ARITHMETIC CIRCUITS IN CMOS VLSI
Adders

Half-adder symbol and operation.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$s$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Adders (2)

Half-adder logic diagram.
Adders (3)

(a) NAND2 logic

(b) NOR-based network

Alternate half-adder logic networks.
Adders (4)

Full-adder symbol and function table.

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$b_i$</th>
<th>$c_i$</th>
<th>$s_i$</th>
<th>$c_{i+1}$</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>1</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Adders (5)

(a) 2-input array

(b) Sum circuit

(c) Carry circuit

CPL full-adder design.
Adders (6)

(a) Gate-level logic

(b) HA-based design

Full-adder logic networks.
Adders (7)

AOI full-adder logic.
Adders (8)

(a) Standard nFET logic

(b) Mirror circuit

Evolution of carry-out circuit.
Adders (9)

Mirror AOI CMOS full-adder.
Adders (10)

Transmission-gate full-adder circuit.
An $n$-bit adder.
A 4-bit ripple-carry adder.

Adders (12)
Worst-case delay through the 4-bit ripple adder.
Adders (14)

4-bit adder-subtractor circuit.
Adders (15)

A basis of the carry look-ahead algorithm.

\[ c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i) \]
Adders (16)

Logic network for 4-bit CLA carry 1
Adders (17)

Sum calculation using the CLA network.
Adders (18)

nFET logic arrays for the CLA terms.
Adders (19)

Possible uses of the nFET logic arrays in Figure 12.18.
Adders (20)

(a) Series-parallel circuit

(b) Mirror equivalent

Static CLA mirror circuit.
Adders (21)

Static mirror circuit for $c_2$. 
Adders (22)

MODL carry circuit.
### Adders (23)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$b_i$</th>
<th>$p_i$</th>
<th>$g_i$</th>
<th>$k_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>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>
</tbody>
</table>

Propagate, generate, and carry-kill values
Adders (24)

Switching network for the carry-out equation.
Adders (25)

(a) Static circuit

(b) Dynamic circuit

Manchester circuit styles.
Adders (26)

Dynamic Manchester carry chain.
An n-bit adder network.
4-bit lookahead carry generator signals.
Adders (29)

Block lookahead generator logic.
Adders (30)

Multilevel CLA block scheme for a 16-bit adder.
Adders (31)

64-bit CLA adder architecture.
Adders (32)

(a) Carry-skip logic

\[ c_{i+4} + c_i \cdot p_{i, i+3} \]

(b) Generalization

\[ c_{i+k} \rightarrow c_i \]

Carry-skip circuitry.
A 16-bit adder using carry-skip circuits.
A 2-level carry-skip adder.
Adders (35)

A 8-bit carry-select adder.
Adders (35)

(a) Symbol

(b) 3-to-2 reduction

Basis of a carry-save adder.
Adders (36)

Creation of an $n$-bit carry-save adder.
Adders (37)

A 7-to-12 reduction using carry-save adders.
Multipliers

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\downarrow \\
\text{a} \\
\text{b}
\end{array}
\quad
\begin{array}{c}
\times \\
\text{a} \\
\times \\
\text{b}
\end{array}
\quad
a \times b
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a \times b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Bit-level multiplier.
Multipliers (2)

\[
\begin{array}{cccc}
    a_3 & a_2 & a_1 & a_0 & \text{multiplicand} \\
\times & b_3 & b_2 & b_1 & b_0 & \text{multiplier} \\
\hline
    a_3 & a_2 & a_1 & a_0 & a_0 & a_0 & a_0 & a_0 \\
\end{array}
\]

\[
\begin{align*}
    + & \quad a_3 & b_1 & a_2 & b_1 & a_1 & b_1 & a_0 & b_1 \\
    + & \quad a_3 & b_2 & a_2 & b_2 & a_1 & b_2 & a_0 & b_2 \\
    + & \quad a_3 & b_3 & a_2 & b_3 & a_1 & b_3 & a_0 & b_3 \\
\hline
    p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 & \text{product} \\
\end{align*}
\]

Multiplication of two 4-bit words.
Shift register for multiplication or division by a factor of 2.
### Multipliers (4)

Alternate view of multiplication process.

<table>
<thead>
<tr>
<th></th>
<th>( a_3 )</th>
<th>( a_2 )</th>
<th>( a_1 )</th>
<th>( a_0 )</th>
<th>( b_3 )</th>
<th>( b_2 )</th>
<th>( b_1 )</th>
<th>( b_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \times )</td>
<td>( (a_3 \ a_2 \ a_1 \ a_0) \times b_0 )</td>
<td>( (a_3 \ a_2 \ a_1 \ a_0) \times b_1 )</td>
<td>( (a_3 \ a_2 \ a_1 \ a_0) \times b_2 )</td>
<td>( (a_3 \ a_2 \ a_1 \ a_0) \times b_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>( p_7 )</td>
<td>( p_6 )</td>
<td>( p_5 )</td>
<td>( p_4 )</td>
<td>( p_3 )</td>
<td>( p_2 )</td>
<td>( p_1 )</td>
<td>( p_0 )</td>
</tr>
</tbody>
</table>

Multiplicand

Multiplier

\( (a \times b_0) \ 2^0 \)

\( (a \times b_1) \ 2^1 \)

\( (a \times b_2) \ 2^2 \)

\( (a \times b_3) \ 2^3 \)

Product
Multipliers (5)

Using a product register for multiplication.

Product register

\[
\begin{align*}
( a \times b_0 ) & \ 2^0 \\
( a \times b_1 ) & \ 2^1 \\
( a \times b_2 ) & \ 2^2 \\
( a \times b_3 ) & \ 2^3 
\end{align*}
\]
Multipliers (6)

Shift-right multiplication sequence.
Multipliers (7)

Register-based multiplier network.
An array multiplier.

Multipliers (8)
Multipliers (9)

Modularized view of the multiplication sequence.
Multipliers (10)

Details for a 4 x 4 array multiplier.
Multipliers (II)

Clocked input registers.
Multipliers (12)

Initial cell placement for the array.
Multipliers (13)

<table>
<thead>
<tr>
<th>$b_{2k+1}$</th>
<th>$b_{2k}$</th>
<th>$b_{2k-1}$</th>
<th>$E_k$</th>
<th>Effect on sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>+1</td>
<td>add $A$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+1</td>
<td>add $A$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+2</td>
<td>shift $A$ left, add</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>take two's ($A$), shift left, add</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>add two's ($A$)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>add two's ($A$)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>add 0</td>
</tr>
</tbody>
</table>

Summary of booth encoded digit operations.