EECS150 - Digital Design
Lecture 10 - Combinational Logic Circuits
Part 1

February 26, 2002
John Wawrzynek
Combinational Logic (CL) Defined

\[ y_i = f_i(x_0, \ldots, x_{n-1}), \text{where } x, y \text{ are } \{0,1\}. \]

- Y is a function of only X.
- If we change X, Y will change immediately (well almost!).
- There is an implementation dependent delay from X to Y.
Adders

Full-adder cell (FA) revisited:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>cin</th>
<th>cout</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>

Fa
Carry-ripple Adder

- Each cell:
  \[ r_i = a_i \text{ XOR } b_i \text{ XOR } c_{in} \]
  \[ c_{out} = a_i c_{in} + a_i b_i + b_i c_{in} = c_{in}(a_i + b_i) + a_i b_i \]

- 4-bit adder:

- What about subtraction?
Subtractors

A - B = A + (-B)

How do we form -B?
1. complement B
2. add 1
Adders (cont.)

Ripple Adder

Ripple adder is inherently slow because, in general, s7 must wait for c7 which must wait for c6 ...

\[ T \propto n, \quad \text{Cost} \propto n \]

How do we make it faster, perhaps with more cost?
Carry Select Adder

\[ T = \frac{T_{\text{ripple_adder}}}{2} + T_{\text{MUX}} \]

\[ \text{COST} = 1.5 \times \text{COST}_{\text{ripple_adder}} + (n+1) \times \text{COST}_{\text{MUX}} \]
Carry Select Adder

- Extending Carry-select to multiple blocks

- What is the optimal # of blocks and # of bits/block?
  - If # blocks too large delay dominated by total mux delay
  - If # blocks too small delay dominated by adder delay

\[ \sqrt{N} \text{ stages of } \sqrt{N} \text{ bits} \quad T \propto \sqrt{N}, \quad \text{Cost} \approx 2\times\text{ripple} + \text{muxes} \]
Carry Select Adder

- $T_{total} = \sqrt{N} \ T_{FA}$
  - assuming $T_{FA} = T_{MUX}$
- For ripple adder $T_{total} = N \ T_{FA}$

- Is $\sqrt{N}$ really the optimum?
  - From right to left increase size of each block to better match delays
  - Ex: 64-bit adder, use block sizes [13 12 11 10 9 8 7]
- How about recursively defined carry select?
Carry Look-ahead Adders

- In general, for n-bit addition best we can achieve is delay $\alpha \log(n)$
- How do we arrange this? (think trees)
- First, reformulate basic adder stage:

<table>
<thead>
<tr>
<th>a b c_i</th>
<th>c_{i+1}</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- carry "kill"
- $k_i = a_i' \cdot b_i'$
- carry "propagate"
- $p_i = a_i \oplus b_i$
- carry "generate"
- $g_i = a_i \cdot b_i$
- $c_{i+1} = g_i + p_i \cdot c_i$
- $s_i = p_i \oplus c_i$
Carry Look-ahead Adders

- Ripple adder using p and g signals:

\[ s_0 = p_0 \oplus c_0 \]
\[ c_1 = g_0 + p_0c_0 \]

\[ s_0 = p_1 \oplus c_1 \]
\[ c_2 = g_1 + p_1c_1 \]

\[ s_0 = p_2 \oplus c_2 \]
\[ c_3 = g_2 + p_2c_2 \]

\[ s_0 = p_3 \oplus c_3 \]
\[ c_4 = g_3 + p_3c_3 \]

- So far, no advantage over ripple adder: \( T \propto N \)
Carry Look-ahead Adders

• Expand carries:

\[
\begin{align*}
c_0 & \\
c_1 &= g_0 + p_0 \cdot c_0 \\
c_2 &= g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_1 p_2 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + \ldots \\
\vdots & \\
\end{align*}
\]

• Why not implement these equations directly to avoid ripple delay?
  – Lots of gates. Redundancies (full tree for each).
  – Gate with high # of inputs.

• Let’s reorganize the equations.
Carry Look-ahead Adders

• “Group” propagate and generate signals:

\[ P = p_i p_{i+1} \ldots p_{i+k} \]
\[ G = g_{i+k} + p_{i+k} g_{i+k-1} + \ldots + (p_{i+1} p_{i+2} \ldots p_{i+k}) g_i \]

\[ C_{out} = G + PC_{in} \]

• P true if the group as a whole propagates a carry to \( c_{out} \)
• G true if the group as a whole generates a carry
• Group P and G can be generated hierarchically.
Carry Look-ahead Adders

9-bit Example of hierarchically generated $P$ and $G$ signals:

$P = P_a P_b P_c$

$G = G_c + P_c G_b + P_b P_c G_a$

$c_0 = \text{Carry Input}$

$c_3 = G_a + P_a c_0$

$c_6 = G_b + P_b c_3$

$c_9 = G + P c_0$
8-bit Carry Look-ahead Adder with 2-input gates.

- \( p_0 \rightarrow g_0 \rightarrow c_0 \rightarrow P_8 = p_0 p_1 \)
- \( c_1 = g_0 + p_0 c_0 \rightarrow G_8 = g_1 + p_1 g_0 \)
- \( c_2 = G_8 + P_8 c_0 \rightarrow P_9 = p_2 p_3 \)
- \( c_3 = g_2 + p_2 c_2 \rightarrow G_9 = g_3 + p_3 g_2 \)
- \( c_4 = G_9 + P_9 c_2 \rightarrow P_a = p_4 p_5 \)
- \( c_5 = g_4 + p_4 c_4 \rightarrow G_a = g_5 + p_5 g_4 \)
- \( c_6 = G_a + P_a c_4 \rightarrow P_b = p_6 p_7 \)
- \( c_7 = g_6 + p_6 c_6 \rightarrow G_b = g_7 + p_7 g_6 \)
- \( c_8 = G_b + P_b c_6 \)

\( P_c = P_8 P_9 \)

\( G_c = G_9 + P_9 G_8 \)

\( P_e = P_c P_d \)

\( G_e = G_d + P_d G_c \)

\( P_d = P_a P_b \)

\( G_d = G_b + P_b G_a \)
Adders in FPGAs