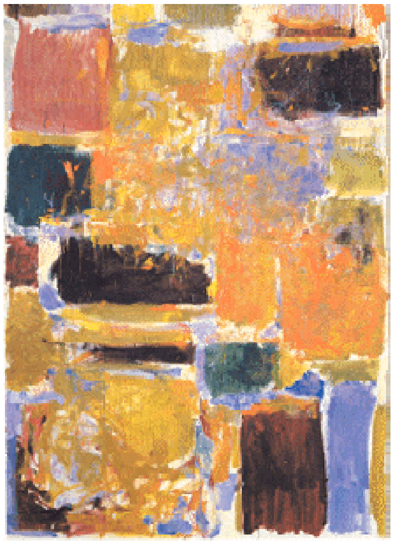


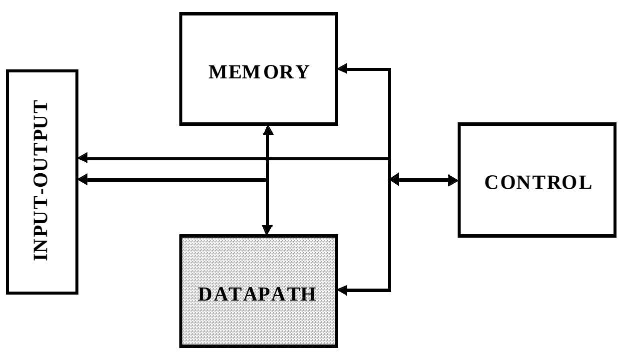
Digital Integrated Circuits

A Design Perspective

Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Arithmetic Circuits

A Generic Digital Processor



Building Blocks for Digital Architectures

Arithmetic unit

- Bit-sliced datapath (adder, multiplier, shifter, comparator, etc.)

Memory

- RAM, ROM, Buffers, Shift registers

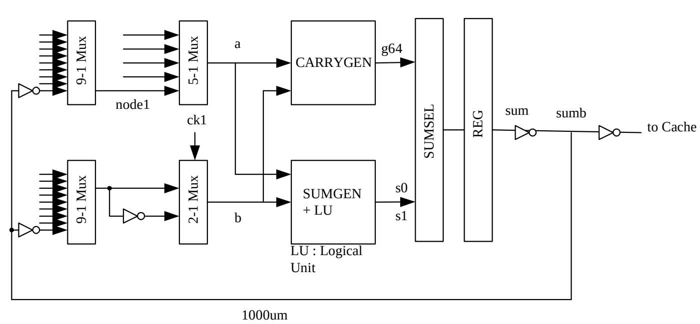
Control

- Finite state machine (PLA, random logic.)
- Counters

Interconnect

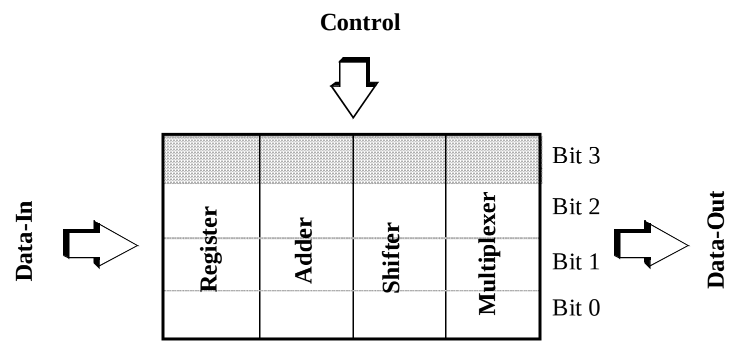
- Switches
- Arbiters
- Bus

An Intel Microprocessor



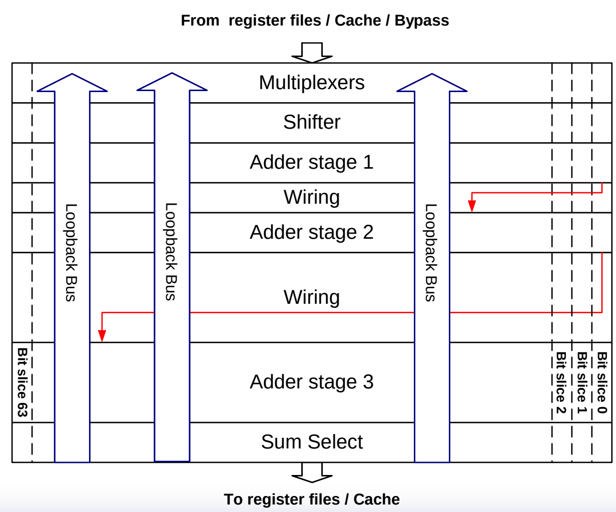
Itanium has 6 integer execution units like this

Bit-Sliced Design

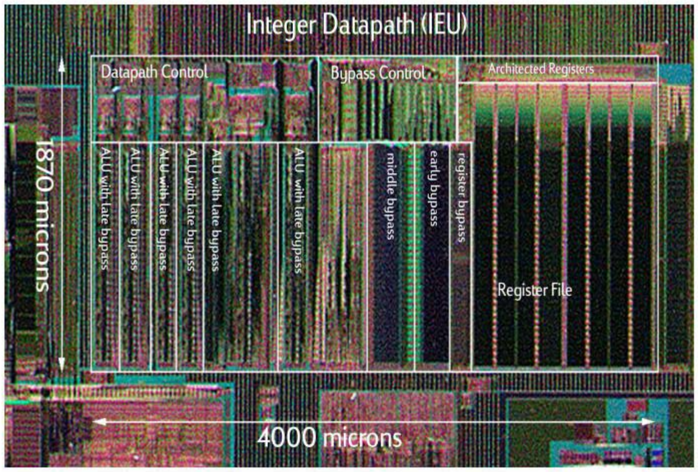


Tile identical processing elements

Bit-Sliced Datapath



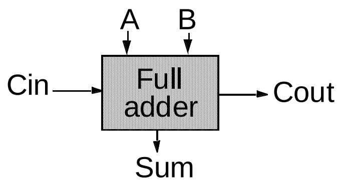
Itanium Integer Datapath



Fetzer, Orton, ISSCC'02

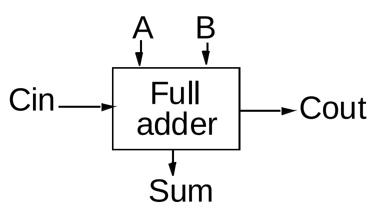
Adders

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



$$\mathbf{S} = \mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C_i}$$

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

$$\mathbf{C_o} = \mathbf{AB + BC_i + AC_i}$$

Express Sum and Carry as a function of P, G, D

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = $A \oplus B$

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

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

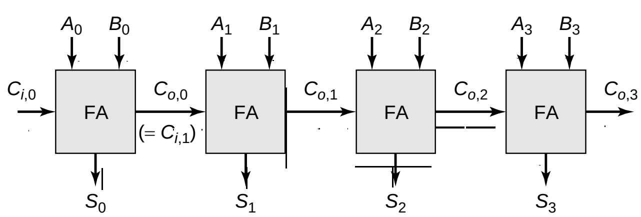
$$S(G, P) = P \oplus C_i$$

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



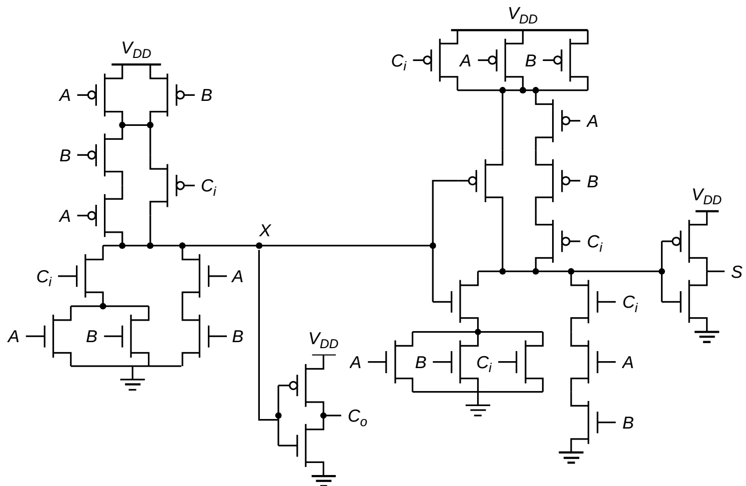
Worst case delay linear with the number of bits

$t_d = O(N)$

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

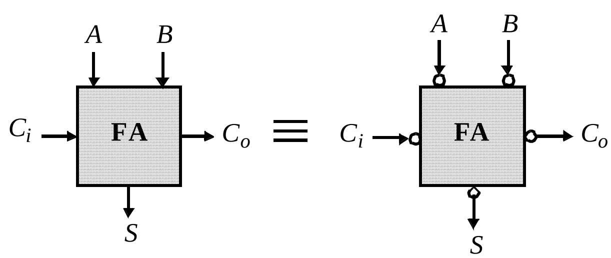
Goal: Make the fastest possible carry path circuit

Complimentary Static CMOS Full Adder



28 Transistors

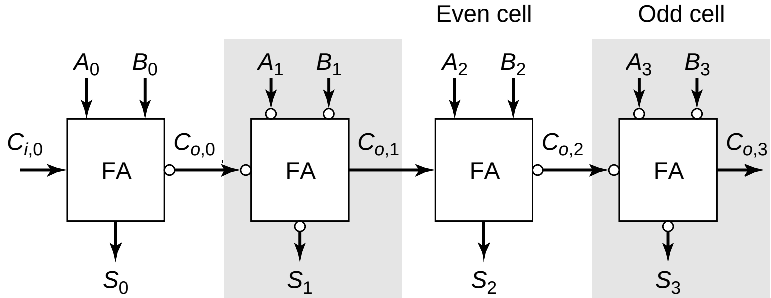
Inversion Property



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

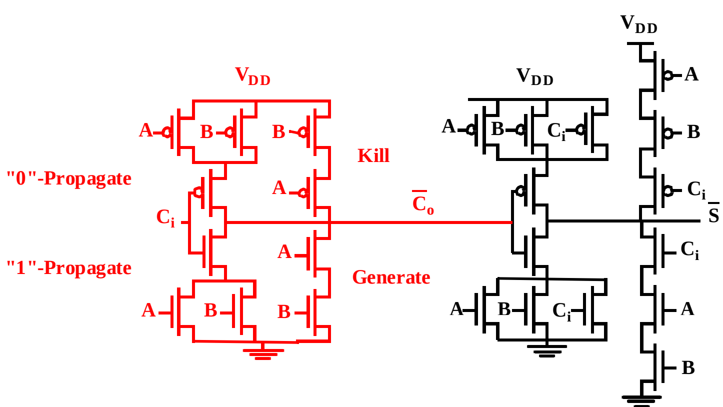
$$\overline{C_o}(A, B, C_i) = C_o(\bar{A}, \bar{B}, \bar{C_i})$$

Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

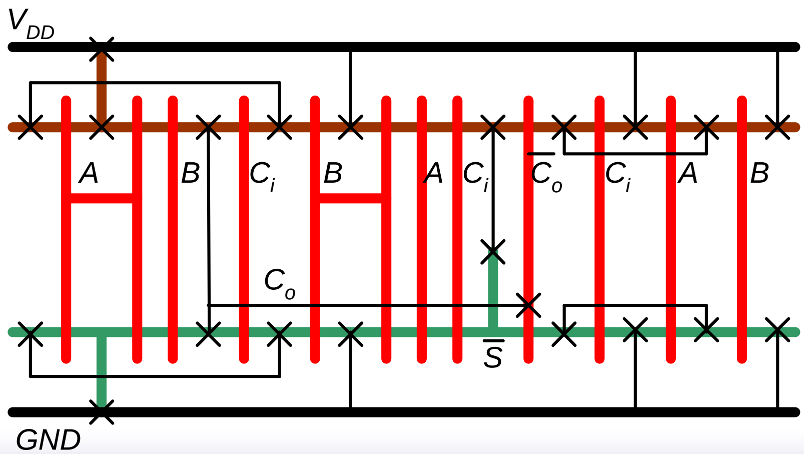
A Better Structure: The Mirror Adder



24 transistors

Mirror Adder

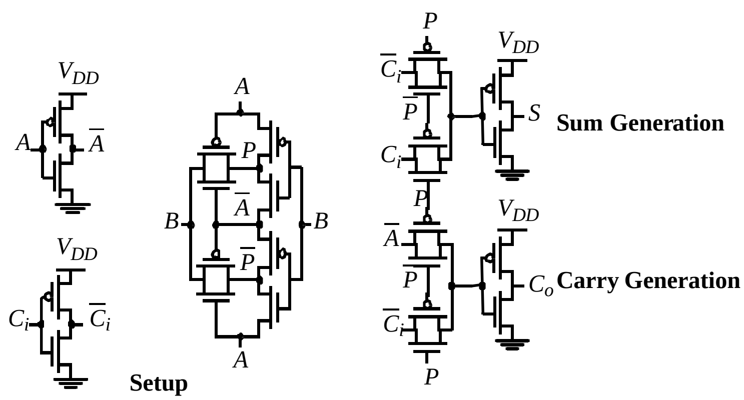
Stick Diagram



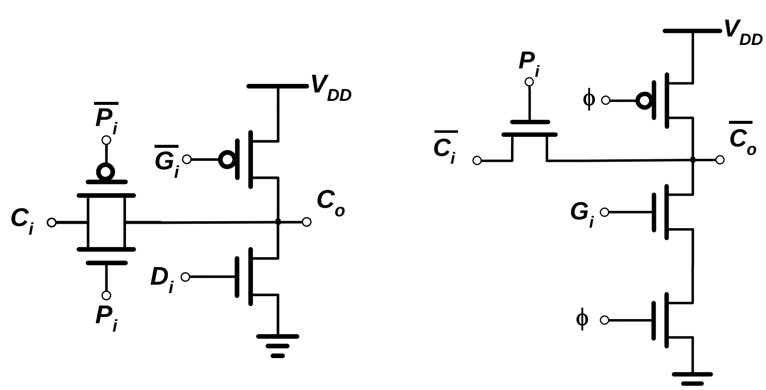
The Mirror Adder

- The NMOS and PMOS chains are completely symmetrical.
  A maximum of two series transistors can be observed in the carry-generation circuitry.
- When laying out the cell, the most critical issue is the minimization of the capacitance at node C_o. The reduction of the diffusion capacitances is particularly important.
- The capacitance at node C_o is composed of four diffusion capacitances, two internal gate capacitances, and six gate capacitances in the connecting adder cell .
- The transistors connected to C_i are placed closest to the output.
- Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.

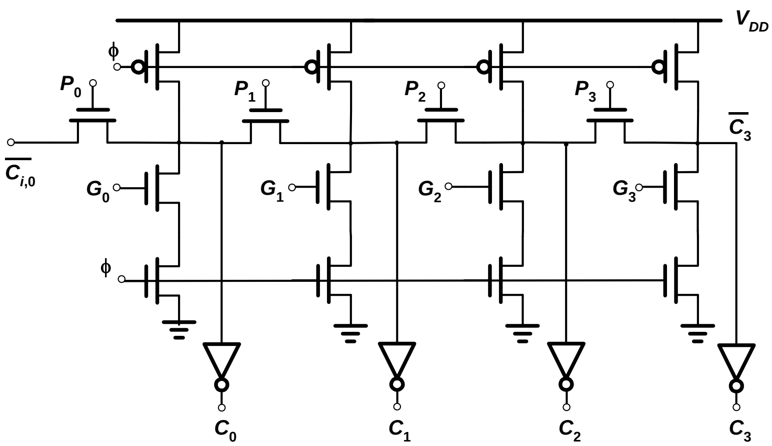
Transmission Gate Full Adder



Manchester Carry Chain

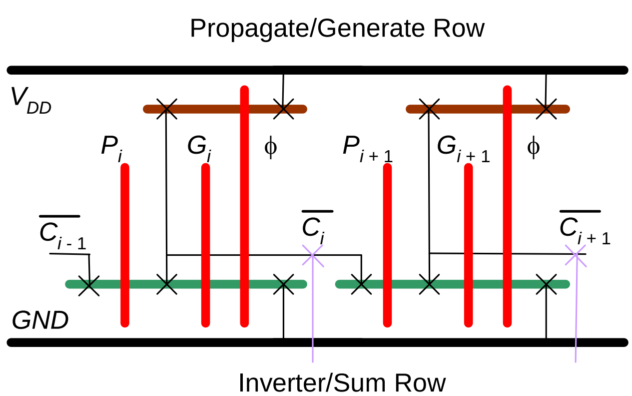


Manchester Carry Chain

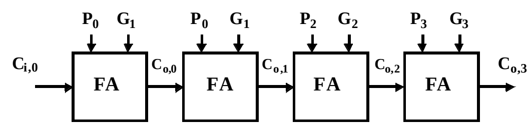


Manchester Carry Chain

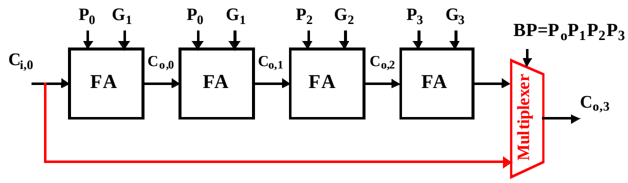
Stick Diagram



Carry-Bypass Adder

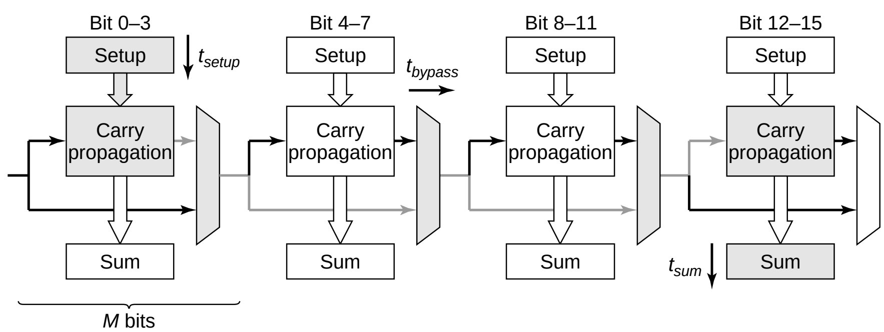


Also called Carry-Skip



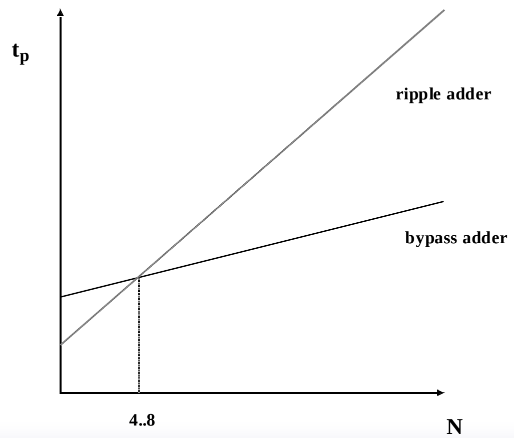
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else “kill” or “generate”.

Carry-Bypass Adder (cont.)

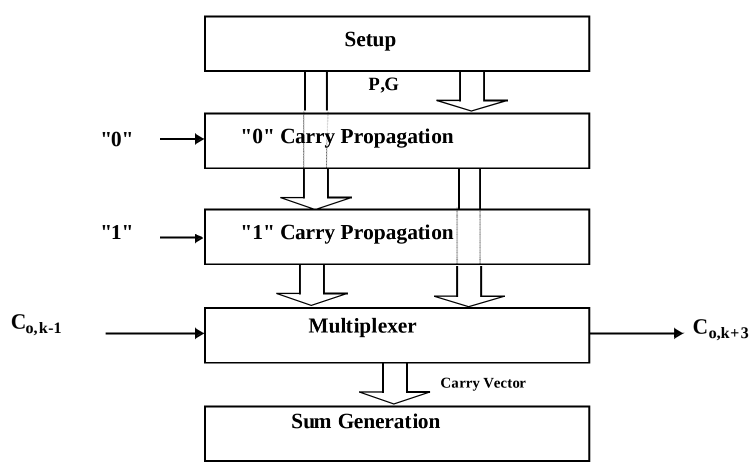


$$t_{adder} = t_{setup} + M t_{carry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

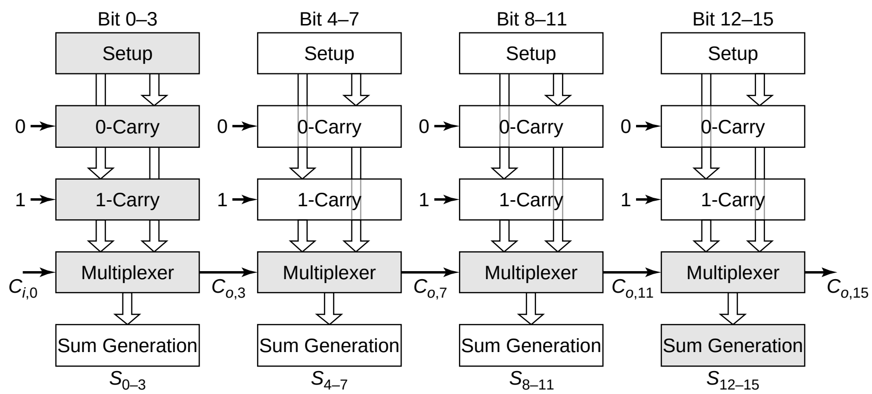
Carry Ripple versus Carry Bypass



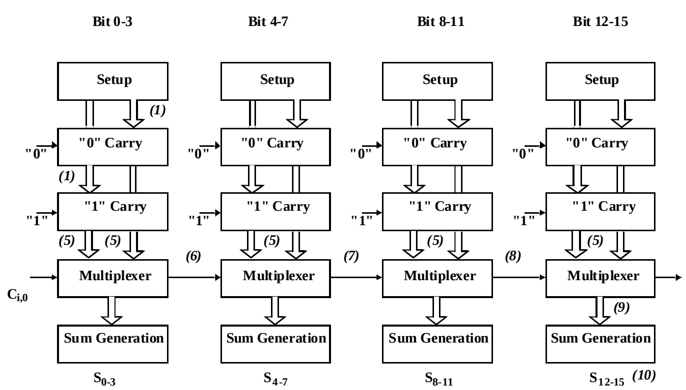
Carry-Select Adder



Carry Select Adder: Critical Path

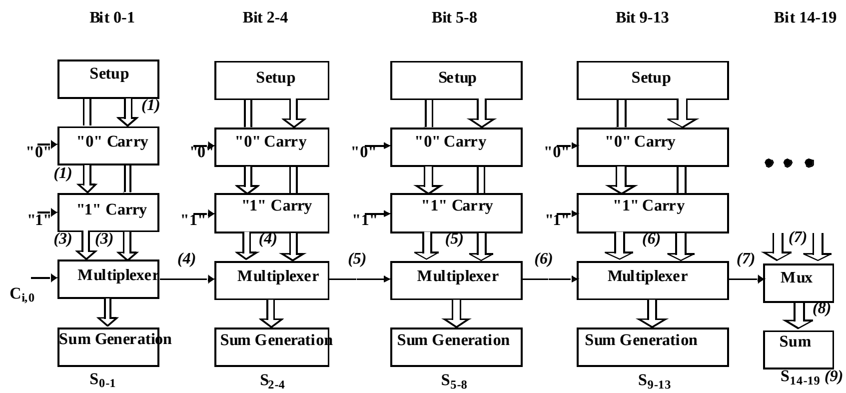


Linear Carry Select



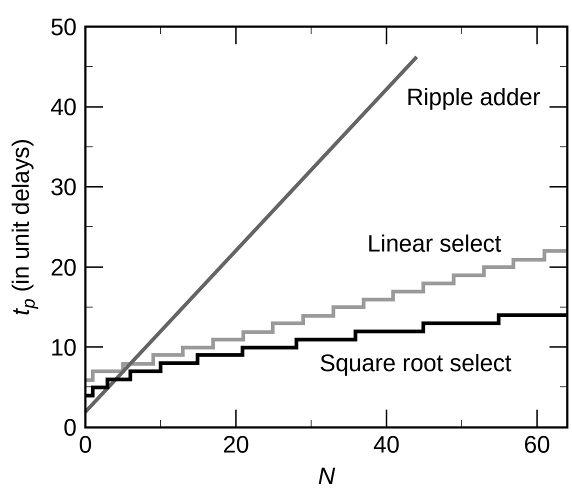
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

Square Root Carry Select

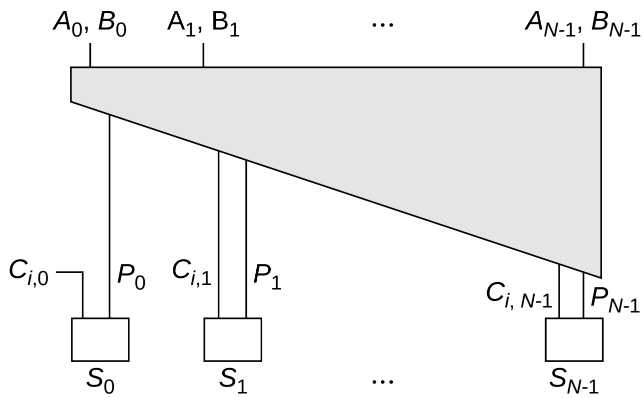


$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Adder Delays - Comparison



LookAhead - Basic Idea



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

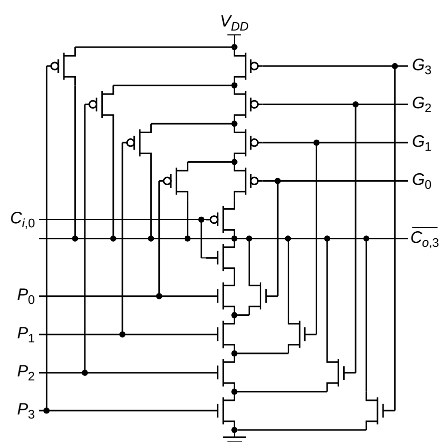
Look-Ahead: Topology

Expanding Lookahead equations:

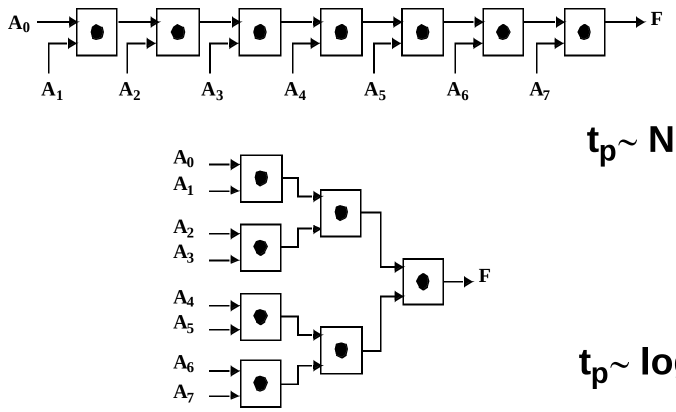
$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

All the way:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})))$$



Logarithmic Look-Ahead Adder



$t_p \sim N$

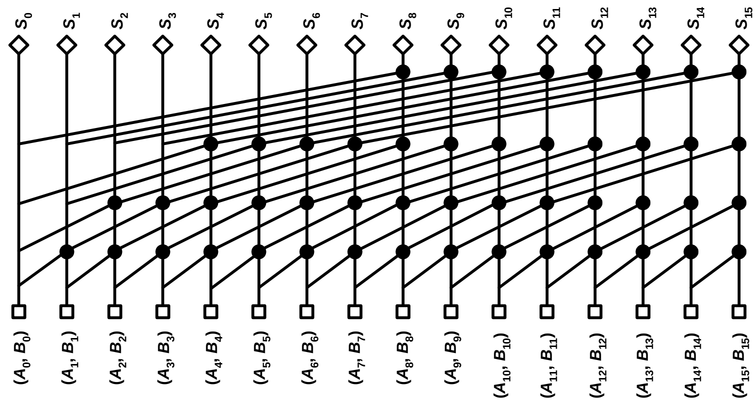
$t_p \sim \log_2(N)$

Carry Lookahead Trees

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

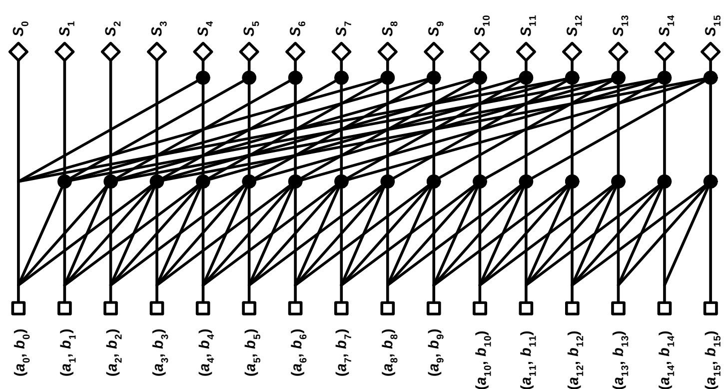
Can continue building the tree hierarchically.

Tree Adders



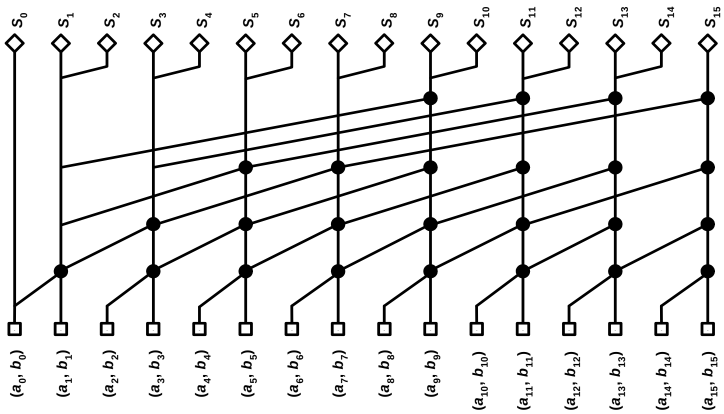
16-bit radix-2 Kogge-Stone tree

Tree Adders



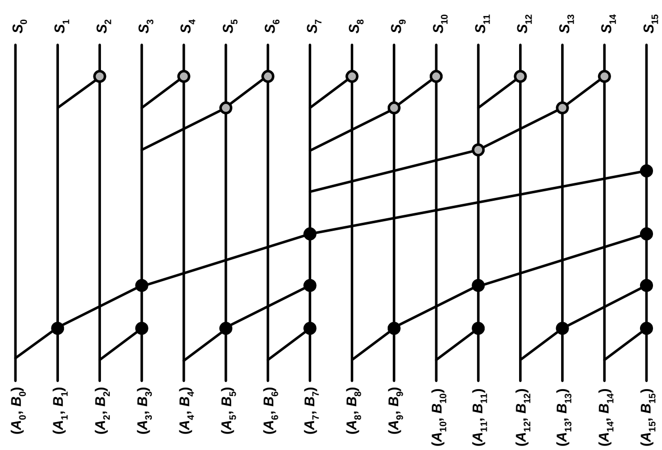
16-bit radix-4 Kogge-Stone Tree

Sparse Trees



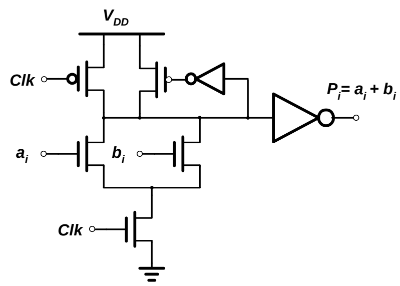
16-bit radix-2 sparse tree with sparseness of 2

Tree Adders

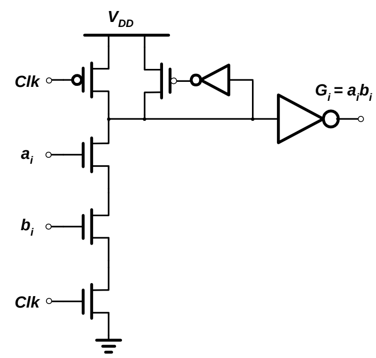


Brent-Kung Tree

Example: Domino Adder

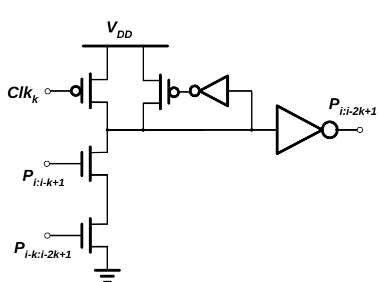


Propagate

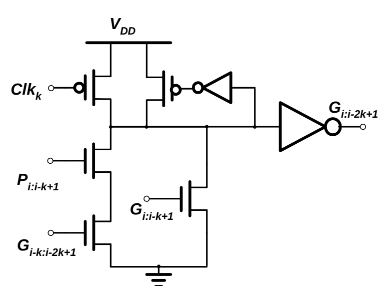


Generate

Example: Domino Adder

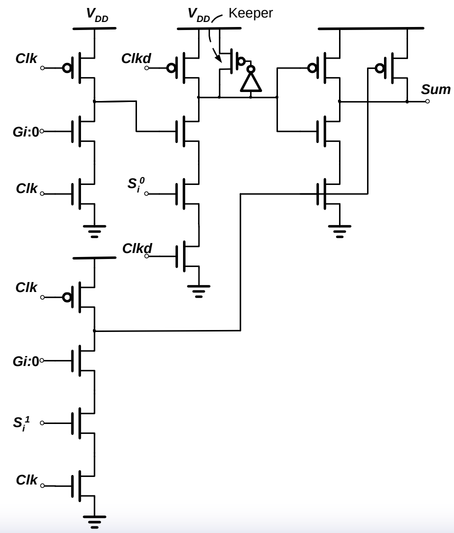


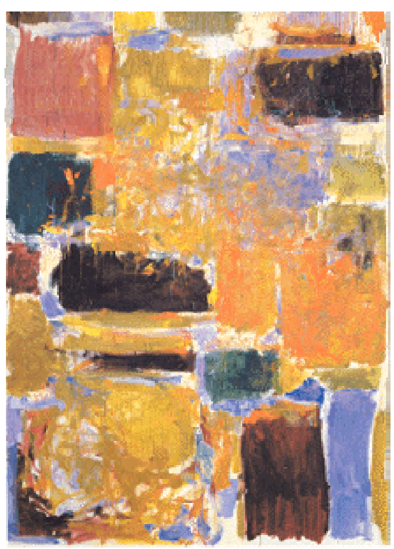
Propagate



Generate

Example: Domino Sum





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

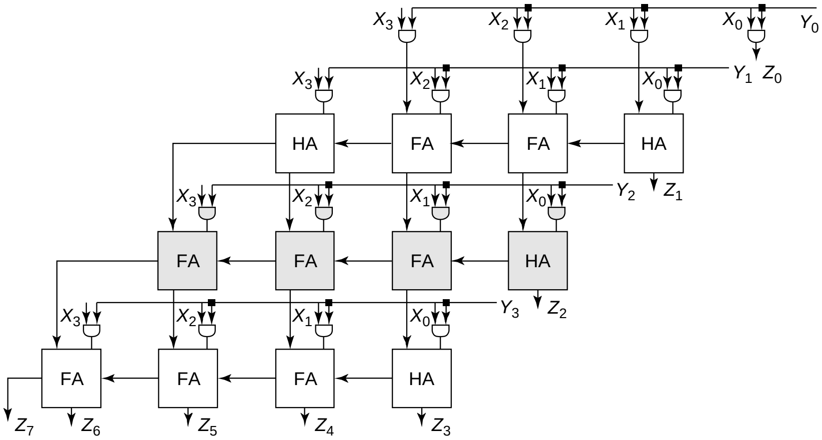
$$X = \sum_{i=0}^{M-1} X_i 2^i$$

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

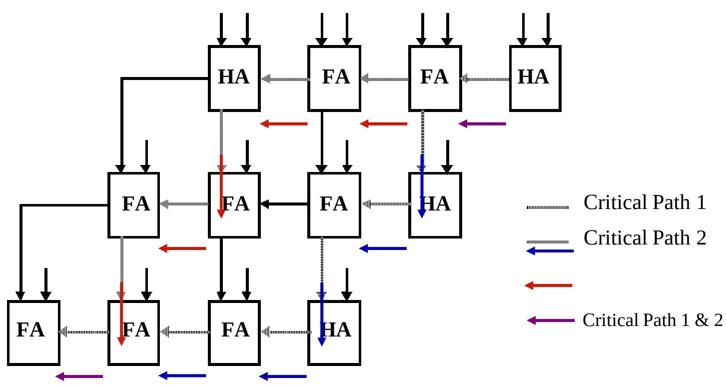
The Binary Multiplication

				1	0	1	0	1	0	Multiplicand
x						1	0	1	1	Multiplier
				1	0	1	0	1	0	Partial products
			1	0	1	0	1	0		
		0	0	0	0	0	0			
+	1	0	1	0	1	0				
	1	1	1	0	0	1	1	1	0	Result

The Array Multiplier

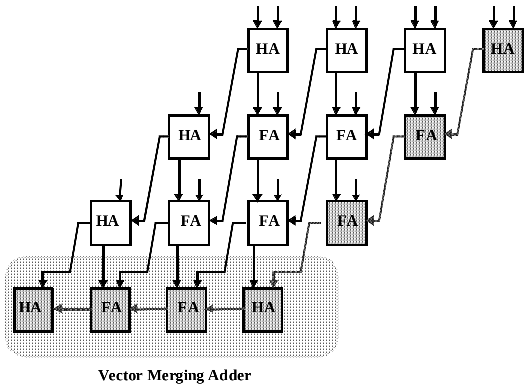


The MxN Array Multiplier — Critical Path



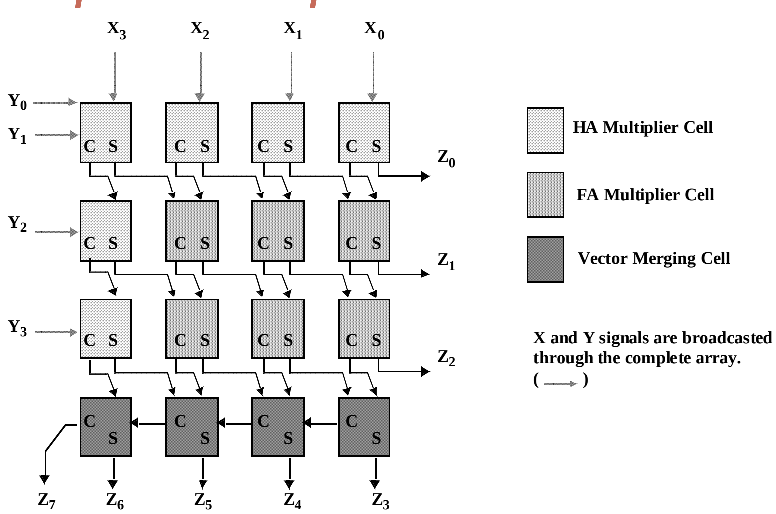
$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

Carry-Save Multiplier

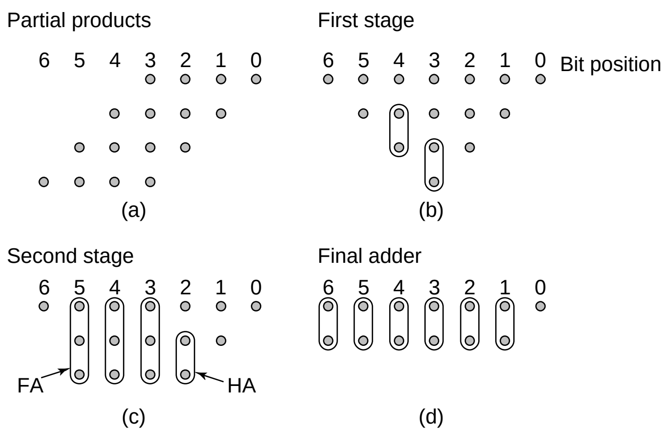


$$t_{mult} = (N-1)t_{carry} + (N-1)t_{and} + t_{merge}]$$

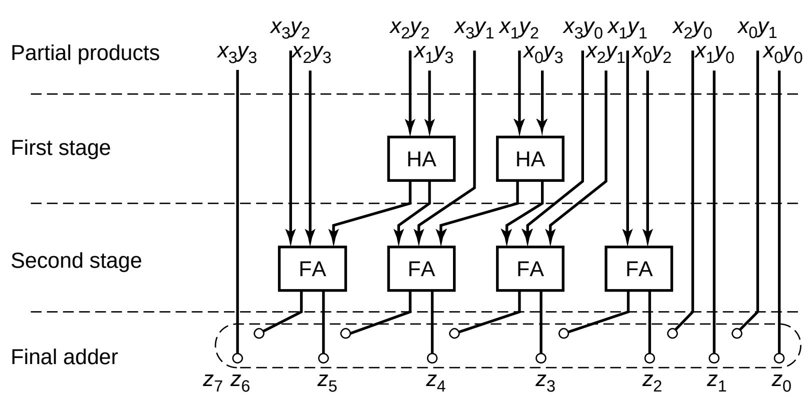
Multiplier Floorplan



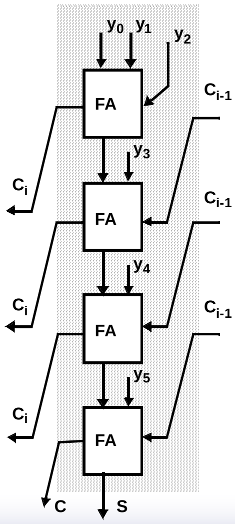
Wallace-Tree Multiplier

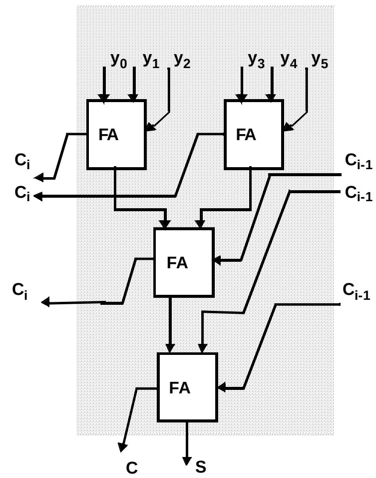


Wallace-Tree Multiplier



Wallace-Tree Multiplier

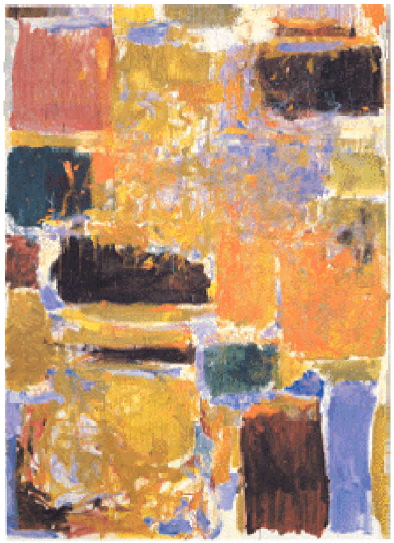




Multipliers —Summary

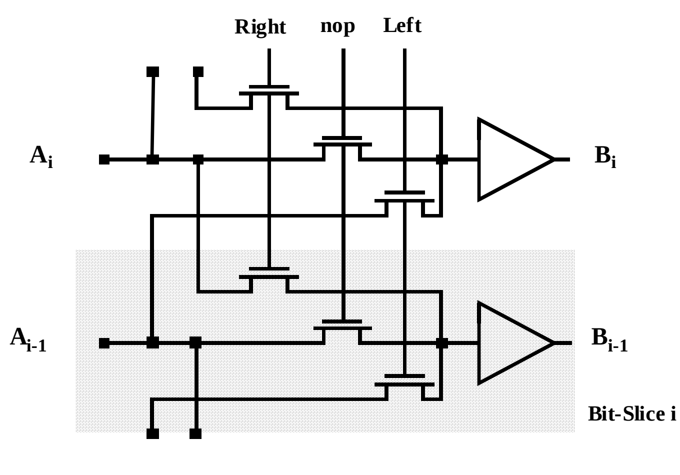
- **Optimization Goals Different Vs Binary Adder**
- **Once Again: Identify Critical Path**
- **Other possible techniques**
  - **- Logarithmic versus Linear (Wallace Tree Mult)**
  - **- Data encoding (Booth)**
  - **- Pipelining**

FIRST GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

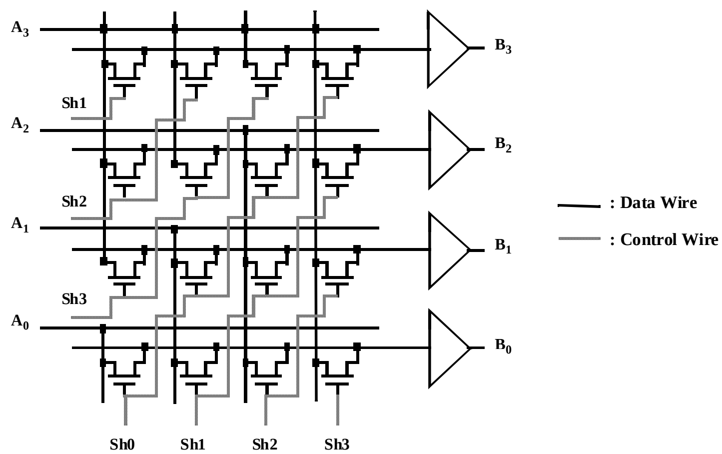


Shifters

The Binary Shifter

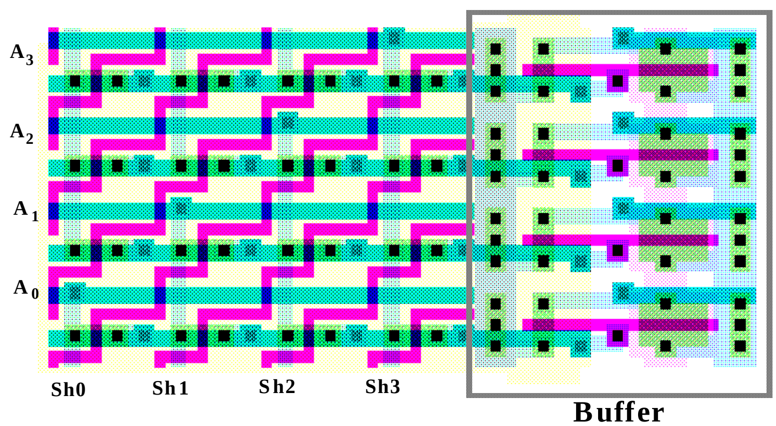


The Barrel Shifter



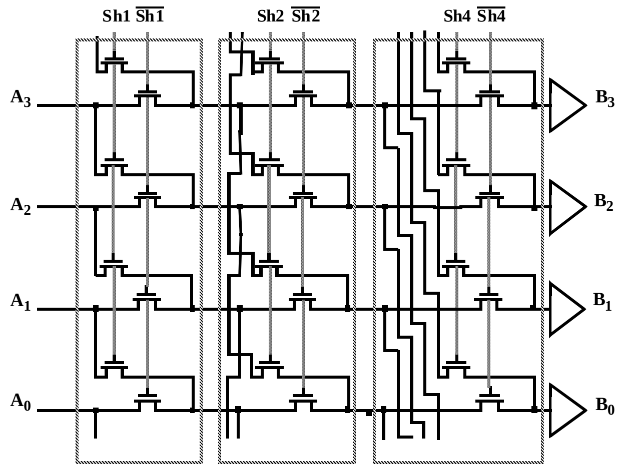
Area Dominated by Wiring

4x4 barrel shifter

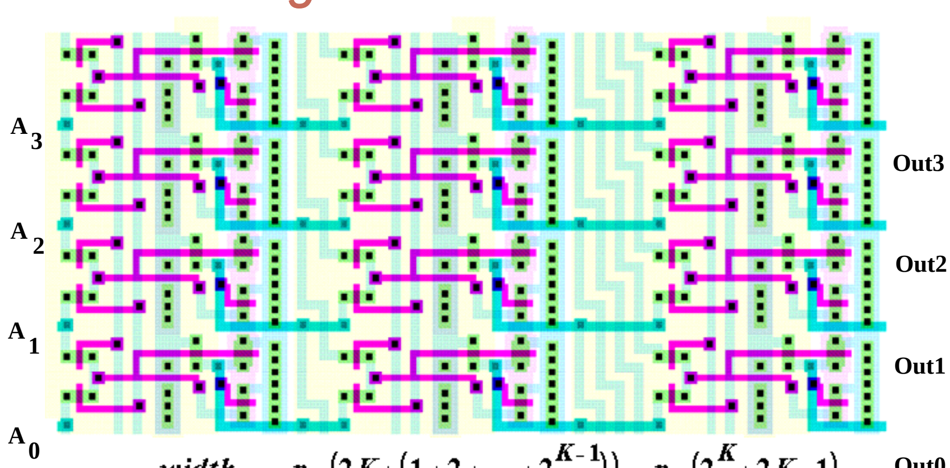


$\text{Width}_{barrel} \sim 2\, p_m\, M$

Logarithmic Shifter



0-7 bit Logarithmic Shifter



$$width_{\log} \approx p_m\left(2K + \left(1 + 2 + \ldots + 2^{K-1}\right)\right) = p_m\left(2^K + 2K - 1\right)$$