Lecture 19
Adders

Announcements

- Midterm 2: Thurs. Nov. 6\textsuperscript{th}, 6:30-8:00pm, 277 Cory
  - Exam starts at 6:30pm sharp
  - Review session: Tues., Nov. 4\textsuperscript{th}, 6:30-7:30pm, Hogan room (521 Cory)

- Project phase 2 out this Thurs., due next Thurs.
Class Material

- Last lecture
  - Ratioed, pass transistor 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
Bit-Sliced Datapath

From register files / Cache / Bypass

- Multiplexers
- Shifter
- Adder stage 1
- Wiring
- Adder stage 2
- Wiring
- Adder stage 3
- Sum Select

To register files / Cache

Itanium Integer Datapath

Fetzer, Orton, ISSCC’02
**Full-Adder**

The binary adder can be represented as:

\[
A \oplus B \oplus C_i = S \\
\overline{A} \overline{B} \overline{C_i} + \overline{A} \overline{B} C_i + A \overline{B} C_i + A B C_i
\]

Where:
- \( S \) = Sum
- \( C_o \) = Carry out
- \( C_i \) = Carry in
- \( A \), \( B \) = Inputs

**The Binary Adder**

\[
S = A \oplus B \oplus C_i \\
\overline{A} \overline{B} \overline{C_i} + \overline{A} \overline{B} C_i + A \overline{B} C_i + A B C_i \\
C_o = A B + B C_i + A C_i
\]
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 ($P$) = $A \oplus B$
- Kill = $A \cdot B$

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

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

\[
t_d = O(N) \\
t_{adder} = (N-1)t_{carry} + t_{sum}
\]

Goal: Make the fastest possible carry path circuit
Complementary Static CMOS Full Adder: “Direct” Implementation

Complementary Static CMOS Full Adder

28 Transistors
**Inversion Property**

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

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

**Minimize Critical Path by Reducing Inverting Stages**

Exploit Inversion Property
A Better Structure: The Mirror Adder

For carry gate:
LE$_C$: 2

24 transistors

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:

$$C_{\text{load}} = C_C + (6+6+9) = 2C_C$$
Sizing the Mirror Adder

\[ C_{\text{load}} = C_{\text{Cl}} + (6+6+9) = 2C_{\text{Cl}} \]
\[ \Rightarrow C_{\text{Cl}} = 21 \]

- 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.
Transmission Gate Full Adder

\[ P = A \oplus B \]

\[ S = \begin{cases} C_i & \text{if } P = 0 \\ \overline{C}_i & \text{if } P = 1 \end{cases} \]

\[ S = P \oplus C_i \]

Sum Generation

Carry Generation

\[ C_o = A \cdot B + P \cdot C_i \]

\[ C_o = \overline{P} \cdot A + P \cdot C_i \]

Manchester Carry Chain

\[ \overline{P}_i \]

\[ C_i \]

\[ \overline{C}_i \]

\[ K_i \]

\[ V_{DD} \]
Carry-Bypass Adder

Idea: If (P0 and P1 and P2 and P3 = 1) then C_{03} = C_0, else “kill” or “generate”.

Carry-Bypass Adder (cont.)

\[ t_{adder} = t_{setup} + (M-1)t_{carry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum} \]
Carry Ripple versus Carry Bypass

Carry-Select Adder

Setup

"0" "0" Carry Propagation

"1" "1" Carry Propagation

C_{n,k-1} Multiplexer

C_{n,k+3} Carry Vector

Sum Generation
**Carry Select Adder: Critical Path**

![Diagram of Carry Select Adder: Critical Path]

**Linear Carry Select**

![Diagram of Linear Carry Select]

Mathematical expression:

\[ t_{add} = t_{setup} + \frac{N}{M} t_{carry} + M t_{max} + t_{sum} \]
**Square Root Carry Select**

![Diagram](image)

\[ t_{add} = t_{setup} + M_{carry} + \sqrt{N} t_{mux} + t_{sum} \]

**Adder Delays - Comparison**

![Graph](image)
Logarithmic (Tree) Adders – Basic Idea

- “Look ahead” across groups of multiple bits to figure out the carry
  - Example with two bit groups:
    \[ P_{1:0} = P_1 \cdot P_0, \ G_{1:0} = G_1 + P_1 \cdot G_0, \Rightarrow C_{out1} = G_{1:0} + P_{1:0} \cdot C_{in} \]

- 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 $\log_{\text{radix}}(N)$
  - Fanout: Maximum logical branching in the tree

Tree Adders

Brent-Kung Tree
**Tree Adders**

16-bit radix-2 Kogge-Stone tree

16-bit radix-2 sparse tree with sparseness of 2
**Tree Adders**

16-bit radix-4 Kogge-Stone Tree

---

**Next Lecture**

- Dynamic Logic