Chapter 7   Variations in Fast Adders

- Simple Carry-Skip Adders
- Multilevel Carry-Skip Adders
- Carry-Select Adders
- Conditional-Sum Adder
- Hybrid Adder Designs
- Optimizations in Fast Adders
Simple Carry-Skip Adders

(a) Ripple-carry adder.

(b) Simple carry-skip adder.
Skip Logic in Carry-Skip Adders

carry = 1 if (a carry is generated) or (C₀ == 1 and Propagation == 1)
Worst Case Delay

\[ T_{\text{delay}} = (b - 1) + (0.5) + \left( \frac{k}{b} - 2 \right) + (b - 1) \]

\[ \approx 2b + \frac{k}{b} - 3.5 \text{ stages (ex: 12.5)} \]

- The worst-case delay in stage 0-3: a carry generated in stage 0 and propagated through 1-3.
  - It is \( b-1 \)

\[ b = \text{fixed block width (ex: 4)} \]

\[ k = \text{number of bits (ex: 16)} \]
What is the optimal block size?

1. set \( \frac{dT}{db} = 0 \)

2. solve for \( b = b_{opt} \)

\[
b_{opt} = \sqrt{\frac{k}{2}}
\]

\[
t = \text{number of blocks} = \frac{k}{b}
\]

\[
t_{opt} = \sqrt{2k}
\]

\[
T_{opt} = 2\sqrt{2k} - 3.5
\]
Can we do better?

Path (1) is one delay longer that Path (2) → block \( t-2 \) can be one bit wider than block \( t-1 \).

Path (1) is one delay longer that Path (3) → block \( l \) can be one bit wider than block \( 0 \).
Variable Block-Width Carry-Skip Adders

Optimal Block Widths:

\[ b \quad b+1 \quad \cdots \quad b + \frac{t}{2} - 1 \quad b + \frac{t}{2} - 1 \quad \cdots \quad b + 1 \quad b \]

\[ b + (b+1) + \cdots + (b + \frac{t}{2} - 1) + (b + \frac{t}{2} - 1) + \cdots + (b+1) + b = k \]

\[ \rightarrow b = \left( \frac{k}{t} \right) - \left( \frac{t}{4} \right) + 1/2 \]
Optimal number of blocks

\[ T_{\text{delay}} = \underbrace{2(b-1)}_{\text{first + last stage}} + \underbrace{(0.5)}_{\text{OR gate}} + \underbrace{(t-2)}_{\text{Skip stages}} = \frac{2k}{t} + \frac{t}{2} - 2.5 \]

1. set \( \frac{dT}{dt} = 0 \)

2. solve for \( t = t_{\text{opt}} \)

\[ t_{\text{opt}} = 2\sqrt{k} \]

\[ b_{\text{opt}} = \left\lceil 1/2 \right\rceil = 1 \quad \text{(stage 0, t-1; goes up to } t_{\text{opt}}/2 = \sqrt{k}) \]

\[ T_{\text{opt}} = 2\sqrt{k} - 2.5 \]
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Fixed-width Carry-Skip</th>
<th>Variable-width Carry-Skip</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{opt}$</td>
<td>$\sqrt{2k}/2$</td>
<td>$2\sqrt{k}$</td>
</tr>
<tr>
<td>$b_{opt}$</td>
<td>$\sqrt{k}/2$</td>
<td>$1 \cdots \sqrt{k} \sqrt{k} \cdots 1$</td>
</tr>
<tr>
<td>$T_{opt}$</td>
<td>$2\sqrt{2k} - 3.5$</td>
<td>$2\sqrt{k} - 2.5$</td>
</tr>
</tbody>
</table>

Conclusion: Variable-width is about 40% faster.
Multilevel Carry-Skip Adders

- **One-level carry-skip adder**

- **Two-level carry-skip adder**

  - 3 delay
Multilevel Carry-Skip Adders
Multilevel Carry-Skip Adders

- Allow carry to skip over several level-1 skip blocks at once.
- Level-2 propagate is AND of level-1 propagates.

Assumptions:
- OR gate is no delay (insignificant delay)
- Basic delay = Skip delay = Ripple delay = Propagate Computation = Sum Computation
Simplifying the Circuit

It doesn’t save any time to skip short carry-chains (1-2 cells long)

optimized
Build the Widest Single-Level Carry-Skip Adder with 8 delays max

\[ \text{Width} = 1 + 3 + 4 + 4 + 3 + 2 + 1 = 18 \text{ bits} \]
Build the Widest Two-Level Carry-Skip Adder with 8 delays max

First, we need a new notation:

\[
T_{\text{produce}} \leq \beta \\
\{\beta, \alpha\} \\
T_{\text{assimilate}} \leq \alpha \\
\gamma \\
\gamma = \min(\beta - 1, \alpha)
\]
8-delay, 2-level, continued
1. Find \( \{\beta, \alpha\} \) for level two

Initial Timing Constraint, Level 2
8-delay, 2-level, continued

2. Given \{\beta, \alpha\} for level two, derive level one

<table>
<thead>
<tr>
<th>Block</th>
<th>(T_{\text{produce}})</th>
<th>(T_{\text{assimilate}})</th>
<th>Number of subblocks</th>
<th>Subblock widths (bits)</th>
<th>Block Width (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>1, 3</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2, 3, 3</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2, 3, 2, 1</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3, 2, 1</td>
<td>6</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2, 1</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Generalization

- Chan et al. [1992] relax assumptions to include general worst-case delays:
  - I(b) Internal carry-propagate delay for the block.
  - G(b) Carry-generate delay for the block.
  - A(b) Carry-assimilate delay for the block.

- Used dynamic programming to obtain optimal configuration.
Carry-Select Adders

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

\[ T_{\text{select-add}}(k) = T_{\text{add}}(k/2) + 1 \]
Carry-select: Carried one step further
Two-Level Carry-Select Adder

Can be pipelined
Compare to Two-Level G-P Adder
Conditional Sum Adder

- The process that led to the two-level carry select adder can be continued . . .

- A logarithmic time conditional-sum adder results if we proceed to the extreme:
  - single bit adders at the top.

- A conditional-sum adder is actually a $(\log_2 k)$-level carry-select adder.
Cost and Delay of a Conditional-Sum Adder

\[ C(k) \approx 2C(k/2) + k + 2 \approx k(\log_2 k + 2) + kC(1) \]

\[ T(k) = T(k/2) + 1 = \log_2 k + T(1) \]

More exact analysis gives actual cost = \((k - 1)(\log_2 k + 1) + kC(1)\)

Top-level block for one bit position of a conditional-sum adder

C(1) and T(1) are the cost and time delay of this circuit.
### Conditional-Sum Example

<table>
<thead>
<tr>
<th>Block width</th>
<th>Block carry-in</th>
<th>Block sum and block carry-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 s c</td>
<td>0 1 0 1 0 0 1 1 0 1 1 0 0 1 0</td>
</tr>
<tr>
<td>1</td>
<td>1 s c</td>
<td>1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 s c</td>
<td>0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>2</td>
<td>1 s c</td>
<td>1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>0 s c</td>
<td>0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>1 s c</td>
<td>0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 1</td>
</tr>
<tr>
<td>8</td>
<td>0 s c</td>
<td>0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>8</td>
<td>1 s c</td>
<td>0 1 1 1 0 0 0 1 0 0 1 0 1 1 1 1</td>
</tr>
<tr>
<td>16</td>
<td>0 s c</td>
<td>0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>16</td>
<td>1 s c</td>
<td>0 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
Hybrid Adder Designed

Hybrids are obtained by combining elements of:

- Ripple-carry adders.
- Carry-lookahead (generate-propagate) adders.
- Carry-skip adders.
- Carry-select adders.
- Conditional-sum adders.

You can obtain adders with

- higher performance.
- greater cost-effectiveness.
- lower power consumption.
Example 1

Carry-Select / Carry-Lookahead

- One- and Two-level carry select adders are essentially hybrids, since the top level k/2- or k/4-bit adders can be of any type.
- Often combined with carry-lookahead adders.
Example 2
Carry-Lookaahead/Carry-Select
Example 3
Multilevel Carry-Lookahead/Carry-Select

\[
\begin{align*}
[60,63] & \quad [48,63] \\
[56,59] & \quad [48,59] \\
[52,55] & \quad [48,55] \\
[48,51] & \\
[44,47] & \quad [32,47] \\
[40,43] & \quad [32,43] \\
[36,39] & \quad [32,39] \\
[32,35] & \\
[28,31] & \quad [16,31] \\
[24,27] & \quad [16,27] \\
[20,23] & \quad [16,23] \\
[16,19] & \\
[12,15] & \quad [0,15] \\
[8,11] & \quad [0,11] \\
[4,7] & \quad [0,7] \\
[0,3] & \\
\end{align*}
\]

Manchester carry chain

\[
\begin{align*}
& [0,55] \\
& [0,47] \\
& [0,31] \\
& [0,39] \\
& [0,39] \\
& [0,31] \\
& [0,23] \\
& c_{56} \\
& c_{48} \\
& c_{40} \\
& c_{32} \\
& c_{24} \\
& c_{16} \\
& c_{12} \\
& c_{8} \\
& c_{0} \\
\end{align*}
\]

\text{to Carry - Select Adders}

\[i,j\] represents the pair of signals \(p_{(i,j)}\) and \(g_{(i,j)}\).

\[0,j\] should really be \([-1,j]\), (since \(c_0\) is taken to be \(g_{-1}\)).

Can be pipelined
Example 4
Ripple-Carry/Carry-Lookahead

Simple and modular
Example 5
Carry Lookahead/Conditional-Sum

- Reduces fan-out required to control the muxes at the lower level (a draw-back of wide conditional sum sum adders).
- Use carry conditional-sum addition in smaller blocks, but form inter-block carries using carry-lookahead.
Open Questions

- Application requirements may shift the balance in favor of a particular hybrid design.
- What combinations are useful for:
  - low power addition.
  - addition on an FPGA.
Optimizations in Fast Adders

- It is often possible to reduce the delay of adders (including hybrids) by optimizing block widths.
- The exact optimal configuration is highly technology dependent.
- Designs that minimize or regularize the interconnect may actually be more cost effective that a design with low gate count.
Other Optimizations

- Assumption: all inputs are available at time zero.
- But, sometimes that is not true:
  - I/O arrive/depart serially, or
  - Different arrival times are associated with input digits, or
  - Different production times are associated with output digits.
- Example: Addition of partial products in a multiplier.