Design of Adders

• Last module:
  – Designing CMOS gate networks
  – Speeding up combinational gates

• This module
  – Adder circuits
  – Simple adders
  – Fast addition
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

\[ S = A \oplus B \oplus C \]
\[ C_{out} = (A \cdot B) \oplus C \]

<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>
Full Adder Design I

- Brute force implementation from equations

\[
S = \oplus \oplus \\
C_{out} = , B, C)
\]
Full Adder Design II

- Factor $S$ in terms of $C_{out}$
  
  $$S = ABC + (A + B + C)(\sim C_{out})$$

- Critical path is usually $C$ to $C_{out}$ in ripple adder
• 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 cf. the clever layout of II
Carry Propagate Adders

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

\[
\begin{array}{c}
A_{N...1} \\
B_{N...1} \\
S_{N...1} \\
C_{in} \\
+ \\
C_{out}
\end{array}
\]

\[
\begin{array}{c}
C_{in} \\
C_{out}
\end{array}
\]

\[
\begin{array}{c}
0000 \\
1111 \\
+0000 \\
1111
\end{array}
\]

\[
\begin{array}{c}
1111 \\
+0000 \\
0000
\end{array}
\]

Carries

\[
A_{4...1} \\
B_{4...1} \\
S_{4...1}
\]
Ripple Carry Adder

- Simplest design: cascade full adders
  - Critical path goes from Cin to Cout
  - Design full adder to have fast carry delay

![Ripple Carry Adder Diagram]
Inversions

• Critical path passes through majority gate
  – Built from minority + inverter
  – Eliminate inverter and use inverting full adder
PGK

• For a full adder, define what happens to carries
  – Generate: \( C_{\text{out}} = 1 \) independent of \( C \)
    • \( G = A \cdot B \)
  – Propagate: \( C_{\text{out}} = C \)
    • \( P = A \oplus B \)
  – Kill: \( C_{\text{out}} = 0 \) independent of \( C \)
    • \( K = \neg A \cdot \neg B \) (i.e., \( \neg K = A + B \))
Generate / Propagate

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

\[ G_{i:j} = \ldots + \ldots \]
\[ P_{i:j} = \ldots \]

- Base case

\[ G_{i:i} \equiv \ldots = \square \]
\[ G_{0:0} \equiv \ldots = \]
\[ P_{i:i} \equiv \ldots = \oplus \]
\[ P_{0:0} \equiv \ldots = \]

- Sum: \[ S_i = \ldots \oplus \]
PG Logic

1: Bitwise PG logic

2: Group PG logic

3: Sum logic

A_4 B_4 A_3 B_3 A_2 B_2 A_1 B_1 C_in

G_4 P_4 G_3 P_3 G_2 P_2 G_1 P_1 G_0 G_0

C_0 C_1 C_2 C_3 C_4

C_out S_4 S_3 S_2 S_1
Ripple Carry Revisited

\[ G_{i:0} = + \]
Ripple Carry PG Diagram

\[ t_{\text{ripple}} = + - + \]
PG Diagram Notation

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

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

Buffer
\[ i:j \]

\[ \begin{align*}
  G_{i:k} \quad P_{i:k} \\
  G_{k-1:j} \quad P_{k-1:j}
\end{align*} \]
**Carry-Skip Adder**

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

![Carry-Skip Adder Diagram]
Carry-Skip PG Diagram

For $k$ $n$-bit groups ($N = nk$)

$$t_{\text{skip}} = \cdots + \left[ \begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \right]$$
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.
CLA PG Diagram

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

16:0 15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0
Higher-Valency Cells
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
Carry-Increment Adder

- Factor initial PG and final XOR out of carry-select

\[ t_{\text{increment}} = r_o + \left[ \right] \]

CS758

Karu Sankaralingam
Variable Group Size

- Also buffer noncritical signals
Tree Adder

• If lookahead is good, lookahead across lookahead!
  – Recursive lookahead gives $O(\log N)$ delay

• Many variations on tree adders
Kogge-Stone
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
  \[ l + f + t = L - 1 \]
Tree Adder Taxonomy

![Diagram of Tree Adder Taxonomy](image-url)
Tree Adder Taxonomy

Kogge-Stone

Sklansky

Brent-Kung

f (Fanout)

l (Logic Levels)

t (Wire Tracks)

0 (2)
1 (3)
2 (5)
3 (9)
0 (4)
1 (5)
2 (6)
3 (8)
2 (4)
1 (2)
0 (1)
3 (7)

CS758

Karu Sankaralingam
Knowles [2, 1, 1, 1]
Ladner-Fischer
Taxonomy Revisited
Summary
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>Ripple Carry</td>
<td>N-1</td>
<td>1</td>
<td>1</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Carry-Skip n=4</td>
<td>N/4 + 5</td>
<td>2</td>
<td>1</td>
<td>1.25N</td>
<td></td>
</tr>
<tr>
<td>Carry-Inc. n=4</td>
<td>N/4 + 2</td>
<td>4</td>
<td>1</td>
<td>2N</td>
<td></td>
</tr>
<tr>
<td>Brent-Kung</td>
<td>(L-1, 0, 0)</td>
<td>2log₂N – 1</td>
<td>2</td>
<td>2N</td>
<td></td>
</tr>
<tr>
<td>Sklansky</td>
<td>(0, L-1, 0)</td>
<td>log₂N</td>
<td>N/2 + 1</td>
<td>0.5log₂N</td>
<td></td>
</tr>
<tr>
<td>Kogge-Stone</td>
<td>(0, 0, L-1)</td>
<td>log₂N</td>
<td>2</td>
<td>Nlog₂N</td>
<td></td>
</tr>
</tbody>
</table>