Chapter 3

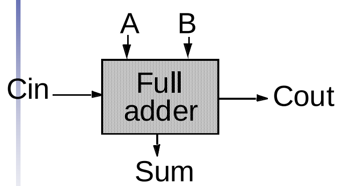
Arithmetic Circuits

Instructor: Prof. Peter Lian
Department of Electrical Engineering & Computer Science
Lassonde School of Engineering
York University

Adders

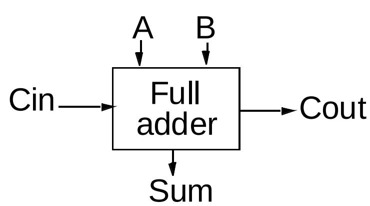
- Ripple-Carry Adder
- Carry-Bypass Adder
- Carry-Select Adder
- Carry-Lookahead Adder

Full-Adder



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

The Binary Adder



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

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

$$C_o = AB + BC_i + AC_i = AB + C_i(A \oplus B)$$

Generate, Propagate, and Delete

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = $A \oplus B$

Delete = $\overline{A}\,\overline{B}$

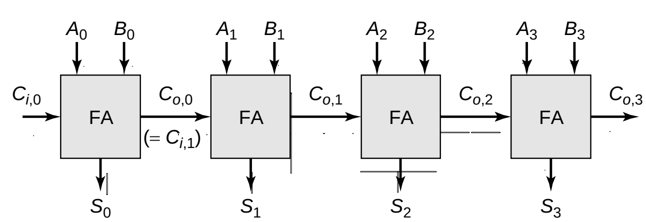
$$C_o(G,P) = G + PC_i$$

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

Can also derive expressions for S and C_o based on D and P

Note that we will be sometimes using an alternate definition for ***Propagate (P) = $A + B$***

The Ripple-Carry Adder



Worst case delay linear with the number of bits

$t_d = O(N)$

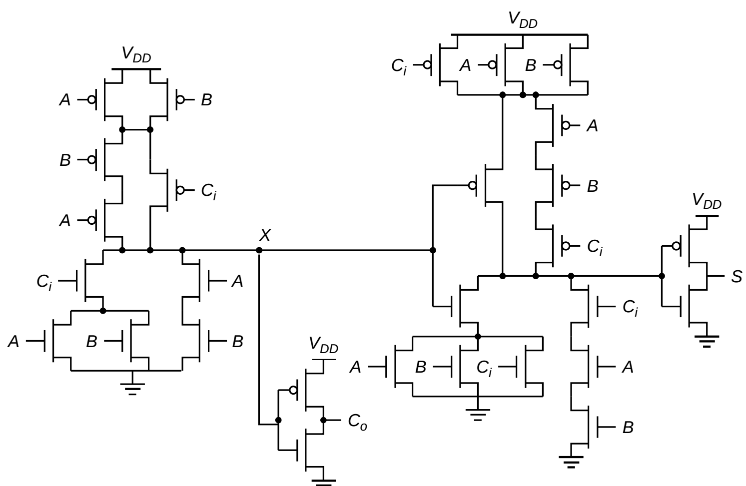
$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$

Goal: Make the fastest possible carry path circuit

Activity 1

Derive the values of A_k and B_k (k=0, …, N-1) so that the worst case delay is obtained for the ripple-carry adder.

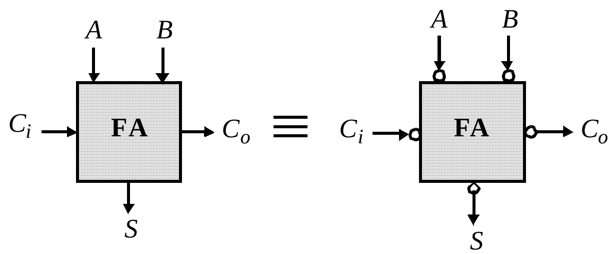
Complimentary Static CMOS Full Adder



$$C_o = AB + BC_i + AC_i, S = ABC_i + \overline{C}_o(A + B + C_i)$$

28 Transistors

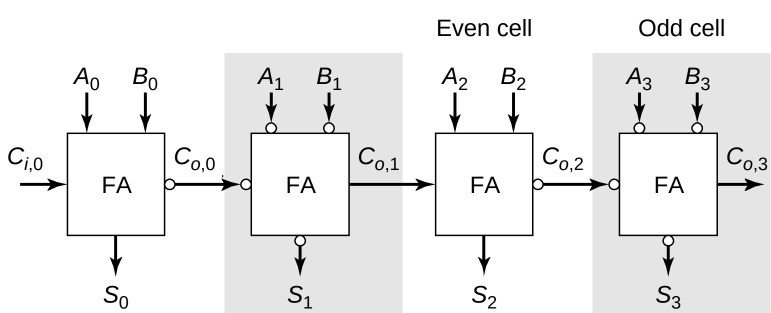
Inversion Property



$$\bar{S}(A,B,C_i) = S(\bar{A},\bar{B},\bar{C}_i)$$

$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C}_i)$$

Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

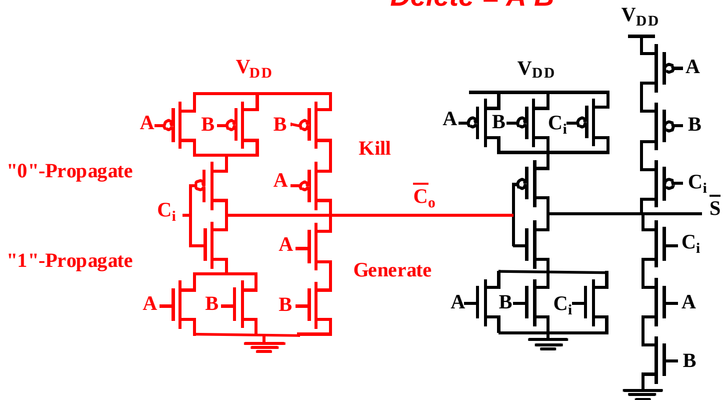
A Better Structure: The Mirror Adder

$$C_o(G,P) = G + PC_i$$
$$S(G,P) = P \oplus C_i$$

Generate (G) = AB

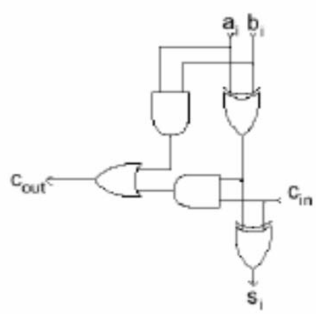
Propagate (P) = $A \oplus B$

Delete = $\overline{A}\,\overline{B}$

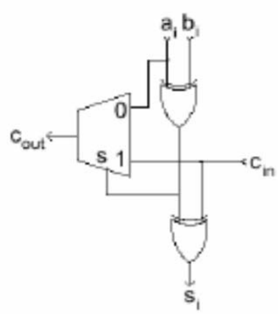


24 transistors

Full Adder Implementation



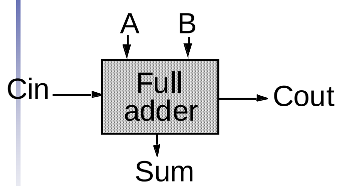
Standard CMOS



Multiplexer-based

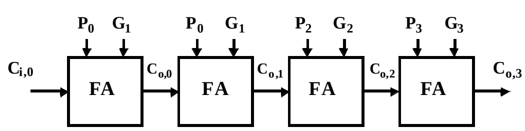
$$S = A \oplus B \oplus C_i$$
$$C_o = AB + C_i(A \oplus B)$$

Full-Adder

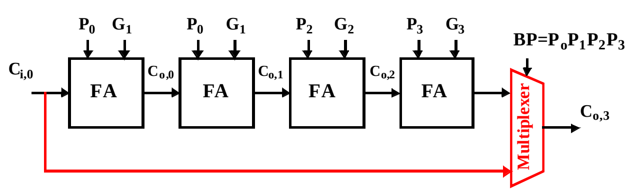


A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Carry-Bypass Adder

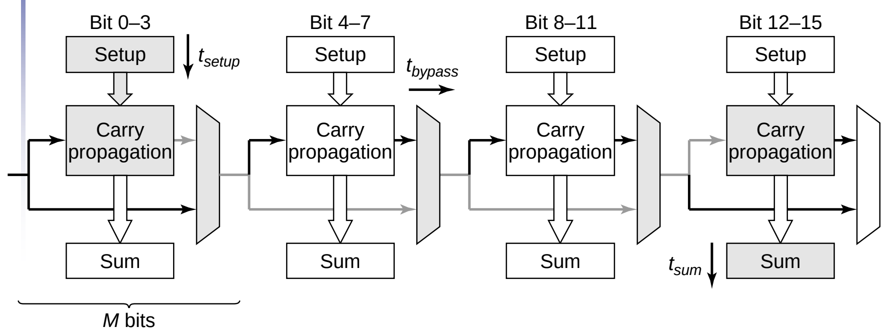


Also called Carry-Skip



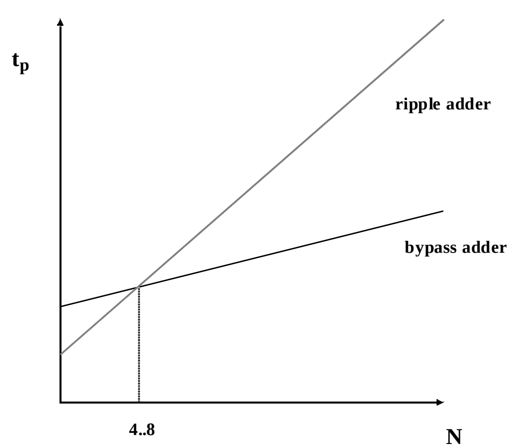
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else "kill" or "generate".

Carry-Bypass Adder

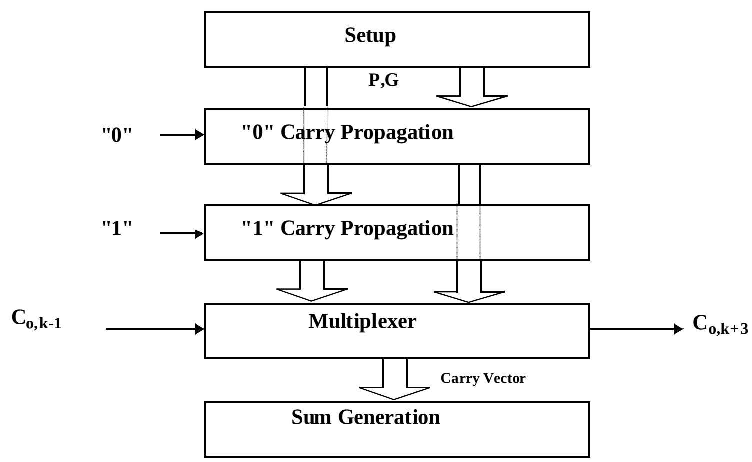


$$t_{adder} = t_{setup} + Mt_{carry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

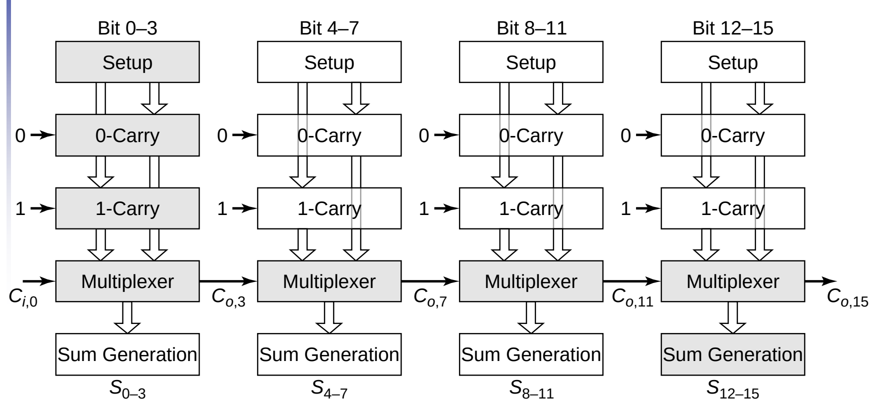
Carry Ripple vs. Carry-Bypass



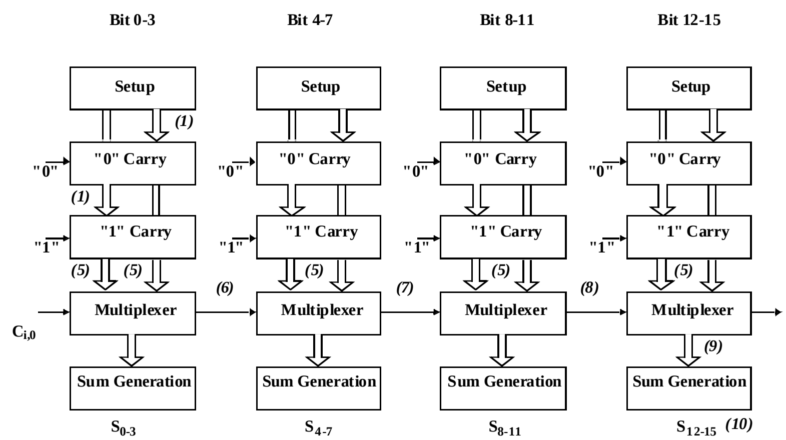
Carry-Select Adder



CSA: Critical Path

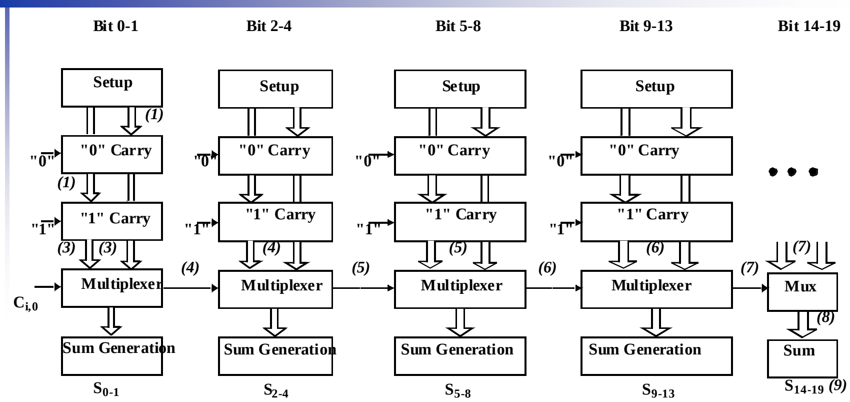


Linear Carry Select



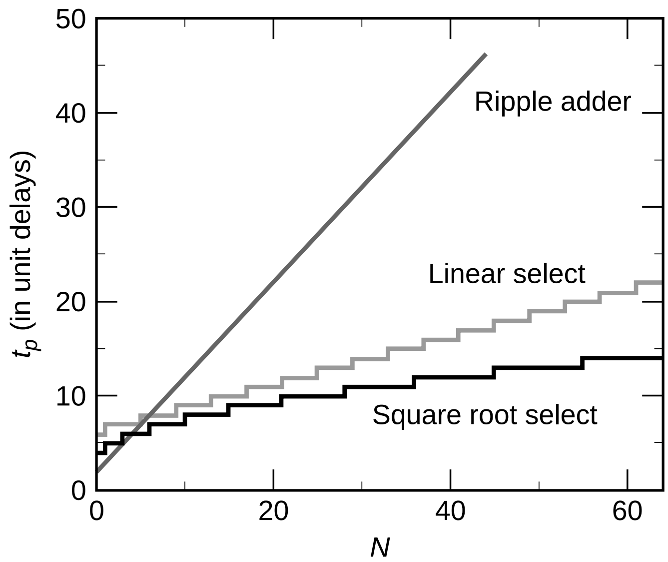
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select



$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N}) t_{mux} + t_{sum}$$

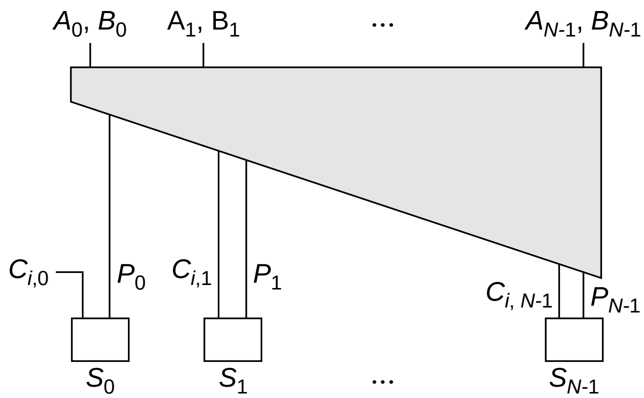
Adder Delays - Comparison



Carry-Lookahead Adders

- Carry-Lookahead Adder – CLA
- Adder trees
  - Radix of a tree
- Logic manipulation
  - Conventional vs. Ling

LookAhead - Basic Idea



$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

Generate, Propagate, and Delete

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Delete = $\overline{A}\,\overline{B}$

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

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

Can also derive expressions for S and C_o based on D and P

Note that we will be sometimes using an alternate definition for

Propagate (P) = A + B

Lookahead Adder

- Lookahead Equations

Position i: $c_i = g_i + p_i c_{i-1}$

Position i+1:
$$\begin{aligned} c_{i+1} &= g_{i+1} + p_{i+1} c_i \\ &= g_{i+1} + p_{i+1}(g_i + p_i c_{i-1}) \\ &= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_{i-1} \end{aligned}$$

Carry exists if:

- Generated in stage i+1
- Generated in stage i and propagated through i+1
- Propagated through both i and i+1

Block Lookahead

- Example for 4th bit carry:

$$\begin{aligned} c_{t+4} = g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i \\ + p_{i+3} p_{i+2} p_{i+1} p_i c_{i-1} \end{aligned}$$

- Block generate and block propagate:

$$G_{i,i+3} = g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i$$
$$P_{i,i+3} = p_{i+3} p_{i+2} p_{i+1} p_i$$
$$c_{i+4} = G_{i,i+3} + P_{i,i+3} c_{i-1}$$

Look-Ahead: Topology

Expanding Lookahead equations:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

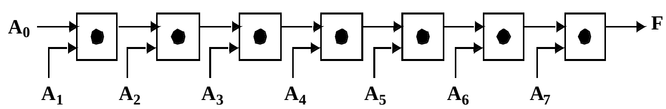
All the way:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0C_{i,0})))$$

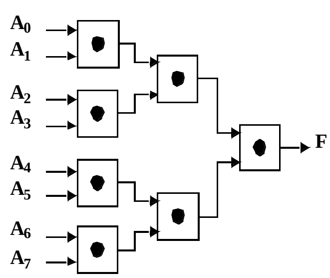
An example:

$$c_{o,3} = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_{i,0}$$

Logarithmic Look-Ahead Adder



$t_p \sim N$



$t_p \sim \log_2(N)$

Carry Lookahead Trees

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

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

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

$$C_{o,3} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{i,0} = G_{3,2} + P_{3,2} C_{o,1}$$

$$\ldots$$

Can continue building the tree hierarchically.

Tree Adders

$$P_G = p_m . p_l$$ *m*: more significant bit

$$G_G = g_m + p_m . g_l$$ *l*: less significant bit

- Start from the input P, G and continue up the tree.
- dot operation:

$$(g, p) = (g_m, p_m) \bullet (g_l, p_l)$$
$$= (g_m + p_m . g_l, p_m . p_l)$$

- Kogge, Stone, Trans on Comp. 1973

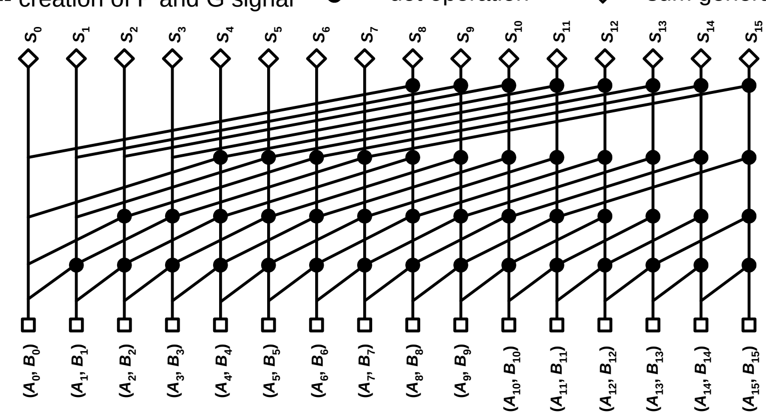
Activity 2

Show that $C_{o,3}$ of a 4-bit ripple-carry adder can be expressed as a function of 2 group carries, i.e.

$C_{o,3}=(G_{3,2}, P_{3,2}).(G_{1,0}, P_{1,0}) .(C_{i,0},0)$

Tree Adders: Radix 2

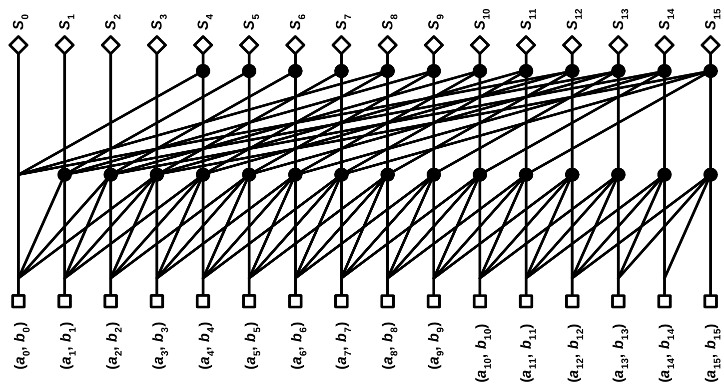
□ -- creation of P and G signal ● -- dot operation ◇ -- sum generation



16-bit radix-2 Kogge-Stone tree:

1. **Building block of order 2**
2. **requires 49 dot operations**

Tree Adders: Radix 4



16-bit radix-4 Kogge-Stone Tree:

1. **Build block of order 4**
2. **Two stages of carry logic**
3. **Complex circuit for group of 4 dot operation**

Ling Adder

- Variation of CLA

Conventional

$$p_i = a_i \oplus b_i$$
$$g_i = a_i \cdot b_i$$
$$G_i = g_i + p_i \cdot G_{i-1}$$
$$S_i = p_i \oplus G_{i-1}$$

Ling's equations

$$t_i = a_i + b_i$$
$$g_i = a_i \cdot b_i$$
$$H_i = g_i + t_{i-1} \cdot H_{i-1}$$
$$S_i = t_i \oplus H_i + g_i t_{i-1} H_{i-1}$$

Ling, IBM J. Res. Dev., 5, 1981

Ling Adder

- Conventional radix-4

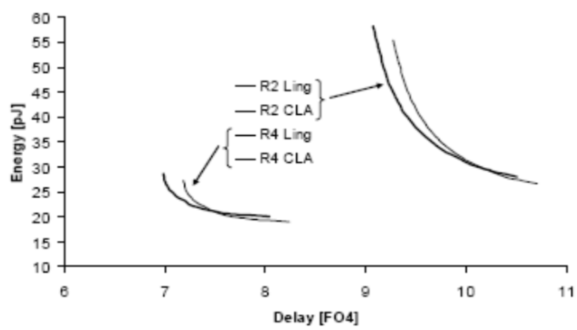
$$G_3 = g_3 + t_3 g_2 + t_3 t_2 g_1 + t_3 t_2 t_1 g_0$$

- Ling radix-4

$$H_3 = g_3 + t_2 g_2 + t_2 t_1 g_1 + t_2 t_1 t_0 g_0$$
$$= g_3 + g_2 + t_2 g_1 + t_2 t_1 g_0$$

  - Reduces the stack height (or width)
  - Reduces input loading

Ling versus CLA



R. Zlatanovici and B. Nikolic, ESSCIRC 2003

CSE4210 Architecture & Hardware for DSP

Multipliers

The Binary Multiplication

$$Z = X \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k$$

$$= \left(\sum_{i=0}^{M-1} X_i 2^i\right)\left(\sum_{j=0}^{N-1} Y_j 2^j\right)$$

$$= \sum_{i=0}^{M-1}\left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j}\right)$$

with

$$X = \sum_{i=0}^{M-1} X_i 2^i$$

$$Y = \sum_{j=0}^{N-1} Y_j 2^j$$

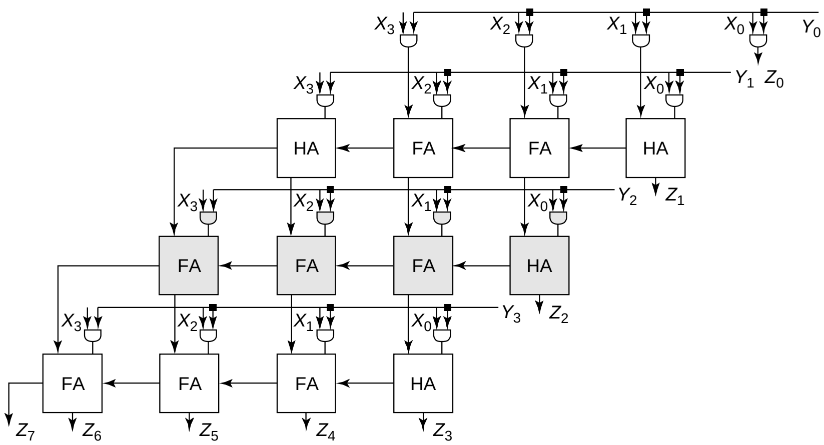
The Binary Multiplication

```
              1  0  1  0  1  0     Multiplicand: N bits
x                   1  0  1  1     Multiplier: M bits
------------------------------
              1  0  1  0  1  0
           1  0  1  0  1  0        AND operation
        0  0  0  0  0  0           Partial products
+    1  0  1  0  1  0
------------------------------
     1  1  1  0  0  1  1  1  0     Result: N+M bits
```

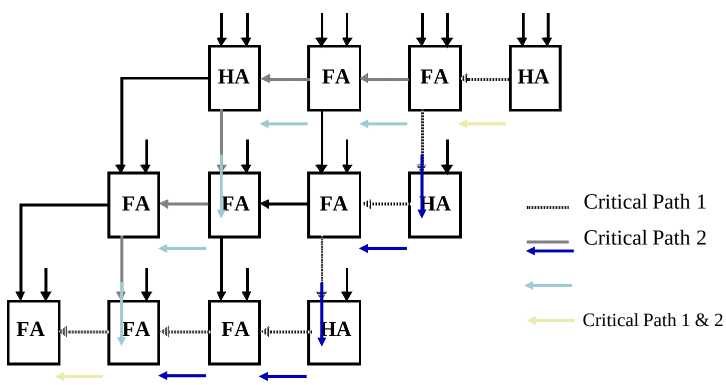
Shift-and-Add Multiplier

- Standard adder and shift-in the multiplicand
- Shift the result as well and add
- N cycles
- Parallel adders and more hardware (adders) instead.

The Array Multiplier

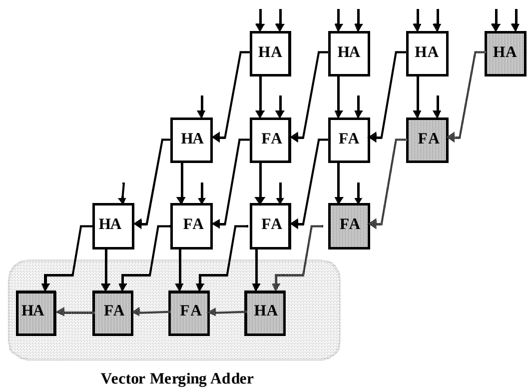


The MxN Array Multiplier— Critical Path



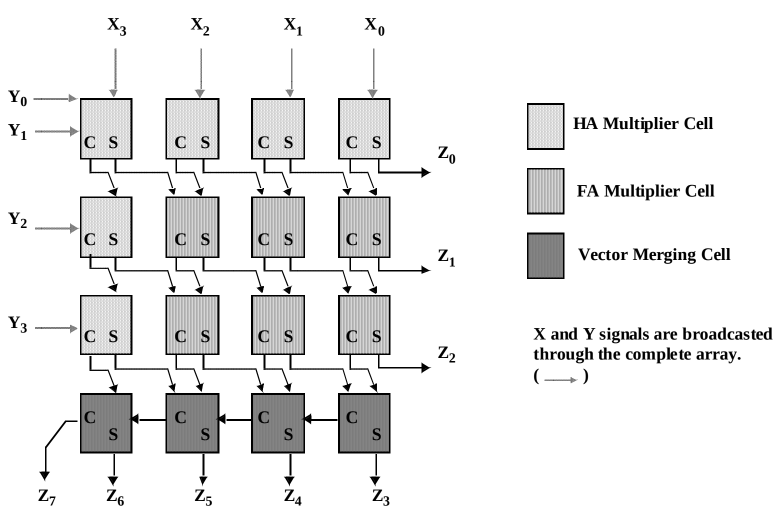
$$t_{mult} \approx \left[(M-1)+(N-2)\right]t_{carry} + (N-1)t_{sum} + t_{and}$$

Carry-Save Multiplier



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

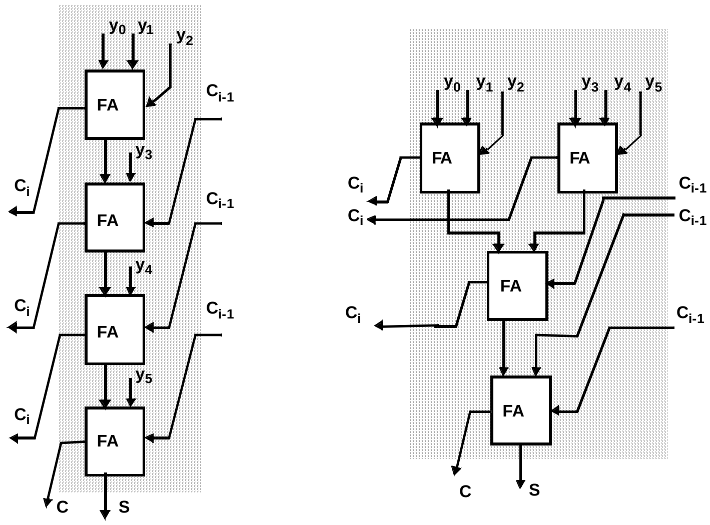
Multiplier Floorplan



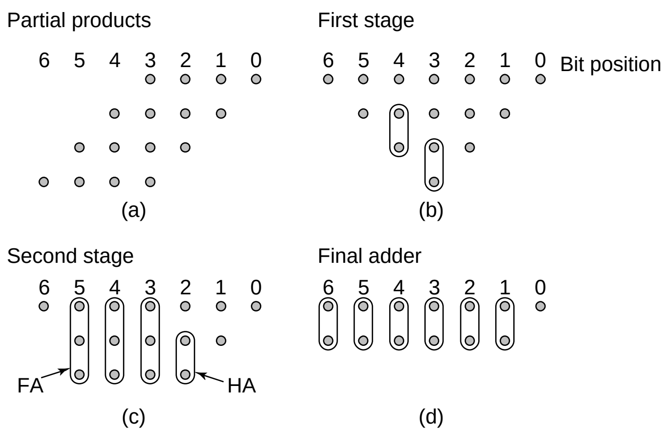
Multipliers

- Partial product generation
- Partial product accumulation
- Final summation

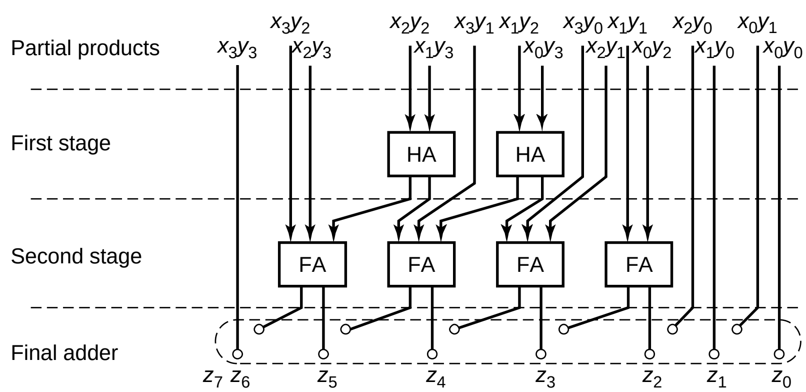
Wallace-Tree Multiplier



Wallace-Tree Multiplier



Wallace-Tree Multiplier



Multipliers —Summary

• Optimization Goals Different Vs Binary Adder

• Once Again: Identify Critical Path

• Other possible techniques
- Logarithmic versus Linear (Wallace Tree Mult)
- Data encoding (Booth)
- Pipelining

FIRST GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

Acknowledgement

- The materials are adopted from “Digital Integrated Circuits: A Design Perspective” by Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, 2nd Edition, 2003, Pearson.