EE241 - Spring 2005
Advanced Digital Integrated Circuits

Lecture 22: Adders

Clock Generation

Delay-Locked Loop (Delay Line Based)

Phase Det U D Charge Pump

\[ f_{\text{REF}} \]

DL

\[ f_o \]

Phase-Locked Loop (VCO-Based)

\[ f_{\text{REF}} \]

PD U D CP

VCO

\[ f_o \]
Phase-Locked Loop Based Clock Generator

Acts also as Clock Multiplier

Loop Components

- **Phase Comparator**
  - Produces UP/DN pulses corresponding to phase difference
- **Charge Pump**
  - Sources/sinks current for duration of UP/DN pulses
- **Loop Filter**
  - Integrates current to produce control voltage
- **Voltage-Controlled Delay Line**
  - Changes delay proportionally to voltage
- **Voltage-Controlled Oscillator**
  - Generates frequency proportional to control voltage
PLL Jitter

DLL Locking

Diagram:

Clock Deskewing

Two clock spines, two DLLs, and a PD that controls them

Clock Ring

Clocks routed in parallel, opposite directions
LCG aligns to the middle

Geannopoulos, ISSCC’98

Shibayama, ISSCC’98
Synchronous Distributed Oscillators

Mizuno, ISSCC’98

Distributed PLLs

Gutnik, ISSCC’2000
Intel Itanium™

Global Clock Distribution

- Balanced H-tree routed in M5 and M6
- Lateral shielding
- Distributes both main and reference clock
- Optimized to account for inductive effects

Regional Clock Distribution

- Distributed array of deskew buffers to reduce process related skew
  - 8 deskew clusters each holding up to 4 buffers
- Regional clock grids driven by modular Regional Clock Drivers
  - M4-M5 grid tailored for the clock load density of the underlying block
  - Full support for scan and clock gating

= Cluster of 4 deskew buffers
= Central Deskew Controller

Rusu, ISSCC’2000
Arithmetic Circuits

- Chapter 11, Rabaey, 2nd ed.
- Selected journal publications
- Books:
  - High-Speed VLSI Arithmetic Units: Adders and Multipliers, by V. Oklobdzija in Chandrakasan et al.
### Full Adder

A | B | Ci | S | Co  | 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 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
The Mirror Adder

Minimize inversions

Mirror Adder Cell
Sizing Mirror Adder

Full Adder Implementation

Standard CMOS

Multiplexer-based
Manchester Carry Chain

Static

Dynamic

- Implement P with pass-transistors
- Implement G with pull-up, kill (delete) with pull-down
- Use dynamic logic to reduce the complexity and speed up

Sizing Manchester Carry Chain

Delay equation

\[ t_p = 0.69 \sum_{i=1}^{N} C_i \left( \sum_{j=1}^{i} R_j \right) = 0.69 \frac{N(N+1)}{2} RC \]

Delay is quadratic with N

- Progressive sizing should help?
Sizing Manchester Carry Chain

- **Stick Diagram**

  $C_{\text{fix}}$ – fixed capacitance at the node (pull-down, pull-up diffusions, metal, + inverter ~15fF)
  $C \sim 2\text{fF/\mu m}$
  $R \sim 10k\Omega \, \mu m$
  When $CW > C_{\text{fix}}$
  small improvements with sizing.
  Loading of the input stage

  \[
  t_p = 0.69 \frac{N(N+1)}{2} RC = 0.69 \frac{N(N+1)}{2} \frac{R}{W} (C_{\text{fix}} + C \cdot W)
  \]

Manchester Carry Chain

- Length of chain is limited to $k = 4$-8
- Standard solution – add inverters
- The overall $N$bit adder delay is a sum of $Nk$ segments (linear)
Carry-Skip Adder

Idea: If (P0 and P1 and P2 and P3 = 1) then $C_{30} = C_0$, else “kill” or “generate”.

MacSorley, Proc IRE 1/61
Lehman, Burla, IRE Trans on Comp, 12/61

Carry-Skip Adder

Critical Path

For N-bit adder with k-bit groups

$$t_d = (k - 1)t_{RCA} + \left( \frac{N}{k} - 2 \right)t_{SKIP} + (k - 1)t_{RCA}$$
Carry-Skip Adder

Critical path delay with constant groups

\[ t_d = 2(k - 1)t_{RCA} + \left( \frac{N}{k} - 2 \right)t_{SKIP} \]
Carry-Skip Adder

Variable Group Length

\[ x_1 = x_4 = 4, \ x_2 = x_7 = 7, \ x_3 = x_8 = 10, \ x_5 = x_6 = 11. \]

\[ t_d = c_1 + \sqrt{c_2 N + c_3} \]

Oklobdzija, Barnes, Arith’85

---

Carry-Skip Adder

![Carry-Skip Adder Diagram]

Carry-Skip Adder

Variable Block Lengths

Manchester Chain with Carry-Skip

Delay model:
Carry-Select Adder

Carry Select Adder: Critical Path

<table>
<thead>
<tr>
<th>Bit 0-3</th>
<th>Bit 4-7</th>
<th>Bit 8-11</th>
<th>Bit 12-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>&quot;0&quot; Carry</td>
<td>&quot;0&quot; Carry</td>
<td>&quot;0&quot; Carry</td>
</tr>
<tr>
<td>&quot;1&quot; Carry</td>
<td>&quot;1&quot; Carry</td>
<td>&quot;1&quot; Carry</td>
<td>&quot;1&quot; Carry</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>Multiplexer</td>
<td>Multiplexer</td>
<td>Multiplexer</td>
</tr>
<tr>
<td>C_{n-3}</td>
<td>C_{n-3}</td>
<td>C_{n-3}</td>
<td>C_{n-3}</td>
</tr>
<tr>
<td>Sum Generation</td>
<td>Sum Generation</td>
<td>Sum Generation</td>
<td>Sum Generation</td>
</tr>
<tr>
<td>S_{k-3}</td>
<td>S_{k-7}</td>
<td>S_{k-11}</td>
<td>S_{k-15}</td>
</tr>
</tbody>
</table>
Linear Carry Select

$$t_{\text{add}} = t_{\text{setup}} + \frac{N}{M} t_{\text{carry}} + M_{\text{max}} + t_{\text{sum}}$$

Square Root Carry Select

$$t_{\text{add}} = t_{\text{setup}} + P t_{\text{carry}} + \sqrt{2N} t_{\text{mux}} + t_{\text{sum}}$$
Conditional Sum Adders

$C_{\text{select-\:add}}(k) = 3C_{\text{add}}(k/2) + k/2 + 1$

$T_{\text{select-\:add}}(k) = T_{\text{add}}(k/2) + 1$

---

Sklansky,
Trans on Comp
6/60

---

Conditional Sum Adders

$x = 10111011010101101101$

$y = 000111001101101101$

$s_i^0 = x_i \oplus y_i$

$s_i^1 = \overline{x_i} \oplus y_i$

$c_i^0 = x_i \cdot y_i$

$c_i^1 = x_i + y_i$

Sklansky,
Trans on Comp
6/60
Two Level Carry-Select Adder

Conditional Sum Adders
TG Conditional Sum

- Serial connection of transmission gates
- Chain length $= 1 + \log_2 n$

Rothermel, JSSC 89
DPL Conditional Sum

CLA

“Conditional carry select”

Carry-Lookahead Adders

- Adder trees
  - Radix of a tree
  - Minimum depth trees
  - Sparse trees
- Logic manipulations
  - Conventional vs. Ling
  - Stack height limiting
Propagate and Generate Signals

Define 3 new variables that ONLY depend on $a_i$, $b_i$

*Generate* ($g_i$) = $a_i b_i$

*Propagate* ($p_i$) = $a_i + b_i$ (could be XOR as well)

*Delete* = $a_i b_i$

\[
c_{out}(g_i, p_i) = g_i + p_i c_{in}
\]
\[
s(g_i, p_i) = g_i \oplus c_{in}
\]

Can also derive expressions for $s$ and $c_{out}$ based on $d_i$ and $p_i$

Carry Lookahead Adder

Lookahead Adder

Lookahead Equations

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

Position $i + 1$: $c_{i+1} = g_{i+1} + p_{i+1} c_i$

$= g_{i+1} + p_{i+1} (g_i + p_i c_{i-1})$

$= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_{i-1}$

Carry exists if:
- generated in stage $i + 1$
- generated in stage $i$ and propagated through $i + 1$
- propagated through both $i$ and $i + 1$

Unrolling of carry recurrence can be continued
- If unrolled to level $k$, resulting in two-level AND-OR structure
  - AND Fan-In = $k + 1$, OR Fan-In = $k + 1$
  - $k + 1$ transistors in the MOS stack
  - Limits $k$ to $2 – 4$
  - Later referred to as a radix of an adder
**Lookahead Adder**

**Mirror Implementation**

![Mirror Implementation Diagram](image)

**Block Lookahead**

Fourth bit carry:

\[ c_{i+4} = g_{i+3} + p_{i+3}g_{i+2} + p_{i+3}p_i g_{i+1} + p_{i+3}p_i + 2 p_{i+1} + 2 p_i + 1 p_i c_{i-1} \]

Block generate and block propagate:

\[ G_{i,i+3} = g_{i+3} + p_{i+3}g_{i+2} + p_{i+3}p_i g_{i+1} + p_{i+3}p_i + 2 p_{i+1} + 2 p_i + 1 g_{i} \]
\[ P_{i,i+3} = p_{i+3}p_i + 2 p_i + 1 p_i \]
\[ c_{i+4} = G_{i,i+3} + P_{i,i+3}c_{i-1} \]
Can create groups of groups, or ‘super-groups’:

\[
G^*_{j+3:j} = G_{j+3} + P_{j+3}G_{j+2} + P_{j+3}P_{j+2}G_{j+1} + P_{j+3}P_{j+2}P_{j+1}G_j
\]

\[
P^*_{j+3:j} = P_{j+3}P_{j+2}P_{j+1}P_j
\]

Delay is \( t_d = c_1 \log N \)
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

\[ P_G = p_m \cdot p_l \quad \text{\(m\) – more significant} \]
\[ G_G = g_m + p_m \cdot g_l \quad \text{\(l\) – less significant} \]

Start from the input \(P, G\), and continue up the tree
2-bit groups, then 4-bit groups, ...

\[ (g, p) = (g_m, p_m) \cdot (g_l, p_l) = (g_m + p_m \cdot g_l, p_m \cdot p_l) \]

Kogge, Stone, Trans on Comp,'73

Radix 2
Tree Adders: Radix 2

16-bit radix-2 Kogge-Stone Tree

Tree Adders: Radix 4

16-bit radix-4 Kogge-Stone Tree
Sparse Trees

16-bit radix-2 sparse tree with sparseness of 2 (Han-Carlson)

Full vs. Sparse Trees

- Sparse trees have less transistors, wires
  - Less power
- Less input loading
- Recovering missing carries
  - Ripple (extra gate delay)
  - Precompute (extra fanout)
- Complex precompute can get into the critical path
Tree Adders: Other Trees

- Ladner-Fischer

\[
\begin{align*}
S_0 &= S_5 \\
S_1 &= S_6 \\
S_2 &= S_7 \\
S_3 &= S_8 \\
S_4 &= S_9 \\
S_5 &= S_{10} \\
S_6 &= S_{11} \\
S_7 &= S_{12} \\
S_8 &= S_{13} \\
S_9 &= S_{14} \\
S_{10} &= S_{15}
\end{align*}
\]

Ling Adder

Variation of CLA

\[
\begin{align*}
p_i &= a_i \oplus b_i \\
g_i &= a_i \cdot b_i \\
G_i &= g_i + p_i \cdot G_{i-1} \\
S_i &= p_i \oplus G_{i-1}
\end{align*}
\]

Ling’s equations

\[
\begin{align*}
t_i &= a_i + b_i \\
g_i &= a_i \cdot b_i \\
H_i &= g_i + t_{i-1} \cdot H_{i-1} \\
S_i &= t_i \oplus H_i + g_i t_{i-1} H_{i-1}
\end{align*}
\]

Ling, IBM J. Res. Dev, 5/81
Ling Adder

Conventional CLA:

\[ G_i = g_i + p_i \cdot G_{i-1} \]

Also:

\[ G_i = g_i + t_i \cdot G_{i-1} \]

Ling’s equation shifts the index of pseudo carry

\[ H_i = g_i + t_{i-1} \cdot G_{i-1} \]

Propagates information on two bits

Doran, Trans on Comp 9/88

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 vs. CLA**

- **Energy [pJ]**
  - R2 Ling
  - R2 CLA
  - R4 Ling
  - R4 CLA

**Static vs. Dynamic**

- **Energy [pJ]**
  - Compound Domino R2
  - Domino R2
  - Domino R4
  - Static R2
Stack Height Limiting

- Transform conventional $G \cdot P$

$$G_4 = G_1 + P_1 \cdot G_{i+1} + G_{i+1} \cdot G_{i+2} + P_1 \cdot P_{i+1} \cdot P_{i+2} \cdot G_{i+3}$$

$$G_4 = (G_{i+1} + G_{i+2} + P_{i+2} \cdot G_{i+3}) \cdot (G_i + P_i \cdot P_{i+1})$$

Park, VLSI Circ’00

---

HP Adder

$$i_4 = P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

Naffziger, ISSCC’96
HP Adder – Differential Domino

Hybrid Adders

Dobberpuhl, JSSC 11/92
DEC Alpha 21064
DEC Adder

- **Combination:**
  - 8-bit tapered pre-discharged Manchester carry chains, with $C_{in} = 0$ and $C_{in} = 1$
  - 32-bit LSB carry-lookahead
  - 32-bit MSB conditional sum adder
  - Carry-select on most significant bits
  - Latch-based timing