Carry Look-ahead Adders

- In general, for n-bit addition
  delay \times \log(n)

- How do we arrange this?
  (arrange carry circuit as a tree)

- First reformulate basic adder stage:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Ci</th>
<th>S</th>
<th>C_{i+1}</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

  carry "kill"  \quad K_i = \overline{a_i} \cdot \overline{b_i}

  carry "propagate"  \quad P_i = a_i \oplus b_i

  carry "generate"  \quad G_i = a_i \cdot b_i

C_{i+1} = G_i + P_i \cdot C_i

S_i = P_i \oplus C_i
Ripple adder using $p$ and $g$ signals:

$$c_0$$

$$(s_0 = p_0 \oplus c_0)$$

$$c_1 = g_0 + p_0 c_0$$

$$(s_1 = p_1 \oplus c_1)$$

$$c_2 = g_1 + p_1 c_1$$

$$\vdots$$

$$c_3$$

$$(s_3 = p_3 \oplus c_3)$$

$$c_4$$

So far, no delay advantage over standard ripple adder delay $\propto n$
- expand carries

\[ C_0 \]
\[ C_1 = g_0 + p_0 C_0 \]
\[ C_2 = g_1 + p_1 C_1 = g_1 + p_1 g_0 + p_1 p_0 C_0 \]
\[ C_3 = g_2 + p_2 C_2 = g_2 + p_2 g_1 + p_1 p_2 g_0 + p_1 p_2 p_0 C_0 \]
\[ C_4 = g_3 + p_3 C_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + \ldots \]

why not implement these equations directly to avoid ripple?

- Lots of gates (redundancies - full tree for each)
- High input gates

Let's reorganize these equations.
"Group" propagate & generate signals

\[
\begin{align*}
\text{Cin} & \quad \downarrow \\
\rightarrow P & = P_i \cdot P_{i+1} \cdots \cdot P_{i+k} \\
\rightarrow G & = g_{i+k} + P_{i+k} \cdot g_{i+k-1} + \\
& \quad \cdots + (P_{i+1} \cdot P_{i+2} \cdots P_{i+k}) g_i \\
\downarrow & \\
\text{Cout} &
\end{align*}
\]

\[P \text{ true if the group as a whole propagates a carry to Cout.}\]

\[G \text{ true if the group as a whole generates a carry.}\]

\[\text{Cout} = G + PC_{in}\]

\[\text{Group P & G can be generated hierarchically.}\]
Example: 9-bit group

\[ C_0 \]

\[ a_0 \rightarrow P_a \]
\[ b_0 \rightarrow \]
\[ a_1 \rightarrow \]
\[ b_1 \rightarrow \]
\[ a_2 \rightarrow \]
\[ b_2 \rightarrow \]
\[ C_3 = G_a + P_a C_0 \rightarrow P = P_a P_b P_c \]

\[ a_3 \rightarrow P_b \]
\[ b_3 \rightarrow \]
\[ a_4 \rightarrow \]
\[ b_4 \rightarrow \]
\[ a_5 \rightarrow \]
\[ b_5 \rightarrow \]
\[ C_6 = G_b + P_b C_3 \]

\[ a_6 \rightarrow P_c \]
\[ b_6 \rightarrow \]
\[ a_7 \rightarrow \]
\[ b_7 \rightarrow \]
\[ a_8 \rightarrow \]
\[ b_8 \rightarrow \]
\[ C_9 = G_1 + P C_0 \]
\[ G_1 = G_c + P_c G_b \]
\[ + P_b P_c G_a \]
16-bit adder example with 2-input gates

\[ C_0 \]

\[ P_0 \rightarrow g_0 \rightarrow S_0 \rightarrow C_1 = g_0 + P_0 C_0 \]

\[ P_1 \rightarrow g_1 \rightarrow S_1 \rightarrow C_2 = g_1 + P_0 C_2 \]

\[ C_2 \]

\[ P_2 \rightarrow g_2 \rightarrow S_2 \rightarrow C_3 = g_2 + P_2 C_2 \]

\[ P_3 \rightarrow g_3 \rightarrow S_3 \]

\[ C_4 \]

\[ P_4 \rightarrow g_4 \rightarrow S_4 \rightarrow C_5 = g_4 + P_4 C_4 \]

\[ P_5 \rightarrow g_5 \rightarrow S_5 \rightarrow C_6 = g_5 + P_5 C_5 \]

\[ C_6 \]

\[ P_6 \rightarrow g_6 \rightarrow S_6 \rightarrow C_7 = g_6 + P_6 C_6 \]

\[ P_7 \rightarrow g_7 \rightarrow S_7 \rightarrow C_8 = g_7 + P_7 C_7 \]

\[ C_8 \]

\[ P_8 = P_0 P_1 \]

\[ G_8 = g_1 + P_1 g_0 \]

\[ P_c = P_8 P_9 \]

\[ C_4 = G_c + P_c C_0 \]

\[ G_c = g_3 + P_3 g_2 \]

\[ C_9 = G_e + P_e C_0 \]

\[ G_e = g_7 + P_7 g_6 \]

\[ P_a = P_4 P_5 \]

\[ G_a = g_5 + P_5 g_4 \]

\[ P_d = P_4 P_5 \]

\[ G_d = g_6 + P_6 g_5 \]

\[ P_e = P_8 P_9 \]

\[ G_e = g_7 + P_7 g_6 \]