x_i\ y_i}$ $\boxed{p_i = x_i \oplus y_i}$ $\boxed{a_i = \bar{x}_i\ \bar{y}_i} = \overline{x_i + y_i}$

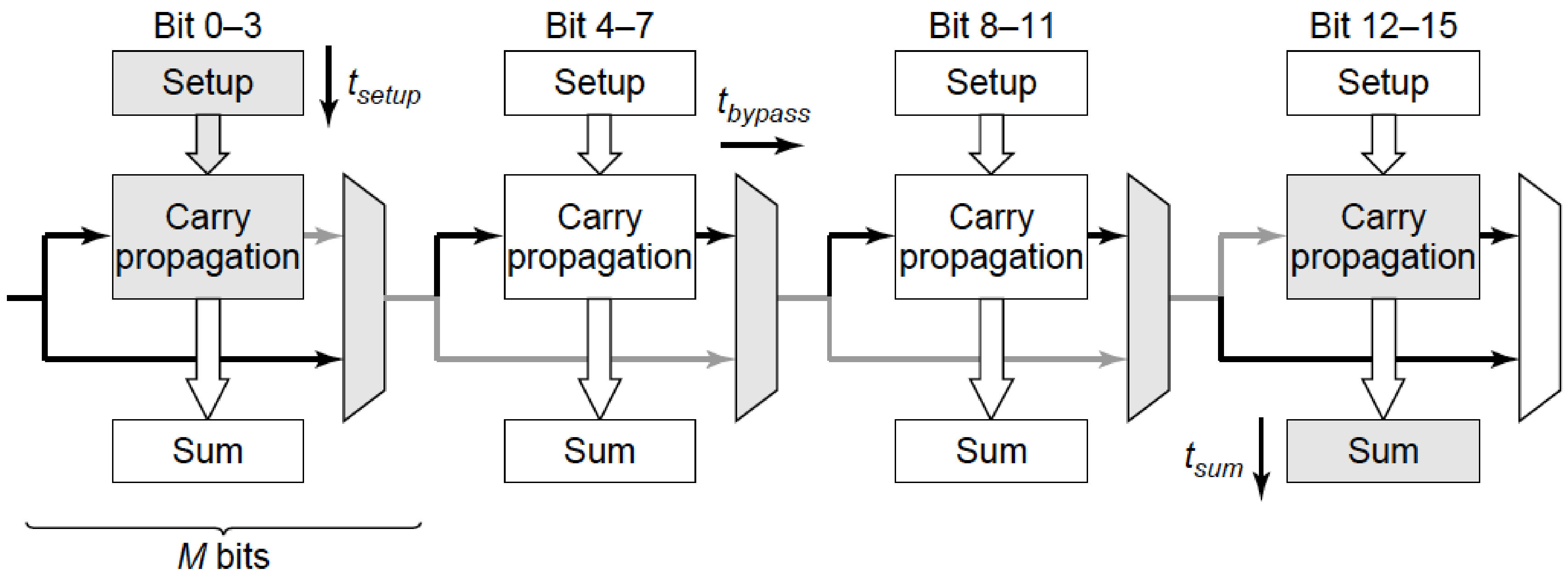
Carry-Skip Adders





Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Skip Adders



$$t_{adder} = t_{setup} + Mt_{carry} + \left(\frac{N}{M} - 1\right) t_{bypass} + (M-1)t_{carry} + t_{sum}$$

Carry-skip adder (CSKA)

- Type a) : partial CPA with fast $c_k \rightarrow c_i$

$$\boxed{\begin{array}{ll} c_i = \overline{P}_{i-1:k} c_i' + P_{i-1:k} c_k & \text{(bit group } (a_{i-1}, \ldots, a_k)) \\ P_{i-1:k} = p_{i-1} p_{i-2} \cdots p_k & \text{(group propagate)} \end{array}}$$

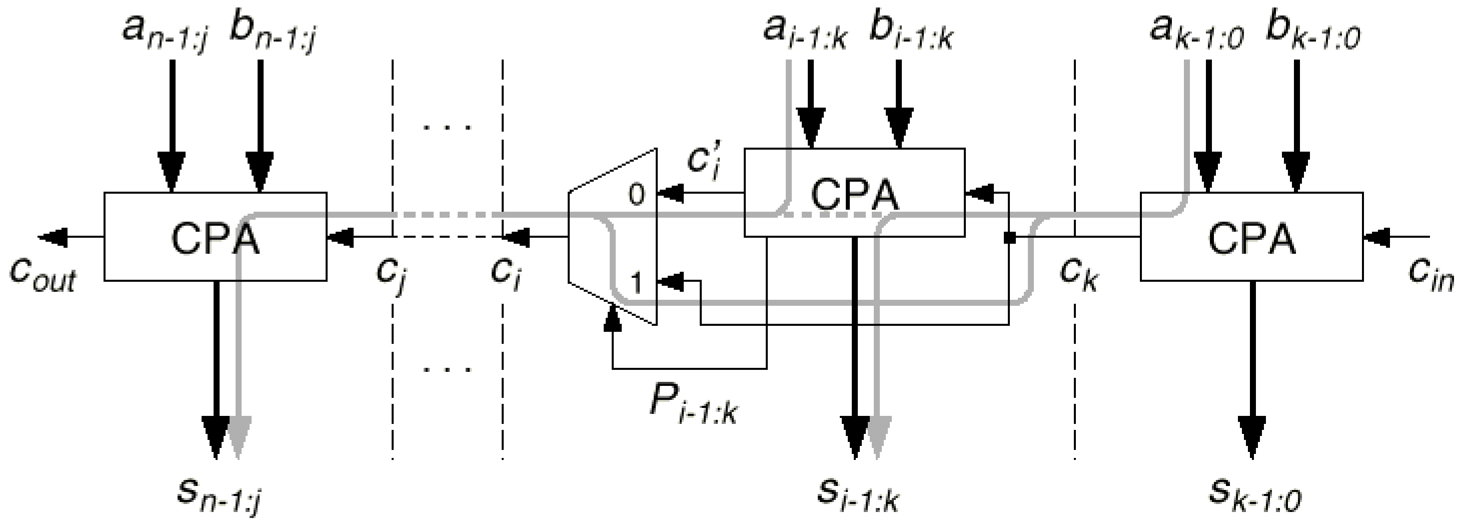
- 1) $P_{i-1:k} = 0$: $c_k \not\rightarrow c_i'$ and c_i' selected ($c_i' \rightarrow c_i$)
  2) $P_{i-1:k} = 1$: $c_k \rightarrow c_i'$ but c_i' **skipped** ($c_i' \not\rightarrow c_i$)

$\Rightarrow$ path $c_k \rightarrow c_i' \rightarrow c_i$ *never sensitized* $\Rightarrow$ fast $c_k \rightarrow c_i$
$\Rightarrow$ *false* path $\Rightarrow$ inherent *logic redundancy* $\Rightarrow$ problems in circuit optimization, timing analysis, and testing

- *Variable* group sizes (faster) : larger groups in the *middle* (minimize delays $a_0 \rightarrow c_k \rightarrow s_{i-1}$ and $a_k \rightarrow c_i \rightarrow s_{n-1}$)

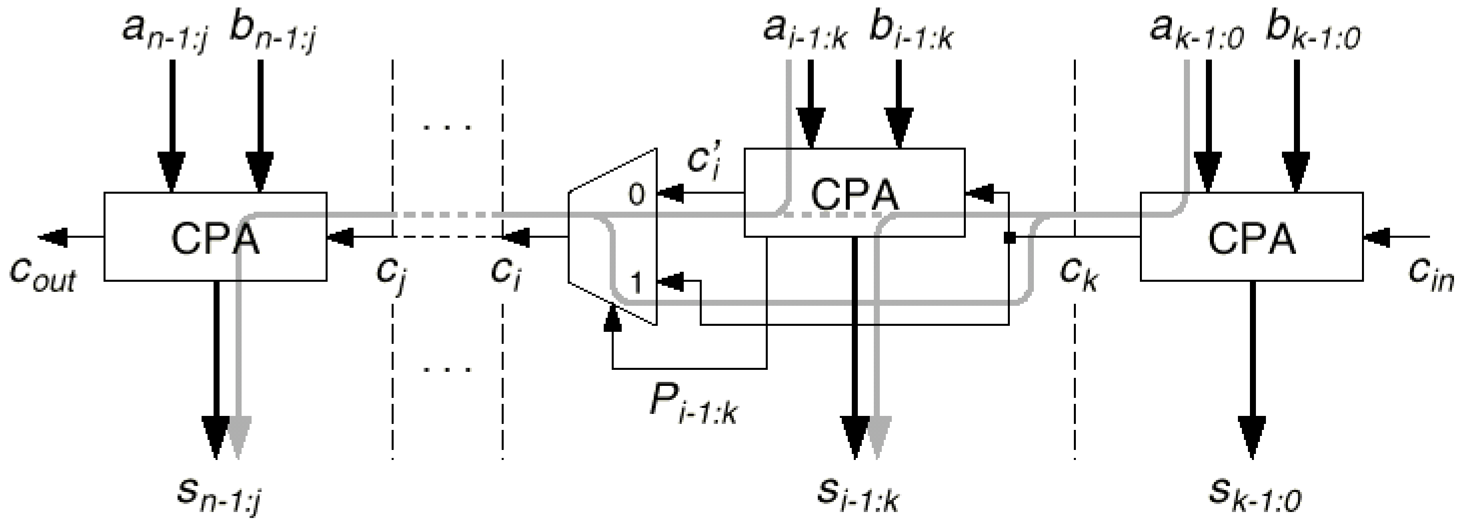
- Partial CPA typ. is RCA or CSKA ($\Rightarrow$ *multilevel* CSKA)
- *Medium* speed-up at *small* hardware overhead (+ AND/bit + MUX/group)

$$\boxed{A \approx 8n\ ,\ T \approx 4n^{1/2}\ ,\ AT \approx 32n^{3/2}}$$



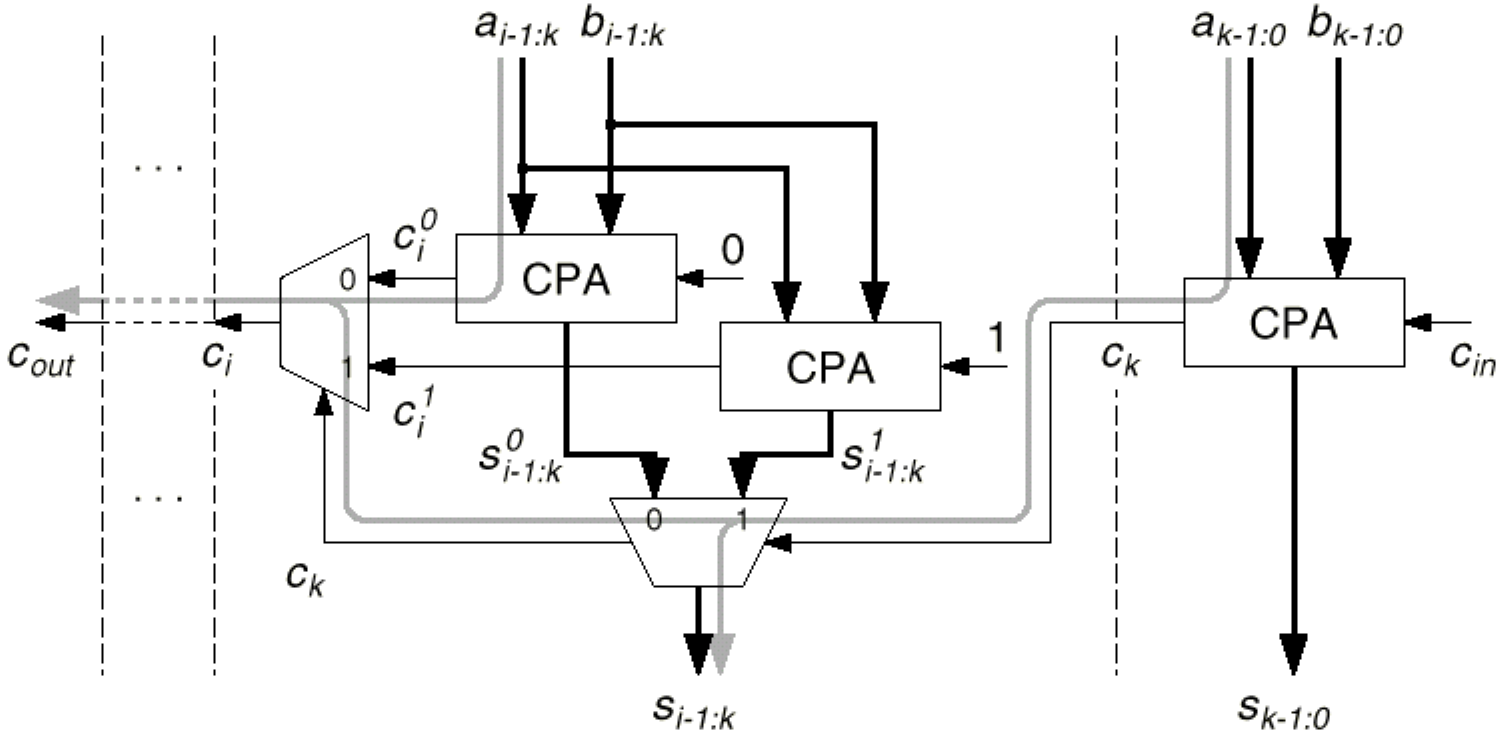
- Partial CPA typ. is RCA or CSKA ($\Rightarrow$ *multilevel* CSKA)
- *Medium* speed-up at *small* hardware overhead (+ AND/bit + MUX/group)

$$\boxed{A \approx 8n\ ,\ T \approx 4n^{1/2}\ ,\ AT \approx 32n^{3/2}}$$



Carry-select adder (CSLA)

$$\boxed{A \approx 14n \;,\; T \approx 2.8n^{1/2} \;,\; AT \approx 39n^{3/2}}$$



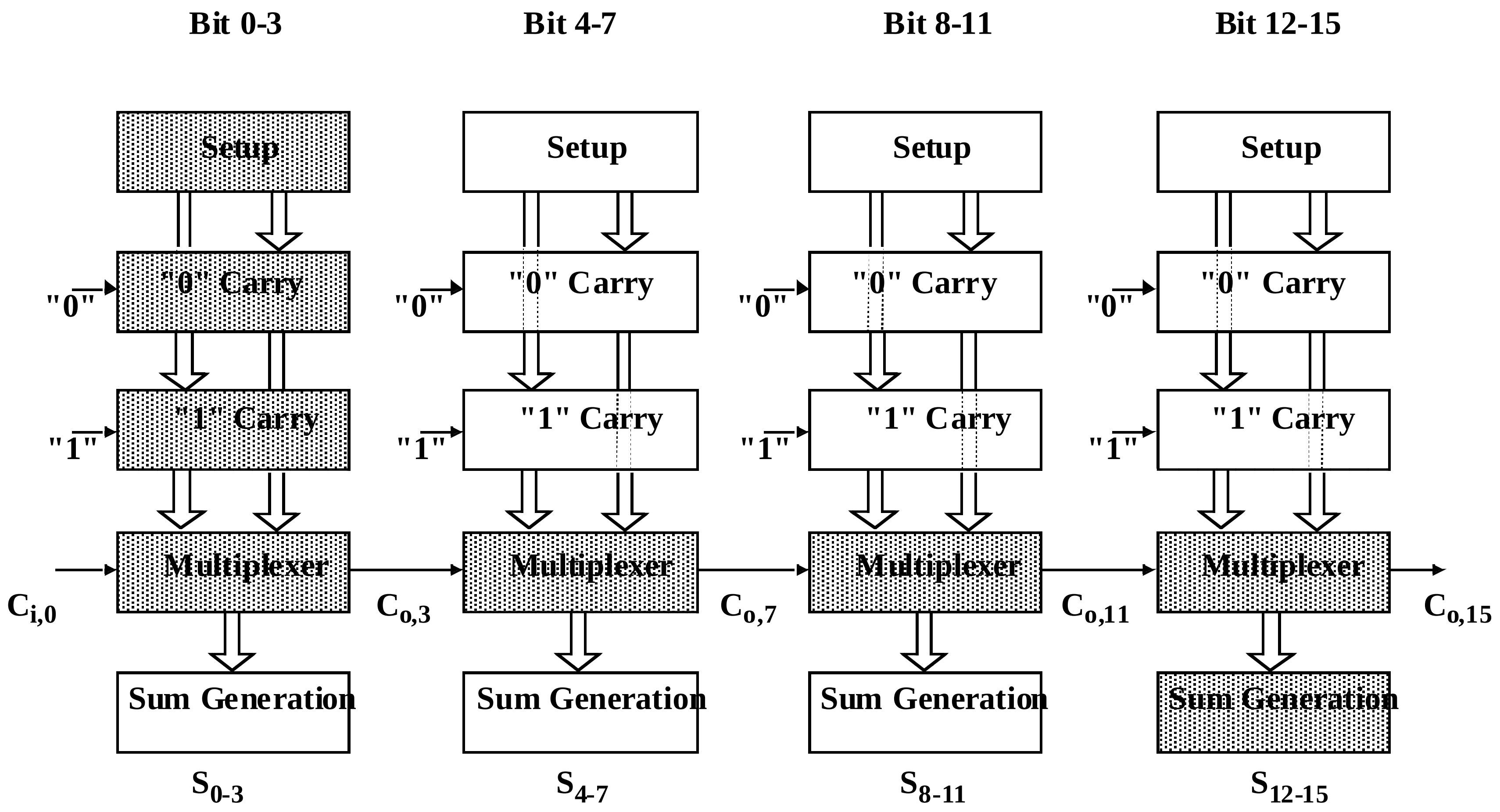
Carry-select adder (CSLA)

- Type a) : partial CPA with fast $c_k \rightarrow c_i$ and $c_k \rightarrow s_{i-1:k}$

$$\boxed{\begin{aligned} s_{i-1:k} &= \overline{c}_k s^0_{i-1:k} + c_k s^1_{i-1:k} \\ c_i &= \overline{c}_k c^0_i + c_k c^1_i \end{aligned}}$$

- *Two CPAs* compute two possible results ($c_{in} = 0/1$), group carry-in c_k **selects** correct one afterwards
- *Variable* group sizes (faster) : larger groups at *end* (MSB) (balance delays $a_0 \rightarrow c_k$ and $a_k \rightarrow c^0_i$)
- Part. CPA typ. is RCA, CSLA ($\Rightarrow$ *multil.* CSLA), or CLA
- *High* speed-up at *high* hardware overhead (+ MUX/bit + (CPA + MUX)/group)

Carry-Select Adders



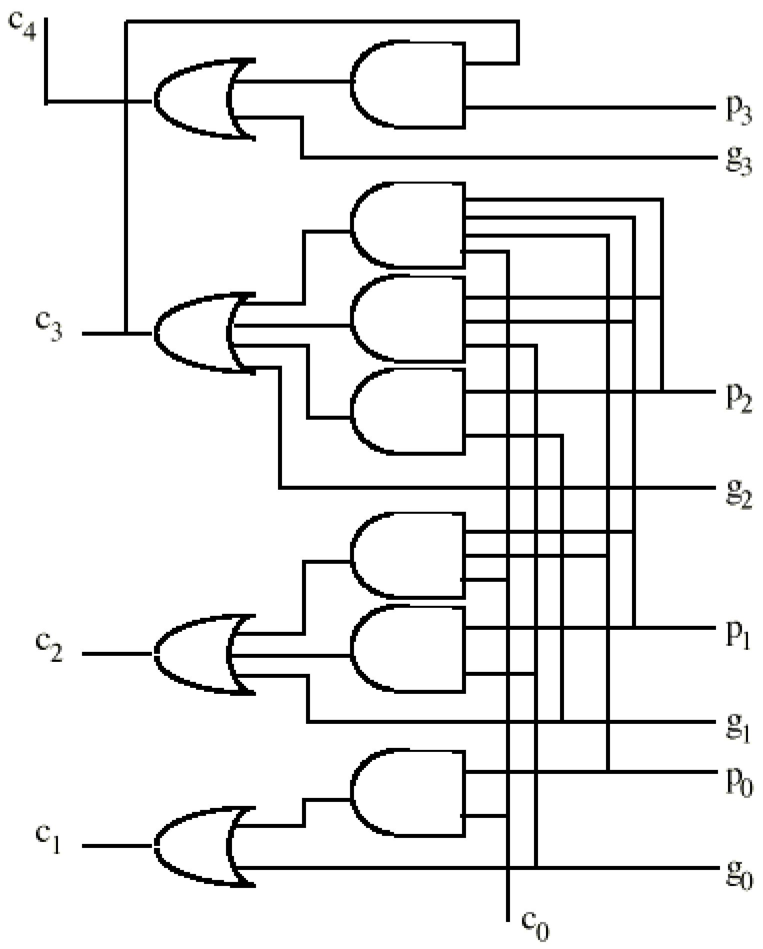
Analysis of carry propagation

carry recurrence

$$
\begin{aligned}
c_i &= g_{i-1} + c_{i-1}p_{i-1} \\
&= g_{i-1} + (g_{i-2} + c_{i-2}p_{i-2})p_{i-1} \\
&= g_{i-1} + g_{i-2}p_{i-1} + c_{i-2}p_{i-2}p_{i-1} \\
&= g_{i-1} + g_{i-2}p_{i-1} + g_{i-3}p_{i-2}p_{i-1} + c_{i-3}p_{i-3}p_{i-2}p_{i-1} \\
&= g_{i-1} + g_{i-2}p_{i-1} + g_{i-3}p_{i-2}p_{i-1} + g_{i-4}p_{i-3}p_{i-2}p_{i-1} \\
&\qquad + c_{i-4}p_{i-4}p_{i-3}p_{i-2}p_{i-1} \\
&= \quad \ldots
\end{aligned}
$$

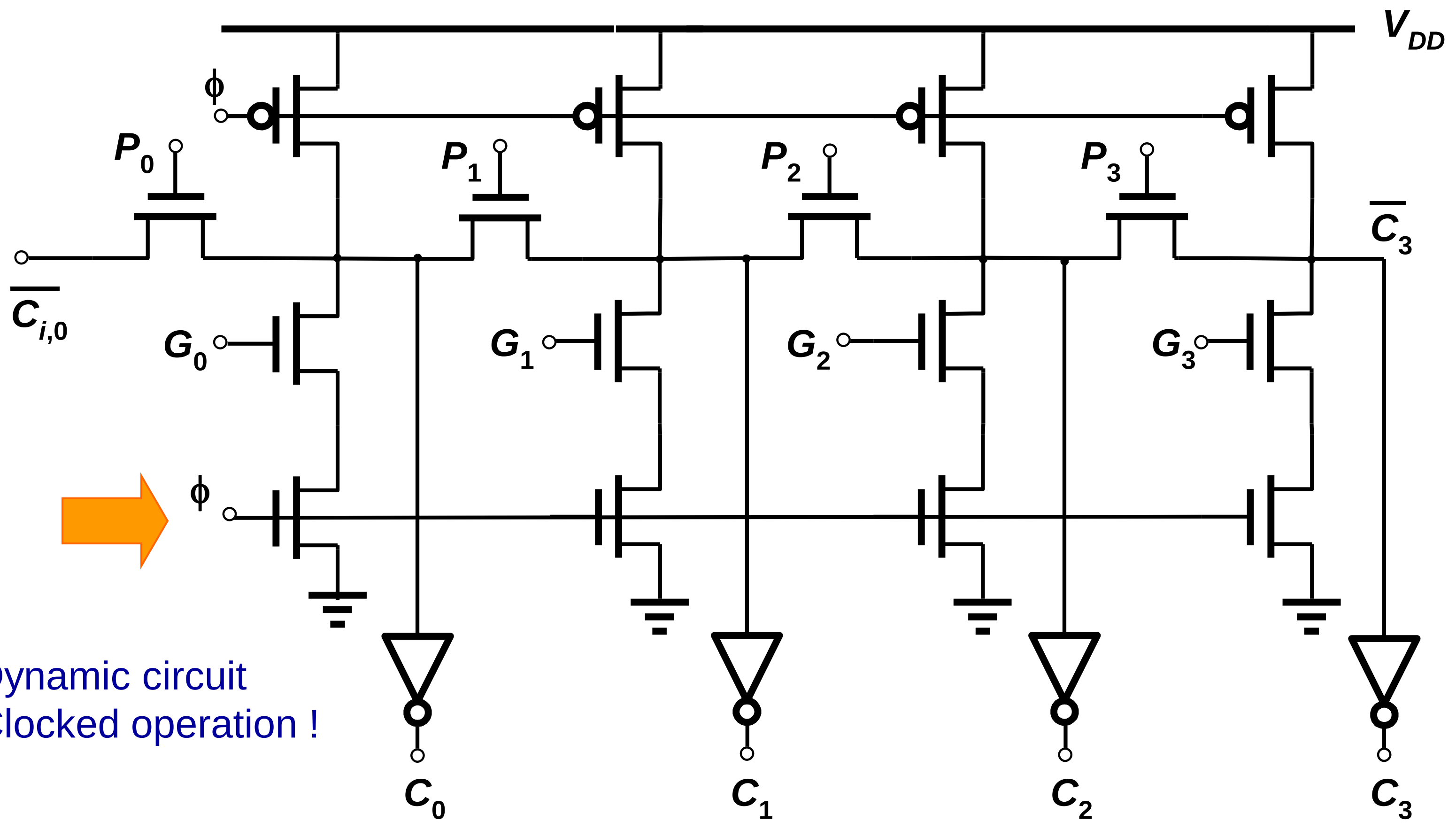
$$
\boxed{
\begin{aligned}
c_4 &= g_3 + g_2p_3 + g_1p_2p_3 + g_0p_1p_2p_3 + c_0p_0p_1p_2p_3 \\
c_3 &= g_2 + g_1p_2 + g_0p_1p_2 + c_0p_0p_1p_2 \\
c_2 &= g_1 + g_0p_1 + c_0p_0p_1 \\
c_1 &= g_0 + c_0p_0
\end{aligned}
}
$$

Carry-Lookahead Adder (CLA)



Four-bit carry network with full lookahead.

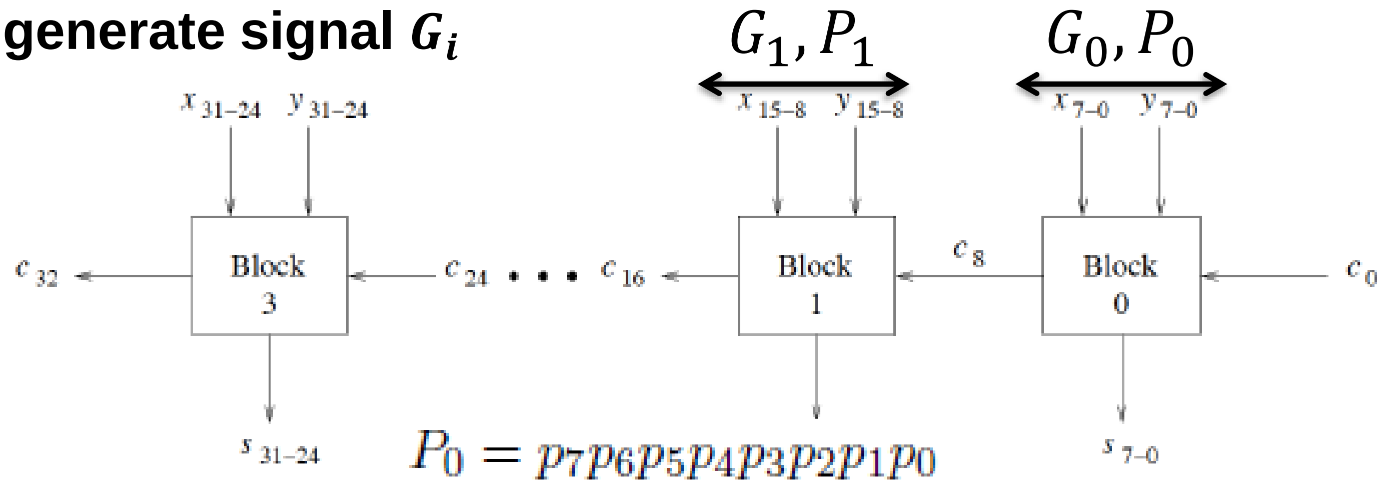
Manchester Carry Chain Circuit



Dynamic circuit
Clocked operation !

CLA from combined (prefix) Generate Propagate Signals

- Define individual sub-adders each comprise multiple bits
- Fore each sub-adder define **propagate signal P_i and a generate signal G_i**



$$P_0 = p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0$$

$$G_0 = g_7 + p_7 g_6 + p_7 p_6 g_5 + \cdots + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0$$

$$c_8 = G_0 + P_0 c_0$$

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 + p_7 p_6 p_5 p_4 g_3 + p_7 p_6 p_5 p_4 p_3 g_2$$
$$+ p_7 p_6 p_5 p_4 p_3 p_2 g_1 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 g_0 + p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$$

Unrolling keeps growing the fanin and fanout ☹ and leads to CLA adder

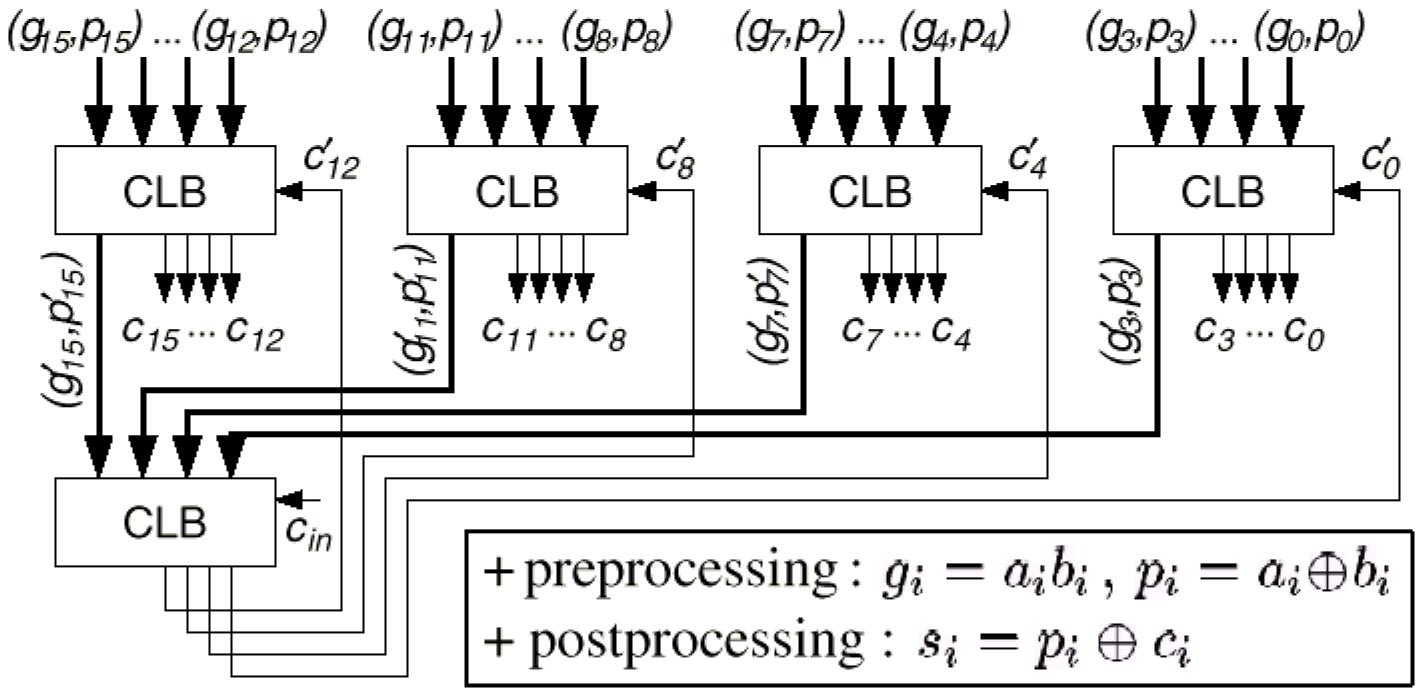
$$c_{16} = G_1 + P_1 c_8$$
$$= G_1 + P_1 G_0 + P_1 P_0 c_0$$

$$P_1 = p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15}$$

Carry-lookahead adder (CLA)

- *High* speed-up at *medium* hardware overhead

$$\boxed{A \approx 14n\ ,\ T \approx 4\log n\ ,\ AT \approx 56n\log n}$$



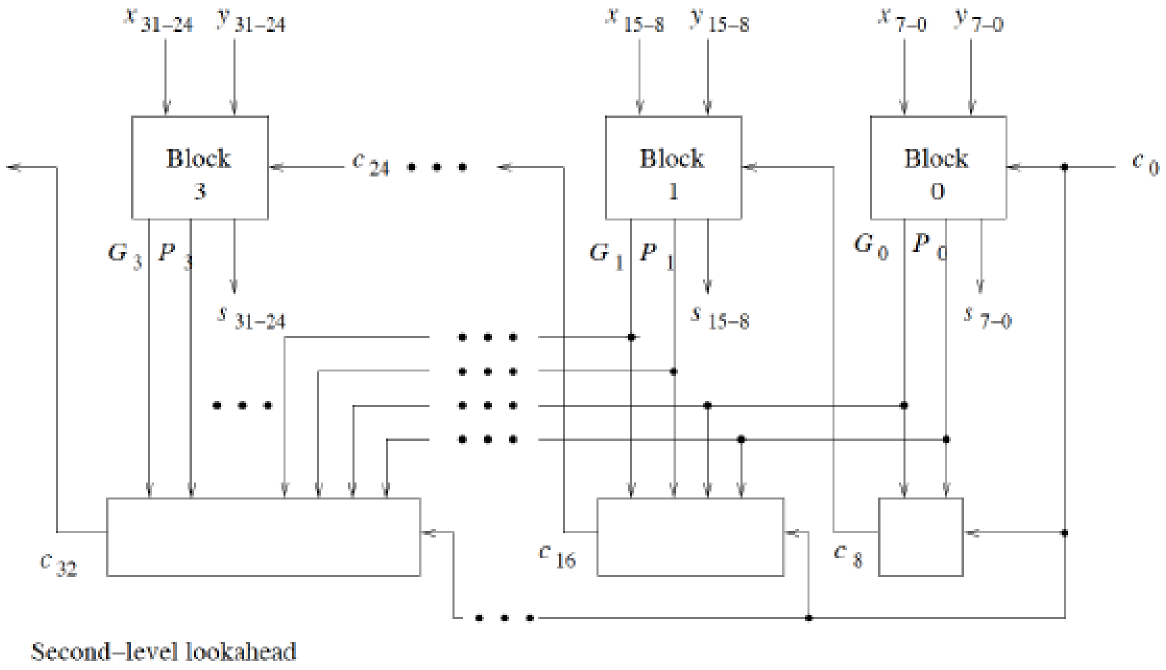
$$\boxed{\begin{array}{l} +\text{preprocessing}: \ g_i = a_i b_i\ ,\ p_i = a_i \oplus b_i \\ +\text{postprocessing}: \ s_i = p_i \oplus c_i \end{array}}$$

Hierarchical CLAs with two levels

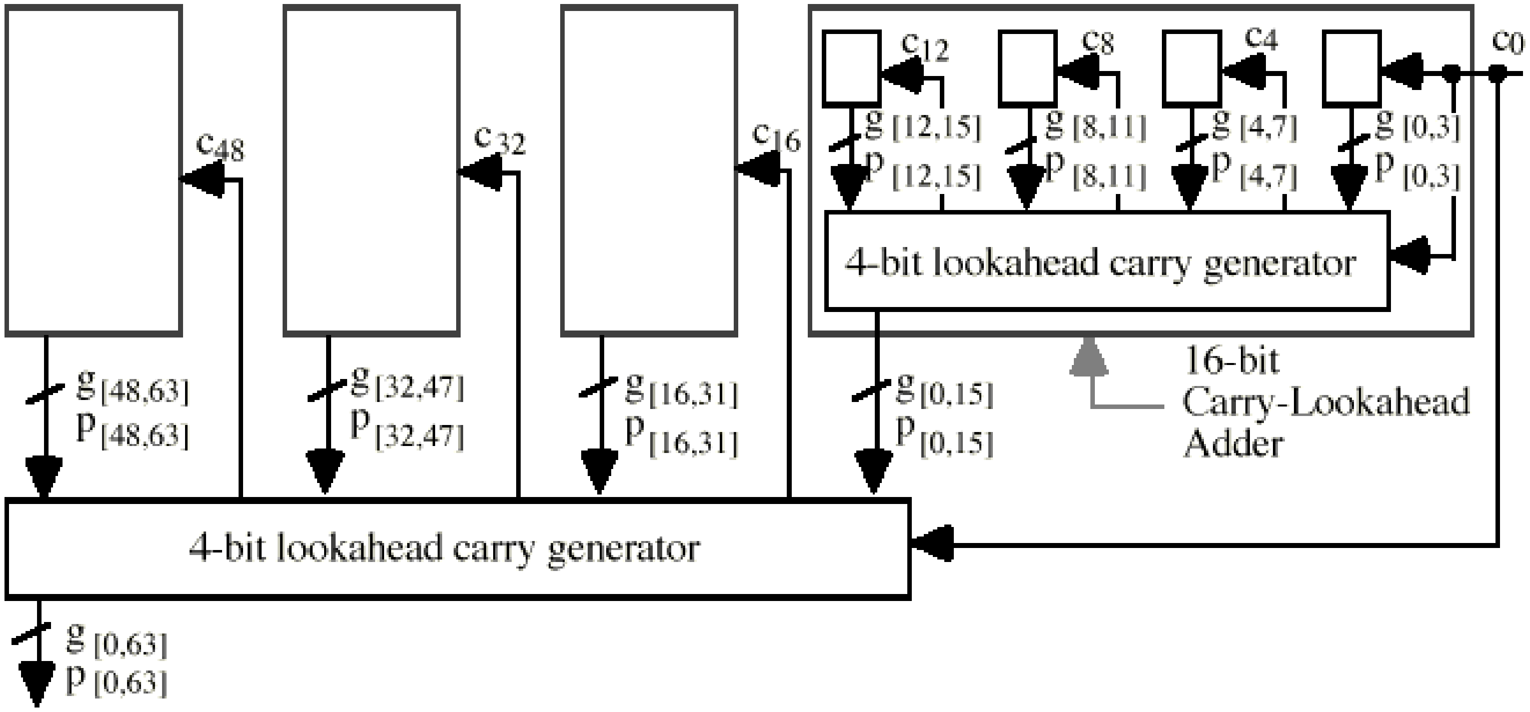
- Objective: decompose the carry propagation into multiple levels of logic
- Keep the hierarchical structure in two levels: fanin and fanout still grow but more slowly with each level

$$c_8 = G_0 + P_0 c_0$$

$$c_{16} = G_1 + P_1 c_8$$
$$= G_1 + P_1 G_0 + P_1 P_0 c_0$$



Carry-lookahead adder (CLA)



Building a 64-bit carry-lookahead adder from 16 4-bit adders and 5 lookahead carry generators.

Prefix Operations

We define a set of signals

$$(G^k_{i:j}, P^k_{i:j})$$

Which define the **combined** $Generate$ and $Propagate$ signals covering the bits from i to j at the k^{th} stage.

- The initial $Generate$ and $Propagate$ signals G_i and P_i will be written as

  $$(G^0_{i:i}, P^0_{i:i})$$

  in this notation.
- The goal is to calculate

  $$(G^k_{n-1:0}, P^k_{n-1:0})$$

  for all n, in any number of k stages.

Parallel Prefix Adders

Parallel Prefix Adders (PPA) represent a systematic approach to designing optimized adder architectures.

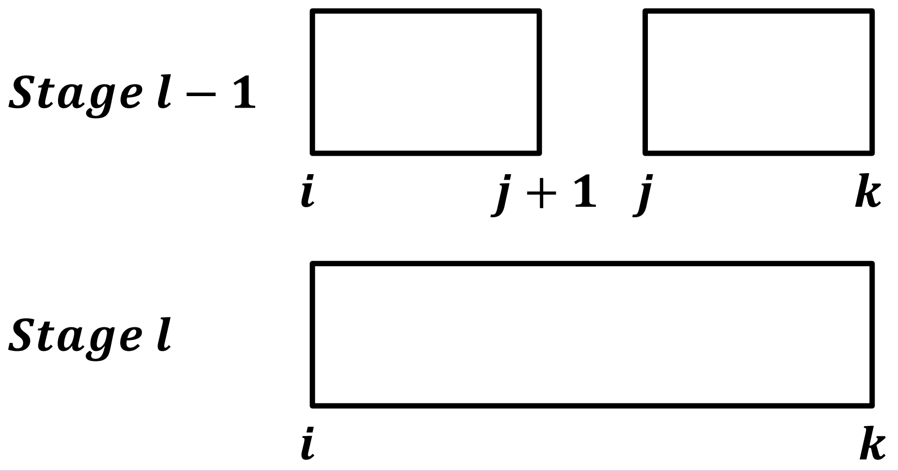
Principle

- Pre-processing (P and G generation)
- Carry Propagation Tree
- Post Processing (S generation)

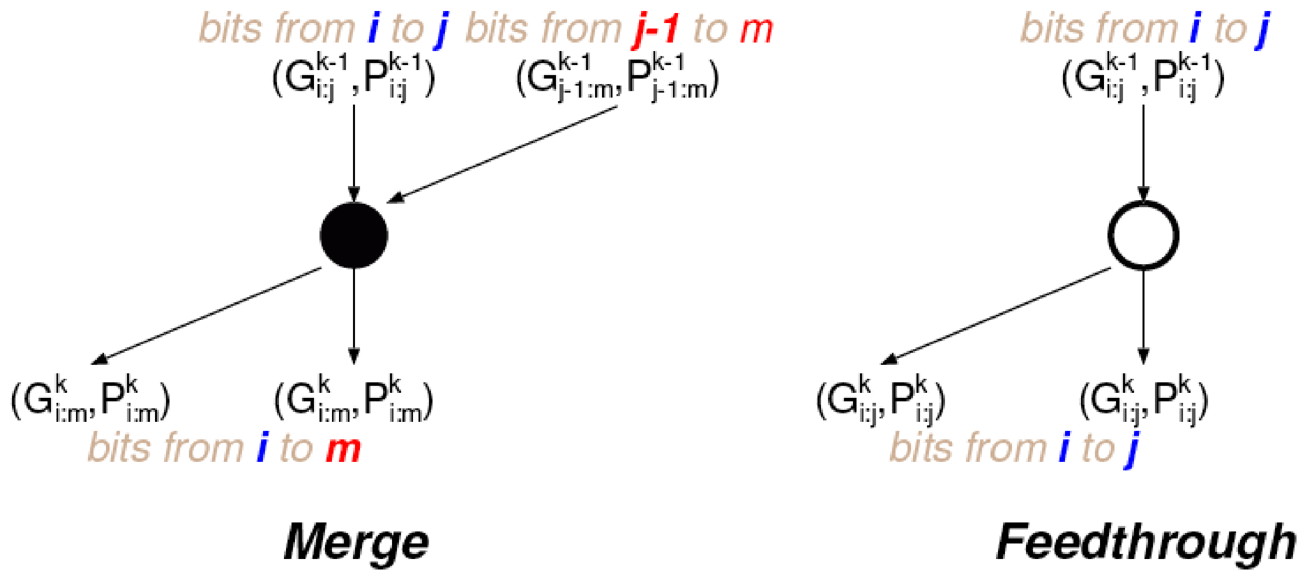
Merging Generate and Propagate Signals

- How to merge signals from two subsequent stages

$$
\begin{aligned}
(G^0_{i:i}, P^0_{i:i}) &= (g_i, p_i) \\
(G^l_{i:k}, P^l_{i:k}) &= (G^{l-1}_{i:j+1}, P^{l-1}_{i:j+1}) \bullet (G^{l-1}_{j:k}, P^{l-1}_{j:k}) \; ; \; k \le j \le i \\
&= (G^{l-1}_{i:j+1} + P^{l-1}_{i:j+1} G^{l-1}_{j:k}, P^{l-1}_{i:j+1} P^{l-1}_{j:k}) \\
c_{i+1} &= G^m_{i:0} \; ; \; i = 0, \ldots, n-1 \; , \; l = 1, \ldots, m
\end{aligned}
$$



Main Parallel Prefix Operators


bits from *i* to *j*
bits from *j-1* to *m*
$(G^{k-1}_{i:j}, P^{k-1}_{i:j})$
$(G^{k-1}_{j-1:m}, P^{k-1}_{j-1:m})$
$(G^{k}_{i:m}, P^{k}_{i:m})$
$(G^{k}_{i:m}, P^{k}_{i:m})$
bits from *i* to *m*
Merge
bits from *i* to *j*
$(G^{k-1}_{i:j}, P^{k-1}_{i:j})$
$(G^{k}_{i:j}, P^{k}_{i:j})$
$(G^{k}_{i:j}, P^{k}_{i:j})$
bits from *i* to *j*
Feedthrough

At each level we have two operations

- **Merge**
  Merges the two (P, G) of **adjacent** bit ranges.
- **Feedthrough**
  No operation, just copies the (P, G) signals to the next stage.

Prefix problem

- Inputs $(x_{n-1}, \ldots, x_0)$, outputs $(y_{n-1}, \ldots, y_0)$, associative binary operator $\bullet$ [11, 13]

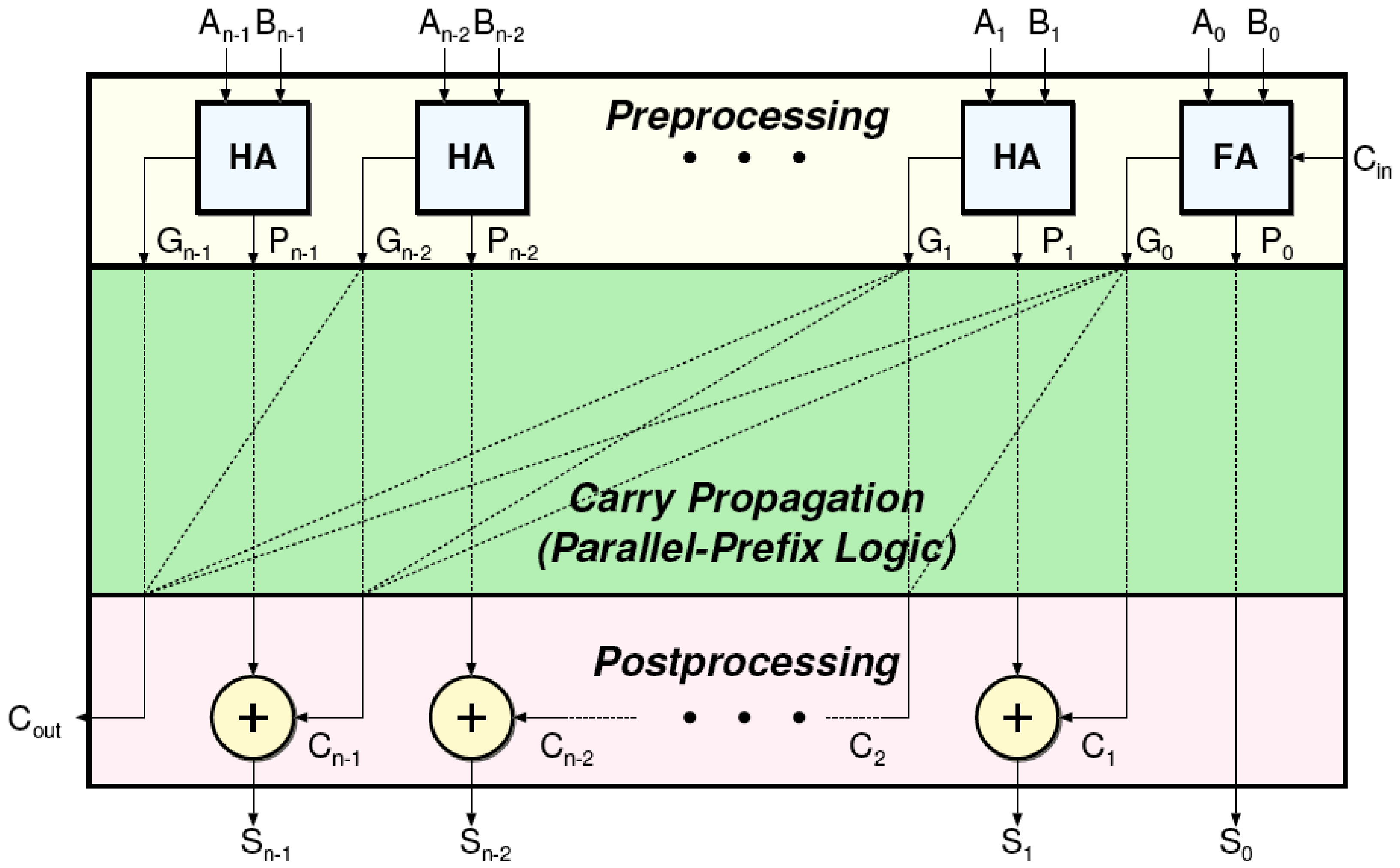
$$\boxed{\begin{array}{l} (y_{n-1}, \ldots, y_0) = (x_{n-1} \bullet \cdots \bullet x_0,\ \ldots,\ x_1 \bullet x_0,\ x_0) \quad \text{or} \\ y_0 = x_0\ ,\ y_i = x_i \bullet y_{i-1}\ ;\ i = 1, \ldots, n-1 \quad \text{(r.m.a.)} \end{array}}$$

- Associativity of $\bullet \Rightarrow$ *tree structures* for evaluation :

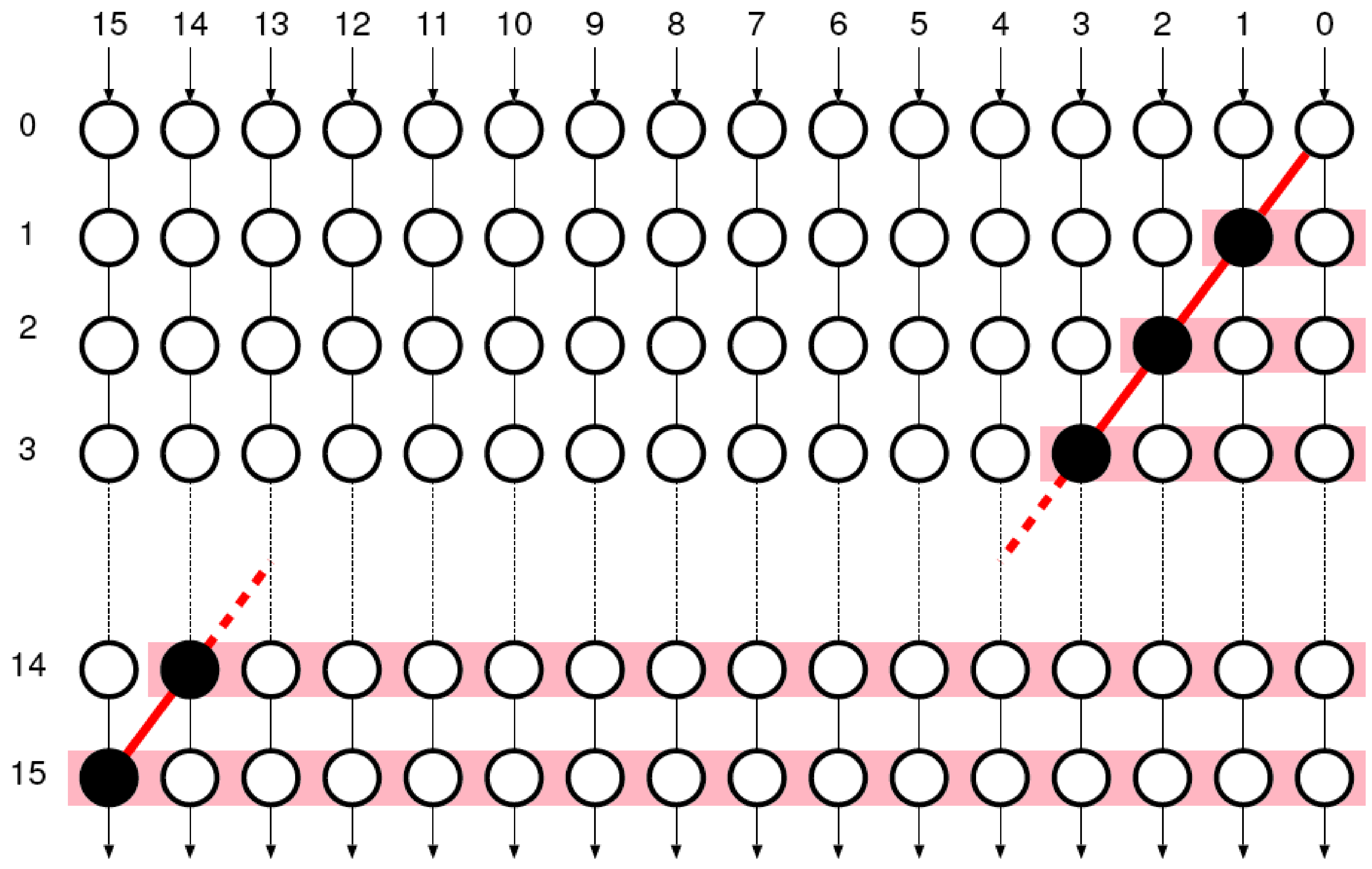
$$\underbrace{x_3 \bullet (\underbrace{x_2 \bullet (\underbrace{x_1 \bullet x_0}_{y_1 = Y^1_{1:0}})}_{y_2 = Y^2_{2:0}})}_{y_3 = Y^3_{3:0}} = \underbrace{(\underbrace{x_3 \bullet x_2}_{Y^1_{3:2}}) \bullet (\underbrace{x_1 \bullet x_0}_{y_1 = Y^1_{1:0}})}_{y_3 = Y^2_{3:0}} \quad \text{, but } y_2 \text{ ?}$$

- *Group variables* $Y^l_{i:k}$: covers bits $(x_k, \ldots, x_i)$ at level l
- *Carry-propagation* is prefix problem : $Y^l_{i:k} = (G^l_{i:k}, P^l_{i:k})$

Basic Structure of Parallel Prefix Adders



Example: Ripple Carry Adder (PPA-RCA)



PPA Performance Metrics

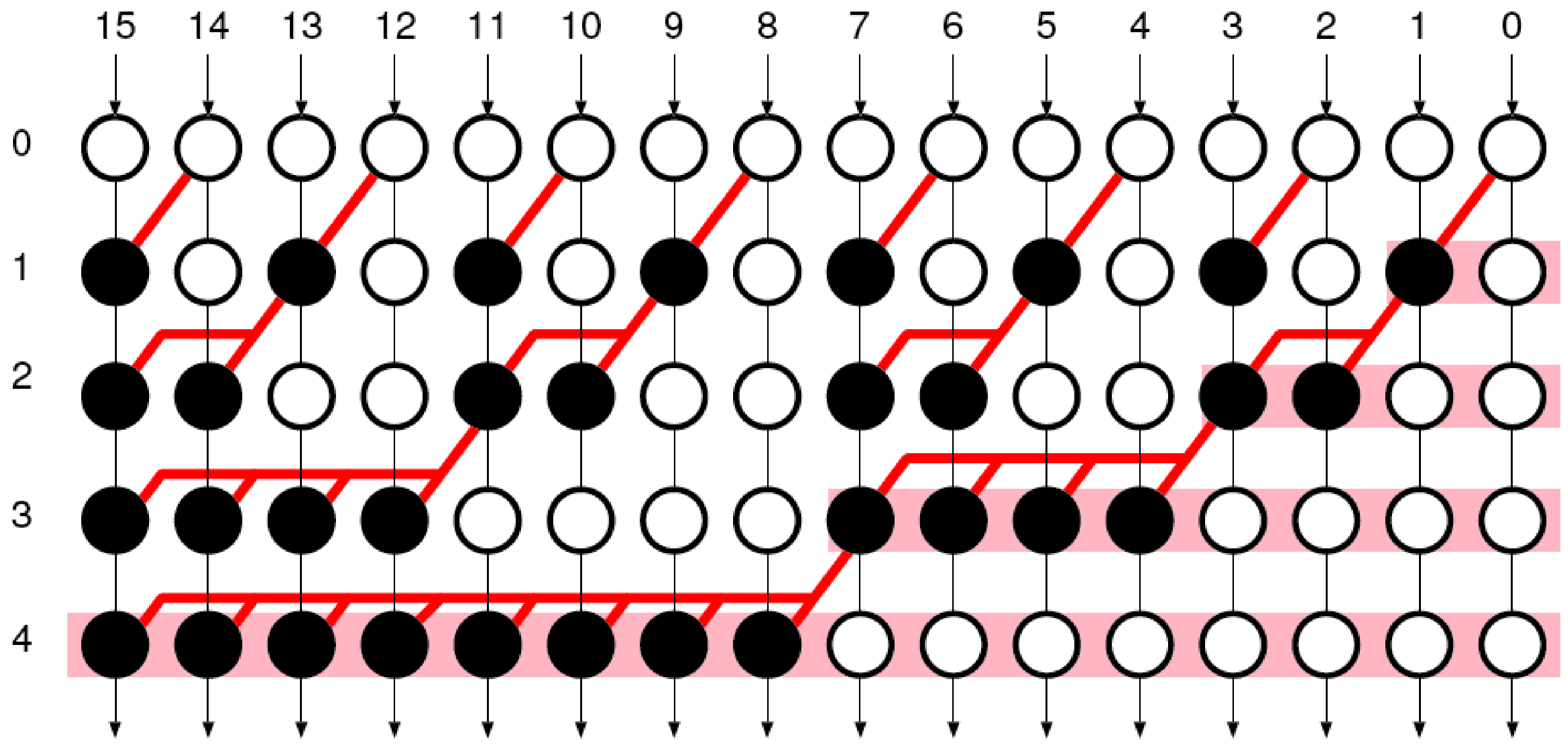
The following parameters define the performance

- **Number of Black Dots**
  Determines the area, since only the **Merge** cells contain logic:

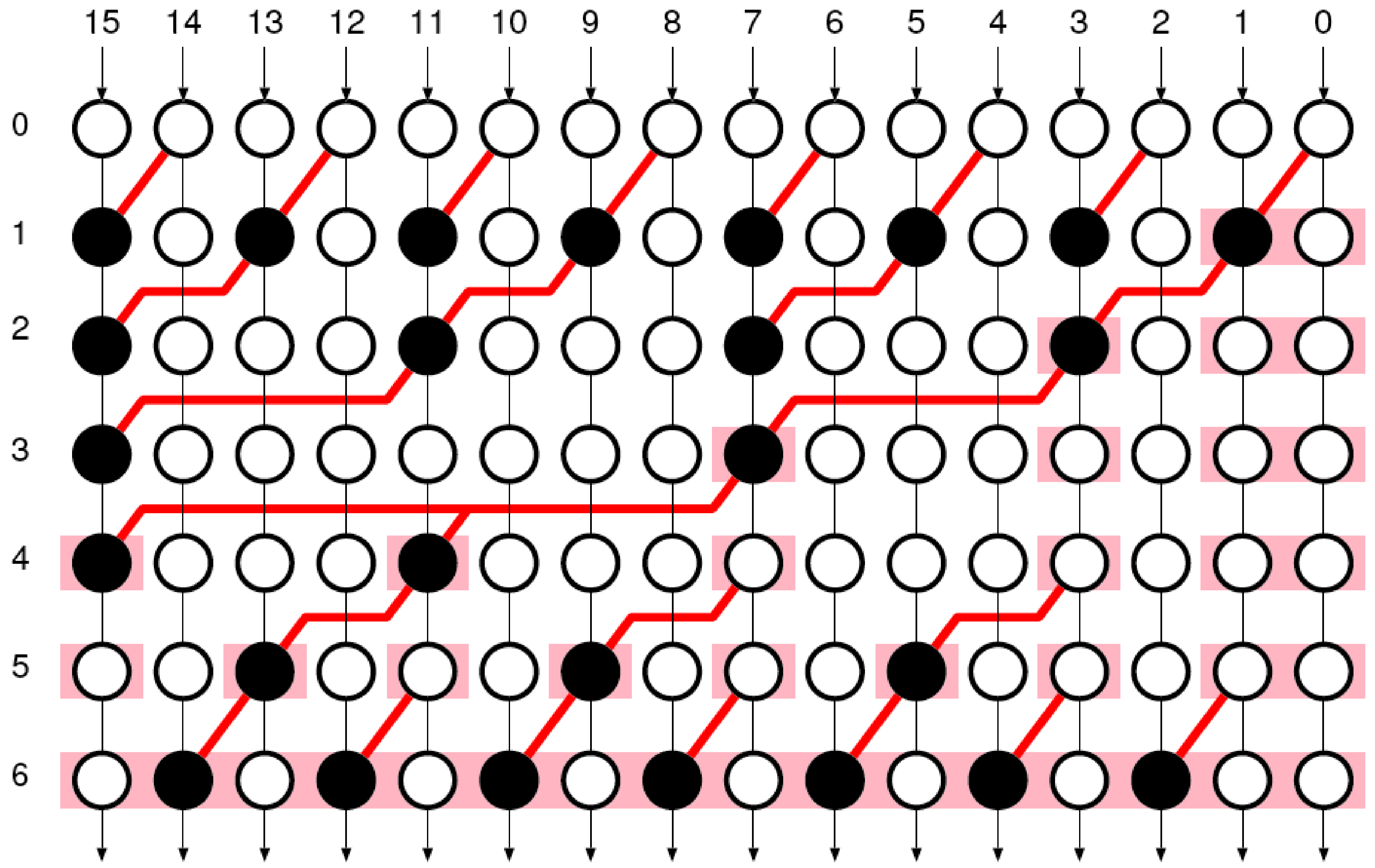
$$(G^k_{i:m}, P^k_{i:m}) = (G^{k-1}_{i:j} + P^{k-1}_{i:j} \cdot G^{k-1}_{j-1:m}, P^{k-1}_{i:j} \cdot P^{k-1}_{j-1:m})$$

- **Number of Stages**
  Determines the critical path $\rightarrow$
  fewer stages result in a faster the adder.
- **Maximum Fan Out**
  Determines the per stage delay.
  Cells with a higher fan out will be slower.

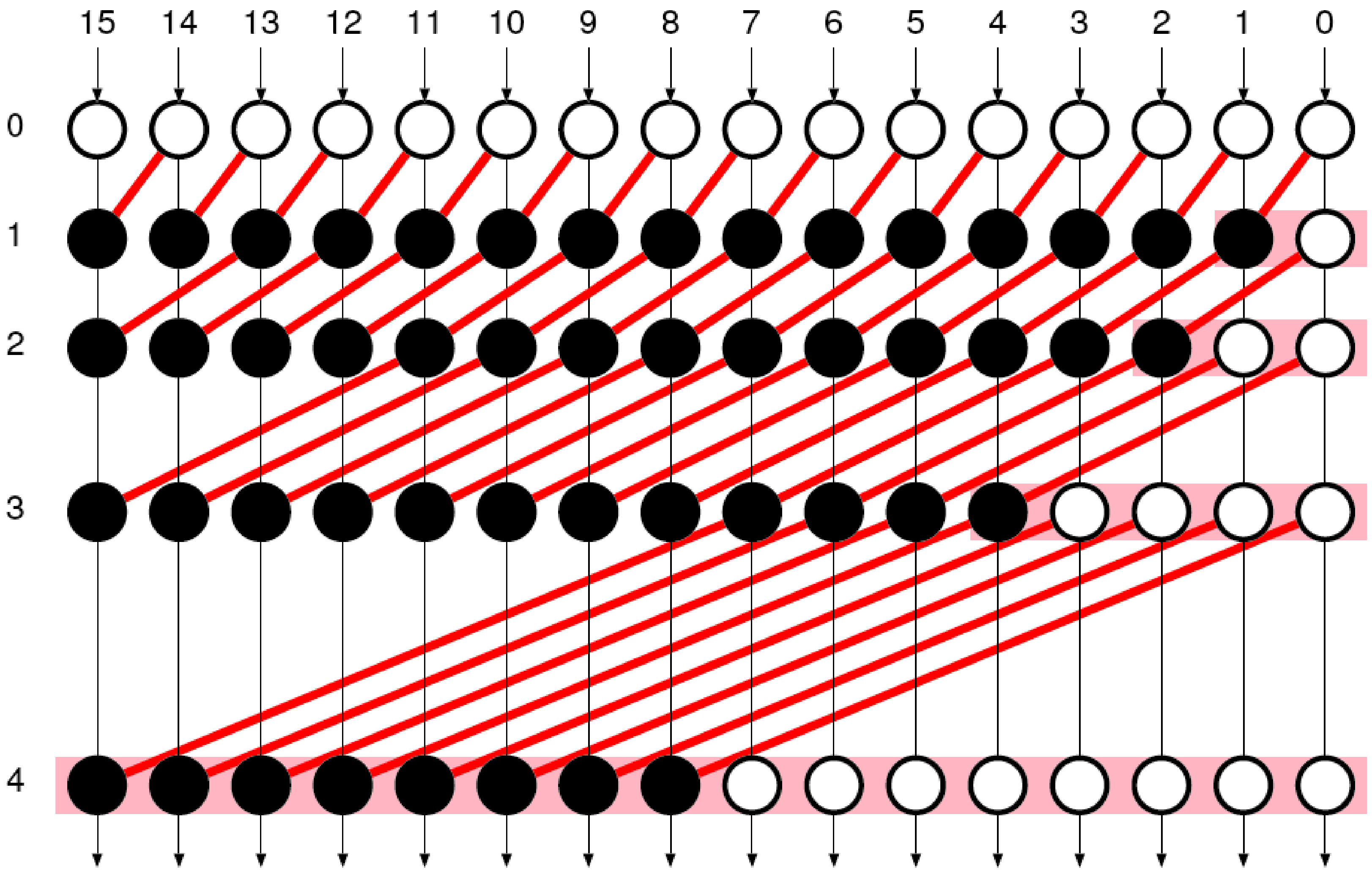
Parallel Prefix Adders: Sklansky (PPA-SK)



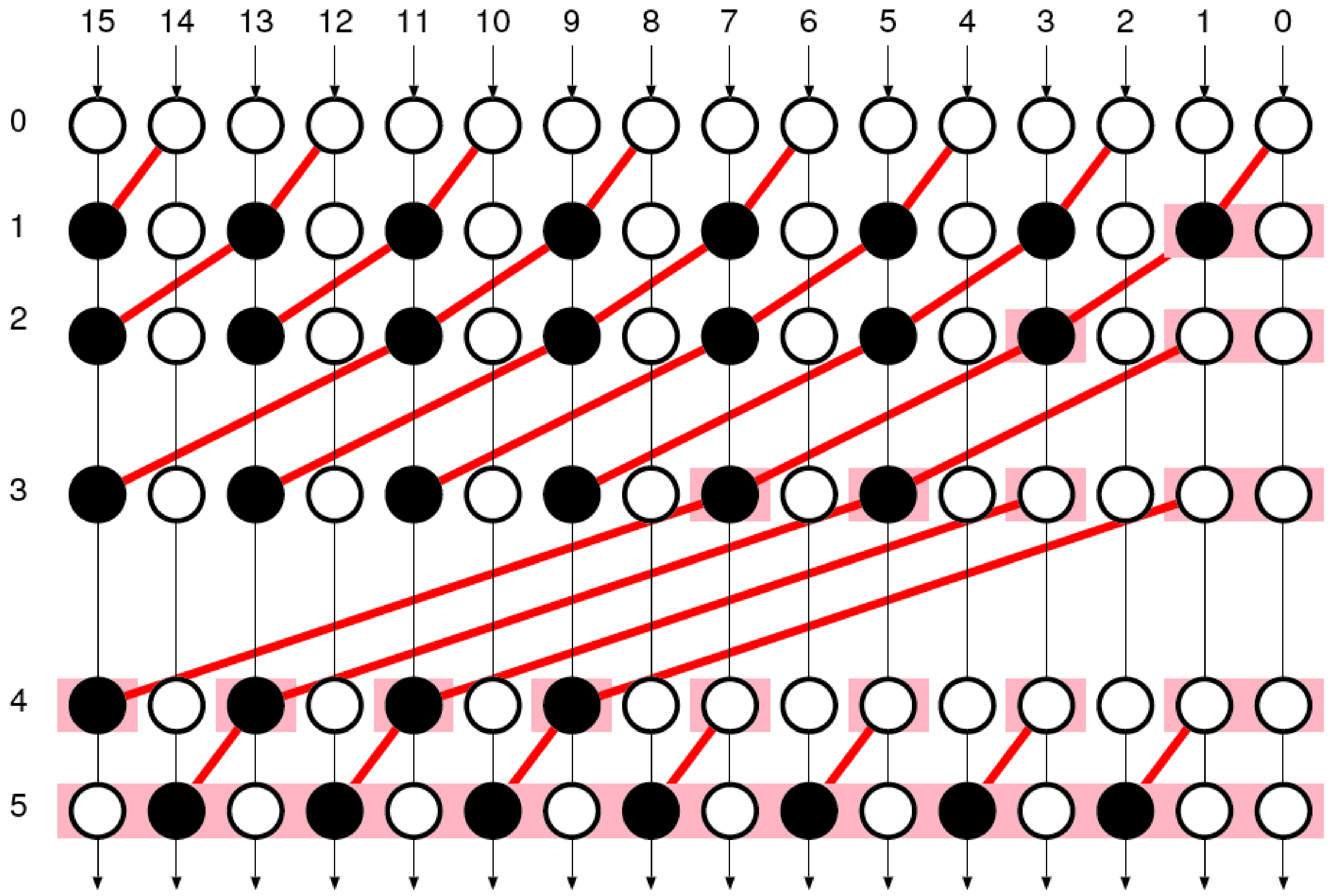
Parallel Prefix Adders: Brent Kung (PPA-BK)



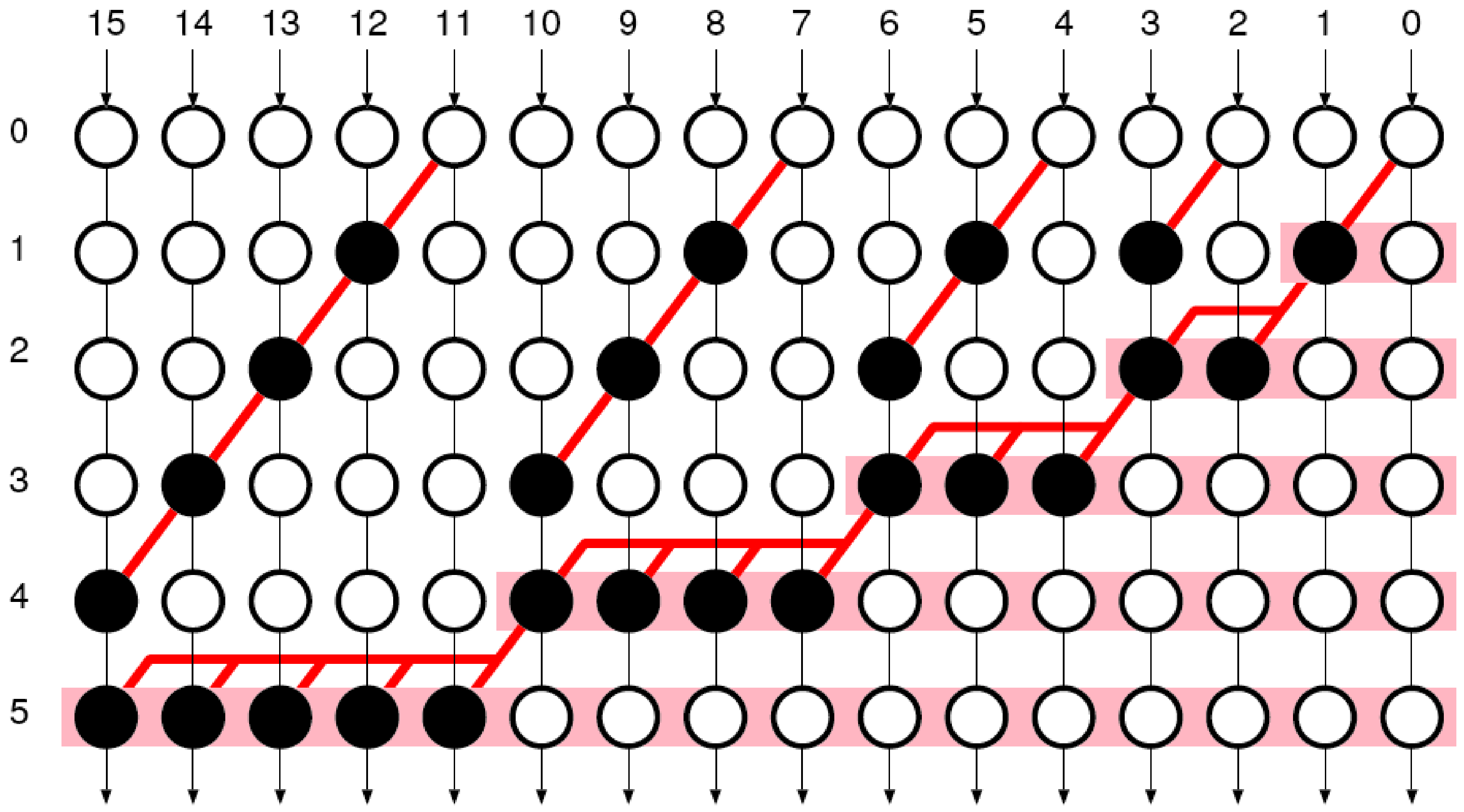
Parallel Prefix Adders: Kogge Stone (PPA-KS)

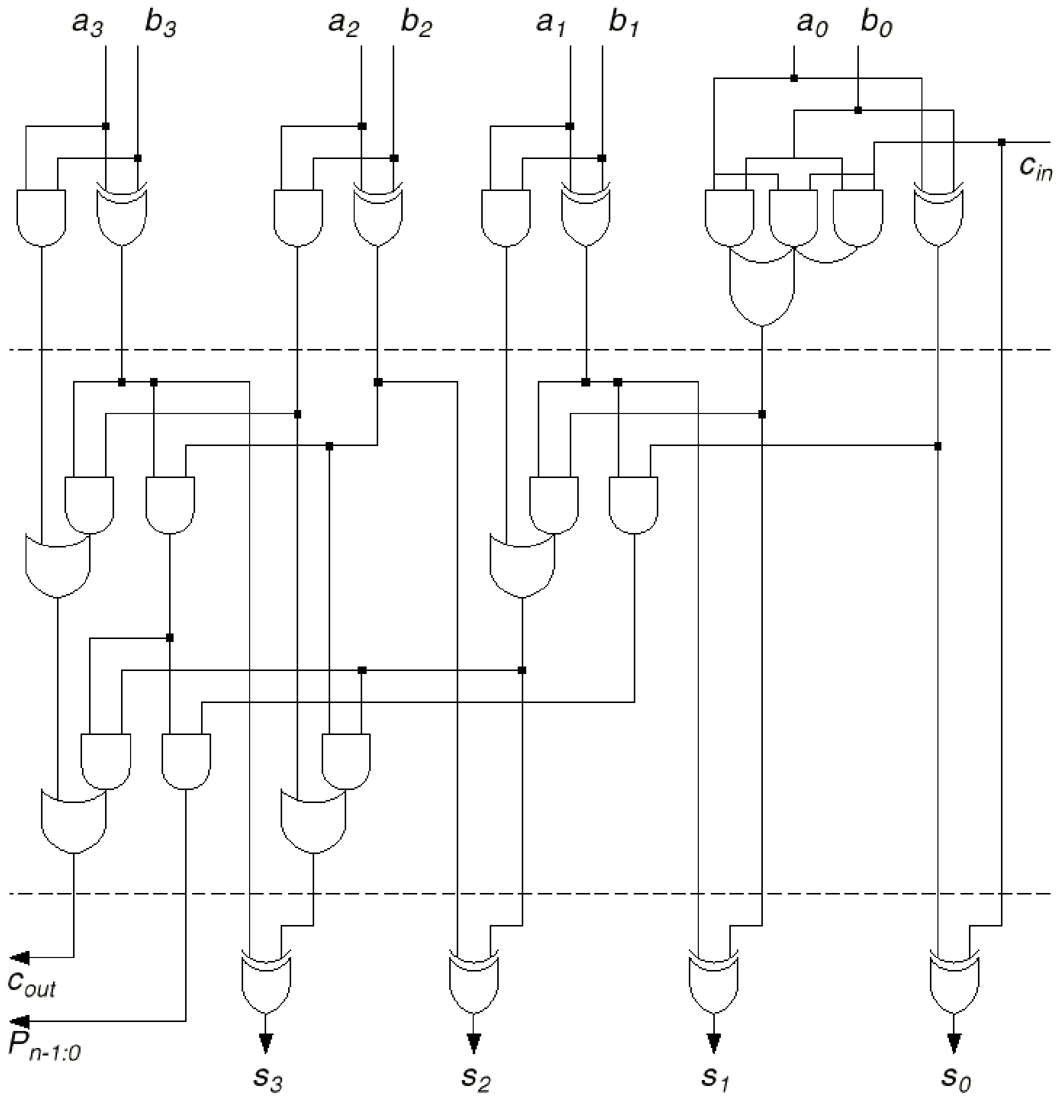


Parallel Prefix Adders: Han Carlson (PPA-HC)



Parallel Prefix Adders: Carry Increment Adder (PPA-CIA)




a_3
b_3
a_2
b_2
a_1
b_1
a_0
b_0
c_{in}
c_{out}
$P_{n-1:0}$
s_3
s_2
s_1
s_0

Performance Summary

Adder	Area	Speed	FO	$\approx A_\bullet$	$\approx T_\bullet$	$\approx FO_{max}$
RCA	small	slow	min	$n-1$	$n-1$	2
SK	large	fast	high	$\frac{n}{2}\log n$	$\log n$	$\frac{n}{2}$
BK	med	med	med	$2n-\log n$	$2\log n-2$	$\log n$
KS	huge	fast	min	$n\log n$	$\log n$	2
HC	large	fast	min	$\frac{n}{2}\log n$	$\log n+1$	2
CIA	med	med	med	$2n-2\sqrt{n}$	$2\sqrt{n}$	$2\sqrt{n}$

The formulas above are approximations

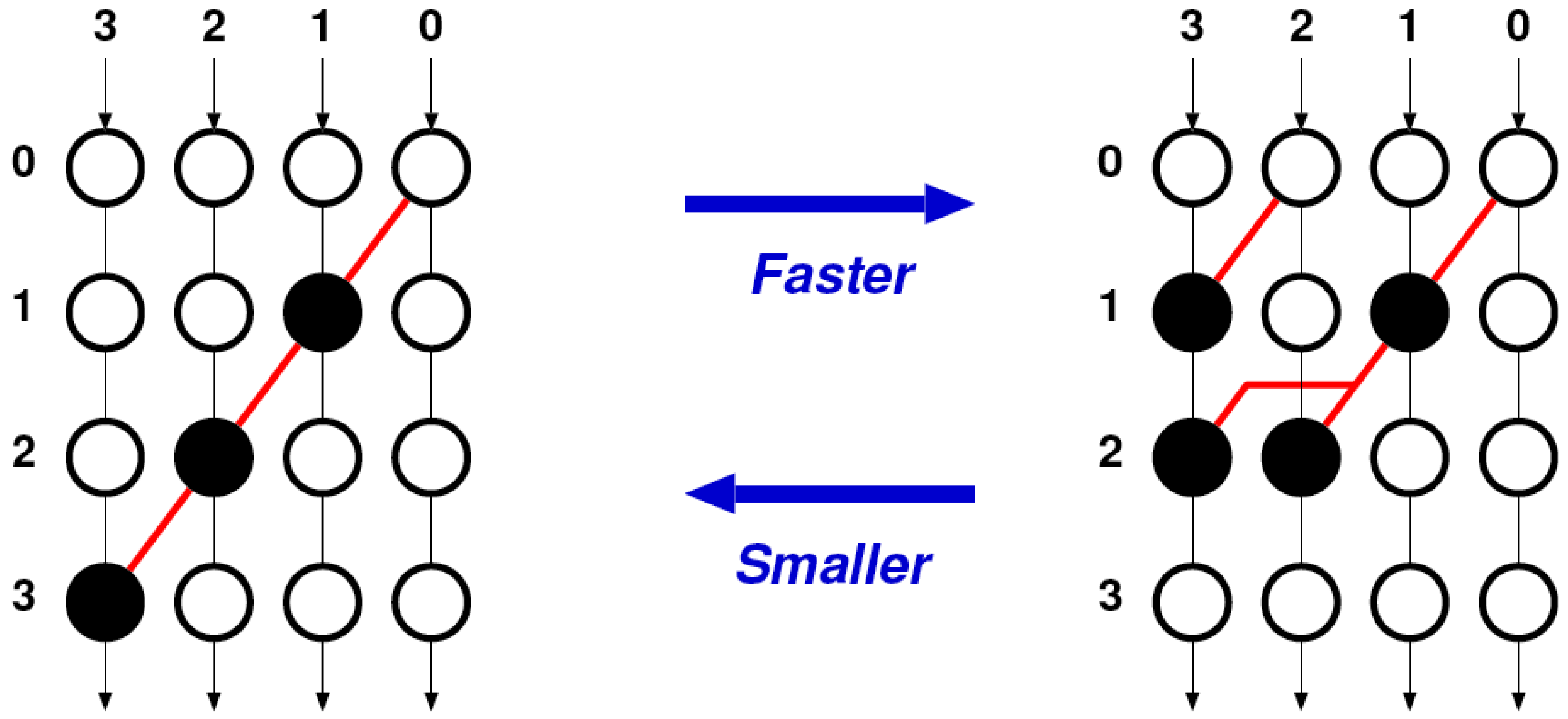
Some adders (KS, HC) have long connections, that are sometimes hard to route, their post layout results differ.

About Adders

Some observations

- **RCA is a very efficient adder**
  It is by far the smallest and simplest adder, especially for small bit sizes it is not too slow either.
- **BK offers good compromise**
  Traditional Carry Look-Ahead Adder (CLA) is BK with 4-bit groups.
- **SK suffers from high fan-out**
  In theory it is very fast, but the large fan out make it difficult to implement.
- **KS and HC are very fast, but suffer from routing**
  These are very large and fast adders, routing is a serious problem.

Simple Rule For Optimizing Parallel Prefix Adders


3
2
1
0
0
1
2
3
Faster
Smaller