EECS 151 Disc 11

Rahul Kumar (session 1)
Yukio Miyasaka (session 2)
Contents

- Adder
- Multiplier
Ripple Carry Adder
Carry Select Adder
Carry Look-ahead Adder
Parallel Prefix Adder

With carry-in, $g_0 = \text{cin}$, $p_0 = 0$, $g_1 = a_0b_0$, $p_1 = a_0 \oplus b_0$, ...

\[
G = g_3 + g_2p_2 + (g_1 + g_0p_1)p_1p_0
\]
\[
= g_3 + g_2p_2 + g_1p_3p_2 + g_0p_3p_1p_0
\]
\[
= c_4
\]

$s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$
Array Multiplier
Carry Save Adder
This is wrong. See Disc 12 slide.
Signed Multiplication

\[ \begin{array}{cccc}
  x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 \\
  + \quad x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 \\
  + \quad x_2 y_2 & x_1 y_2 & x_0 y_2 \\
  + \quad x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 \\
  + \quad 1 & 1 & 1 & 1 \\
\end{array} \]

Diagram of a signed multiplication circuit using full adders (FA) and half adders (HA).
Booth Multiplier (Radix 4)

- Reduce #partial-products by looking at 2 bits (actually 3) at a time.
- We don’t want to add A*3, so sub A and then add 4*A in the next partial product.
- We also need to sub 2*A instead of add 2*A to cancel the side-effect.
- Magically, Booth multiplier works for signed multiplication just by sign-extending the multiplier (B).
- Can use the optimization in the previous page for sign-extended partial products.

<table>
<thead>
<tr>
<th>$B_{K+1}$</th>
<th>$B_K$</th>
<th>$B_{K-1}$</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>add 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>add 2*A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>sub 2*A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>add 0</td>
</tr>
</tbody>
</table>
Practice Problems
(Taken from HW in previous semesters)
Radix-4 Kogge-Stone Adder

Recall from the lecture that Kogge-Stone adder is a parallel prefix form carry look-ahead adder (CLA). In this problem, we'd like to design and analyze a 16-bit radix-4 Kogge-Stone adder.

(a) Write down the Boolean functions of the gate implementation of following building blocks for the radix-4 Kogge-Stone adder. You may use AND, OR and XOR only. Also, how should you handle cases where certain 4-input prefix cells have fewer than 4 pairs of input?
Radix-4 Kogge-Stone Adder

Pre-processing: \[ G_i = A_iB_i, \quad P_i = A_i \oplus B_i \]

4-input prefix: \[ G_{t;i} = G_i + (P_i(G_k + (P_k(G_j + P_jG_i)))) \]
\[ P_{t;i} = P_iP_kP_jP_i \]

Post-processing: \[ S_i = P_i \oplus G_{i-1:0} \]
Radix-4 Kogge-Stone Adder

(b) Which of the following are true for radix-4 Kogge-Stone adder?

___ Compared to Radix-2 adders, Radix-4 adders reduce the depth of the tree by a factor of 4 (excluding the input and output stages).

___ An N-bit Radix-4 adders has $O(\sim \log_4(N))$ in time.

___ It is possible to implement a 32-bit Kogge-Stone adder using only the building blocks in part (a).
Radix-4 Kogge-Stone Adder

F. Radix-4 adders reduce the depth by a factor of 2.
T. But do keep in mind that each stage takes longer time.
T. You might need to leave some intermediate P,G outputs unconnected, though.
Radix-4 Kogge-Stone Adder

(c) Suppose given the following propagation delays:

\[ t_{AND} = 5\text{ps}, \quad t_{OR} = 4\text{ps}, \quad t_{XOR} = 7\text{ps} \]

Derive the critical path of the 16-bit Kogge-Stone adder based on your implementation in part (a), ignoring the delays in routing. (Hint: If you are not sure about the topology, take a look at the slides in Lecture 19.)
Radix-4 Kogge-Stone Adder

The longest path will go through: 1 pre-processing cell + 2 four-input prefix cells + 1 post-processing cell. So the total delay is:

\[
t_{\text{critical}} = \max\{5\text{ps}, 7\text{ps}\} + 2 \times (5\text{ps} + 4\text{ps} + 5\text{ps} + 4\text{ps} + 5\text{ps} + 4\text{ps}) + 7\text{ps} \\
= 7\text{ps} + 2 \times 27\text{ps} + 7\text{ps} \\
= 68\text{ps}
\]
(e) **(251A Only)** In reality, those prefix cells will not be built using the basic 2-input AND, OR, XOR gates. Instead, they will be built as a big CMOS gate (and inverters). Draw the schematic of the 4-input prefix cell for $\tilde{G}_{l,i}$ and $\tilde{P}_{l,i}$ respectively with the minimum number of transistors.
Radix-4 Kogge-Stone Adder
Booth Multiplication

Refer to the following table of the behavior of Booth recoding.

<table>
<thead>
<tr>
<th>$B_{K+1}$</th>
<th>$B_K$</th>
<th>$B_{K-1}$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>add 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>add A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>add 2A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>sub 2A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>sub A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>add 0</td>
</tr>
</tbody>
</table>

Answer should be in the format of:

Suppose $A=1010$ \hspace{1cm} $B=1001$

$(B[3:1]=010)$: \hspace{1cm} add $A$,

$(B[3:1]=100)$: \hspace{1cm} sub $2A$

$\ldots$

result \hspace{1cm} $= A - (A<<2) + \ldots$

\hspace{4cm} $= 0000(1010) - 00(1010)00 + \ldots$

\hspace{4cm} $= \ldots$

Write down the sequence of operation and the final result given the following unsigned two input numbers:

01100111 (A)
$x$ 10110010 (B)
Booth Multiplication

\[(B[1:-1]=100)\] sub 2A,
\[(B[3: 1]=001)\] add A,
\[(B[5: 3]=110)\] sub A,
\[(B[7: 5]=101)\] sub A,
\[(B[9: 7]=001)\] add A

\[
\text{result } = -(2A) + (A \ll 2) - (A \ll 4) - (A \ll 6) + (A \ll 8)
\]
\[
= -0000000(01100111)0 + 000000(01100111)00
\]
\[
- 0000(01100111)0000 - 00(01100111)000000
\]
\[
+ (01100111)0000000
\]
\[
= 0100 0111 1001 1110 (18334 \text{ in decimal})
\]