MULTIPLIERLESS FILTER DESIGN

- Realization of filters without full-fledged multipliers


- Partly based on following papers:

FIR-FILTER DIRECT FORM

- FIR = finite impulse response
- Difference equation:
  \[ y[n] = \sum_{k=0}^{N} b_k \cdot x[n - k] \]

FIR-FILTER TRANSPOSED FORM

- Computationally equivalent to direct form
- Can be obtained by reversing order of final addition (associativity) followed by retiming
- Now, all multiplications share one input
IIR FILTER

- IIR = infinite impulse response
- Difference equation:

\[ y[n] = \sum_{k=1}^{N} a_k \cdot y[n-k] + \sum_{k=0}^{N} b_k \cdot x[n-k] \]

IIR-FILTER DIRECT FORM 1

• Distinguish between:
  - Multiplication of two variables
  - Multiplication of one variable by a constant (scaling) ⇒ opportunities of optimization

• Constants:
  - Can be considered as given
  - Can be specially chosen

• Implementation:
  - One-to-one
  - Resource sharing
  - In software, on processor without hardware multiplier

MULTIPLICATION
ELEMENARY SCHOOL ALGORITHM

\[
\begin{array}{c}
0 & 1 & 1 & 0 \\
\times & 1 & 0 & 0 & 1 \\
\hline
0 & 1 & 1 & 0 \\
+ & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 1 & 0 \\
+ & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 1 & 0 \\
\hline
0 & 1 & 1 & 0 & 1 & 1 & 0 \\
\end{array}
\]


 ARRAY MULTIPLIER

- Array multiplier is an efficient layout of a combinational (parallel-parallel) multiplier.
- Array multipliers may be pipelined to decrease clock period at the expense of latency.

ARRAY MULTIPLIER ORGANIZATION

UNSIGNED 4X4 ARRAY MULTIPLIER

© Sabih H. Gerez, University of Twente, The Netherlands
2'S COMPLEMENT MULTIPLICATION (1)

- An n-bit number $X$, and an m-bit number $Y$:

\[
X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i
\]

\[
Y = -y_{m-1}2^{m-1} + \sum_{i=0}^{m-2} y_i 2^i
\]

2’S COMPLEMENT MULTIPLICATION (2)

- Product:

\[
P = XY = x_{n-1}y_{m-1}2^{m+n-2} + \\
\sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_i y_j 2^{i+j} + \\
-2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i - 2^{m-1} \sum_{i=0}^{n-2} x_i y_{m-1} 2^i
\]

2’S COMPLEMENT MULTIPLICATION (3)

- Note that: $-x \cdot 2^n = -2^n + \bar{x} \cdot 2^n$

- and:

\[
\sum_{i=0}^{k} -2^i = 1 - 2^{k+1}
\]

- Therefore:

\[
-2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i = 2^{n-1} \sum_{i=0}^{m-2} -2^i + 2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i
\]

\[
= -2^{n+m-2} + 2^{n-1} + 2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i
\]

2’S COMPLEMENT MULTIPLICATION (4)

- The product becomes:

\[
P = XY = x_{n-1}y_{m-1}2^{n+m-2} + \\
\sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_i y_j 2^{i+j} - 2^{n+m-1} + 2^{n-2} + 2^{m-2}
\]

+ \[
+2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i + 2^{m-1} \sum_{i=0}^{n-2} x_i y_{m-1} 2^i
\]
BAUGH-WOOLEY MULTIPLIER

- Algorithm for two's-complement multiplication.
- Careful processing of partial products leads to:
  - Array with only additions, no subtractions
  - No hardware for sign extensions in upper left corner
- Achieved by:
  - Negation of some partial products
  - Injection of ones in some array positions

BOOTH MULTIPLIER

- Encoding scheme to reduce number of stages in multiplication.
- Performs two bits of multiplication at once; requires half the stages.
- Each stage is slightly more complex than an adder.

BOOTH ENCODING

- Two's-complement form of multiplier:
  \[ y = -2^n y_n + 2^{n-1} y_{n-1} + 2^{n-2} y_{n-2} + \ldots \]
- Rewrite using \( 2^a = 2^{a+1} - 2^a \):
  \[ y = 2^n(y_{n-1} - y_n) + 2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + 2^{n-3}(y_{n-4} - y_{n-3}) + \ldots \]
  \[ y = 2^{n-1}(2(y_{n-1} - y_n) + (y_{n-2} - y_{n-1})) + 2^{n-2}(2(y_{n-3} - y_{n-2}) + (y_{n-4} - y_{n-3})) + \ldots \]
- Consider first two terms: by looking at three bits of \( y \), we can determine whether to add \( x \), \( 2x \), \( -x \), \( -2x \), or 0 to partial product.
**BOOTH ACTIONS**

<table>
<thead>
<tr>
<th>$y_i$ $y_{i-1}$ $y_{i-2}$</th>
<th>increment $(2(y_{i-1} - y_i) + y_{i-2} - y_{i-1})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1x</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1x</td>
</tr>
<tr>
<td>0 1 1</td>
<td>2x</td>
</tr>
<tr>
<td>1 0 0</td>
<td>-2x</td>
</tr>
<tr>
<td>1 0 1</td>
<td>-1x</td>
</tr>
<tr>
<td>1 1 0</td>
<td>-1x</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

**BOOTH EXAMPLE**

- $x = 011001 (25_{10})$, $y = 101110 (-18_{10})$
- $y_1y_0y_{-1} = 100$, $P_1 = P_0 - (10 \cdot 011001) = 1111001110$. $-2 \cdot 1 \cdot x$
- $y_3y_2y_{1} = 111$, $P_2 = P_1 + 0 = 1111001110$. $0 \cdot 4 \cdot x$
- $y_5y_4y_{3} = 101$, $P_3 = P_2 - 0110010000 = 1100011110 (-45_{10})$. $-1 \cdot 16 \cdot x$

**SCALING: BOUNDS ON ADDITIONS (1)**

- Consider multiplication of $x$ by $71 = 1000111_2$.
- Additions-only solution:
  $71x = (x << 6) + (x << 2) + (x << 1) + x$
  (realized by means of 3 shifts and 3 additions; shifts by a constant costs only wires in hardware)
- Subtractions-only solution:
  $71x = ((x << 7) - x) - (x << 5) - (x << 4) - (x << 3)$
  (realized by means of 4 shifts and 4 subtractions)
SCALING: BOUNDS ON ADDITIONS (2)

- In general, if \( b \) is the number of bits, \( z \) the number of zeros and \( o \) the number of ones (\( b = z + o \)):
  - The additions-only solution requires \( o - 1 \) additions.
  - The subtractions-only solution requires \( z + 1 \) subtractions.
- There is always a solution with at most \( b/2 + O(1) \) additions or subtractions (just take the cheapest of the two solutions).
- The average cost is also \( b/2 + O(1) \).
- Booth encoding has also the same cost.
- Can it be done better?

SIGNED POWER-OF-TWO REPRESENTATION

- Uses three-valued digits instead of binary digits: 0, 1, \( \overline{1} \)
- A \( 1 \) at position \( k \) means a contribution of \( 2^k \) to the final value (as usual).
- A \( \overline{1} \) at position \( k \) means a contribution of \( -2^k \) to the final value.
- Example: \( 101\overline{1}00\overline{1} = 64 + 16 - 8 - 1 = 71 \)

CANONICAL SIGNED-DIGIT (CSD)

- Special case of signed-digit power-of-two, with minimal number of non-zero digits
- Canonical = unique encoding
- When used to minimize additions in constant multiplication, reduces number of operations to \( b/3 + O(1) \) in average, but still \( b/2 + O(1) \) in worst case.
- Example: \( 100100\overline{1} = 64 + 8 - 1 = 71 \)

TWO’S COMPLEMENT TO CSD CONVERSION (1)

- Two’s complement number: \( X = x_{n-1}x_{n-2} \ldots x_1x_0 \)
- Target: \( C = c_{n-1}c_{n-2} \ldots c_1c_0 \)
- Start from LSB and proceed to MSB using table on next slide
- Dummy value: \( x_n = x_{n-1} \)
- Carry-in, initialized to 0.
2’S COMPLEMENT TO CSD CONVERSION (2)

<table>
<thead>
<tr>
<th>carry-in</th>
<th>$x_{i+1}$</th>
<th>$x_i$</th>
<th>carry-out</th>
<th>$c_i$</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Hewlitt & Swarzlander, Table 2

CSD NOT OPTIMAL

- CSD has minimal number of non-zeros, but is still not optimal for the “single constant multiplication” problem.
- How come?

SINGLE-CONSTANT MULTIPLICATION

- Number of operations can be reduced by allowing shifting and adding intermediate results
- Example, goal is to multiply by $45 = 101101_2 = 10\overline{1}0\overline{1}01$

Voronenko & Pueschel, Figure 2

MULTIPLE-CONSTANT MULTIPLICATION

- Even more opportunities for optimization occur when multiple constants can be optimized at the same time (think of the transposed form of a FIR filter).
- Example:
COMPUTATIONAL COMPLEXITY

- The optimization of the implementation for both the single-constant and multiple-constant multiplication problems is NP-complete.
- Powerful heuristics are available.
- Try SPIRAL on-line application:
  [http://spiral.ece.cmu.edu/mcm/gen.html](http://spiral.ece.cmu.edu/mcm/gen.html)

CHOOSING THE COEFFICIENTS

- Until now, the discussion was about implementing filters with given constant coefficients as efficiently as possible.
- It is even more interesting to take cheap implementation as a criterion during filter design. A problem description could e.g. be:
  - Given a number \( T \), construct a filter with at most \( T \) non-zero bits in its set of coefficients while at the same time satisfying the usual criteria such as "bandwidth", "pass band ripple", etc.