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>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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- carry “kill”
- $k_i = a_i' \cdot b_i'$
- carry “propagate”
- $p_i = a_i \oplus b_i$
- carry “generate”
- $g_i = a_i \cdot b_i$

\[
c_{i+1} = g_i + p_i c_i
\]
\[
s_i = p_i \oplus c_i
\]
Carry Look-ahead Adders

- "Group" propagate and generate signals:

\[
P = p_i p_{i+1} \ldots p_{i+k}
\]

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

- 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

- Group P and G can be generated hierarchically.

\[
C_{\text{out}} = G + PC_{\text{in}}
\]
Carry Look-ahead Adders

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

\[ P = P_a P_b P_c \]

\[ G = G_c + P_c G_b + P_b P_c G_a \]

\[ c_9 = G + P_c c_0 \]
\[ p = a \oplus b \]
\[ g = ab \]
\[ s = p \oplus c_i \]
\[ c_{i+1} = g + c_i p \]

8-bit Carry Look-ahead Adder

\[ P = P_a P_b \]
\[ G = G_b + G_a P_b \]
\[ C_{out} = G + c_{in} P \]
8-bit Carry Look-ahead Adder with 2-input gates.
Carry look-ahead Wrap-up

- Adder delay $\alpha \log_2 N$ (up then down the tree).
- Cost?
- Can be applied with other techniques. Group P & G signals can be generated for sub-adders, but another carry propagation technique (for instance ripple) used within the group.
- Other more complex techniques exist that can bring the delay down below $O(\log N)$, but are only efficient for very wide adders.
Bit-serial Adder

- 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 may be optional.

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

Diagram:

```
+-------------------+  +-------------------+  +-------------------+
| n-bit shift registers |  | reset |  | FA |  | n-bit shift register |
| A                  |  | FF   |  | c  |  | R                  |
| B                  |  |      |  | s  |  |                    |
```
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.

How do we map a 2-bit adder to one slice?
Multiplication

\[
\begin{array}{cccccc}
a_3 & a_2 & a_1 & a_0 & \text{Multiplicand} \\
b_3 & b_2 & b_1 & b_0 & \text{Multiplier} \\
\hline
X & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 \\
\hline
\text{Product} & a_1b_0 + a_0b_1 & a_0b_0
\end{array}
\]

Many different circuits exist for multiplication. Each one has a different balance between speed (performance) and amount of logic (cost).
"Shift and Add" Multiplier

- Sums each partial product, one at a time.
- In binary, each partial product is shifted versions of A or 0.

Control Algorithm:
1. \( P \leftarrow 0, A \leftarrow \text{multiplicand}, \ B \leftarrow \text{multiplier} \)
2. If LSB of B==1 then add A to P else add 0
3. Shift \([P][B]\) right 1
4. Repeat steps 2 and 3 \(n-1\) times.
5. \([P][B]\) has product.

- Cost \( \alpha n, T = n \) clock cycles.
- What is the critical path for determining the min clock period?
“Shift and Add” Multiplier

Signed Multiplication:

Remember for 2’s complement numbers MSB has negative weight:

\[ X = \sum_{i=0}^{N-2} x_i 2^i - x_{n-1} 2^{n-1} \]

ex: \(-6 = 11010_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 - 1 \cdot 2^4\)
\[ = 0 + 2 + 0 + 8 - 16 = -6 \]

- Therefore for multiplication:
  a) subtract final partial product
  b) sign-extend partial products

- Modifications to shift & add circuit:
  a) adder/subtractor
  b) sign-extender on P shifter register
Array Multiplier

Generates all $n$ partial products simultaneously.

What is the critical path?
Carry-save Addition

• Speeding up multiplication is a matter of speeding up the summing of the partial products.
• “Carry-save” addition can help.
• Carry-save addition passes (saves) the carries to the output, rather than propagating them.

Example: sum three numbers, 
\[ 3_{10} = 0011, \ 2_{10} = 0010, \ 3_{10} = 0011 \]
\[ \begin{align*}
3_{10} & \quad 0011 \\
+ \quad 2_{10} & \quad 0010 \\
\text{c} & \quad 0100 = 4_{10} \\
\text{s} & \quad 0001 = 1_{10}
\end{align*} \]

• In general, carry-save addition takes in 3 numbers and produces 2.
• Whereas, carry-propagate takes 2 and produces 1.
• With this technique, we can avoid carry propagation until final addition.
Carry-save Circuits

- When adding sets of numbers, carry-save can be used on all but the final sum.
- Standard adder (carry propagate) is used for final sum.
Array Multiplier using Carry-save Addition

Fast carry-propagate adder
Carry-save Addition

CSA is associative and communitive. For example:

\[((X_0 + X_1) + X_2) + X_3\] = \[(X_0 + X_1) + (X_2 + X_3)\]

- A balanced tree can be used to reduce the logic delay.
- This structure is the basis of the **Wallace Tree Multiplier**.
- Partial products are summed with the CSA tree. Fast CPA (ex: CLA) is used for final sum.
- Multiplier delay \(\alpha \log_{3/2} N + \log_2 N\)