

# Chapter 5

## Arithmetic Circuits

### SKEE2263 Digital Systems

Mun'im/Ismahani/Izam

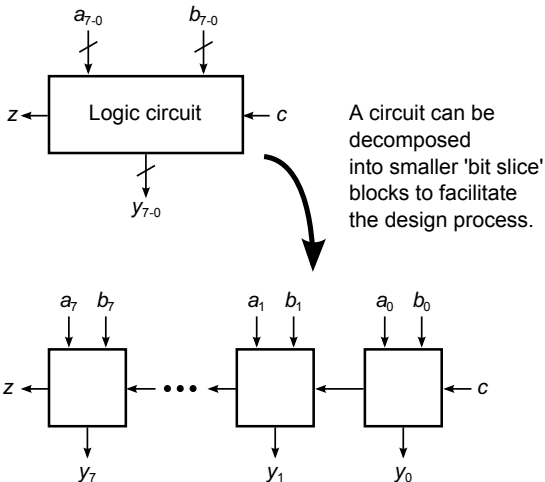
{munim@utm.my,e-izam@utm.my,ismahani@fke.utm.my}

March 21, 2016

# Table of Contents

- 1 Iterative Designs
- 2 Adders
- 3 High-Speed Adders
- 4 Subtractors
- 5 Condition Codes
- 6 Multipliers
- 7 High-Speed Multipliers

# Iterative Designs

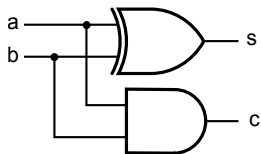


# Half Adder

<i>a</i>	<i>b</i>	<i>c</i>	<i>s</i>
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

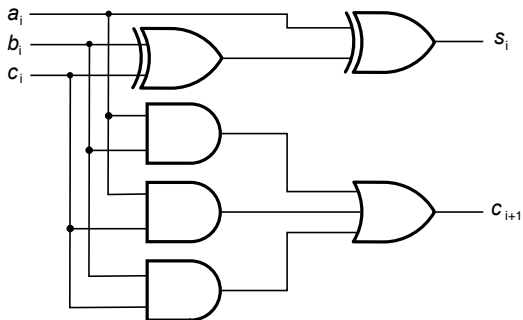
$a_i$	$b_i$	$c_i$	$c_{i+1}$	$s_i$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

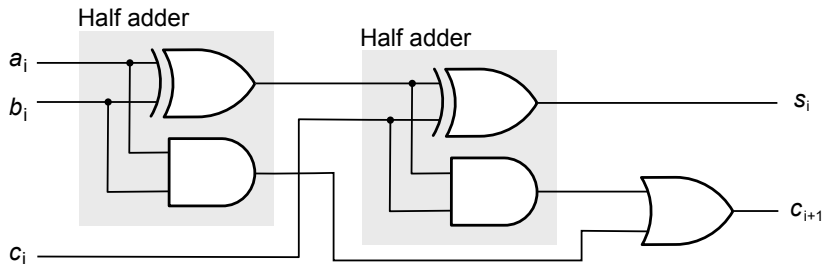
# Full Adder

## Two-Level Realization



# Full Adder

## Multi-Level, Hierarchical Realization



$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$= a_i \oplus b_i \oplus c_i$$

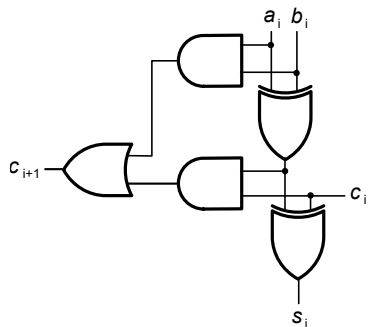
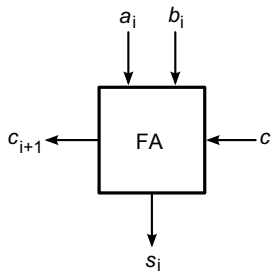
$$c_{i+1} = a_i b_i + (a_i \oplus b_i)c_i$$

$$= a_i b_i + (a_i + b_i)c_i$$

$$= a_i b_i + a_i c_i + b_i c_i$$

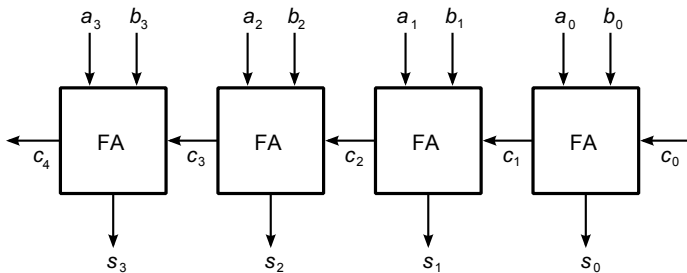
# Full Adder

## Schematic for Cascadable Module

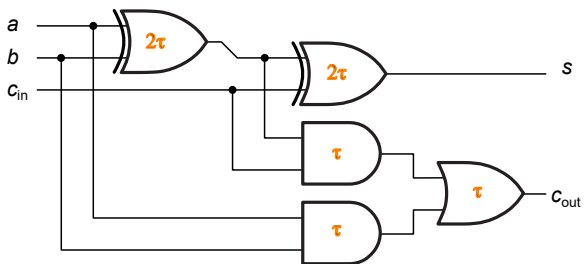




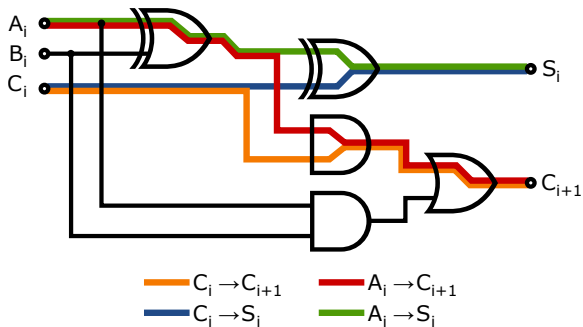
# 4-bit Ripple Adder



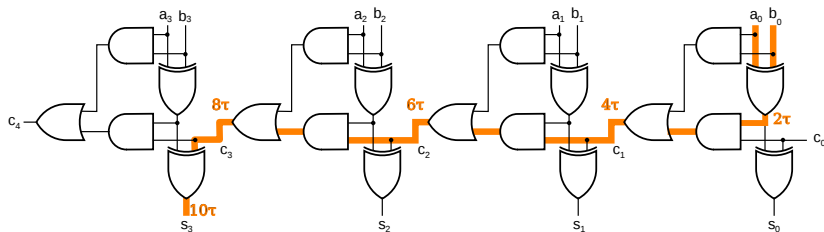
# Delay Analysis of Cascadable Full Adder



## Delay Analysis of Cascadable Full Adder



## Delay Analysis of 4-bit Adder



Delay expression in terms of  $\tau$ :

$$t_p = [4 + 2(n - 2) + 2]\tau = [2n + 2]\tau$$

In "O"-notation, delay is linear with number of bits

$$O(n)$$

Generic delay expression:

$$t_p = (n - 1)t_{\text{carry}} + t_{\text{sum}}$$

# Carry Lookahead Adder

## Analysis of Carry Behavior

$a_i$	$b_i$	$c_i$	$c_{i+1}$	<b>Decision</b>	$g_i$	$p_i$	
0	0	0	0	"Carry kill"	0	0	
0	0	1	0		0	0	
0	1	0	0	"Carry propagate"	0	1	
0	1	1	1		$c_i$	0	1
1	0	0	0		0	1	
1	0	1	1		0	1	
1	1	0	1	"Carry generate"	1	1	
1	1	1	1		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  $a_i b_i = 1$ .

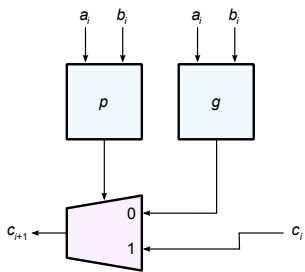
$$g_i = a_i b_i$$

- **Carry propagate** ( $p_i$ ):

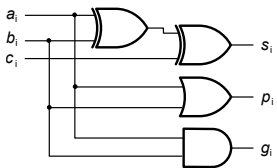
propagates the carry-in ( $c_i$ ) to a stage to the next stage (i.e.,  $c_{i+1} \leftarrow c_i$ ) if the “half sum”  $a_i \oplus b_i = 1$ .

$$p_i = a_i + b_i$$

# Carry Lookahead Adder



$p_i, g_i$  w.r.t. carries



Partial Full Adder (PFA)

# Carry Lookahead Adder

## Carry-out Equations

$$c_1 = g_0 + p_0c_0$$

$$c_2 = g_1 + p_1c_1$$

$$= g_1 + p_1(g_0 + p_0c_0)$$

$$= g_1 + p_1g_0 + p_1p_0c_0$$

$$c_3 = g_2 + p_2c_2$$

$$= g_2 + p_2(g_1 + p_1g_0 + p_1p_0c_0)$$

$$= g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

$$c_4 = g_3 + p_3c_3$$

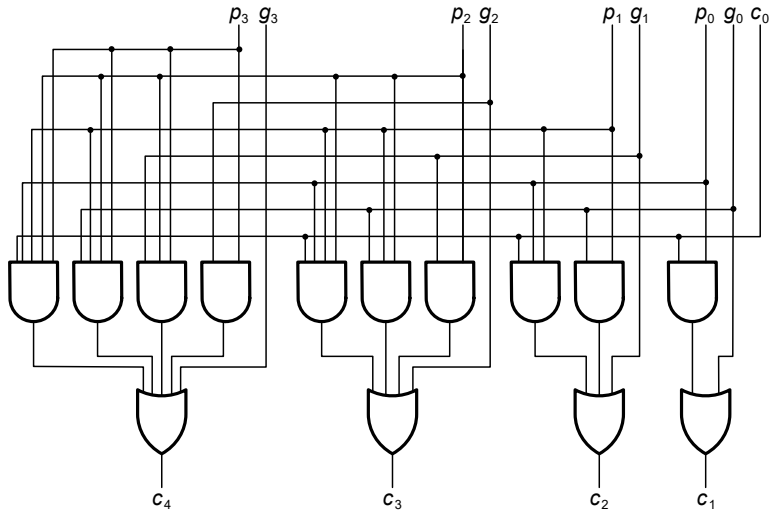
$$= g_3 + p_3(g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0)$$

$$= g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$$



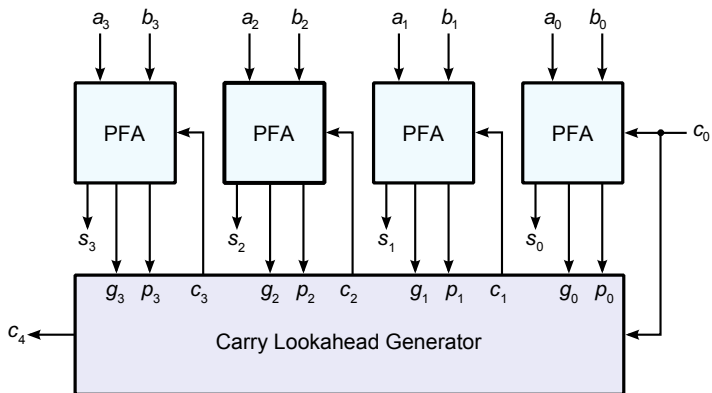
# Carry Lookahead Adder

## Carry Lookahead Generator (CLG)



# Carry Lookahead Adder

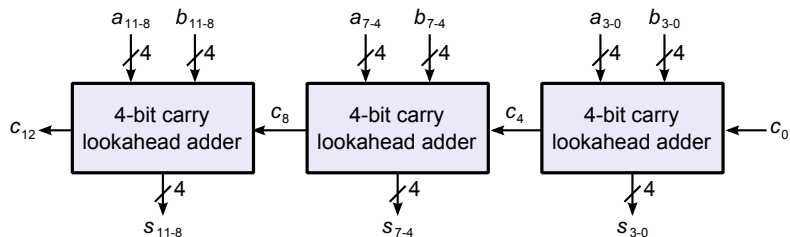
## 4-bit Carry Lookahead Adder



- Maximum gate delay for the carry generation is  $3 \tau$ .
- The full adders introduce two more gate delays.
- $\therefore$  the worst case path is  $5 \tau$ .

# Carry Lookahead Adder

## 12-bit Cascaded Carry Lookahead Adders

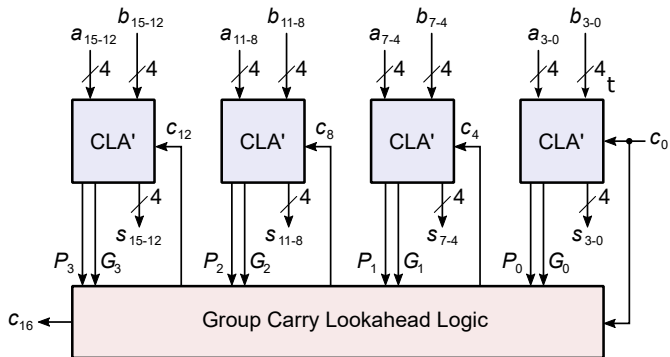


$$\begin{aligned}
 \text{Delay to calculate } C_{12} &= \text{delay}(C_0 \rightarrow C_4) + \text{delay}(C_4 \rightarrow C_8) + \text{delay}(C_8 \rightarrow C_{12}) \\
 &= 3\tau + 2\tau + 2\tau \\
 &= 7\tau
 \end{aligned}$$

$$\begin{aligned}
 \text{Delay to calculate } S_{11} &= \text{delay}(C_0 \rightarrow C_{12}) + 2\tau \\
 &= 7\tau + 2\tau \\
 &= 9\tau
 \end{aligned}$$

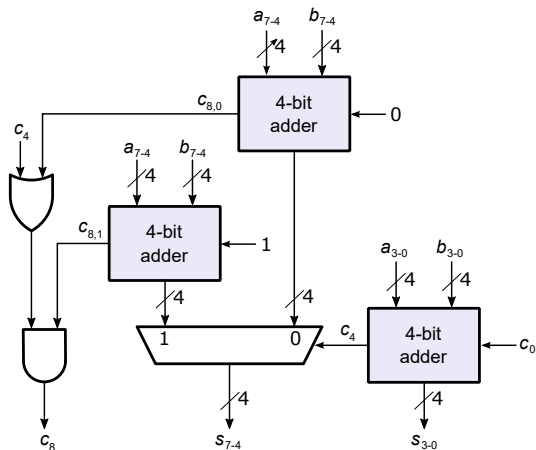
# Carry Lookahead Adder

## 16-bit Carry Lookahead Adder



- Total delay =  $10 \tau$ .

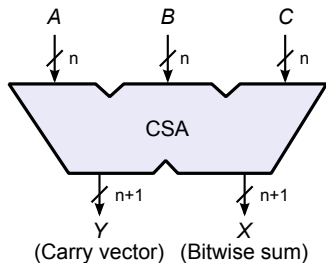
# Carry Select Adder



- Total delay = delay of 1 4-bit adder +  $2 \tau$ .

# Carry Save Adder

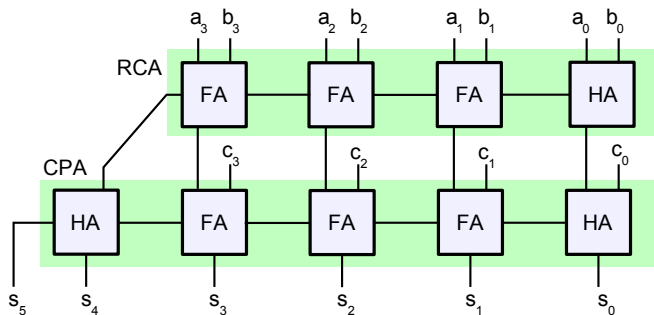
3:2 Compression Costing Just  $2\tau$



$$\begin{array}{r}
 A \quad 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\
 B \quad 0 \ 1 \ 0 \ 1 \ 0 \ 1 \\
 + \ C \quad 1 \ 1 \ 1 \ 1 \ 0 \ 1 \\
 \hline
 S \quad 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \\
 C \quad 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \\
 \hline
 \Sigma \quad 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1
 \end{array}$$

# Carry Save Adder

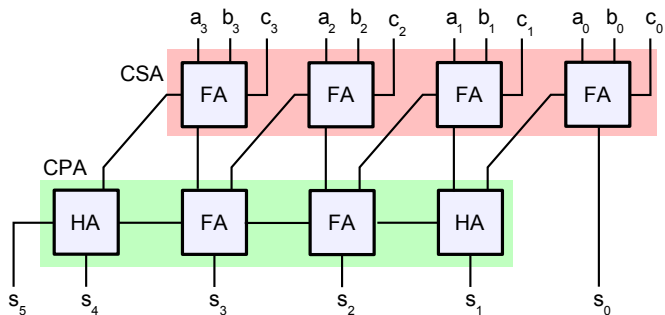
Adding 3 numbers using Ripple Carry Adders only



Delay  $30\tau$

# Carry Save Adder

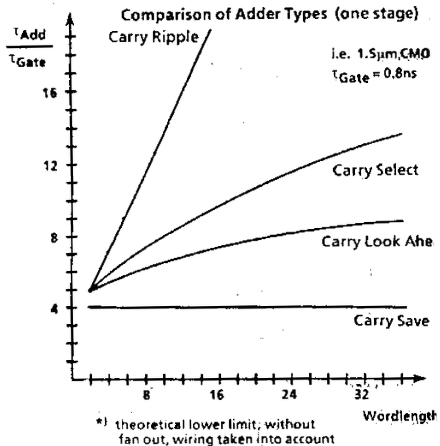
Adding 3 numbers using Carry Save Adders



Delay  $14\tau$



# Adder Delay Summary



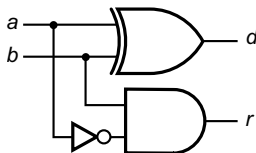
# Half Subtractor

Input		Output	
<i>a</i>	<i>b</i>	<i>r</i> (borrow)	<i>d</i> (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	
$a$	$b$	$r_{in}$	$r_{out}$	$d$
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

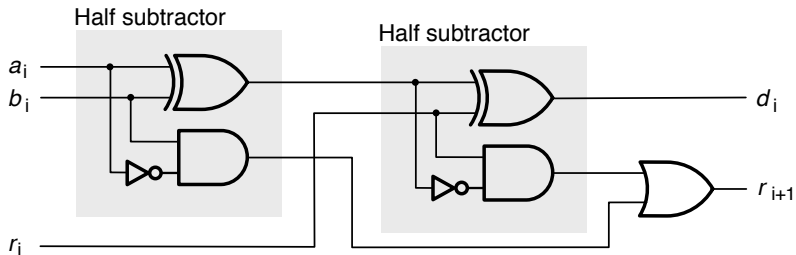
$$d = a'b'c + a'bc' + ab'c' + abc$$

$$= a \oplus b \oplus c$$

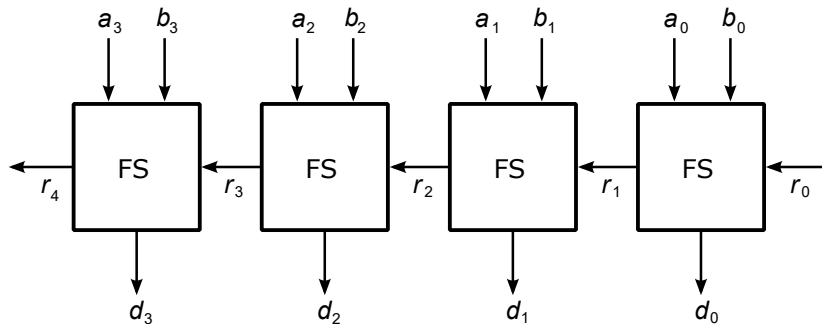
$$c_{out} = a'b'c + a'bc' + a'bc + abc$$

$$= a'c + a'b + bc$$

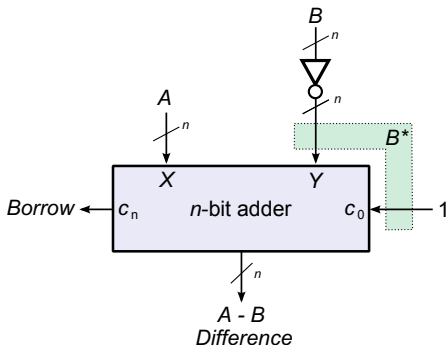
# Full Subtractor



# Ripple Subtractor



# Subtraction by Adding



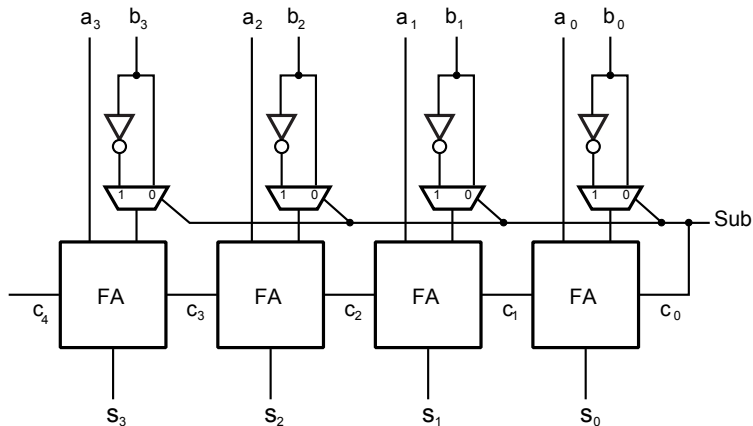
$$\begin{aligned}
 S &= A + \overline{B} + 1 \\
 &= A + B^* \\
 &= A + (2^n - B) \\
 &= A - B + 2^n \\
 &= A - B
 \end{aligned}$$

where:

$\overline{B}$ : Ones' complement of  $B$

$B^*$ : Two's complement of  $B$

# Adder/Subtractor



# Flags

<b>Code</b>	<b>Name</b>	<b>Significance</b>
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.

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

## N Flag

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

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

# C Flag

The carry out from the adder

$$C = c_n$$

# 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

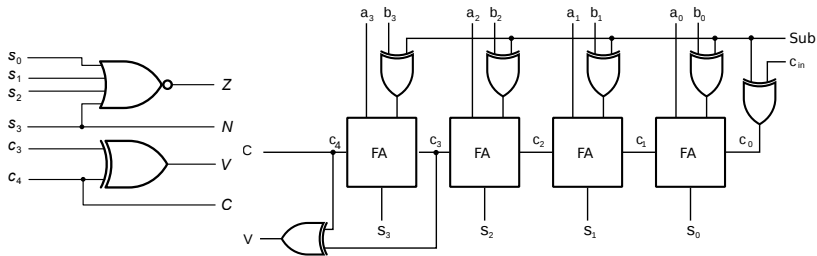
Method 1:

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

Method 2:

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

# Flags Circuit



## Deriving Other Relations

<b>Math Symbol</b>	<b>Description</b>	<b>Flags (Unsigned)</b>	<b>Flags (Signed)</b>
=	Equal		Z
≠	Not equal		Z'
<	Less than	C'	$N \oplus V$
≤	Less than or equal	$C' + Z$	$Z + (N \oplus V)$
≥	Greater than or equal	C	$N \odot V$
>	Greater than	$C \cdot Z'$	$Z' \cdot (N \odot V)$

# Binary Multiplication

$$\begin{array}{r}
 \phantom{\times} 1\ 1\ 0\ 1 \quad (13)_{10} \quad \text{Multiplicand M} \\
 \times 1\ 0\ 1\ 1 \quad (11)_{10} \quad \text{Multiplier Q} \\
 \hline
 \phantom{\times} 1\ 1\ 0\ 1 \\
 \phantom{\times} 1\ 1\ 0\ 1 \\
 \phantom{\times} 0\ 0\ 0\ 0 \\
 \phantom{\times} 1\ 1\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \quad (143)_{10} \quad \text{Product P}
 \end{array}$$

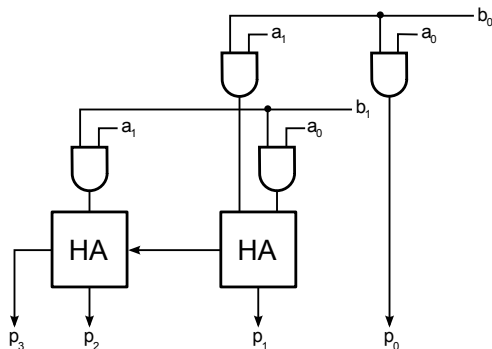
} Partial Products

Multiplication is all about:

- 1 Generating the partial products
- 2 Adding the partial products

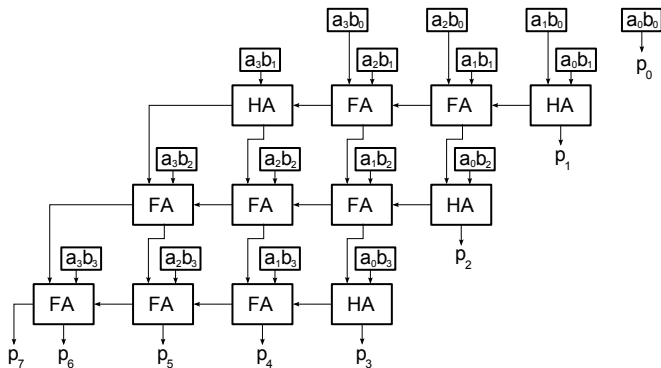
## 2 × 2 Multiplication

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

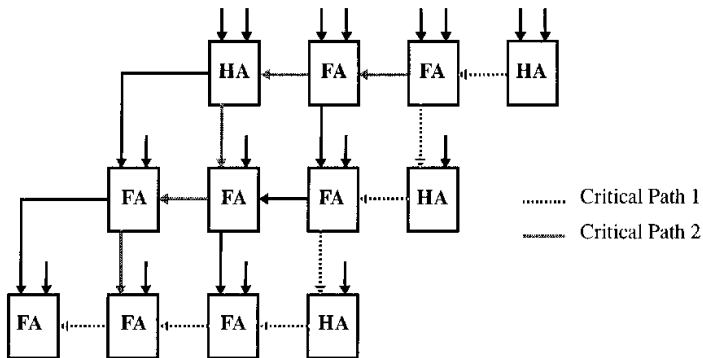




# Ripple Carry Array Multiplier



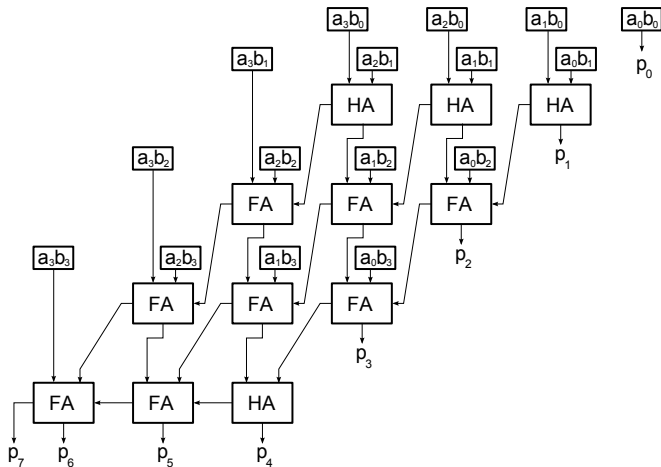
# $M \times N$ Array Multiplier Critical Path



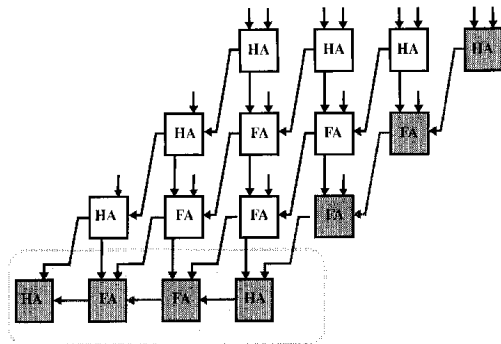
$$t_{mult} = [(M-1) + (N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

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

# Carry Save Adder Multiplier



# Carry Save Adder Multiplier Critical Path



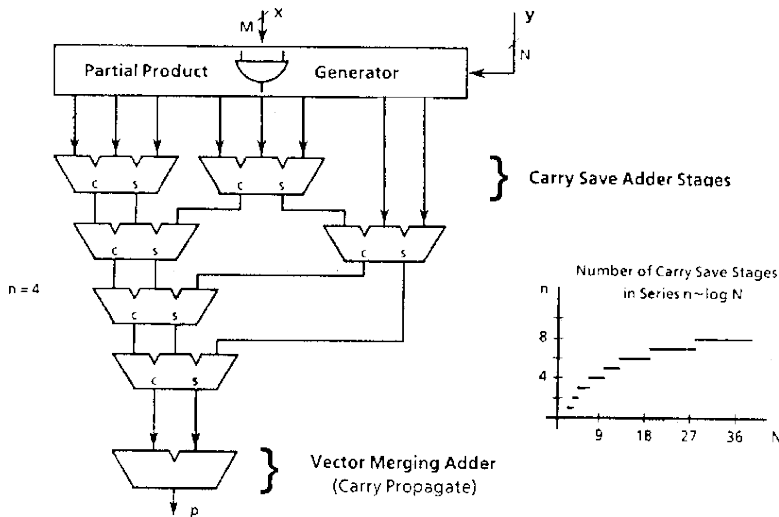
Vector Merging Adder

$$t_{mult} = (N-1)t_{carry} + (N-1)t_{and} + t_{merge}$$

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

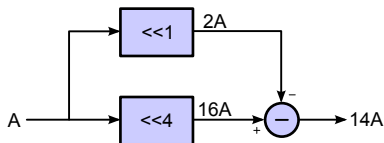
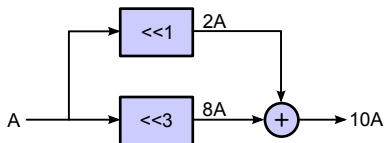
# Wallace Tree Multiplier

An application of carry save adder

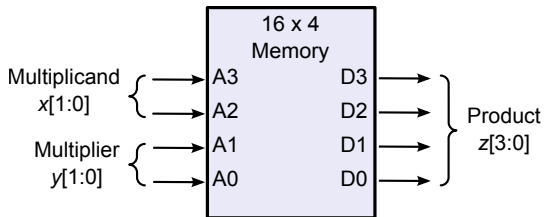


## Multiplication with Constants

- 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