Lecture 11: Adders

Deming Chen

Slides based on the initial set from David Harris
Outline

- Single-bit Addition
- Carry-Ripple Adder
- Carry-Skip Adder
- Carry-Lookahead Adder
- Carry-Select Adder
- Carry-Increment Adder
- Tree Adder

Readings: 11.1-11.2.2.8
Single-Bit Addition

Half Adder
\[ S = A \oplus B \]
\[ C_{out} = A \cdot B \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Full Adder
\[ S = A \oplus B \oplus C \]
\[ C_{out} = MAJ(A, B, C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
For a full adder, define what happens to carries (in terms of A and B)
- Generate: $C_{out} = 1$ independent of C
  - $G = A \cdot B$
- Propagate: $C_{out} = C$
  - $P = A \oplus B$
- Kill: $C_{out} = 0$ independent of C
  - $K = \sim A \cdot \sim B$
Full Adder Design I

- Brute force implementation from eqns

\[ S = A \oplus B \oplus C \]

\[ C_{\text{out}} = \text{MAJ}(A, B, C) \]
Full Adder Design II

- Factor $S$ in terms of $C_{out}$
  \[ S = ABC + (A + B + C)(\neg C_{out}) \]
- Critical path is usually $C$ to $C_{out}$ in ripple adder
Layout

- Clever layout circumvents usual line of diffusion
  - Use wide transistors on critical path
  - Eliminate output inverters
Full Adder Design III

- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area
Carry Propagate Adders

- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?

\[
\begin{array}{c}
\text{A}_{N-1} \quad \text{B}_{N-1} \\
\text{S}_{N-1} \\
\text{C}_{\text{out}} \quad \text{C}_{\text{in}}
\end{array}
\]

\[
\begin{array}{c}
0000 \\
1111 \\
+0000 \\
1111 \\
0000 \\
\text{carries} \\
\text{A}_{4...1} \quad \text{B}_{4...1} \\
\text{S}_{4...1}
\end{array}
\]
Carry-Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from $C_{in}$ to $C_{out}$
  - Design full adder to have fast carry delay

![Carry-Ripple Adder Diagram](image)
Generate / Propagate

- Equations often factored into G and P
- Generate and propagate for groups spanning i:j

\[ G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \]
\[ P_{i:j} = P_{i:k} \cdot P_{k-1:j} \]

- Base case

\[ G_{i:i} \equiv G_i = A_i \cdot B_i \]
\[ P_{i:i} \equiv P_i = A_i \oplus B_i \]

- Sum:

\[ S_i = P_i \oplus G_{i-1:0} \]
PG Logic

1: Bitwise PG logic
2: Group PG logic
3: Sum logic
Carry-Ripple Revisited

\[ G_{i:0} = G_i + P_i \cdot G_{i-1:0} \]
Carry-Ripple PG Diagram

\[ t_{\text{ripple}} = t_{\text{pg}} + (N - 1)t_{\text{AO}} + t_{\text{xor}} \]
PG Diagram Notation

Black cell

\[ i:k \quad k-1:j \]

\[ \text{Black cell} \]

\[ i:j \]

Gray cell

\[ i:k \quad k-1:j \]

\[ \text{Gray cell} \]

Buffer

\[ i:j \]

\[ \text{Buffer} \]
Carry-Skip Adder

- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal

```
C_{out} \rightarrow A_{16:13} B_{16:13} \rightarrow P_{16:13} \rightarrow \ldots \rightarrow A_{4:1} B_{4:1} \rightarrow P_{4:1} \rightarrow C_{in}
```
For k n-bit groups (N = nk)

\[ t_{skip} = t_{pg} + \left[ 2(n - 1) + (k - 1) \right] t_{AO} + t_{xor} \]
Variable Group Size

Delay grows as $O(\sqrt{N})$
Carry-Lookahead Adder

- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel.
- Uses higher-valency cells with more than two inputs.

Diagram:

- $A_{16:13}$, $B_{16:13}$
- $G_{16:13}$, $P_{16:13}$
- $S_{16:13}$, $C_{12}$
- $A_{12:9}$, $B_{12:9}$
- $G_{12:9}$, $P_{12:9}$
- $S_{12:9}$, $C_{8}$
- $A_{8:5}$, $B_{8:5}$
- $G_{8:5}$, $P_{8:5}$
- $S_{8:5}$, $C_{4}$
- $A_{4:1}$, $B_{4:1}$
- $G_{4:1}$, $P_{4:1}$
- $S_{4:1}$, $C_{in}$
- $C_{out}$
Carry-Select Adder

- Trick for critical paths dependent on late input X
  - Precompute two possible outputs for X = 0, 1
  - Select proper output when X arrives
- Carry-select adder precomputes n-bit sums
  - For both possible carries into n-bit group
Tree Adder

- If lookahead is good, lookahead across lookahead!
  - Recursive lookahead gives $O(\log N)$ delay
- Many variations on tree adders
Sklansky

Adders

CMOS VLSI Design 4th Ed.
Kogge-Stone Adders

Adders

CMOS VLSI Design 4th Ed.

24
Tree Adder Taxonomy

- Ideal N-bit tree adder would have
  - \( L = \log N \) logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels

- Describe adder with 3-D taxonomy \((l, f, t)\)
  - Logic levels: \( L + l \)
  - Fanout: \( 2^f + 1 \)
  - Wiring tracks: \( 2^t \)

- Known tree adders sit on plane defined by
  \[ l + f + t = L - 1 \]
Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Classification</th>
<th>Logic Levels</th>
<th>Max Fanout</th>
<th>Tracks</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carry-Ripple</td>
<td></td>
<td>N-1</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>Carry-Skip n=4</td>
<td></td>
<td>N/4 + 5</td>
<td>2</td>
<td>1</td>
<td>1.25N</td>
</tr>
<tr>
<td>Carry-Inc. n=4</td>
<td></td>
<td>N/4 + 2</td>
<td>4</td>
<td>1</td>
<td>2N</td>
</tr>
<tr>
<td>Brent-Kung</td>
<td>(L-1, 0, 0)</td>
<td>2log₂N – 1</td>
<td>2</td>
<td>1</td>
<td>2N</td>
</tr>
<tr>
<td>Sklansky</td>
<td>(0, L-1, 0)</td>
<td>log₂N</td>
<td>N/2 + 1</td>
<td>1</td>
<td>0.5 Nlog₂N</td>
</tr>
<tr>
<td>Kogge-Stone</td>
<td>(0, 0, L-1)</td>
<td>log₂N</td>
<td>2</td>
<td>N/2</td>
<td>Nlog₂N</td>
</tr>
</tbody>
</table>