Carry-ripple Adder Revisited

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

- 4-bit adder:  
  "Full adder cell"

- What about subtraction?
Subtractor

A - B = A + (-B)

How do we form -B?
1. complement B
2. add 1

Delay in Ripple Adders

- Ripple delay amount is a function of the data inputs:

- However, we usually only worry about the worst case delay on the critical path. There is always at least one set of input data that exposes the worst case delay.
Adders (cont.)

Ripple Adder

Ripple adder is inherently slow because, in general, \( s_7 \) must wait for \( c_7 \) which must wait for \( c_6 \) …

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

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

Carry Select Adder

\[
T = T_{\text{ripple adder}} / 2 + T_{\text{MUX}}
\]

\[
\text{COST} = 1.5 \cdot \text{COST}_{\text{ripple adder}} + (n+1) \cdot \text{COST}_{\text{MUX}}
\]
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

\[ \sqrt{N} \text{ stages of } \sqrt{N} \text{ bits} \]
\[ T \propto \sqrt{N}, \]
\[ \text{Cost} = 2 \times \text{ripple + muxes} \]

Carry Select Adder

- Compare to ripple adder delay:

\[ T_{\text{total}} = 2 \sqrt{N} \] \( T_{\text{FA}} - T_{\text{FA}} \) assuming \( T_{\text{FA}} = T_{\text{MUX}} \)

For ripple adder \( T_{\text{total}} = N \) \( T_{\text{FA}} \)

"cross-over" at \( N=3 \), Carry select faster for any value of \( N>3 \).

- 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 [12 11 10 9 8 7 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</th>
<th>b</th>
<th>c_i</th>
<th>c_{i+1}</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>

- Carry “kill” \( k_i = a_i \oplus b_i \)
- Carry “propagate” \( p_i = a_i \land b_i \)
- Carry “generate” \( g_i = a_i b_i \)

\[ c_{i+1} = g_i + p_i c_i \]
\[ s_i = p_i \oplus c_i \]

Ripple adder using \( p \) and \( g \) signals:

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

- Expand carries:
  \[c_0 = g_0 + p_0 \cdot c_0\]
  \[c_1 = g_1 + p_1 \cdot c_1 = g_1 + p_1g_0 + p_1p_0c_0\]
  \[c_2 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0\]
  \[c_3 = g_3 + p_3c_3 = g_3 + p_3g_2 + p_3p_2g_1 + \ldots\]
  \[c_4 = g_4 + p_4c_4 = g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 + \ldots\]

- 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.

- “Group” propagate and generate signals:
  \[
  \begin{align*}
  P &= p_i \cdot p_{i+1} \cdots p_{i+k} \\
  G &= g_{i+k} + p_{i+k}g_{i+k-1} + \ldots + (p_{i+1}p_{i+2} \cdots p_{i+k})g_i \\
  \end{align*}
  \]

  - P true if the group as a whole propagates a carry to \(c_{\text{out}}\)
  - G true if the group as a whole generates a carry

  \[c_{\text{out}} = G + P \cdot c_{\text{in}}\]

  - Group P and G can be generated hierarchically.
Carry Look-ahead Adders

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

$$
P = P_a P_b P_c
$$

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

8-bit Carry Look-ahead Adder

$$
P = P_a P_b
\quad G = G_b + G_a P_b
$$
8-bit Carry Look-ahead Adder with 2-input gates.

- Addition of 2 n-bit numbers:
  - takes n clock cycles,
  - uses 1 FF, 1 FA cell, plus registers
  - the bit streams may come from or go to other circuits, therefore the registers are optional.

Bit-serial Adder

- A, B, and R held in shift-registers.
  - Shift right once per clock cycle.
  - Reset is asserted by controller.

- Addition of 2 n-bit numbers:
  - takes n clock cycles,
Adders on the Xilinx Virtex

- Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Virtex-E CLB supports two separate carry chains, one per Slice. The height of the carry chains is two bits per CLB.
- The arithmetic logic includes an XOR gate and AND gate that allows a 2-bit full adder to be implemented within a slice.
- Cin to Cout delay = 0.1ns, versus 0.4ns for F to X delay.

Spring 2005 EECS150 - Lec20-cl1 Page 17