Chapter 5
Arithmetic Circuits
SKEE2263 Digital Systems

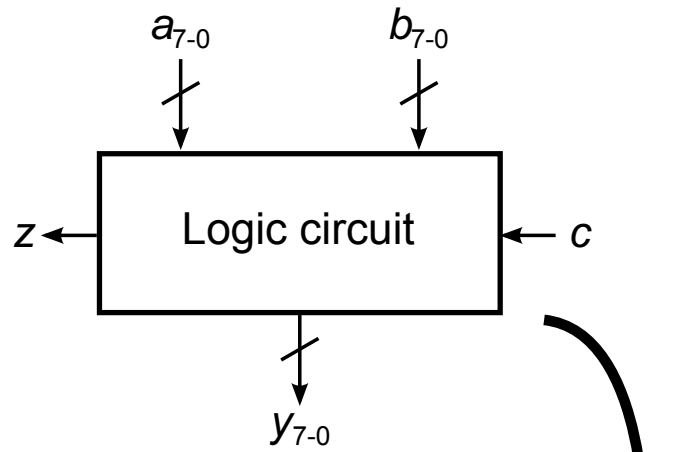
Mun'im Zabidi {munim@utm.my}
Ismahani Ismail {ismahani@fke.utm.my}
Izam Kamisian {e-izam@utm.my}

Faculty of Electrical Engineering, Universiti Teknologi Malaysia

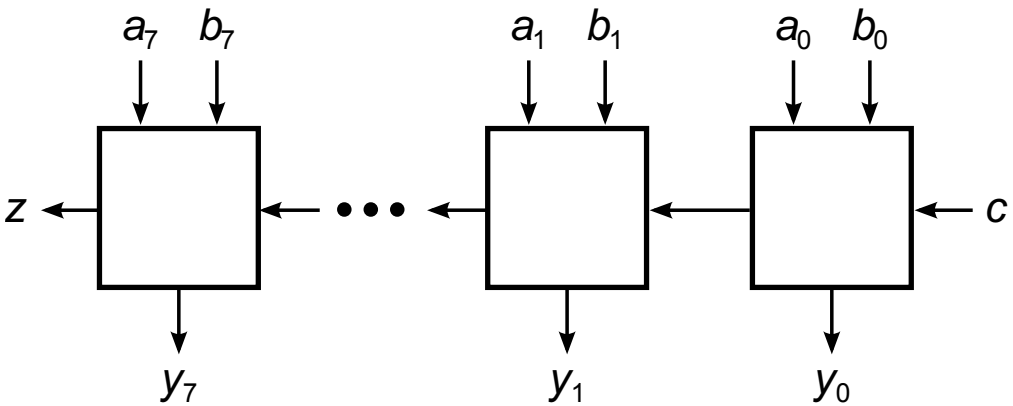
March 11, 2018

Table of Contents

Iterative Designs



A circuit can be decomposed into smaller 'bit slice' blocks to facilitate the design process.

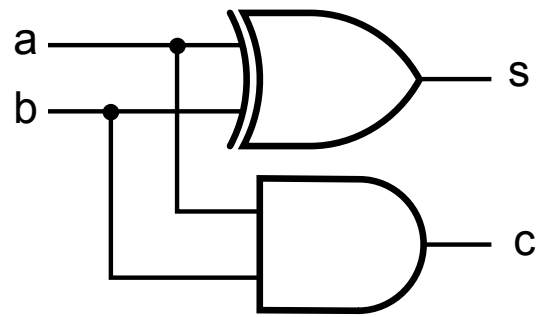


Half Adder

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$s = a \oplus b$$

$$c = ab$$



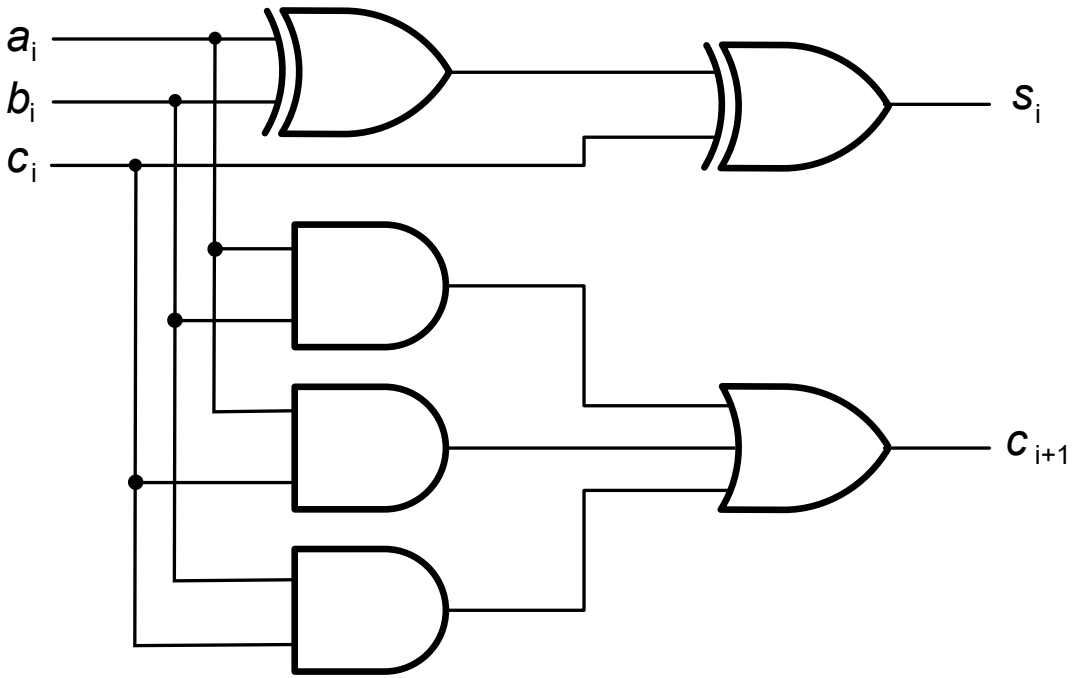
Full Adder

Boolean Expressions

a_i	b_i	c_i	c_{i+1}	s_i
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

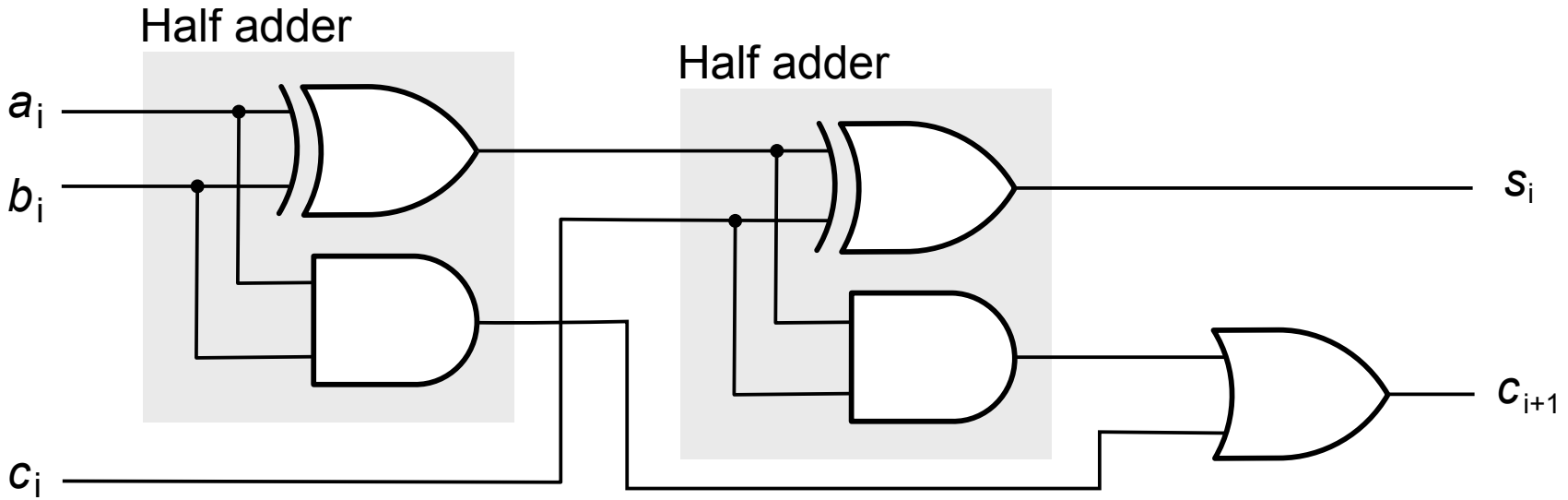
$$
\begin{aligned}
s_i &= a_i \oplus b_i \oplus c_i \\
c_{i+1} &= a_i b_i + a_i c_i + b_i c_i
\end{aligned}
$$

Full Adder
Two-Level Realization



Full Adder

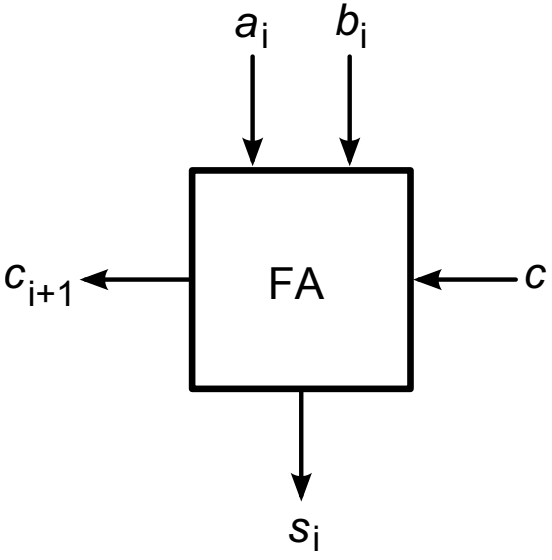
Multi-Level, Hierarchical Realization

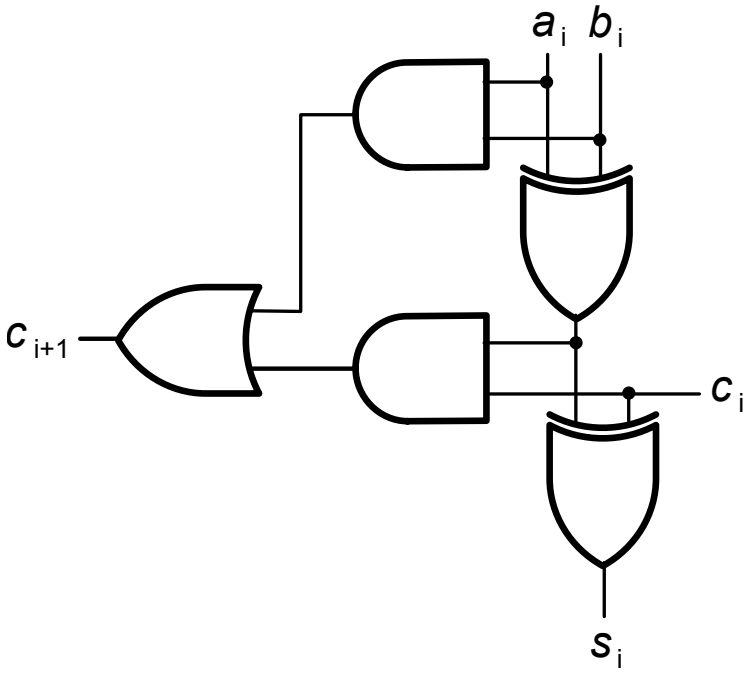


$$
\begin{aligned}
s_i &= (a_i \oplus b_i) \oplus c_i \\
&= a_i \oplus b_i \oplus c_i \\
c_{i+1} &= a_i b_i + (a_i \oplus b_i) c_i \\
&= a_i b_i + (a_i + b_i) c_i \\
&= a_i b_i + a_i c_i + b_i c_i
\end{aligned}
$$

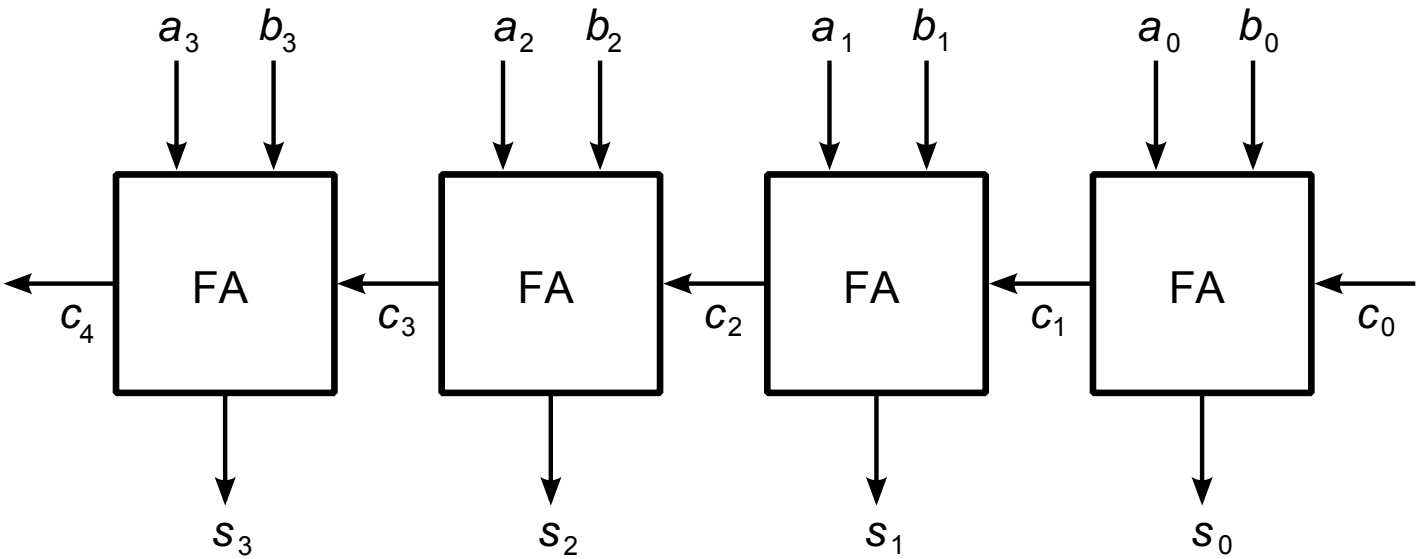
Full Adder

Schematic for Cascadable Module

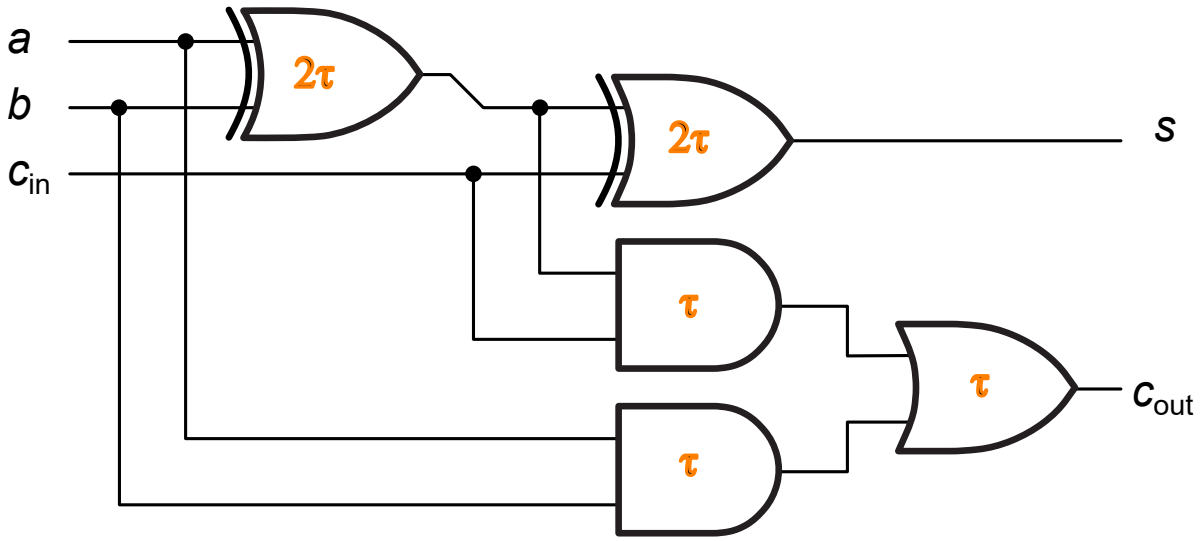




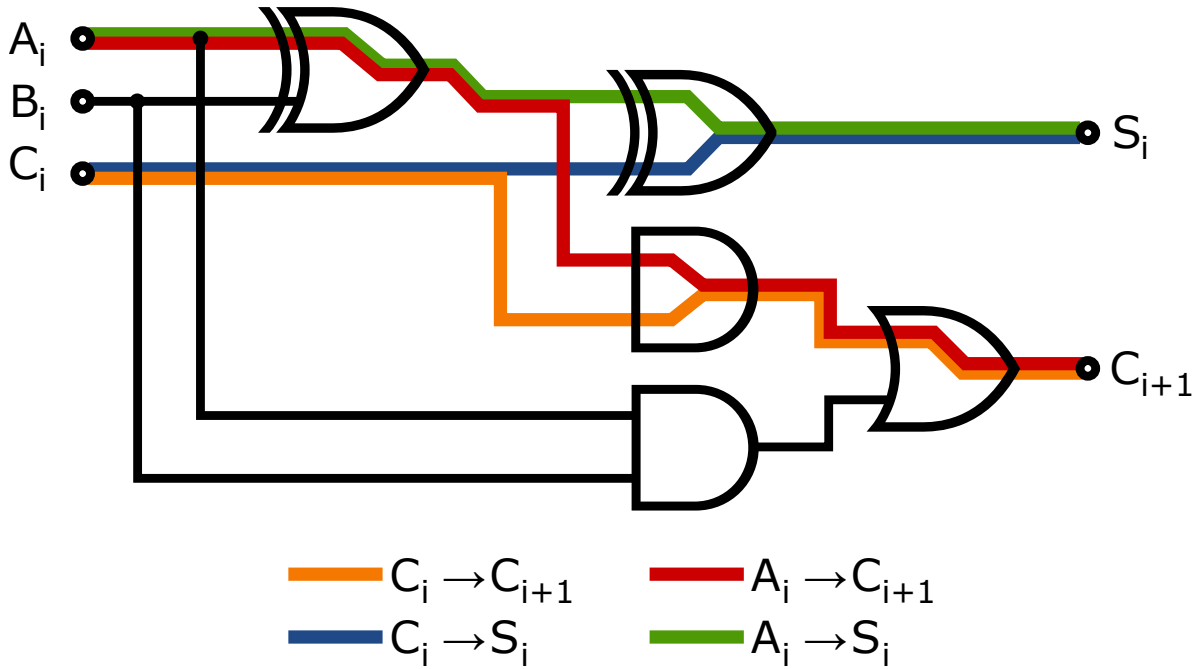
4-bit Ripple Adder



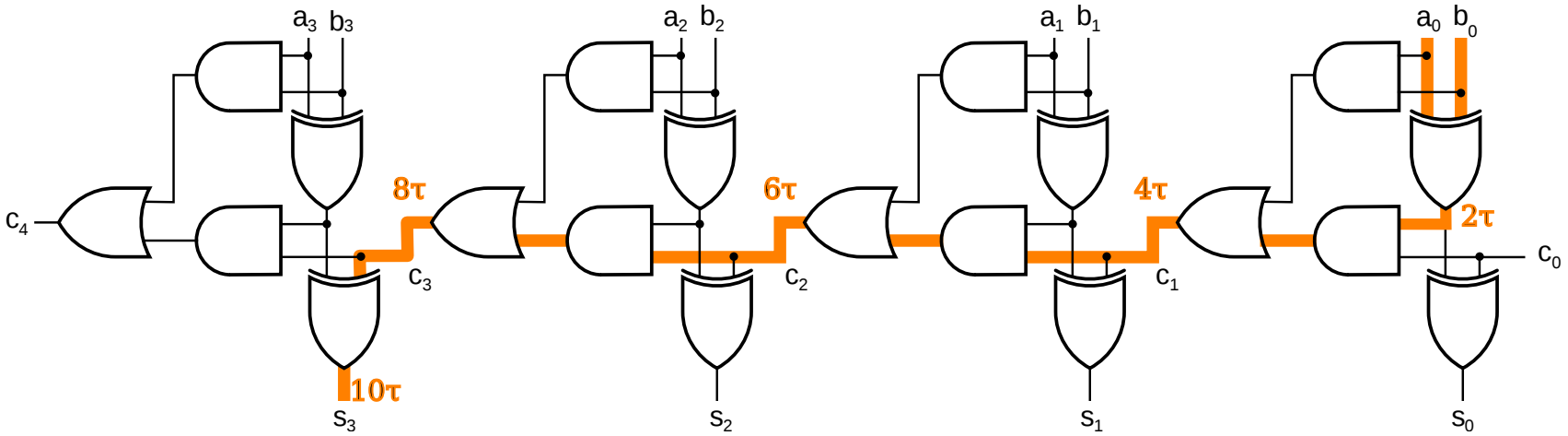
Delay Analysis of Cascadable Full Adder



Delay Analysis of Cascadable Full Adder



Delay Analysis of n-bit Ripple Carry Adder



Delay expression in terms of τ:

$$\begin{aligned} t_p &= \text{delay}_{\text{stage 0}} + \text{delay}_{\text{intermediate stages}} + \text{delay}_{\text{stage } n-1} \\ &= 4\tau + [2(n-2)]\tau + 2\tau \\ &= [2n+2]\tau \end{aligned}$$

In “Big(O)”-notation, delay is linear with number of bits

$$O(n)$$

Carry Lookahead Adder

Analysis of Carry Behavior

a_i	b_i	c_i	c_{i+1}		**Decision**	g_i	p_i
0	0	0	0	0	"Carry	0	0
0	0	1	0		kill"	0	0
0	1	0	0	c_i	"Carry	0	1
0	1	1	1		propagate"	0	1
1	0	0	0			0	1
1	0	1	1			0	1
1	1	0	1	1	"Carry	1	1
1	1	1	1		generate"	1	1

$c_{i+1} = 1$:

- when $a_i = b_i = 1$ (regardless of the carry in) or,
- when $a_i \neq b_i$ and the carry-in (from the previous stage) is 1

Carry Lookahead Adder

Derivation of p_i, g_i

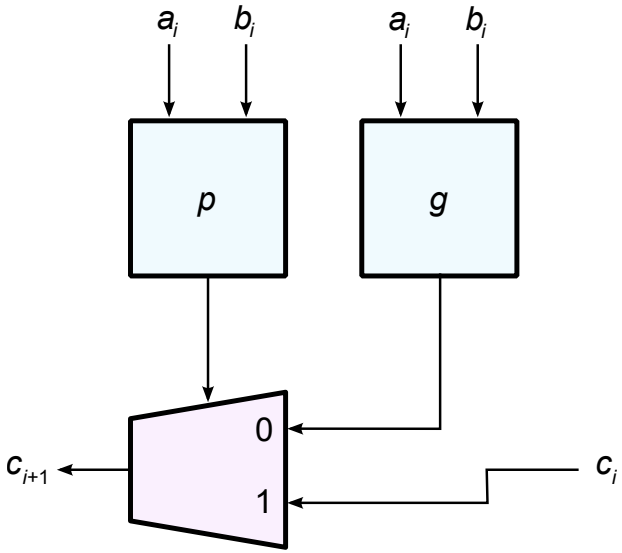
- **Carry generate** (g_i):
  a signal that is true when $a_i b_i = 1$.

$$g_i = a_i b_i$$

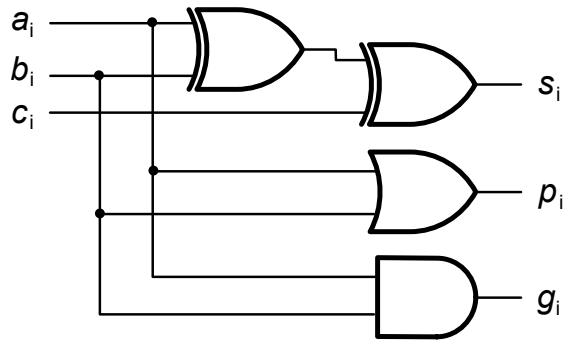
- **Carry propagate** (p_i):
  propagates the carry-in (c_i) to a stage to the next stage (i.e., $c_{i+1} \leftarrow c_i$) if the "half sum" $a_i \oplus b_i = 1$.

$$p_i = a_i + b_i$$

Carry Lookahead Adder



p_i, g_i w.r.t. carries



Partial Full Adder (PFA)

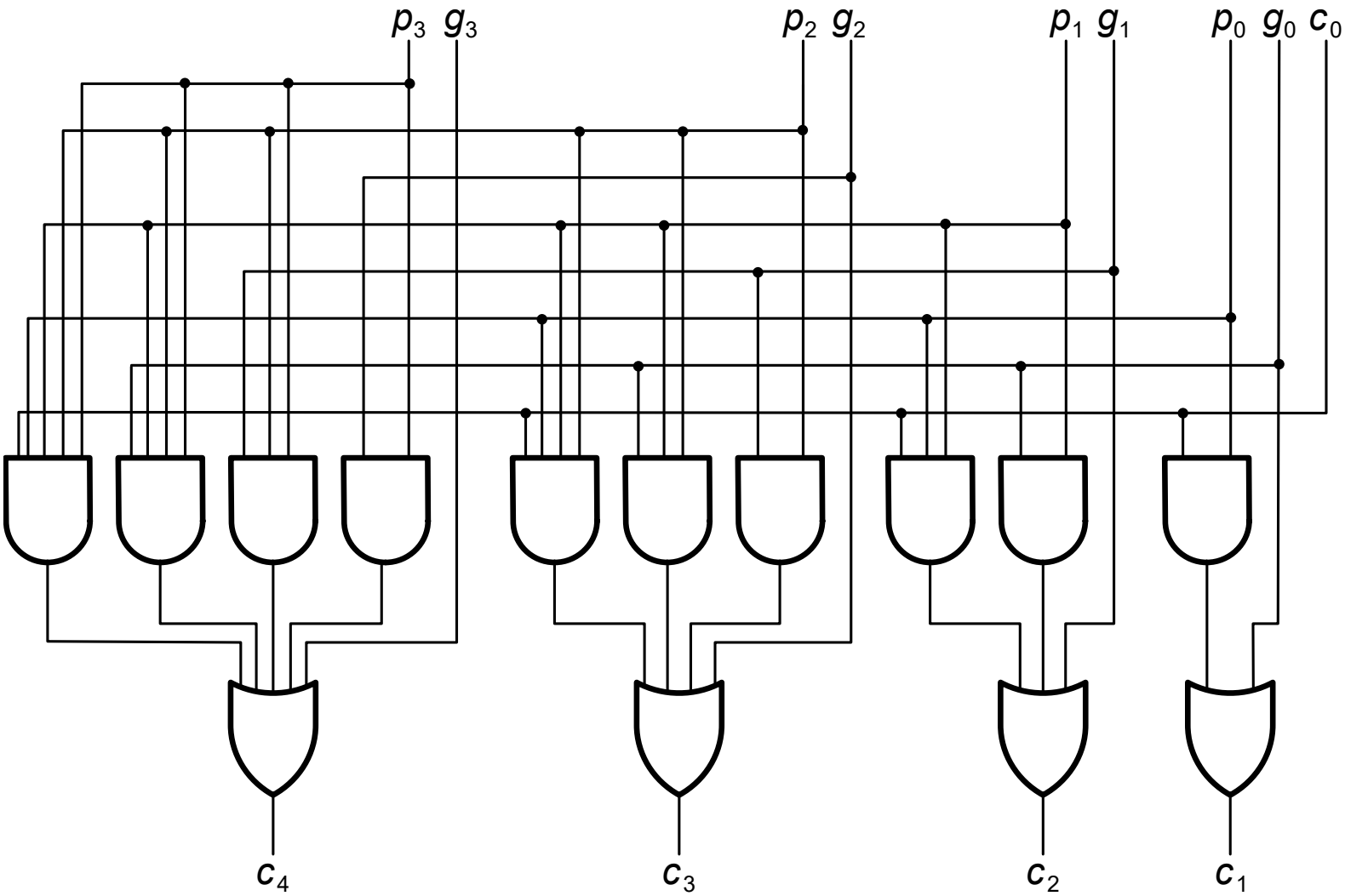
Carry Lookahead Adder

Carry-out Equations

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + p_1 (g_0 + p_0 c_0) \\
&= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0) \\
&= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 c_3 \\
&= g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\
&= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}
$$

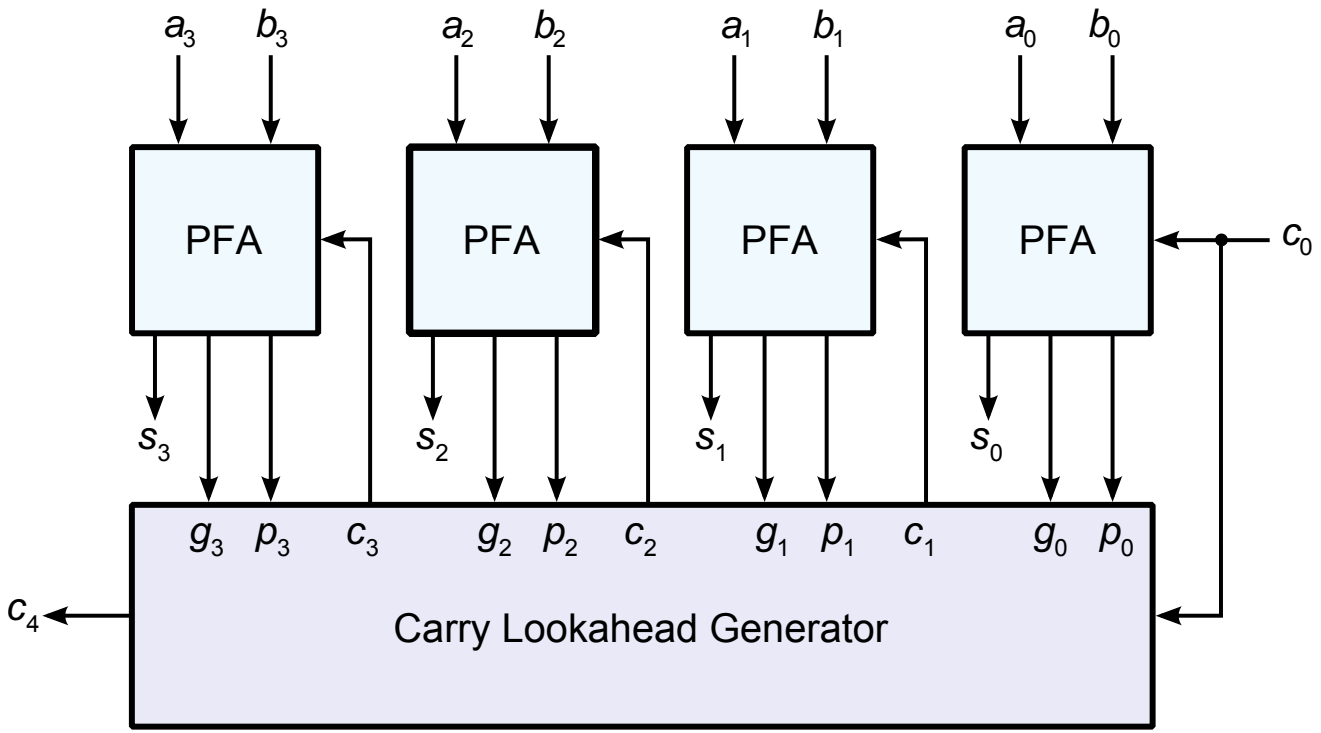
Carry Lookahead Adder

Carry Lookahead Generator (CLG)



Carry Lookahead Adder

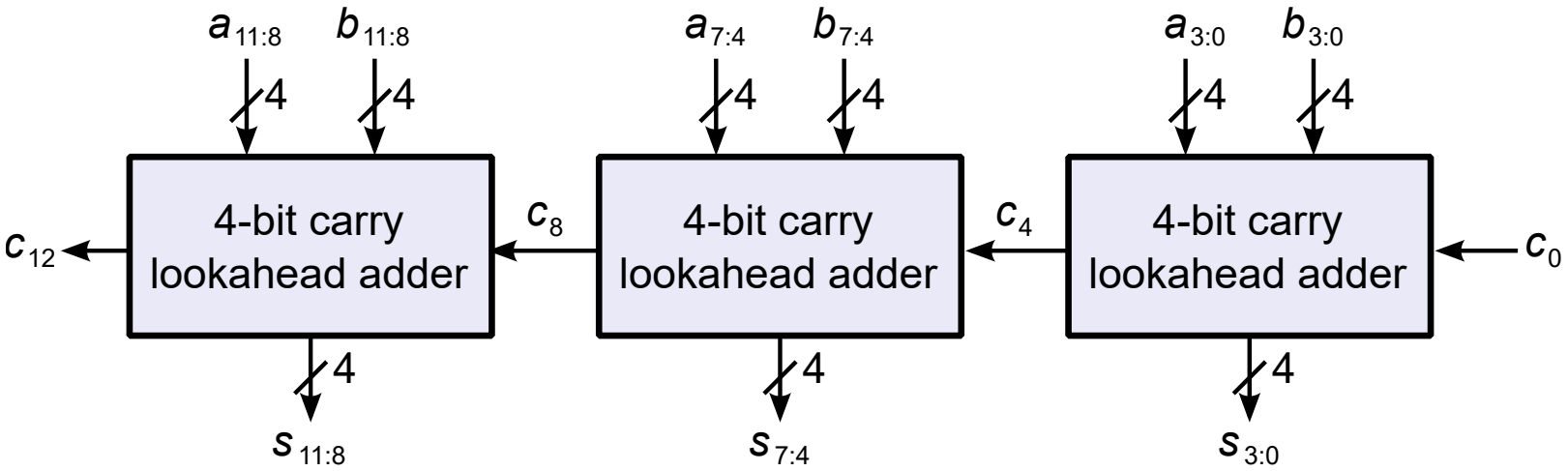
4-bit Carry Lookahead Adder



$$
\begin{aligned}
a_i, b_i \rightarrow p_i, g_i &= \tau \\
c_0 \rightarrow c_4 &= 2\tau \\
c_3 \rightarrow s_3 &= 2\tau \\
\hline
\text{Total} &= 5\tau
\end{aligned}
$$

Carry Lookahead Adder

12-bit Cascaded Carry Lookahead Adders



Generate p_i, g_i once for whole cascade $= \tau$

$c_i \rightarrow c_{i+4}$ for each block $= m \times 2\tau$

$c_8 \rightarrow s_{11}$ $= 2\tau$

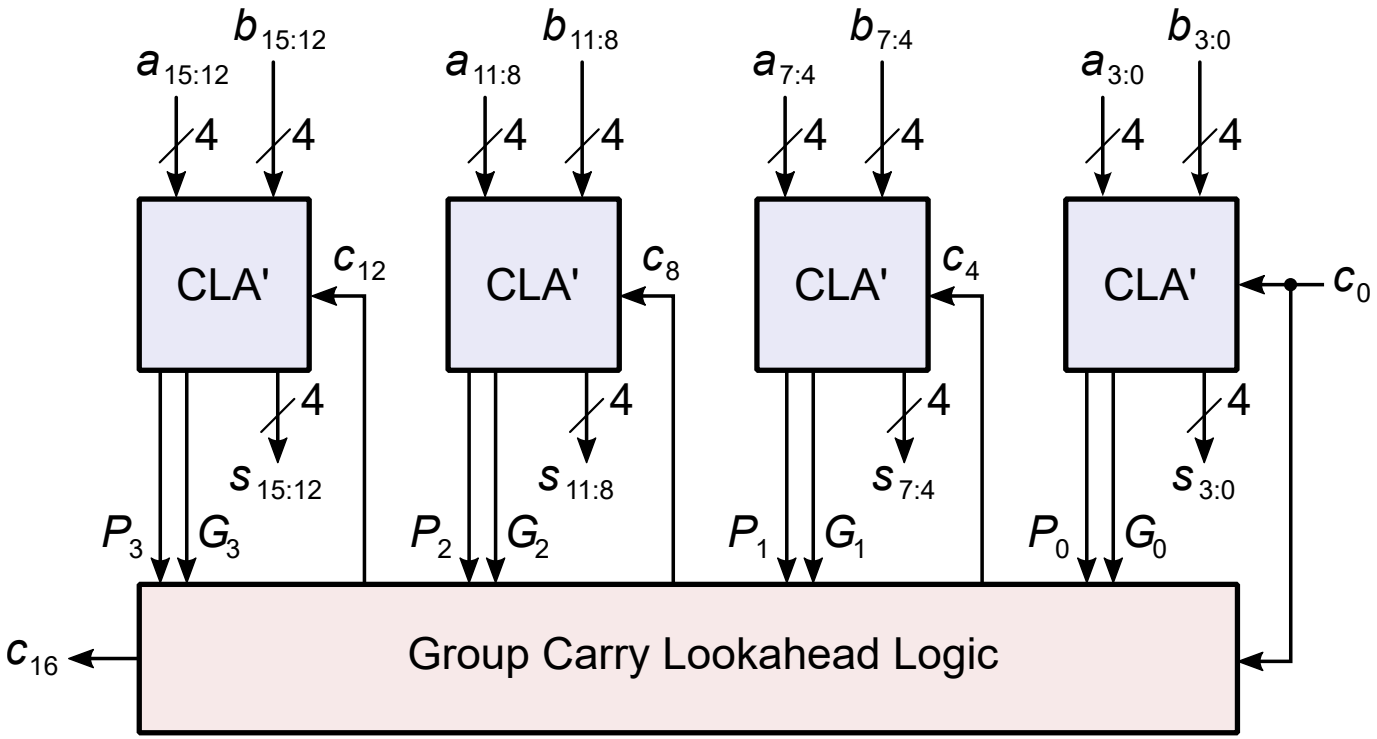
Total $= [1 + 2m + 2]\tau = [2m + 3]\tau$

For 12-bit CLA, $m = 3$:

$$[2m + 3]\tau = [2 \times 3 + 3]\tau = 9\tau$$

Carry Lookahead Adder

16-bit Hierarchical Carry Lookahead Adder



- Total delay = 8 τ.

Carry Lookahead Adder

16-bit Hierarchical Carry Lookahead Adder

- Group generate

$$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

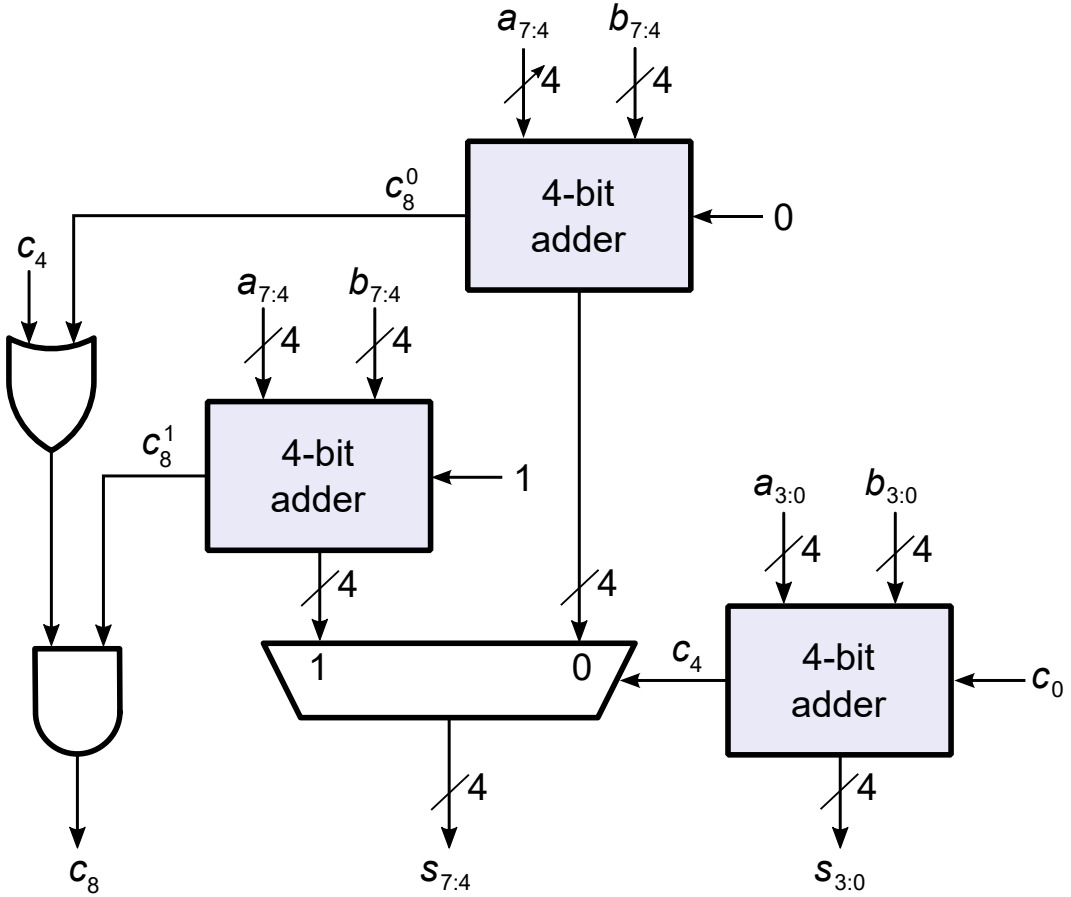
- Group propagate

$$P_0 = p_3 p_2 p_1 p_0$$

- Interblock carries:

$$\begin{aligned}
c_4 &= G_0 + P_0 c_0 \\
c_8 &= G_1 + P_1 c_4 \\
&= G_1 + P_1 G_1 + P_1 P_0 c_0 \\
c_{12} &= G_2 + P_2 c_8 \\
&= G_2 + P_2 G_1 + P_2 P_1 G_1 + P_2 P_1 P_0 c_0 \\
c_{16} &= G_3 + P_3 c_{12} \\
&= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0
\end{aligned}$$

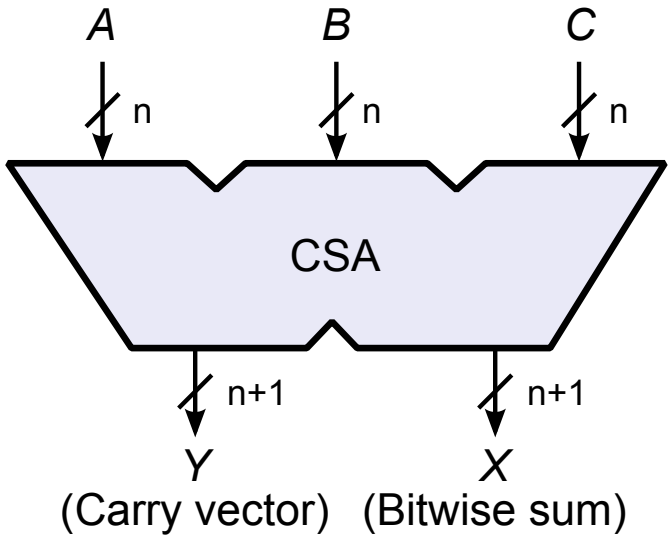
Carry Select Adder



- Total delay = delay of 1 4-bit adder + 2 τ.

Carry Save Adder

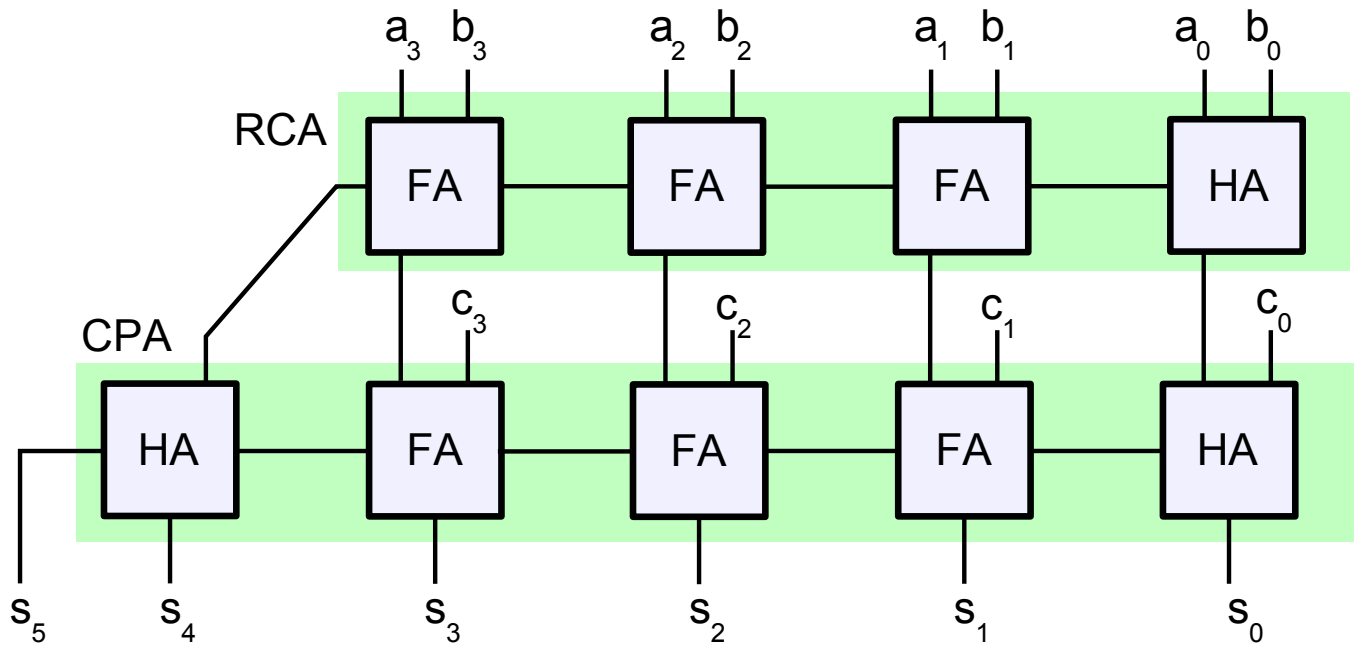
3:2 Compression Costing Just 2τ



$$
\begin{array}{cccccccc}
 & A & 0 & 0 & 1 & 0 & 1 & 1 \\
 & B & 0 & 1 & 0 & 1 & 0 & 1 \\
+ & C & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
 & S & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
 & C & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\hline
 & \sum & 1 & 0 & 1 & 1 & 1 & 0 & 1
\end{array}
$$

Carry Save Adder

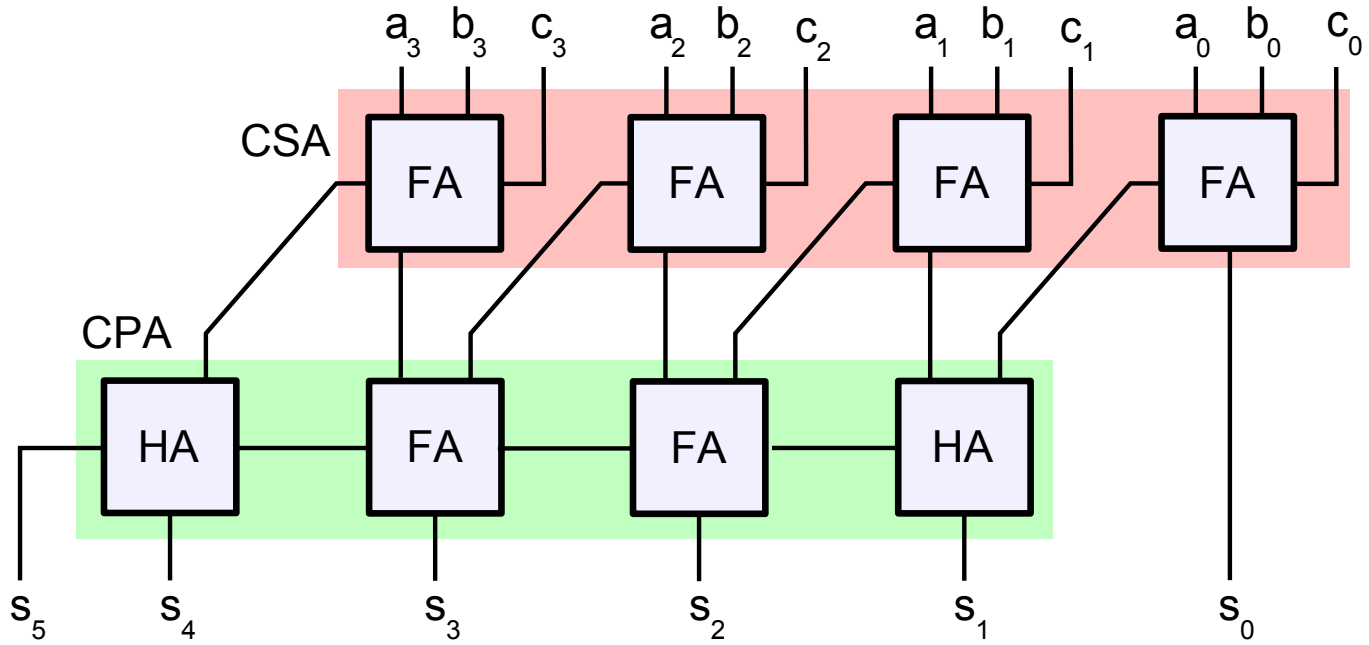
Adding 3 numbers using Ripple Carry Adders only



Delay 14τ

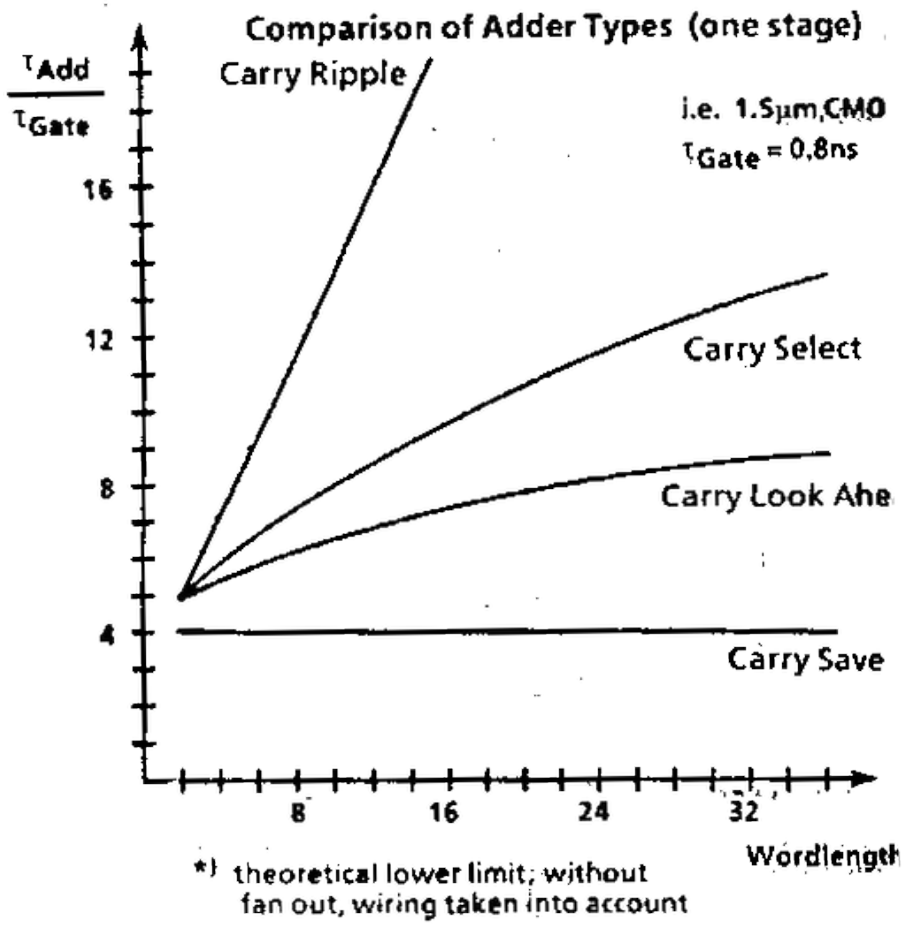
Carry Save Adder

Adding 3 numbers using Carry Save Adders



Delay 12τ

Adder Delay Summary

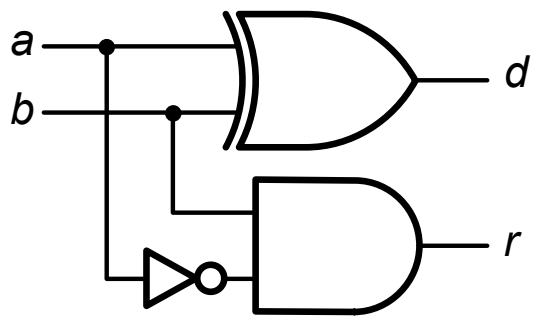


Source: `http://www.slideshare.net/ankitgoel/cmos-arithmetic-circuits`

Half Subtractor

Input		Output	
a	b	r (borrow)	d (difference)
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$$d = a'b + ab'$$
$$= a \oplus b$$
$$r = a'b$$

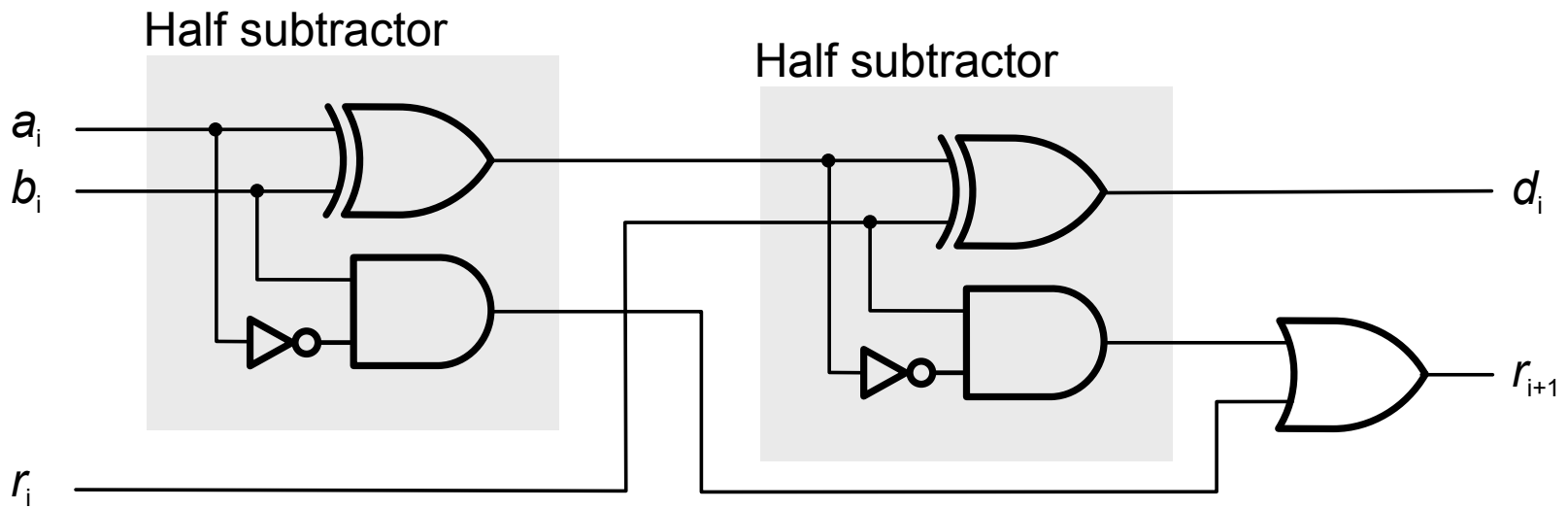


Full Subtractor

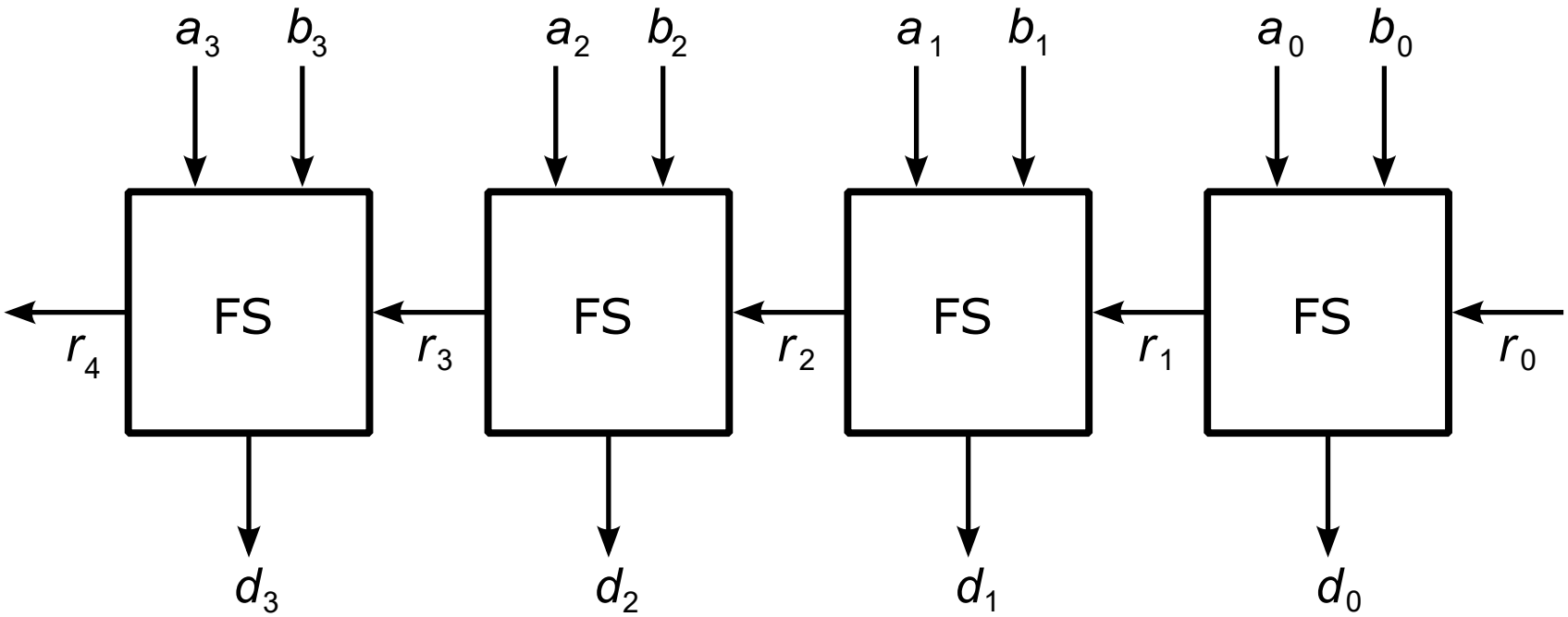
Input			Output	
a	b	r_i	r_{i+1}	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$
\begin{aligned}
d &= a'b'r_i + a'br_i' + ab'r_i' + abr_i \\
&= a \oplus b \oplus r_i \\
r_{i+1} &= a'b'r_i + a'br_i' + a'br_i + abr_i \\
&= a'r_i + a'b + br_i
\end{aligned}
$$

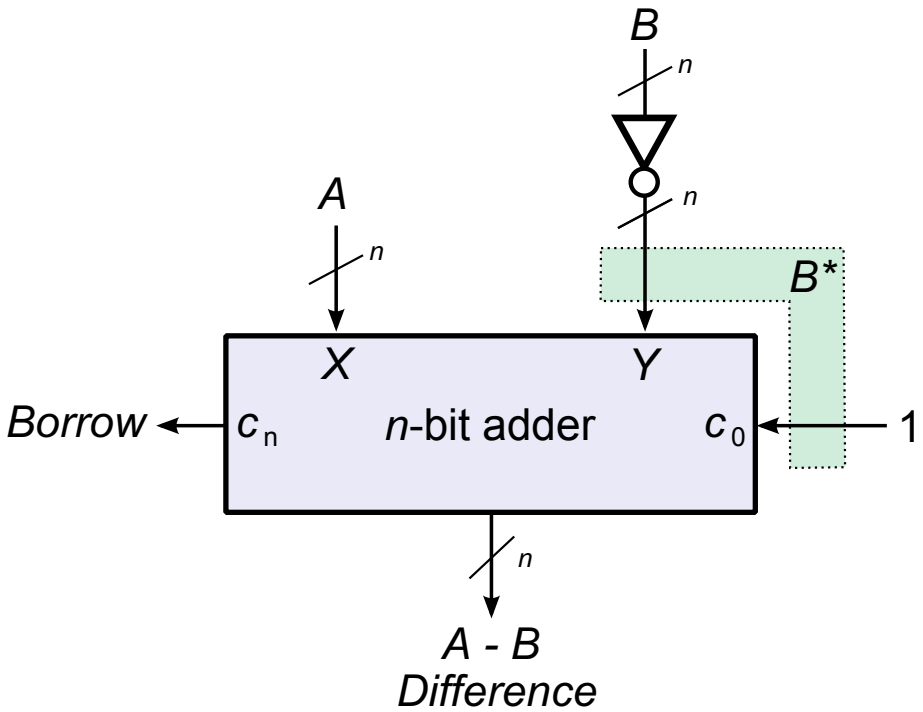
Full Subtractor



Ripple Subtractor



Subtraction by Adding

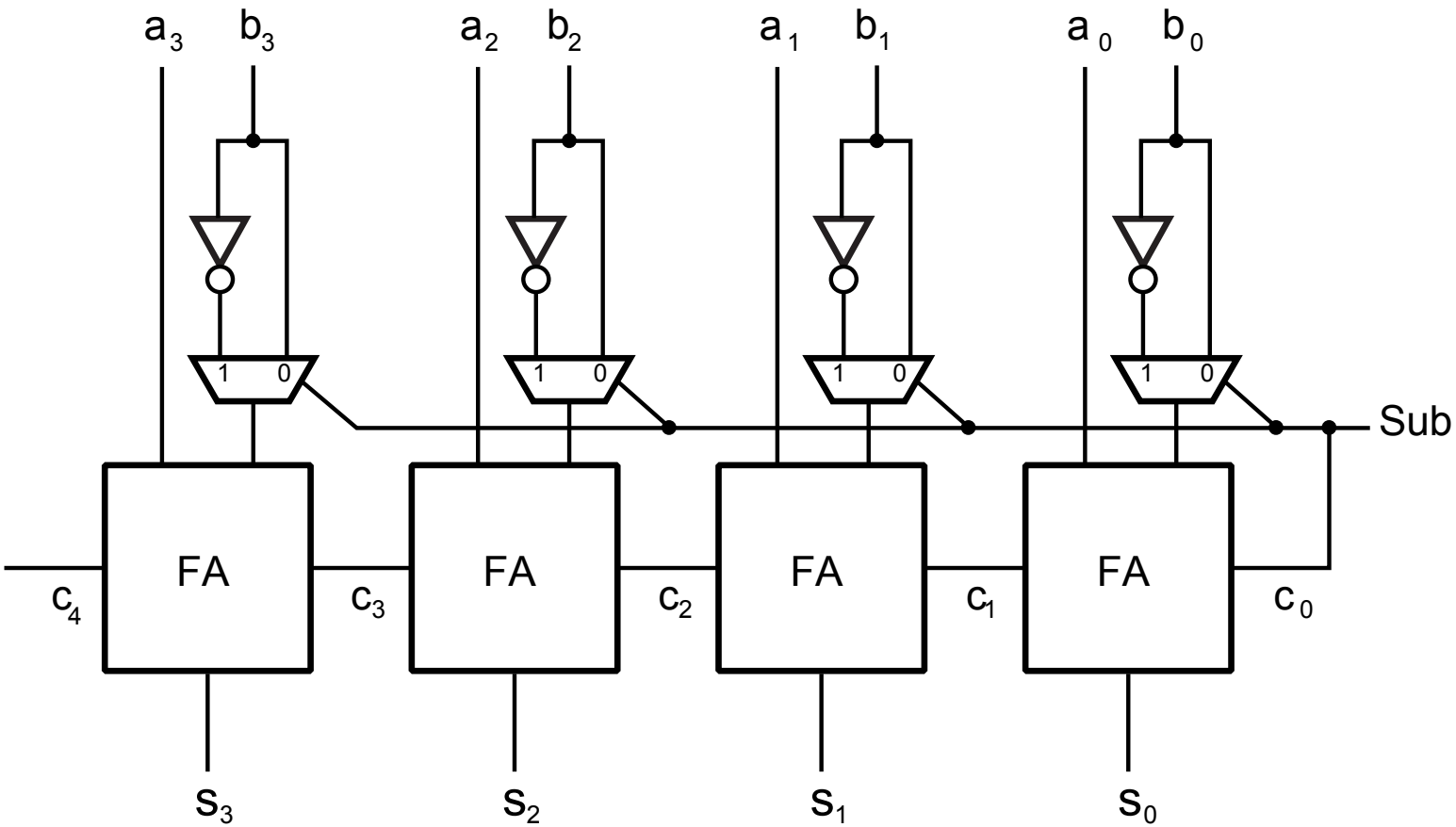


$$
\begin{aligned}
S &= A + \overline{B} + 1 \\
&= A + B^* \\
&= A + (2^n - B) \\
&= A - B + 2^n \\
&= A - B
\end{aligned}
$$

where:

- $\overline{B}$: Ones' complement of B
- B^*: Two's complement of B

Adder/Subtractor



Flags

Code	Name	Significance
Z	Zero	Result is zero
N	Negative	Result is < 0
C	Carry	The most significant bit produced a carry
V	Overflow	Result has too many bits to be represented correctly

Z Flag

True when all the bits in an n-bit result are 0s.

$$\begin{aligned} \mathrm{Z} &= s'_{n-1} s'_{n-2} \cdots s'_1 s'_0 \\ &= (s_{n-1} + s_{n-2} + \cdots + s_1 + s_0)' \end{aligned}$$

N Flag

The negative flag (N) is simply the sign bit. When it is high, the result was less than zero.

$$\mathrm{N} = s_{n-1}$$

C Flag

The carry out from the adder

$$\mathrm{C} = c_n$$

V Flag

True when a calculation produces a result that is greater than a register can store

a_{n-1}	b_{n-1}	s_{n-1}	c_{n-1}	c_{n-2}	V
0	0	0	0	0	0
0	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	0

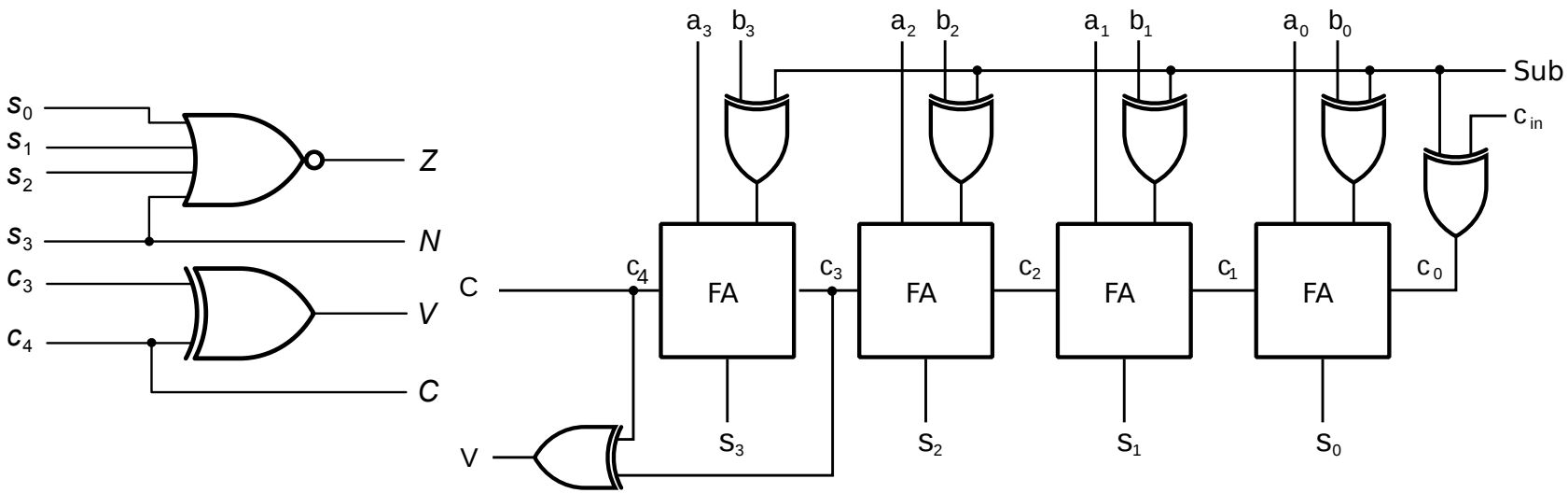
Method 1:

$$V = a_{n-1} b_{n-1} s'_{n-1} + a'_{n-1} b'_{n-1} s_{n-1}$$

Method 2:

$$V = c_n \oplus c_{n-1}$$

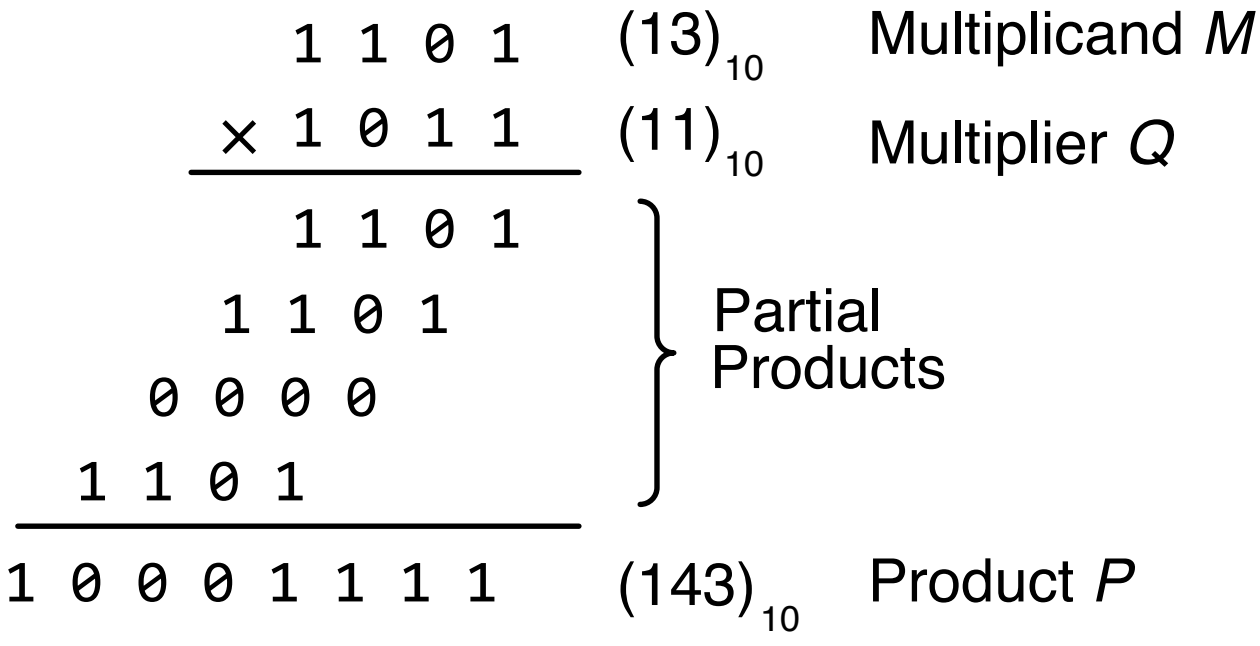
Flags Circuit



Deriving Other Relations

Math Symbol	**Description**	**Flags (Unsigned)**	**Flags (Signed)**
$=$	Equal	Z	
$\neq$	Not equal	Z'	
$<$	Less than	C'	$N \oplus V$
$\leq$	Less than or equal	C' + Z	$Z + (N \oplus V)$
$\geq$	Greater than or equal	C	$N \odot V$
$>$	Greater than	$C \cdot Z'$	$Z' \cdot (N \odot V)$

Binary Multiplication

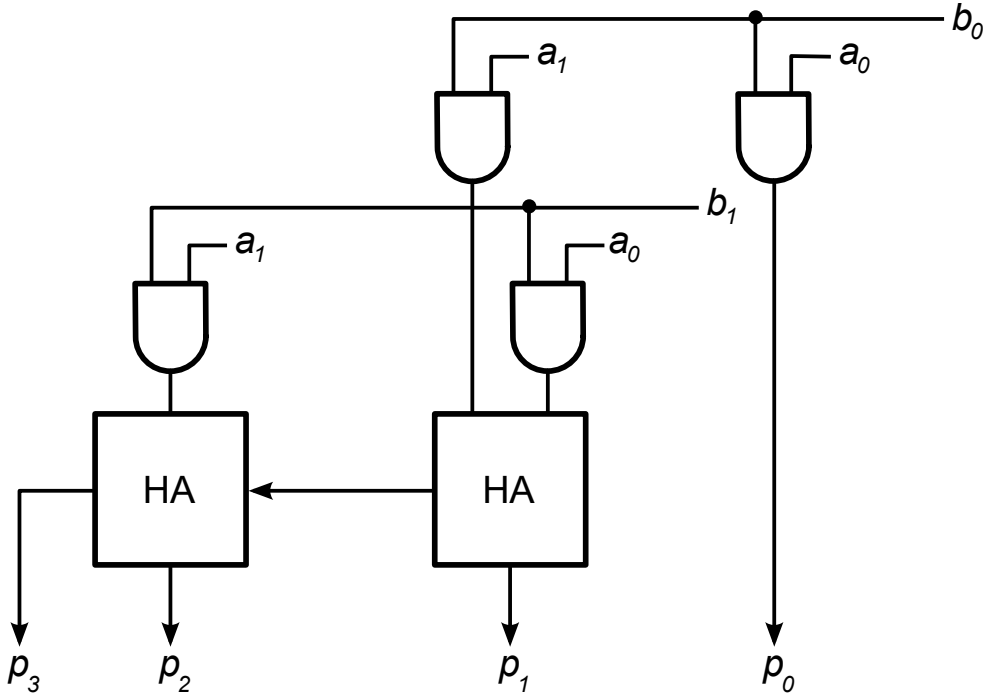


Multiplication is all about:

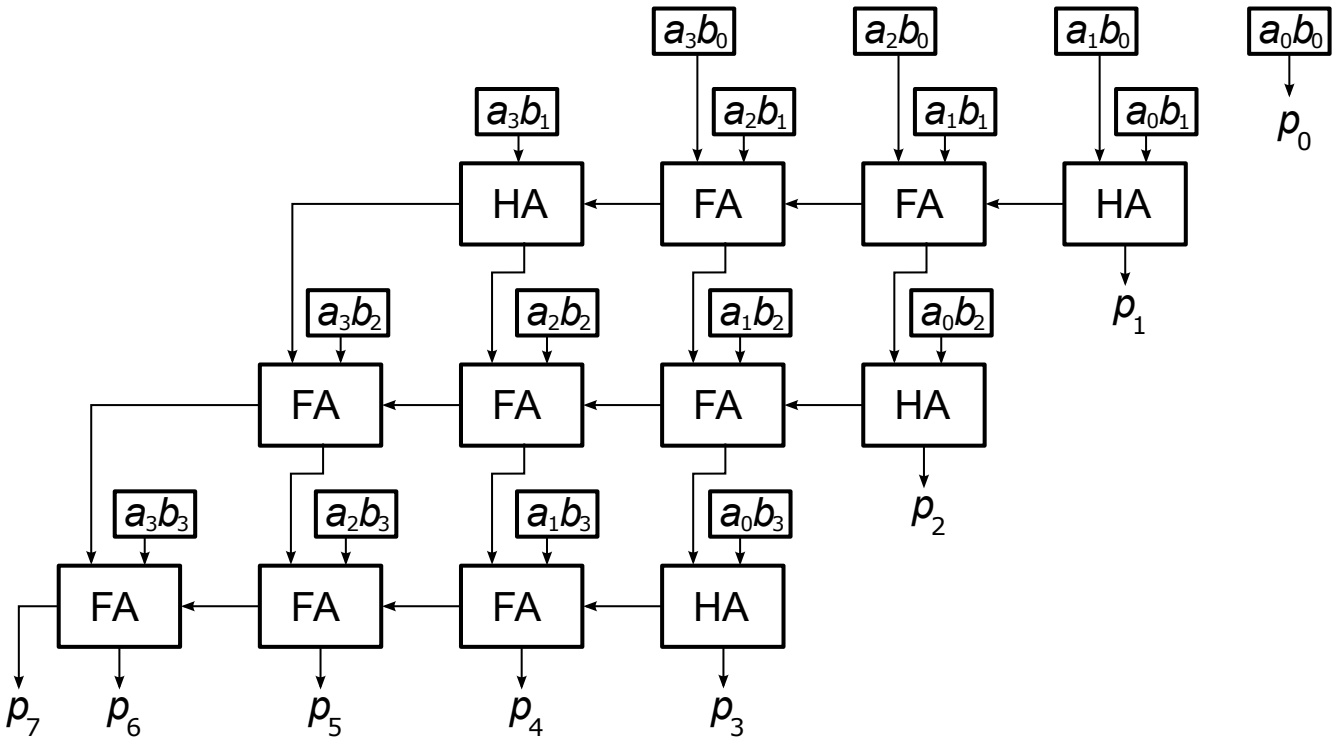
1. Generating the partial products
2. Adding the partial products

2×2 Multiplication

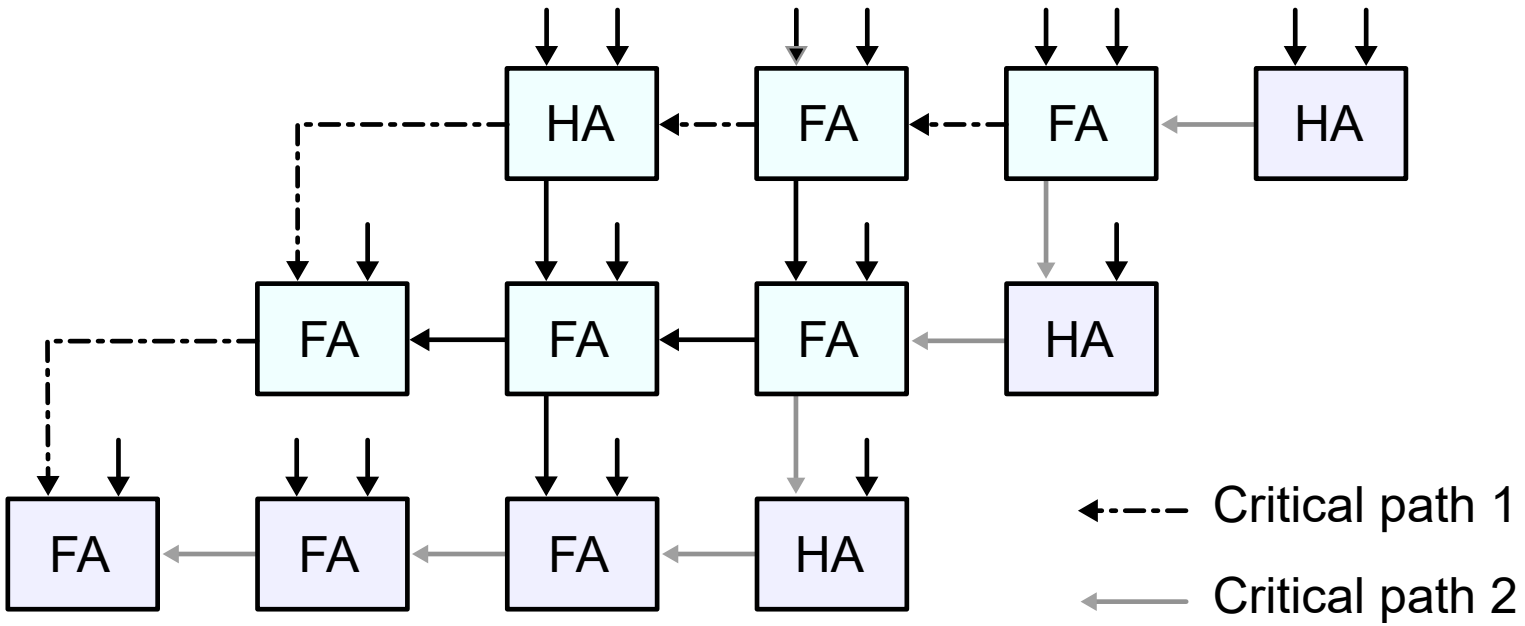
$$
\begin{array}{cccc}
 & & a_1 & a_0 \\
 & \times & b_1 & b_0 \\
\hline
 & & a_1 b_0 & a_0 b_0 \\
 & a_1 b_1 & a_0 b_1 & \\
\hline
p_3 & p_2 & p_1 & p_0
\end{array}
$$



Ripple Carry Array Multiplier

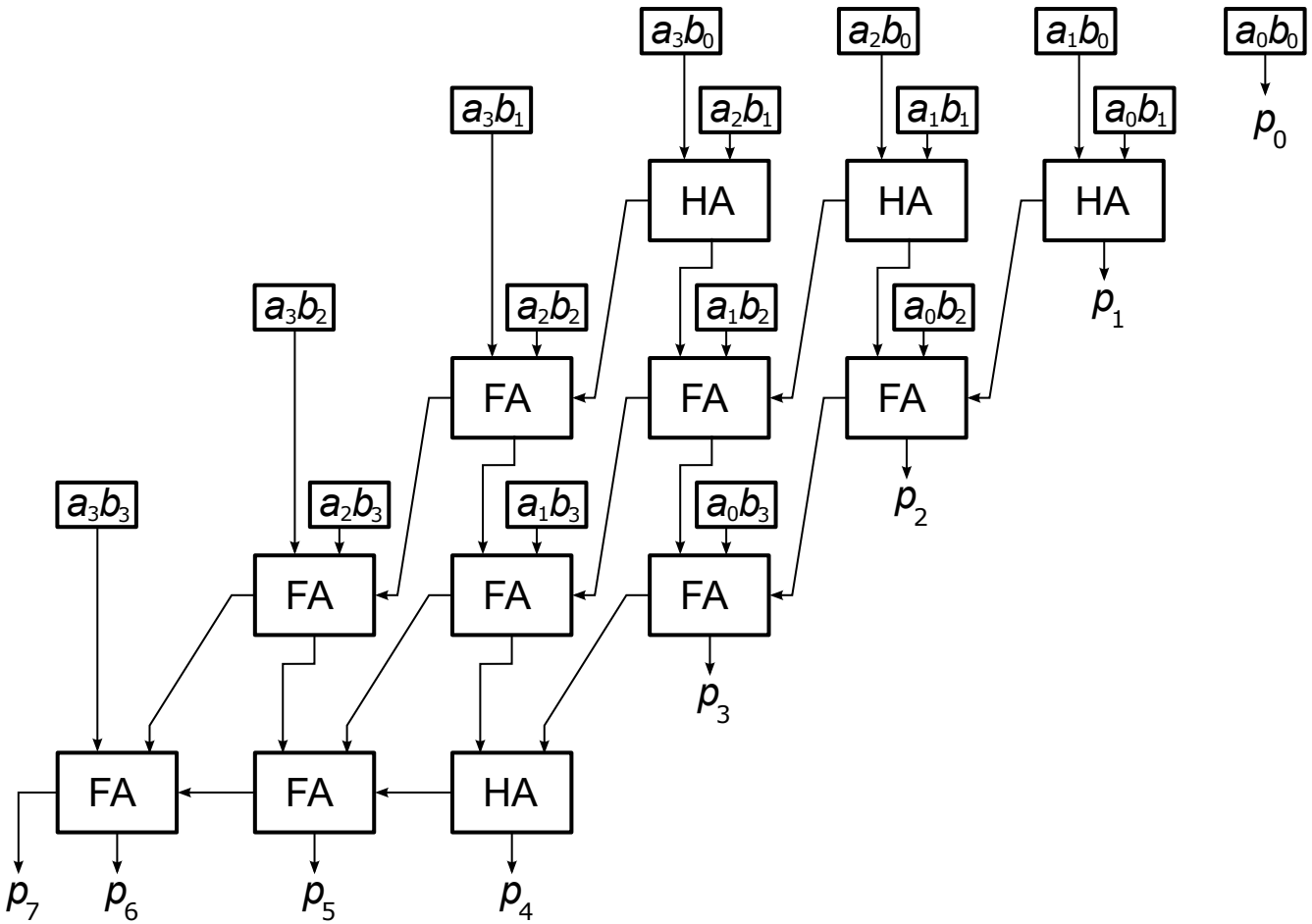


$M \times N$ Array Multiplier Critical Paths

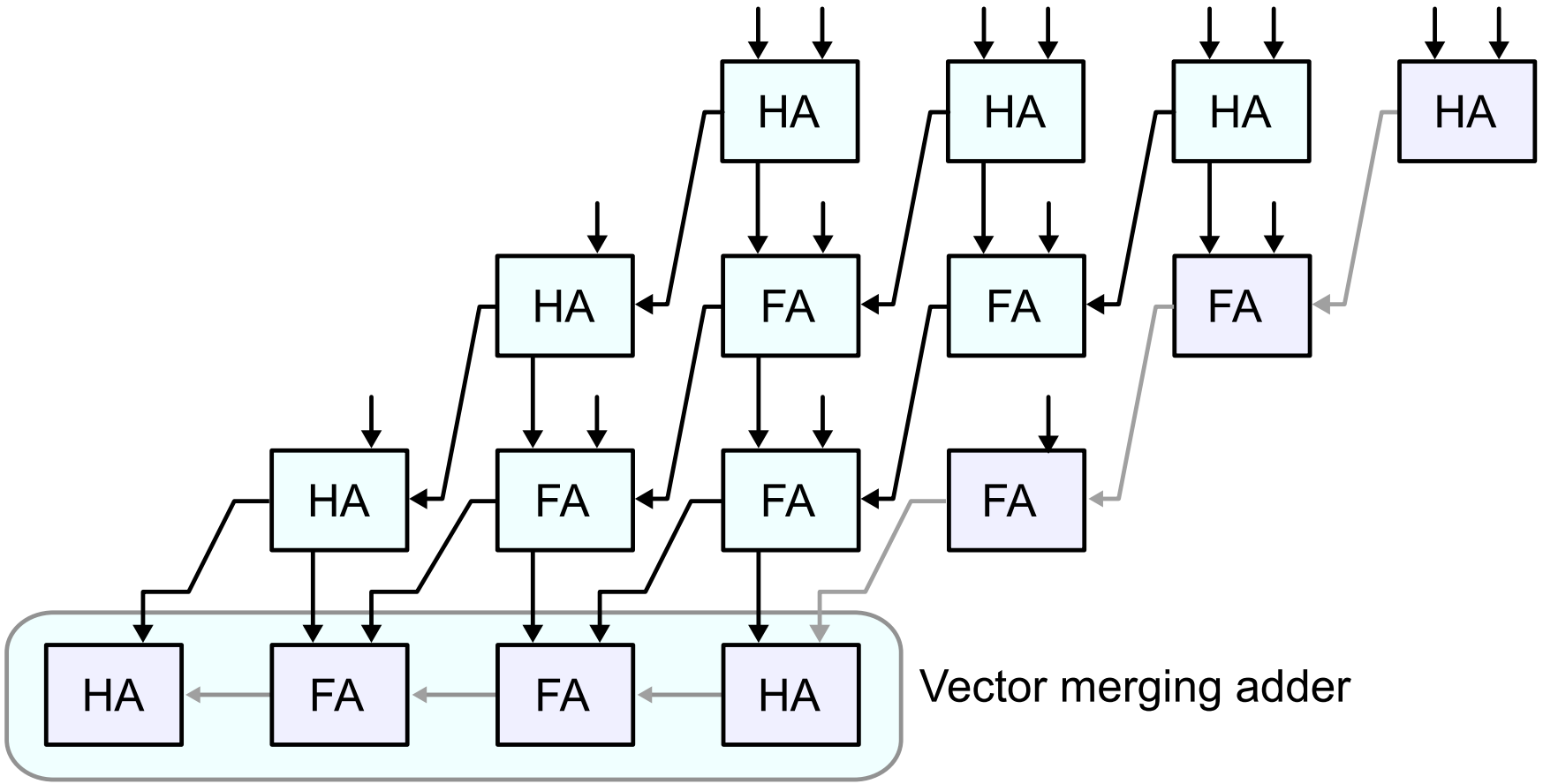


Source: `http://www.slideshare.net/ankitgoel/cmos-arithmetic-circuits`

Carry Save Adder Multiplier



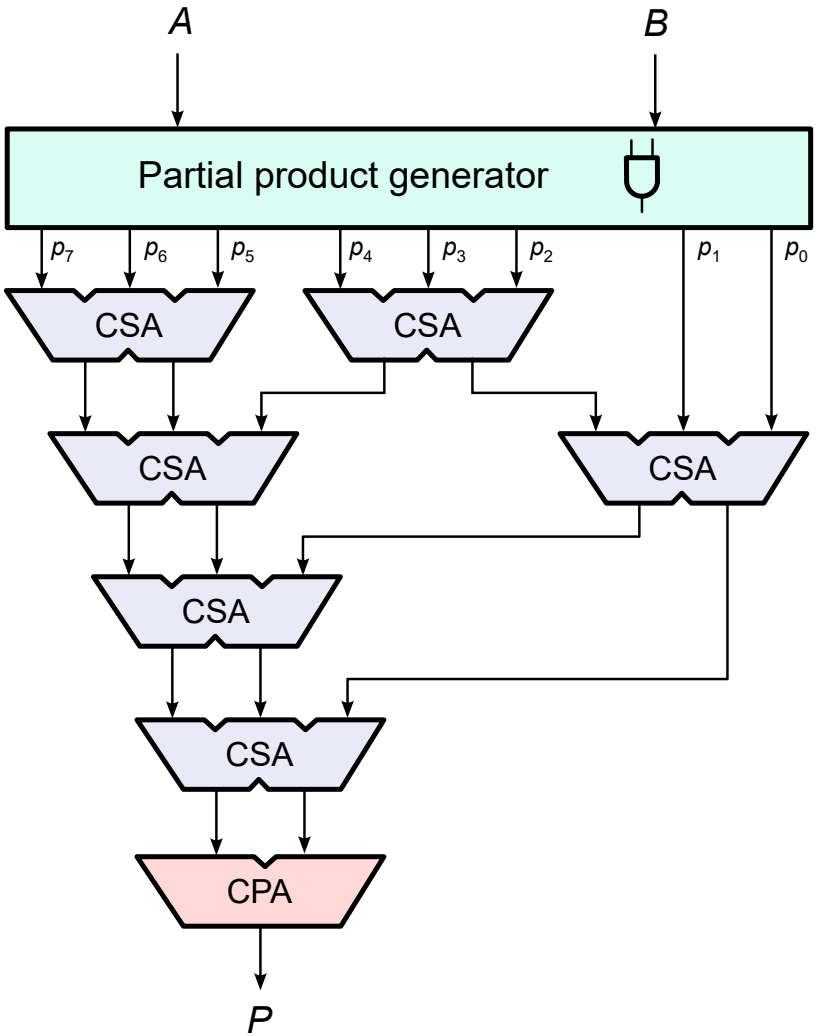
Carry Save Adder Multiplier Critical Path



Source: http://www.slideshare.net/ankitgoel/cmos-arithmetic-circuits

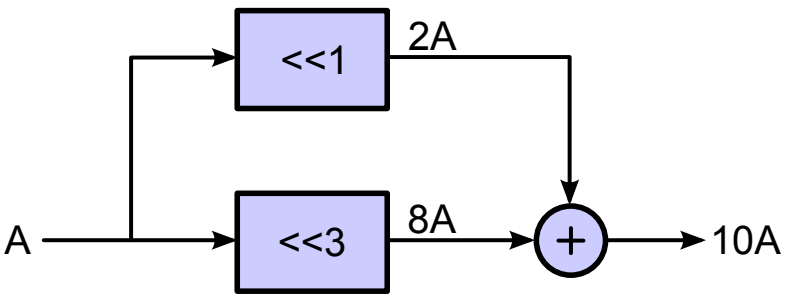
Wallace Tree Multiplier

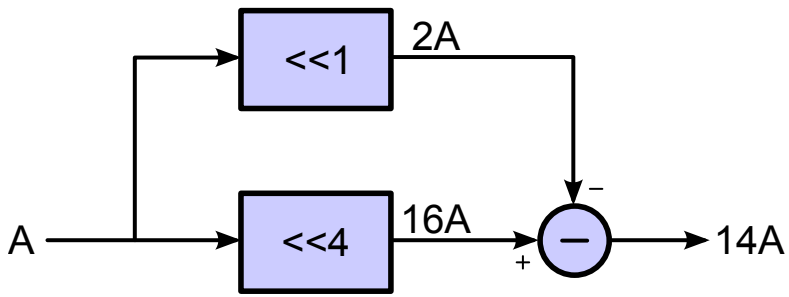
An application of carry save adder



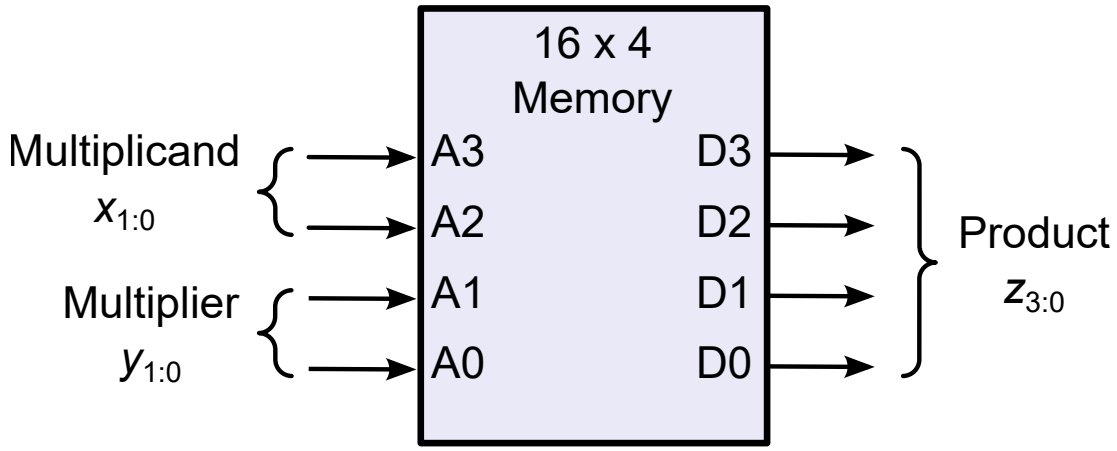
Multiplication with Constants

- Constants with long strings of 0s or 1s can be multiplied very efficiently using just a few adders.
- Remember that shifting a number one bit = multiplication by 2





Lookup Table (LUT) Multipliers



A_3A_2	A_1A_0			
	00	**01**	**10**	**11**
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001