EE141-Fall 2012 Digital Integrated Circuits

Lecture 20
Adders

Announcements

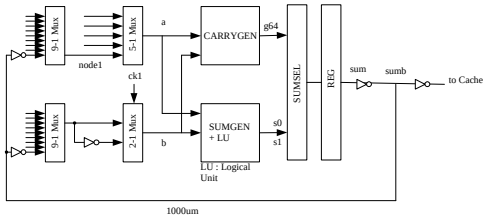
- Midterm 2: Thurs. Nov. 1st, 6:30-8:00pm, 60 Evans
  - Exam starts at 6:30pm sharp
  - Review session: Tues., Oct. 30th, 6pm, 550 Cory
- Project phase 2 out this Thurs., due next Fri.

Class Material

- Last lecture
  - Dynamic logic
- Today's lecture
  - Adders
- Reading
  - Chapter 11

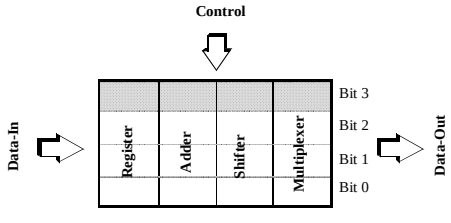
Adders

An Intel Microprocessor



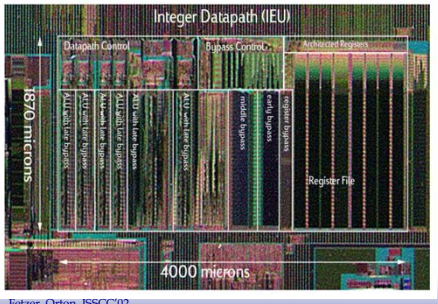
Itanium has 6 64-bit integer execution units like this

Bit-Sliced Design



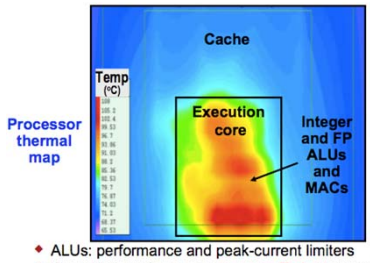
Tile identical processing elements

Itanium Integer Datapath



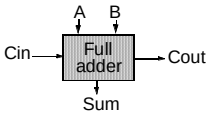
Fetzer, Orton, ISSCC'02

Data Paths Are Thermal Hogs



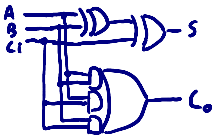
- ALUs: performance and peak-current limiters
- Goal: high-performance energy-efficient design

Full-Adder



A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	kill
0	0	1	1	0	kill
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



$$S = A \oplus B \oplus C_i$$
$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$
$$C_o = AB + BC_i + AC_i$$

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

Define 3 new variables which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Kill = $\bar{A}\,\bar{B}$

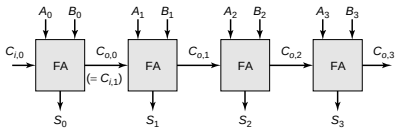
$$C_o(G,P) = G + PC_i$$
$$S(G,P) = P \oplus C_i$$

Can also derive expressions for *S* and C_o based on *K* and *P*

Note that we will sometimes use an alternate definition for

Propagate (P) = A + B

Simplest Adder: Ripple-Carry



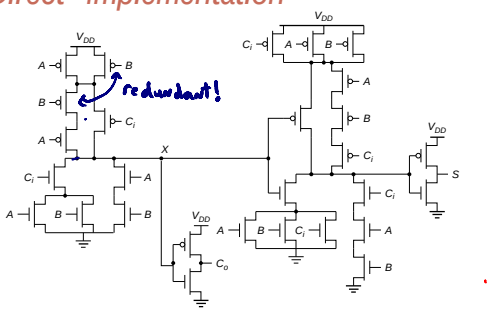
Worst case delay linear with the number of bits

$t_d = O(N)$

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

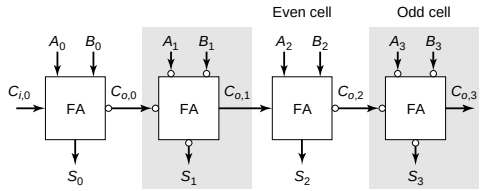
Goal: Make the fastest possible carry path circuit

Complementary Static CMOS Full Adder: "Direct" Implementation



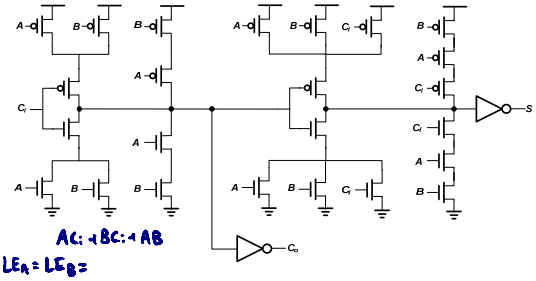
28 Transistors

Minimize Critical Path by Reducing Inverting Stages



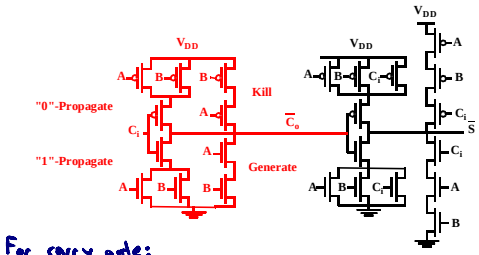
Exploit Inversion Property

Complementary Static CMOS Full Adder



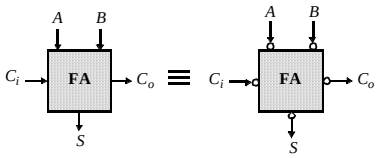
28 Transistors

A Better Structure: The Mirror Adder



24 transistors

Inversion Property



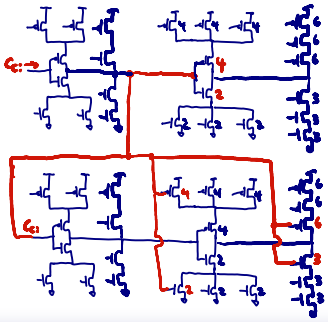
$$\overline{S}(A,B,C_i) = S(\overline{A},\overline{B},\overline{C_i})$$

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

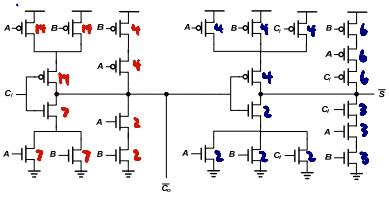
Sizing the Mirror Adder: Fanout

- Since LE of carry gate is 2, want *f* of 2 to get EF of 4
- Use min. size sum gates to reduce load on carry.
- Total load on carry gate is:

$C_{load} = C_{Ci} + (6+6+9)$
$C_{load} = 2C_{Ci}$



Sizing the Mirror Adder



- $C_{load} = C_{Ci} + (6+6+9) = 2C_{Ci}$

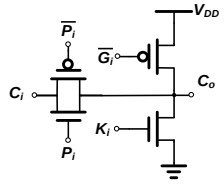
  → $C_{Ci} = 21$

- Minimum size G and K stacks to reduce diffusion loading

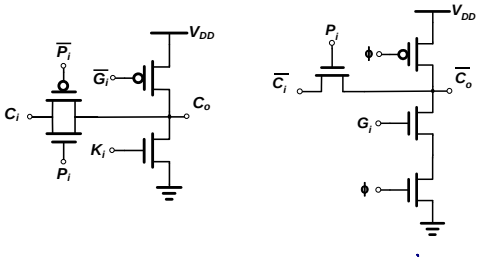
Mirror Adder Summary

- **The NMOS and PMOS chains are completely symmetrical. Maximum of two series transistors in the carry-generation gate.**
- **When laying out the cell, the most critical issue is the minimization of the capacitance at node C_o. Reduction of the diffusion capacitances is particularly important.**
- **Carry signals are critical - transistors connected to C_i are placed closest to the output.**
- **Only the transistors in the (propagate) carry chain have to be optimized for speed. All transistors in the sum stage can be minimal size.**

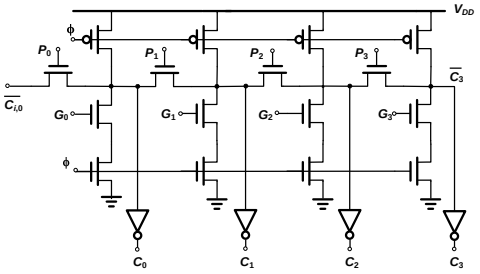
Manchester Carry Chain



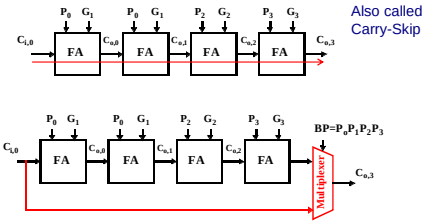
Dynamic Manchester Carry Chain



Manchester Carry Chain

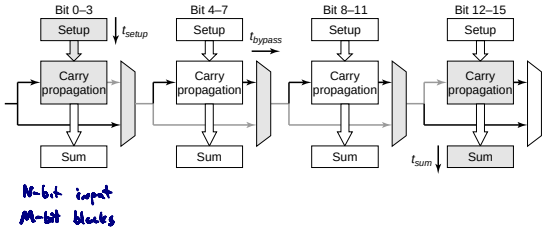


Carry-Bypass Adder



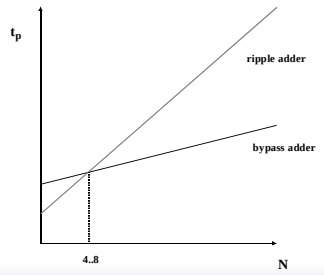
Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{o3} = C_0$, else "kill" or "generate".

Carry-Bypass Adder (cont.)

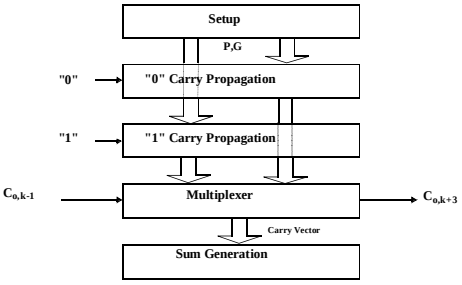


$$t_{adder} = t_{setup} + (M\text{-}1)t_{carry} + (N/M\text{-}1)t_{bypass} + (M\text{-}1)t_{carry} + t_{sum}$$

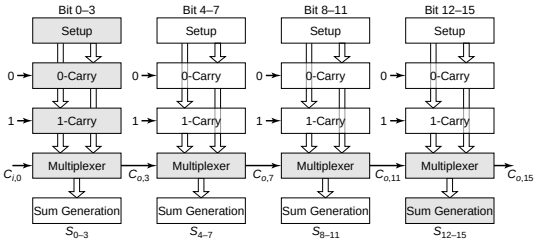
Carry Ripple versus Carry Bypass



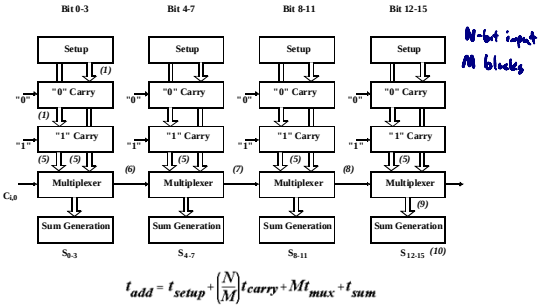
Carry-Select Adder



Carry Select Adder: Critical Path

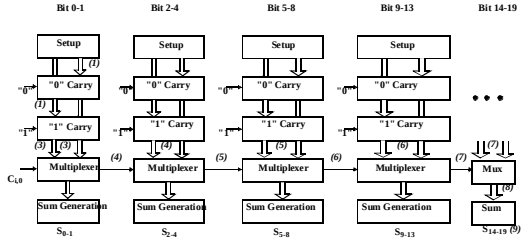


Linear Carry Select



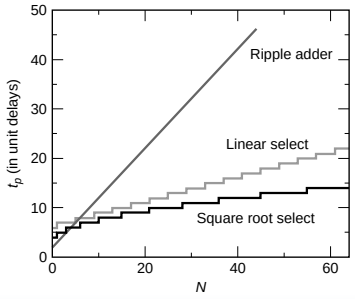
$$t_{add} = t_{setup} + \left(\frac{N}{M}\right)t_{carry} + Mt_{mux} + t_{sum}$$

Square Root Carry Select



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

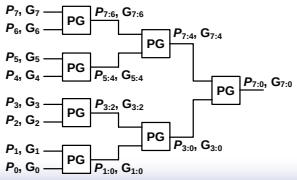
Adder Delays - Comparison



Logarithmic (Tree) Adders – Basic Idea

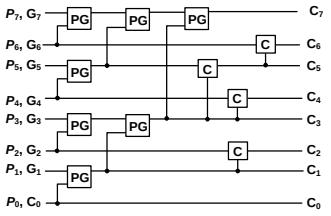
- "Look ahead" across groups of multiple bits to figure out the carry
  - Example with two bit groups:

    $P_{1:0} = P_1 \cdot P_0$, $G_{1:0} = G_1 + P_1 \cdot G_0$, → $C_{out1} = G_{1:0} + P_{1:0} \cdot C_{in}$
- Combine these groups in a tree structure:
  - Delay is now ~$\log_2$(N)
  - Instead of ~N



Rest of the Tree

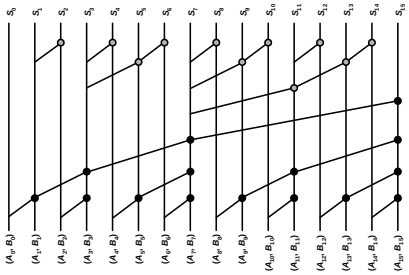
- Previous picture shows only half of the algorithm
  - Need to generate carries at individual bit positions too



Many Kinds of Tree Adders

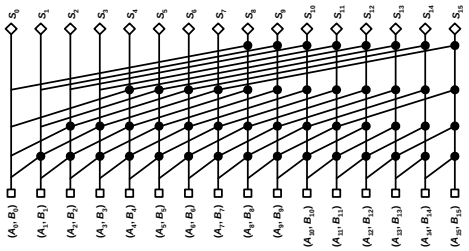
- Many ways to construct these tree (or "carry lookahead") adders
  - Many of these variations named after the people who first came up with them
- Most of these vary three basic parameters:
  - Radix: how many bits are combined in each PG gate
    - Previous example was radix 2; often go up to radix 4
  - Tree Depth: how many stages of logic you go through to get the final carry. Must be at least $\log_{Radix}$(N)
  - Fanout: Maximum logical branching in the tree

Tree Adders



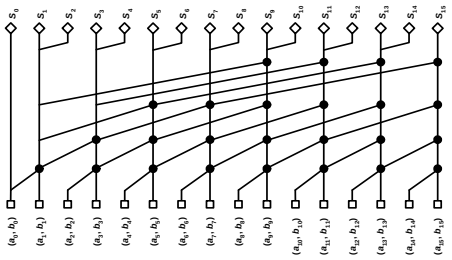
Brent-Kung Tree

Tree Adders



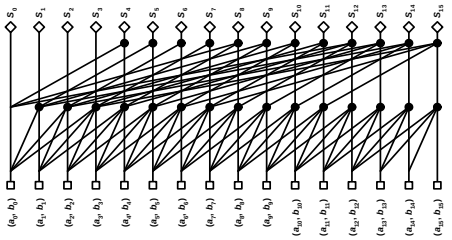
16-bit radix-2 Kogge-Stone tree

Tree Adders



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

Tree Adders



16-bit radix-4 Kogge-Stone Tree

Next Lecture

- Domino Logic