Discussion Section 11

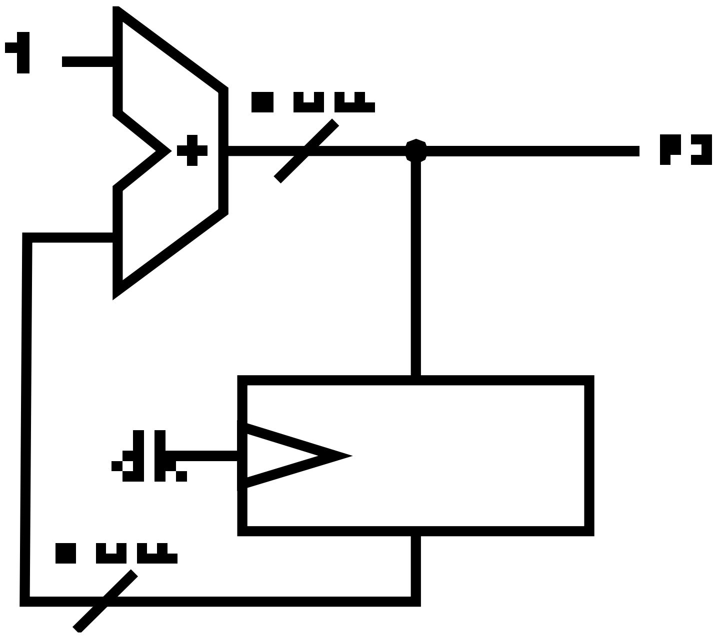
Sean Huang

April 16, 2021

Berkeley
UNIVERSITY OF CALIFORNIA

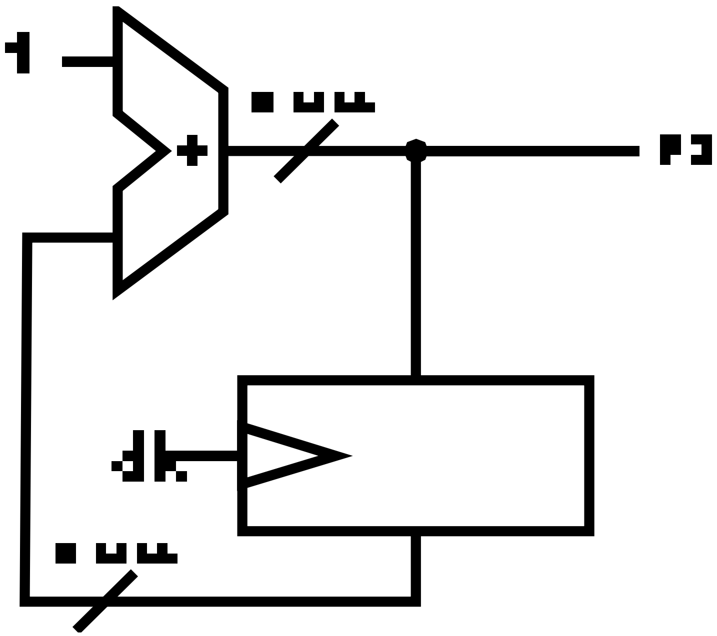
Counter Blocks

- 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?



Berkeley
UNIVERSITY OF CALIFORNIA

Counter Blocks

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

a	b	c	d	a'	b'	c'	d'
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

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

B_{k+1}	B_k	B_{k-1}	Action
0	0	0	Add 0
0	0	1	Add A
0	1	0	Add A
0	1	1	Add 2A
1	0	0	Sub 2A
1	0	1	Sub A
1	1	0	Sub A
1	1	1	Add 0

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

B_{k+1}	B_k	B_{k-1}	Action
0	0	0	Add 0
0	0	1	Add A
0	1	0	Add A
0	1	1	Add 2A
1	0	0	Sub 2A
1	0	1	Sub A
1	1	0	Sub A
1	1	1	Add 0

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

11[0] Sub A

```
   010101
 × 01101100
 ---------
 -010101
```

Booth Recoding Example

For first bit pair assume previous pair is 00

- 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)

```
                          010101
                      × 01101100
                      ----------
11[0]   Sub A           -010101
10[1]   Sub A         -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

```
                          010101
                       × 01101100
                       ----------
11[0]   Sub A            -010101
10[1]   Sub A          -010101
01[1]   Add 2A       +010101
                     ------------
                      01000110111
```

Berkeley
UNIVERSITY OF CALIFORNIA

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

$$
\begin{array}{cccccccccc}
 & & & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & a_3b_0 & a_3b_0 & a_3b_0 & a_3b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
+ & a_3b_1 & a_3b_1 & a_3b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
+ & a_3b_2 & a_3b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
- & a_3b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0
\end{array}
$$

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

$$
\begin{array}{rcccccccc}
 & & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & a_3b_0 & a_3b_0 & a_3b_0 & a_3b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & & & +1 & & & \\
+ & a_3b_1 & a_3b_1 & a_3b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & & & +1 & & & & \\
+ & a_3b_2 & a_3b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 & & & +1 & & & & & \\
- & a_3b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
 & & +1 & & & & & & \\
\hline
 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0
\end{array}
$$

Berkeley
UNIVERSITY OF CALIFORNIA

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!

$$
\begin{array}{ccccccccc}
 & & & & & & & & a_3a_2a_1a_0 \\
 & & & & & & & & \times\ b_3b_2b_1b_0 \\
\hline
 & & & & & \overline{a_3b_0} & a_2b_0 & a_1b_0 & a_0b_0 \\
+ & & & & \overline{a_3b_1} & a_2b_1 & a_1b_1 & a_0b_1 & \\
+ & & & \overline{a_3b_2} & a_2b_2 & a_1b_2 & a_0b_2 & & \\
- & & \overline{a_3b_3} & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
- & & 1 & 1 & 1 & 1 & & & \\
\hline
 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0
\end{array}
$$

Berkeley
UNIVERSITY OF CALIFORNIA

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

$$
\begin{array}{ccccccccc}
 & & & & & & & & a_3a_2a_1a_0 \\
 & & & & & & & & \times\ b_3b_2b_1b_0 \\
\hline
 & & & & & \overline{a_3b_0} & a_2b_0 & a_1b_0 & a_0b_0 \\
+ & & & & \overline{a_3b_1} & a_2b_1 & a_1b_1 & a_0b_1 & \\
+ & & & \overline{a_3b_2} & a_2b_2 & a_1b_2 & a_0b_2 & & \\
+ & & \overline{a_3b_3} & \overline{a_2b_3} & \overline{a_1b_3} & \overline{a_0b_3} & & & \\
+ & & & & & 1 & & & \\
- & & 1 & 1 & 1 & 1 & & & \\
\hline
 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0
\end{array}
$$

2's complement
-A = ~A + 1

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

$$
\begin{array}{ccccccccc}
 & & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & \overline{a_3b_0} & a_2b_0 & a_1b_0 & a_0b_0 & \\
+ & & & \overline{a_3b_1} & a_2b_1 & a_1b_1 & a_0b_1 & & \\
+ & & \overline{a_3b_2} & a_2b_2 & a_1b_2 & a_0b_2 & & & \\
+ & \overline{a_3b_3} & \overline{a_2b_3} & \overline{a_1b_3} & \overline{a_0b_3} & & & & \\
+ \; 1 & & & & 1 & & & & \\
\hline
c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 &
\end{array}
$$

2's complement
-A = ~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

Berkeley
UNIVERSITY OF CALIFORNIA

Baugh-Wooley Multiplier

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

11111111111111101001

00000000000000001001

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

11111111111111101001

+1

00000000000000001001

+1

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)

+1

1111111111111110001001

101001

+1

Berkeley UNIVERSITY OF CALIFORNIA

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)

←+1

1111111100000000001001

101001

+1