Booth-Encoded Multiplier

For Advanced VLSI Design
Fall 2002
台大電機系吳安宇教授
Outline

- Review of Adders
- Array Multiplier
- Canonical Multiplier Recoding
  - Radix-2 SD Representation
  - Canonical Recording Algorithm
  - Advantage in performing multiplication
  - Radix-4 SD Representation
- String Recording and Booth Multiplier
  - String Property
  - Converting to Radix-4 Notation
  - Operation in high-speed Multiplier
**Conventional Full Adder**

\[
\begin{align*}
  c_{i+1} &= x_i \cdot y_i + c_i \cdot (x_i \oplus y_i) \\
  s_i &= x_i \oplus y_i \oplus c_i
\end{align*}
\]

**Multiplexer-based Manchester adder**

Define:  
\[
g_i = x_i \cdot y_i \quad p_i = x_i \oplus y_i \quad \oplus: \text{XOR}
\]

(module A)

\[
\begin{align*}
  c_{i+1} &= g_i + p_i \cdot c_i = p_i g_i + p_i \cdot c_i \\
  s_i &= p_i \oplus c_i = p_i c_i + p_i \cdot c_i
\end{align*}
\]

(module B)

(module A)

\[
\begin{align*}
  g_i \quad c_i
\end{align*}
\]

\[
\begin{align*}
  MUX \quad p_i \quad \overline{p_i}
\end{align*}
\]

(module B)

\[
\begin{align*}
  MUX \quad c_i
\end{align*}
\]
Fig. E.1 Block diagram of a Manchester adder.
8-bits multiplexer-based Manchester adder
(Appendix E)
**Carry Look-ahead Adders**

\[
\begin{align*}
  c_{i+1} &= g_i + p_i \cdot c_i \\
  s_i &= p_i \oplus c_i \\
  c_1 &= c_1 \\
  c_2 &= g_1 + p_1 \cdot c_1 \\
  s_1 &= p_1 \oplus c_1 \\
  c_3 &= g_2 + p_2 \cdot c_2 \\
  &= g_2 + p_2 \cdot (g_1 + p_1 \cdot c_1) \\
  &= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot c_1 \\
  s_2 &= p_2 \oplus c_2 \\
  &= p_2 \oplus (g_1 + p_1 \cdot c_1)
\end{align*}
\]

Diagram:
- Generate \( S_2 \) without \( C_2 \)
- No carry propagation delay
**Carry-Select Adders (8x8 bits)**

- Critical path = 6$t_{mux}$
- 29 Multiplexers
- 8 And gates
Carry-Save Adders \((a_i + b_i + C_i = 2C_{i+1} + S_i)\)
Carry-Save and Carry Propagate Adder (CSA/CPA)
Multi-input Carry-Save Adders

- 4-input CSA tree
- 5-input CSA tree
- 6-input CSA tree
Array Multiplier
Array Multiplier

Notation:

\[ X = \sum_{i=0}^{m-1} x_i \cdot 2^i \quad Y = \sum_{i=0}^{n-1} y_i \cdot 2^i \]

Product:

\[ P = X \times Y = \left( \sum_{i=0}^{m-1} x_i 2^i \right) \cdot \left( \sum_{j=0}^{n-1} y_j 2^j \right) \]

\[ = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left( x_i y_j \right) 2^{i+j} \]

\[ = \sum_{k=0}^{m+n-1} P_k 2^k \]
Array Multiplier (cont.)

**FIGURE 8.35** Array multiplier cell

**TABLE 8.2** 4-bit Multiplier Partial Products

<table>
<thead>
<tr>
<th></th>
<th>X3</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
<th>Y3</th>
<th>Y2</th>
<th>Y1</th>
<th>Y0</th>
<th>Multiplicand</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3Y0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>Y0</td>
<td>X3Y0</td>
<td>X0Y0</td>
</tr>
<tr>
<td>X3Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X3</td>
<td>X2</td>
<td>Y1</td>
<td>Y1</td>
<td>X3Y1</td>
<td>X0Y1</td>
</tr>
<tr>
<td>X3Y2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X3</td>
<td>Y2</td>
<td>X1</td>
<td>Y2</td>
<td>X3Y2</td>
<td>X0Y2</td>
</tr>
<tr>
<td>X3Y3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X3</td>
<td>Y2</td>
<td>Y3</td>
<td>Y3</td>
<td>X3Y3</td>
<td>X0Y3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P7</th>
<th>P6</th>
<th>P5</th>
<th>P4</th>
<th>P3</th>
<th>P2</th>
<th>P1</th>
<th>P0</th>
<th>Product</th>
</tr>
</thead>
</table>
4x4 bits Array Multiplier Structure
Array Multiplier (cont.)

- $n \times n$ multiplier: $n(n-2)$ Full adders
  - $n$ Half adders
  - $n^2$ AND gates

- Worst-case Delay: $(2n + 1) \tau_g$
  
  where $\tau_g$: worst-case adder delay

- It can be drawn as a “Squared Array” for better full-custom layout
Squared Version: for full custom Layout
Pipelined Serial/parallel multiplier

Serial Stream
LSB first
X

Extra delay

Serial/parallel multiplier

Partial Sum In

Pipelined Serial/parallel multiplier
Canonical Multiplier Recoding
Radix-2 SD Representation

- Signed Digit (SD), Radix-2 allows \{ \overline{1}, 0, 1 \}

<table>
<thead>
<tr>
<th>Signed-Digit Vectors</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0 \ 0 \ 0 \ 0 \ 1 \ 1))_2</td>
<td>(2 + 1 = 3)</td>
<td>2</td>
</tr>
<tr>
<td>((0 \ 0 \ 0 \ 1 \ 0 \ \overline{1}))_2</td>
<td>(4 - 1 = 3)</td>
<td>2</td>
</tr>
<tr>
<td>((0 \ 0 \ 1 \ \overline{1} \ 0 \ \overline{1}))_2</td>
<td>(8 - 4 - 1 = 3)</td>
<td>3</td>
</tr>
<tr>
<td>((0 \ \overline{1} \ \overline{1} \ 0 \ \overline{1}))_2</td>
<td>(16 + 8 - 4 - 1 = 3)</td>
<td>4</td>
</tr>
<tr>
<td>((1 \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1}))_2</td>
<td>(32 - 16 - 8 - 4 - 1 = 3)</td>
<td>5</td>
</tr>
<tr>
<td>((0 \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1}))_2</td>
<td>(8 - 4 - 2 + 1 = 3)</td>
<td>4</td>
</tr>
<tr>
<td>((0 \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1}))_2</td>
<td>(16 - 8 - 4 - 2 + 1 = 3)</td>
<td>5</td>
</tr>
<tr>
<td>((1 \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1} \ \overline{1}))_2</td>
<td>(32 - 16 - 8 - 4 - 2 + 1 = 3)</td>
<td>6</td>
</tr>
</tbody>
</table>
Radix-2 SD Representation (cont.)

- A minimal SD vector \( D = (D_{n-1}, D_{n-2}, \ldots, D_1, D_0) \) that contains no adjacent nonzero digit is called a “Canonical Signed-Digit (CSD)” vector.
  \[
  D_i \times D_{i-1} = 0, \text{ for } 1 \leq i \leq n-1
  \]

- Reitwiesner shows that there is a unique CSD representation for any digit number \( \alpha \) with a fixed word-length \( n \).
  ( criteria: a \( (n+1) \)-digit with a leading zero digit )
Canonical Recording Algorithm

Given an \((n + 1)\)-digit binary vector \(B = B_n B_{n-1} \cdots B_1 B_0\) with \(B_n = 0\) and \(B_i \in \{0, 1\}\) for \(0 \leq i \leq n - 1\). We wish to obtain the \((n + 1)\)-digit canonical SD vector \(D = D_n D_{n-1} \cdots D_1 D_0\) with \(D_i = \{1, 0, 1\}\) and \(D_n = 0\) such that both vectors \(D\) and \(B\) represent the same value.

\[
\alpha = \sum_{i=0}^{n} B_i \times 2^i = \sum_{i=0}^{n} D_i \times 2^i
\]

**Step 1.** Start with the low-order end of \(B\) by setting the index \(i = 0\) and initial carry \(C_0 = 0\).

**Step 2.** Examine two adjacent bits \(B_{i+1}\) and \(B_i\) of vector \(B\) conditioned by the carry-in \(C_i\) and generate the next carry \(C_{i+1}\) according to the same rule of conventional binary arithmetic, that is, \(C_{i+1} = 1\) if and only if there are two or three 1's among the three inputs \(B_{i+1}, B_i,\) and \(C_i\).

**Step 3.** Generate the \(i\)th digit \(D_i\) of vector \(D\) by the following arithmetic equation

\[
D_i = B_i + C_i - 2C_{i+1}
\]

**Step 4.** Increment the index \(i\) by one and check if \(i = n\). Go to Step 2 if no, and halt otherwise.
Canonical Recording Algorithm

\[ D_i = B_i + C_i - 2C_{i+1} \]

<table>
<thead>
<tr>
<th>i</th>
<th>Bi+1</th>
<th>Bi</th>
<th>Ci</th>
<th>Ci+1</th>
<th>Di</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>i1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

e.g. \( B = (0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1) \)  
\( D \) has weight of 4  
\( B \) has weight of 5

\[ D = (0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1) \]  
\( B \in D : \{0,1\} \in \{-1,0,1\} \)
Advantage in performing multiplication

- Since $D_i \in \{-1,0,1\}$
- Only the addition of the multiplicand of $(A)$ or $(-A)$ are required per each cycle.
- Zeros correspond to “shift” in multiplication
- 33% fewer nonzero terms than conventional 2’s complement numbers
CSD Multiplication

A CSD multiplier using linear arrangement of adders to compute $x \times 0.101001001010101$
Radix-4 SD Representation

- Since $D$ must be canonical, $D_i \times D_i + 1 = 0$, only five possible choices of the digit pair $\{D_{i+1}, D_i\}$ are available.
  - $\{10, 01, 00, 01, 10\} \Leftrightarrow \{2, 1, 0, 1, 2\}$
  - excluding $\{11, 11, 11, 11\}$

Note: $\overline{11} = 01$
- $\overline{11} = 01$
- $\overline{11} = 101$
Radix-4 SD Representation (cont.)

Multiplier (B2) = (001010111)

D₂ = (010101001)  (Radix-2)

Radix-4 CSD

D₄ = (0 2 2 2 1)  (Radix-4)

Fast Recoded Multiplier with “two-digits” Shifting Per cycle

Multiplicand (A) 0, +/- A, +/- 2A in 5 cycles

Multiplier (NEG, ZERO, TWO) control signals
String Recording and Booth Multiplier

Column position: \( \ldots, i+k, i+k-1, i+k-2, \ldots, i, i-1 \)

Bit Content: \( \ldots, 0, 1, 1, \ldots, 1, 0 \)

\[ \rightarrow \text{Bit Content: } \ldots, 1, 0, \ldots, 0, 1, 0 \]

\( k \) consecutive 1’s

Add \( (k-1) \) consecutive 0’s and subtract
## String Property

\[ 2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + \ldots + 2^{i+1} + 2^i \]

**Example:**

\[ 001000\overline{100} = 00111100 \]

- **(4-1) 0's**
- **4 1's**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Remark on String Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_i )</td>
<td>( B_{i-1} )</td>
<td>( D_i )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
D_i = \begin{cases} 
0, & \text{if } B_i = B_{i-1} \\
1, & \text{if } B_i < B_{i-1} \\
1, & \text{if } B_i > B_{i-1} 
\end{cases}
\]
String Property (cont.)

with \( B_n = B_{-1} = 0, \quad B = B_{n-1}, B_{n-2}, \ldots, B_1, B_0 \)

\( \Rightarrow \) Binary Vector \( B = (0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0)_2 \)

To String vector \( D = (0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0)_{SD} \)

(non-canonical)
Converting to Radix-4

Radix-2:

\[
(D_{i+1}, D_i) = \{ \overline{10}, \begin{pmatrix} 01 \\ 11 \end{pmatrix}, 00, \begin{pmatrix} 01 \\ 11 \end{pmatrix}, 10 \} \]

⇒ Radix-4:

\[
(F_i) = \{ \overline{2}, \overline{1}, 0, 1, 2 \} \]
Operation in Multiplier (B)

- Shift right partial product, if \( B_i = B_{i-1} \)
- Add “A”, then shift right, if \( B_i < B_{i-1} \)
- Subtract “A”, then shift right, if \( B_i > B_{i-1} \)

\[
D_i = \begin{cases} 
0, & \text{if } B_i = B_{i-1} \\
1, & \text{if } B_i < B_{i-1} \\
\overline{1}, & \text{if } B_i > B_{i-1} 
\end{cases}
\]
Booth’s Multiplier

Figure 5.12 The 4-bit scanning pattern of a 32-bit generalized Booth’s multiplier implemented with carry-save adders as shown in Fig. 5.11.
Booth's Multiplier

Figure 5.11 The schematic logic showing the necessary multiplier decode, multiple generation, and addition using CSA/CPA for a 4-bit scanning generalized Booth's multiplier.