Discussion Section 11

Sean Huang
April 16, 2021
Counter Blocks

- Simple to implement
  - Just an add 1 every clock cycle!
- Register to remember the count
Counter Blocks

- Simple to implement
  - Just an add 1 every clock cycle!
- Register to remember the count
- Adders are expensive
  - Too general
  - Do we really need an entire adder to just count by 1 each time?
Counter Blocks

- Determine next value combinatorially
- Use the same tools as encoding state machines to design next count logic
  - K–maps, Boolean algebra, Truth tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>a'</th>
<th>b'</th>
<th>c'</th>
<th>d'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<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>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Terminal Count (tc)

- Common output of counters
- Flag set when either the counter reaches its maximum/minimum value or a set threshold value
- Useful for using counters in state machines
Booth Recoding

• Do partial products every 2 bits instead of 1 for fewer operations
• Traditional partial products
  – 0
  – A
• Higher-radix has more partial products
  – 0
  – A
  – 2A
  – 3A

<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>
Booth Recoding

- Higher-radix has more partial products, which we can decompose as such
  - $0 = 0$
  - $A = A$
  - $2A = 4A - 2A$
  - $3A = 4A - A$

- $4A$ is simply $A$ shifted 2 to the left, so this is the same as adding $A$ to the next partial sum

<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>
Booth Recoding Example

- For first bit pair, implied $B_{k-1} = 0$
- First pair is 11, $B_{k-1} = 0$
  - Perform 4A – A
  - Put down –A for this partial product

For first bit pair assume previous pair is 00

\[
\begin{array}{c}
010101 \\
\times 01101100 \\
\hline
11010101 \\
-010101 \\
\hline
01110100
\end{array}
\]
Booth Recoding Example

- For first bit pair, implied $B_{k-1}=0$
- First pair is 11, $B_{k-1}=0$
  - Perform $4A - A$
  - Put down $-A$ for this partial product
- Next pair is 10, $B_{k-1}=1$
  - Perform $4A - 2A$
  - Put down $-A$ ($-2A + A$ from last partial)

For first bit pair assume previous pair is 00

\[
\begin{array}{c}
010101 \\
\times 01101100 \\
\hline
\end{array}
\]

\[
11[0] \quad \text{Sub A} \quad 10[1] \quad \text{Sub A}
\]

\[
-010101 \\
-010101
\]
Booth Recoding Example

- For first bit pair, implied $B_{k-1} = 0$
- First pair is 11, $B_{k-1} = 0$
  - Perform $4A - A$
  - Put down $-A$ for this partial product
- Next pair is 10, $B_{k-1} = 1$
  - Perform $4A - 2A$
  - Put down $-A$ ($-2A + A$ from last partial)
- Last pair 01, $B_{k-1} = 1$
  - Perform $+2A$

For first bit pair assume previous pair is 00

\[
\begin{array}{c}
\text{010101} \\
\times \text{01101100} \\
\hline
\end{array}
\]

\[
\begin{array}{c}
11[0] \quad \text{Sub A}
\end{array}
\]

\[
\begin{array}{c}
10[1] \quad \text{Sub A}
\end{array}
\]

\[
\begin{array}{c}
01[1] \quad \text{Add 2A}
\end{array}
\]

\[
\begin{array}{c}
\text{-010101} \\
\text{-010101} \\
\text{+010101} \\
\hline
\text{01000110111}
\end{array}
\]
Baugh-Wooley Multiplier

• Booth recoding does not really work well for signed multiplication
Baugh-Wooley Multiplier

- Must sign extend and subtract last partial for signed multiplication

\[
\begin{array}{cccccccccccccc}
& a_3 & a_2 & a_1 & a_0 \\
\times & b_3 & b_2 & b_1 & b_0 \\
\hline
& a_3 b_0 & a_3 b_0 & a_3 b_0 & a_3 b_0 \\
+ & a_3 b_1 & a_3 b_1 & a_3 b_1 & a_3 b_1 \\
+ & a_3 b_2 & a_3 b_2 & a_3 b_2 & a_3 b_2 \\
- & a_3 b_3 & a_3 b_3 & a_3 b_3 & a_3 b_3 \\
\hline
& c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
\end{array}
\]
Baugh-Wooley Multiplier

- Must sign extend and subtract last partial for signed multiplication
- Can remove sign extension by adding a 1 at the MSB of each partial product

\[
\begin{align*}
&\phantom{+} a_3a_2a_1a_0 \\
&\times b_3b_2b_1b_0 \\
&+ a_3b_0 a_3b_0 a_3b_0 a_3b_0 a_3b_0 a_2b_0 a_1b_0 a_0b_0 \\
&+ a_3b_1 a_3b_1 a_3b_1 a_3b_1 a_2b_1 a_1b_1 a_0b_1 \\
&+ a_3b_2 a_3b_2 a_3b_2 a_3b_2 a_2b_2 a_1b_2 a_0b_2 \\
&- a_3b_3 a_3b_3 a_3b_3 a_3b_3 a_2b_3 a_1b_3 a_0b_3
\end{align*}
\]

\[+1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1\]

\[c_7 \quad c_6 \quad c_5 \quad c_4 \quad c_3 \quad c_2 \quad c_1 \quad c_0\]
Baugh-Wooley Multiplier

- Must sign extend and subtract last partial for signed multiplication
- Can remove sign extension by adding a 1 at the MSB of each partial product
- Remember to subtract this constant at the end!
Baugh-Wooley Multiplier

- Must sign extend and subtract last partial for signed multiplication
- Can remove sign extension by adding a 1 at the MSB of each partial product
- Remember to subtract this constant at the end!
- Subtraction at the end can be re-represented

\[
\begin{array}{ccccccccc}
 a_3 & a_2 & a_1 & a_0 \\
 \times & b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]

\[
\begin{array}{ccccccccc}
 a_3 b_0 & a_2 b_0 & a_1 b_0 & a_0 b_0 \\
+ \quad a_3 b_1 & a_2 b_1 & a_1 b_1 & a_0 b_1 \\
+ \quad a_3 b_2 & a_2 b_2 & a_1 b_2 & a_0 b_2 \\
+ \quad a_3 b_3 & a_2 b_3 & a_1 b_3 & a_0 b_3 & 1 \\
\end{array}
\]

\[
2's \ complement \quad -A = \sim A + 1
\]

\[
\begin{array}{ccccccc}
 c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
\end{array}
\]
Baugh-Wooley Multiplier

- Must sign extend and subtract last partial for signed multiplication
- Can remove sign extension by adding a 1 at the MSB of each partial product
- Remember to subtract this constant at the end!
- Subtraction at the end can be re-represented

$$\begin{align*}
a_3 & a_2 a_1 a_0 \\
x & b_3 b_2 b_1 b_0
\end{align*}$$

\[
\begin{array}{cccccccc}
+ & a_3 b_0 & a_2 b_0 & a_1 b_0 & a_0 b_0 \\
+ & a_3 b_1 & a_2 b_1 & a_1 b_1 & a_0 b_1 \\
+ & a_3 b_2 & a_2 b_2 & a_1 b_2 & a_0 b_2 \\
+ & a_3 b_3 & a_2 b_3 & a_1 b_3 & a_0 b_3 \\
+ & 1 & 1 & 1 & 1
\end{array}
\]

2's complement

\[-A = \sim A + 1\]
Appendix A

Why the sign extension can be ignored in Baugh–Wooley
Baugh-Wooley Multiplier

- Consider both the case of a negative (2’s complement) and positive number

101001

001001
Baugh-Wooley Multiplier

- Consider both the case of a negative (2’s complement) and positive number
- Sign extend by a few bits

\[
\begin{align*}
111111111111111101001 \\
000000000000000001001
\end{align*}
\]
Baugh-Wooley Multiplier

- Consider both the case of a negative (2’s complement) and positive number
- Sign extend by a few bits
- Add 1 at the original sign bit

\[
\begin{array}{c}
111111111111111101001 \\
+1
\end{array}
\]

\[
\begin{array}{c}
000000000000000001001 \\
+1
\end{array}
\]
Baugh-Wooley Multiplier

- In the positive case
  - Extension all 0, can ignore all except inverted sign bit
- In the negative case
  - 1 carries to next sign extension bit
  - Carry chains all the way until all sign extension bits are 0
  - Drop carry out (won’t affect final sum)
Baugh-Wooley Multiplier

- In the positive case
  - Extension all 0, can ignore all except inverted sign bit

- In the negative case
  - 1 carries to next sign extension bit
  - Carry chains all the way until all sign extension bits are 0
  - Drop carry out (won’t affect final sum)

\[
\begin{align*}
11111111 \quad &00000000 \quad 01001 \\
\leftarrow +1 &
\end{align*}
\]