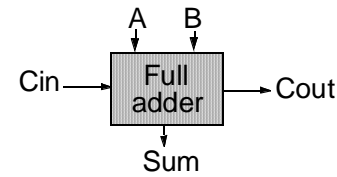


CSE/EE 462: VLSI Design Fall 2006 Adders and Multipliers

Jay Brockman

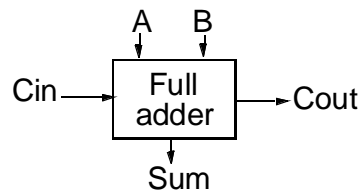
[Adapted from Mary Jane Irwin and Vijay Narananan, CSE Penn State
and Rabaey's *Digital Integrated Circuits*, ©2002, J. Rabaey et al.]

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



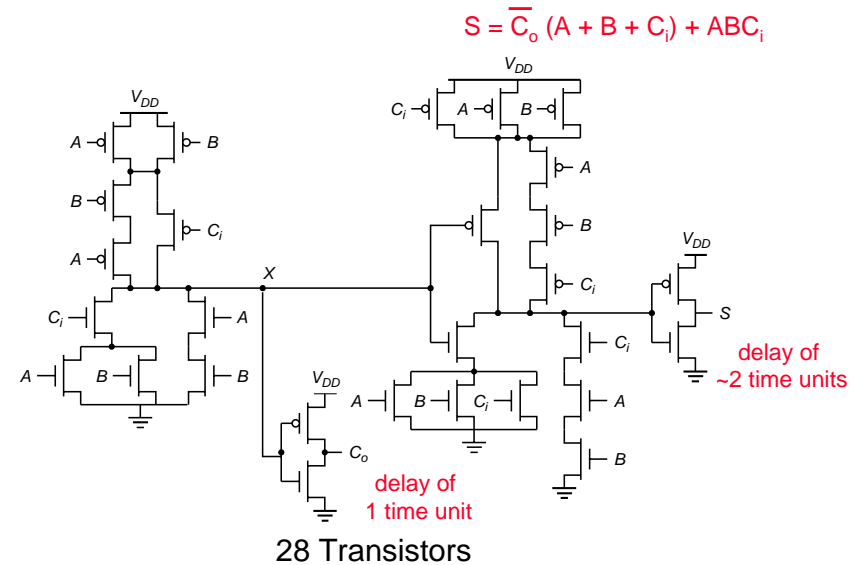
$$S = A \oplus B \oplus C_i$$

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

$$C_o = AB + BC_i + AC_i$$

$$S = \bar{C}_o (A + B + C_i) + ABC_i$$

Complimentary Static CMOS Full Adder



Verilog Model with Delay

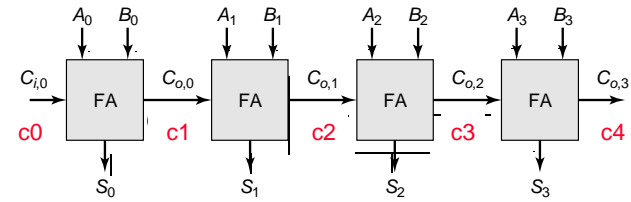
```
module fadd(sum, cout, a, b, cin);
  output sum, cout;
  input a, b, cin;

  assign #2 sum = a ^ b ^ cin;  xor
  assign #1 cout = a & b | b & cin | a & cin;
endmodule
```

after delay of 2 time unit

after delay of 1 time units

The Ripple-Carry Adder



Worst case delay linear with the number of bits

$$t_d = O(N)$$

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

Goal: Make the fastest possible carry path circuit

Verilog Model

```
module ripple4(sum, c4, c3, c2, c1, a, b, c0);
  output [3:0] sum;
  output c4, c3, c2, c1;
  input [3:0] a;
  input [3:0] b;
  input c0;

  fadd bit0(sum[0], c1, a[0], b[0], c0);
  fadd bit1(sum[1], c2, a[1], b[1], c1);
  fadd bit2(sum[2], c3, a[2], b[2], c2);
  fadd bit3(sum[3], c4, a[3], b[3], c3);
endmodule
```

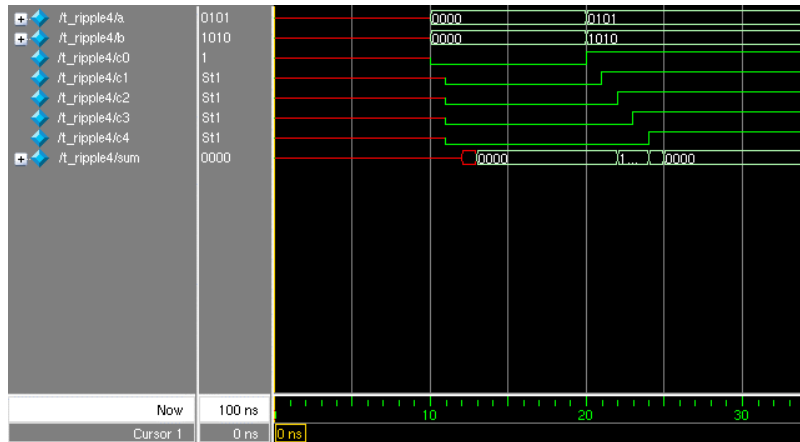
Verilog Testbench

```
module t_ripple4();
  wire [3:0] sum;
  wire c4, c3, c2, c1;
  reg [3:0] a;
  reg [3:0] b;
  reg c0;

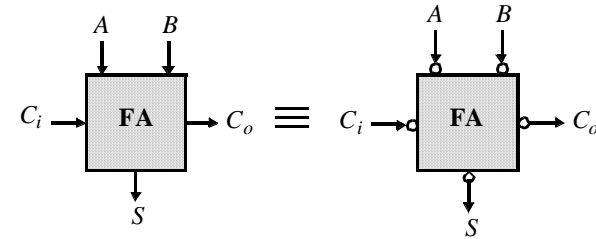
  ripple4 ripple4_1(sum, c4, c3, c2, c1, a, b, c0);

  initial begin #100 $finish; end
  initial begin
    #10 a = 4'b0000; b = 4'b0000; c0 = 0;
    #10 a = 4'b0101; b = 4'b1010; c0 = 1;
  end
endmodule
```

Modelsim Simulation Results



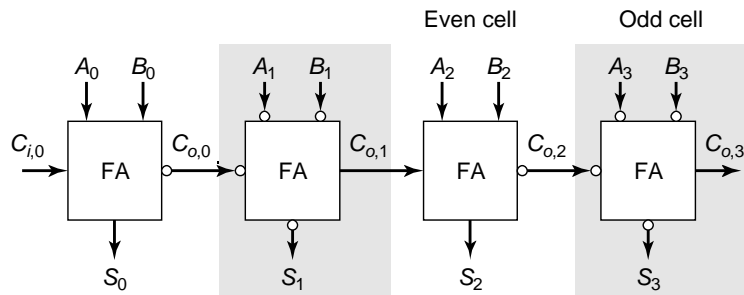
Inversion Property



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

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

Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

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 ⊕ B

Delete = $\bar{A} \bar{B}$

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

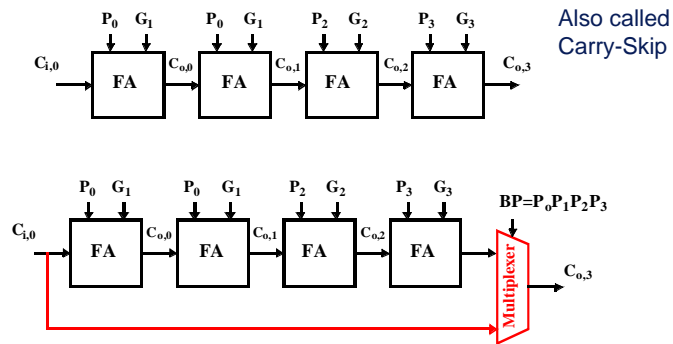
$$S(G, P) = P \oplus C_i$$

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

Note that we will be sometimes using an alternate definition for

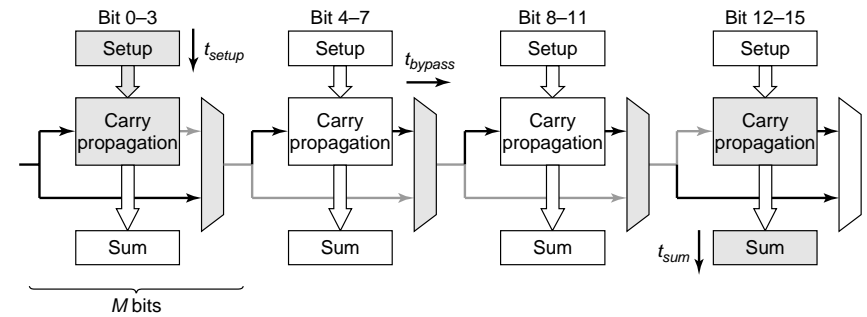
Propagate (P) = A + B

Carry-Bypass Adder



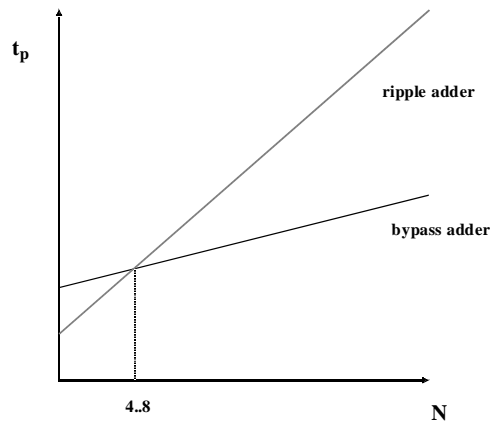
Idea: If (P_0 and P_1 and P_2 and $P_3 = 1$) then $C_{03} = C_0$, else "kill" or "generate".

Carry-Bypass Adder (cont.)



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

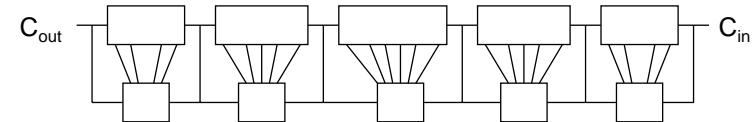
Carry Ripple versus Carry Bypass



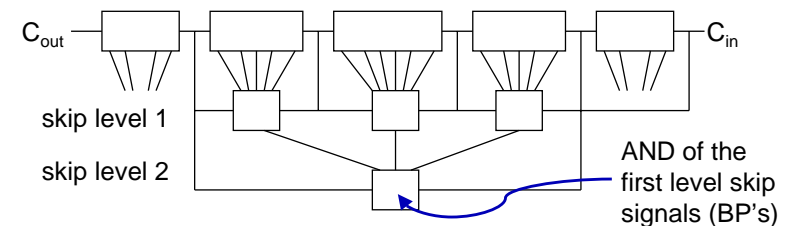
Carry-Skip Adder Extensions

Variable block sizes

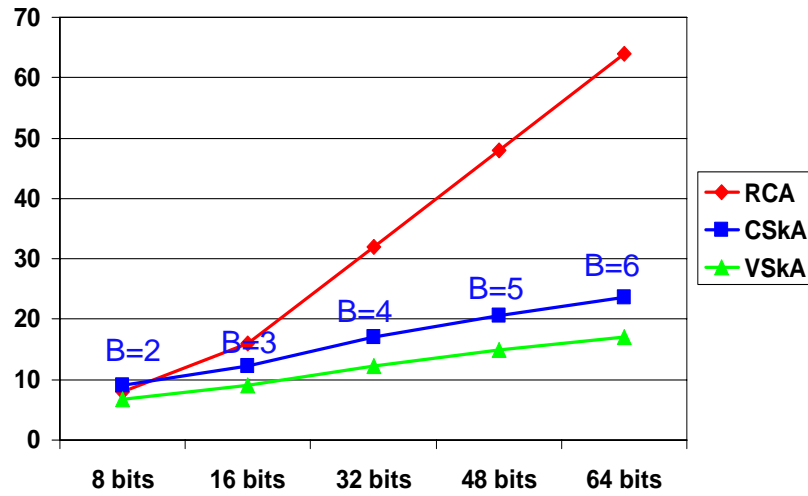
- A carry that is generated in, or absorbed by, one of the inner blocks travels a shorter distance through the skip blocks, so can have bigger blocks for the **inner carries** without increasing the overall delay



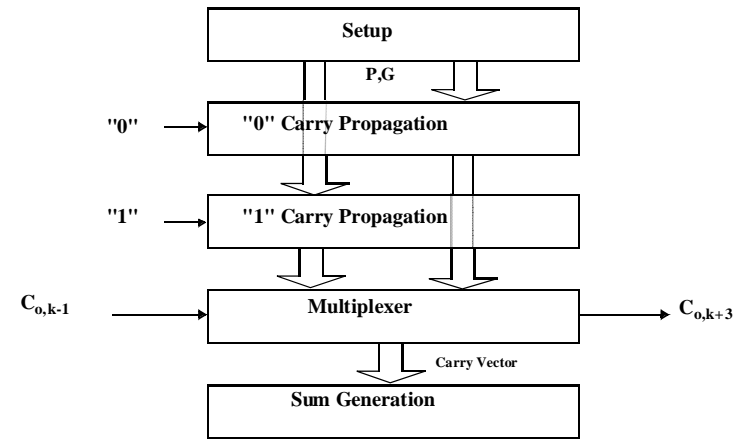
Multiple levels of skip logic



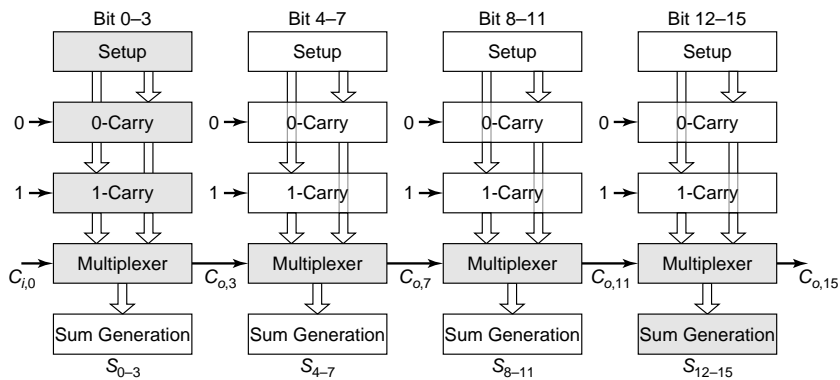
Carry-Skip Adder Comparisons



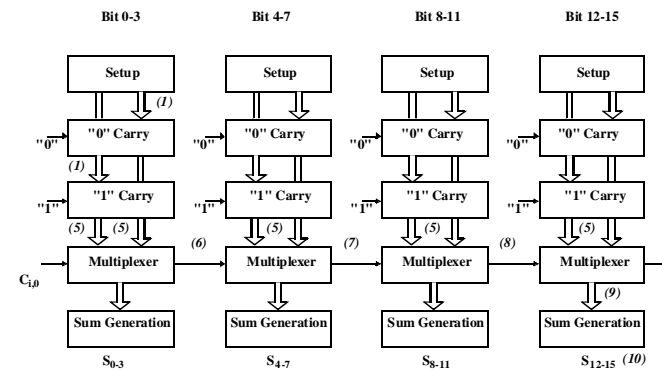
Carry-Select Adder



Carry Select Adder: Critical Path

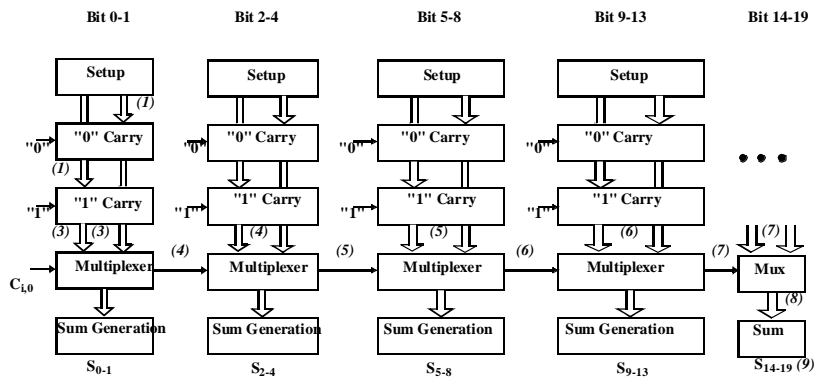


Linear Carry Select



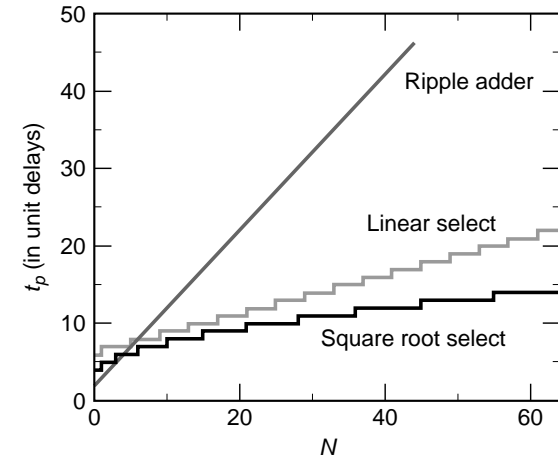
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right)t_{carry} + Mt_{mux} + t_{sum}$$

Square Root Carry Select

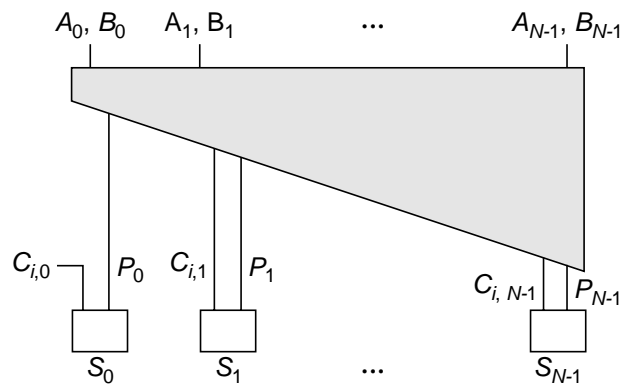


$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}$$

Adder Delays - Comparison



LookAhead - Basic Idea



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

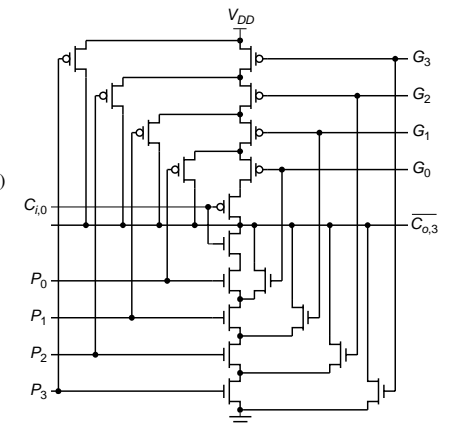
Look-Ahead: Topology

Expanding Lookahead equations:

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

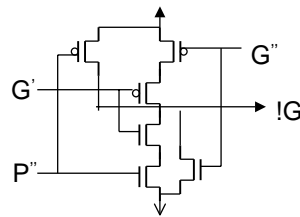
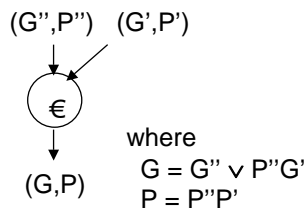
All the way:

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



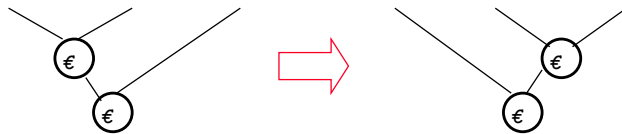
Parallel Prefix Adders (PPAs)

- Define carry operator ϵ on (G,P) signal pairs



- ϵ is associative, i.e.,

$$[(g''', p''') \epsilon (g'', p'')] \epsilon (g', p') = (g''', p''') \epsilon [(g'', p'') \epsilon (g', p')]$$

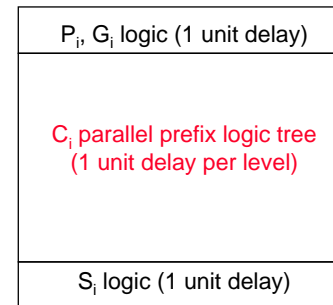


PPA General Structure

- Given P and G terms for each bit position, computing all the carries is equal to finding *all* the prefixes in parallel

$$(G_0, P_0) \epsilon (G_1, P_1) \epsilon (G_2, P_2) \epsilon \dots \epsilon (G_{N-2}, P_{N-2}) \epsilon (G_{N-1}, P_{N-1})$$

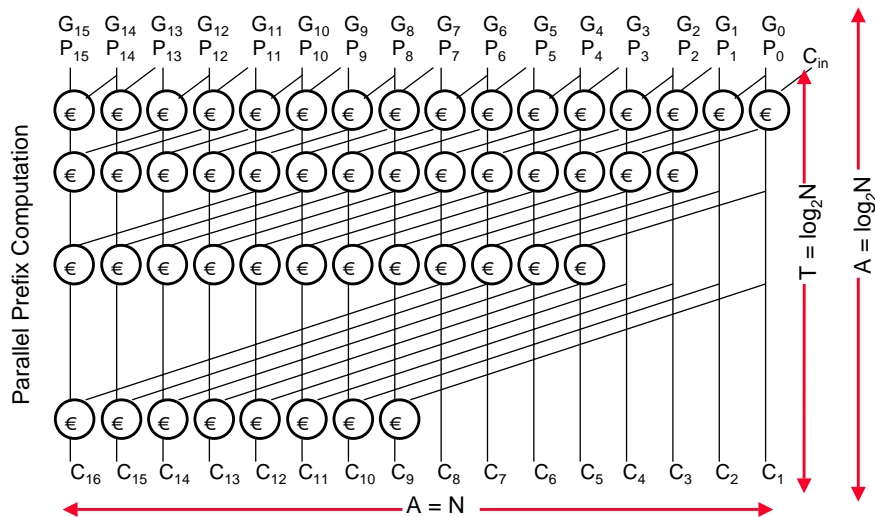
- Since ϵ is associative, we can group them in any order
 - but note that it is *not* commutative



- Measures to consider

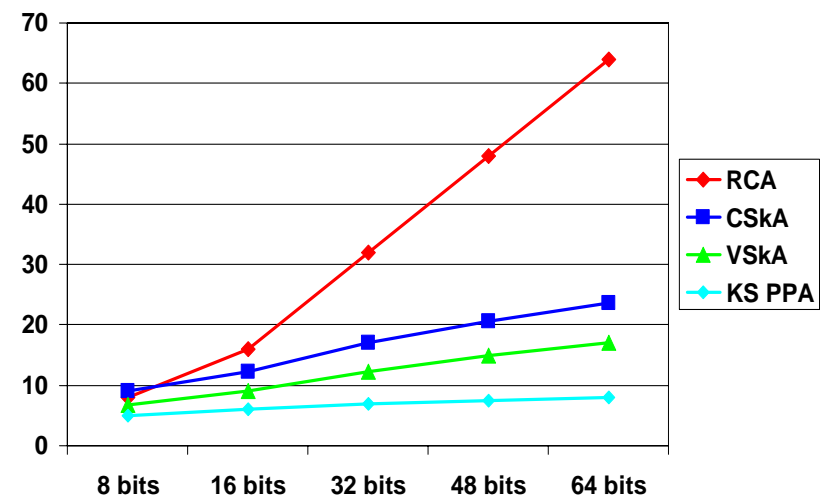
- number of ϵ cells
- tree cell depth (time)
- tree cell area
- cell fan-in and fan-out
- max wiring length
- wiring congestion
- delay path variation (glitching)

Kogge-Stone PPF Adder

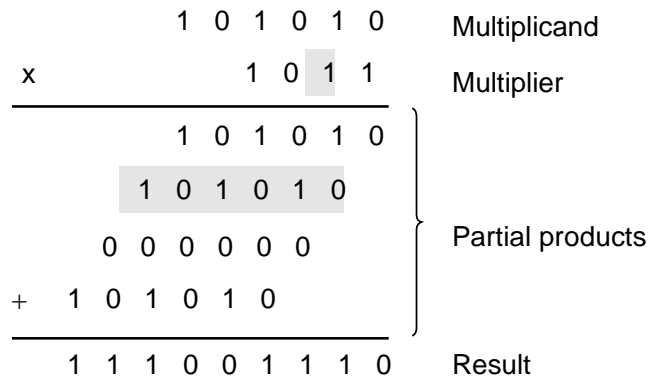


$$T_{add} = t_{setup} + \log_2 N t_{\epsilon} + t_{sum}$$

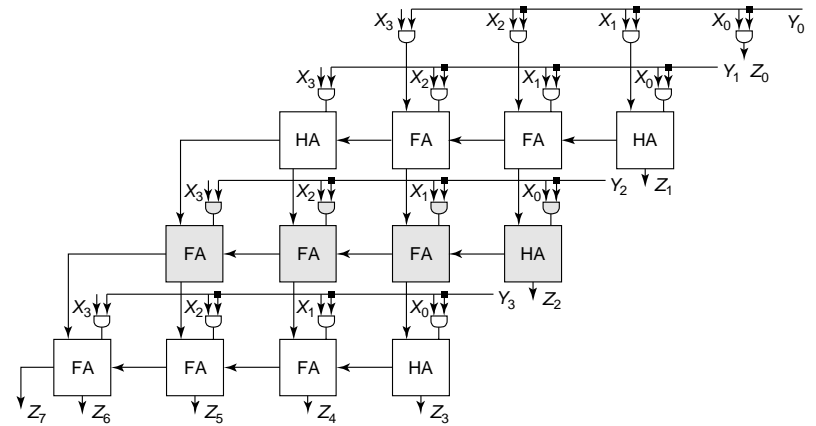
More Adder Comparisons



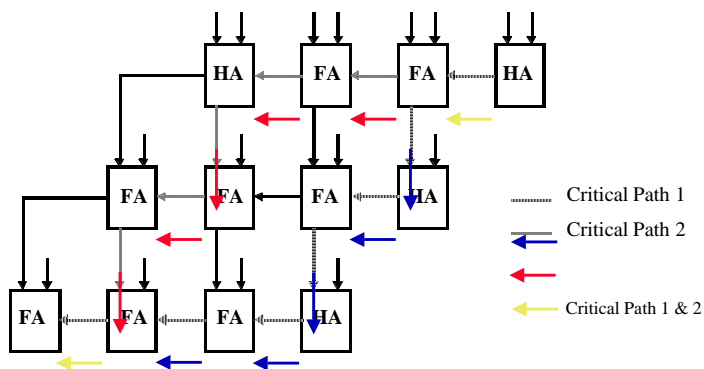
Binary Multiplication



The Array Multiplier

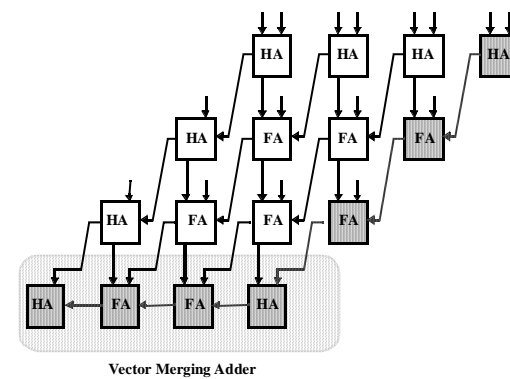


The MxN Array Multiplier— Critical Path



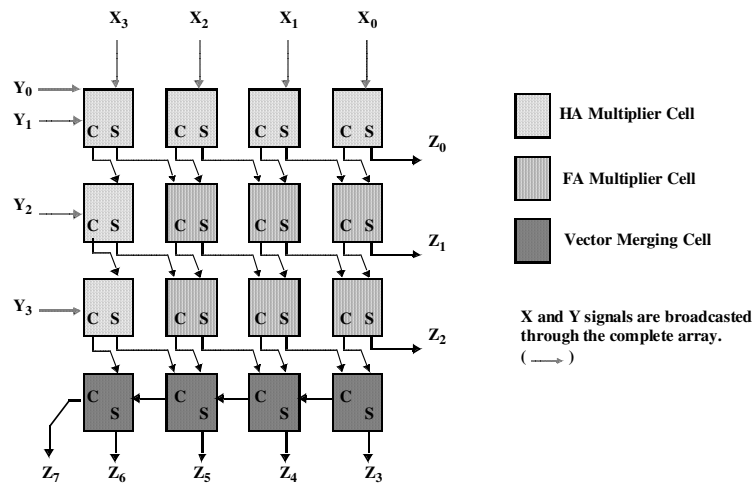
$$t_{mult} = [(M-1) + (N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

Carry-Save Multiplier



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

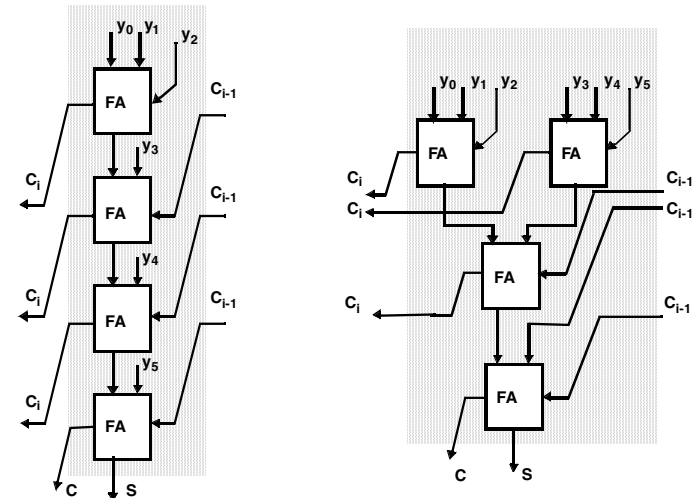
Multiplier Floorplan



CSE/EE 462 L04 Adders and Multipliers.33

Brockman, ND, 2006

Wallace-Tree Multiplier



CSE/EE 462 L04 Adders and Multipliers.34

Brockman, ND, 2006

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

CSE/EE 462 L04 Adders and Multipliers.35

Brockman, ND, 2006