MULTIPLIERLESS FILTER DESIGN
Implementation of Digital Signal Processing

Sabih H. Gerez
University of Twente

STATE OF PROJECT MAP

• Please fill the poll:
  – A: Not yet started with Project MAP
  – B: Working on the *pen-and-paper* part of Project MAP
  – C: Working on the *Arx* part of Project MAP
  – D: Done with Project MAP

STATE OF PROJECT GFS

• Please fill the poll:
  – A: Not yet seriously started with Project GFS
  – B: Spent between 1 and 5 hours on Project GFS
  – C: Spent more than 5 hours on Project GFS

MULTIPLIERLESS FILTER DESIGN

• Realization of filters without full-fledged multipliers
• Some slides based on support material by W. Wolf for his book *Modern VLSI Design, 3rd edition*. © W
• Partly based on following papers:
TOPICS

- Multiplier wrap-up:
  - Array multiplier
  - Booth multiplier
- Filter structures: direct, transposed and hybrid forms
- Canonical signed digit
- Optimal single and multiple-constant multiplication
- Choosing coefficients

MULTIPLICATION

- 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
    [ How does that work? ]

ELEMENTARY SCHOOL ALGORITHM

0 1 1 0 multiplicand
x 1 0 0 1 multiplier
0 1 1 0 partial product
+ 0 0 0 0
0 0 1 1 0
+ 0 0 0 0
0 0 0 1 1 0
+ 0 1 1 0
0 1 1 0 1 1 0

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**

```
0 1 1 0
x 1 0 0 1
+ 0 0 0 0
0 0 1 1 0
+ 0 0 1 1 0
+ 0 1 1 0
0 1 1 0 1 1 0
```

- Skew array for rectangular layout

**UNSIGNED 4X4 ARRAY MULTIPLIER**

**ARRAY MULTIPLIER COMPONENTS**

- AND gates
- FULL ADDERS
- HALF ADDERS

```
A -> Full adder -> Sum
B -> Carry in
      -> Carry out
```

```
A -> Half adder -> Sum
B -> Carry in
      -> Carry out
```

- Fast multiplication amounts to reducing the critical path.

- What is the main issue when doing signed multiplications?

**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:

\[ P = XY = x_{n-1}y_{m-1}2^{m+n-2} + \sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_iy_j2^{i+j} + \]

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

---

2’S COMPLEMENT MULTIPLICATION (3)

- Note that: \( -x \cdot 2^n = -2^n + x \cdot 2^n \)
- and: \( \sum_{i=0}^{k} -2^i = 1 - 2^{k+1} \)
- Therefore:

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

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

---

2’S COMPLEMENT MULTIPLICATION (4)

- The product becomes:

\[ P = XY = x_{n-1}y_{m-1}2^{n+m-2} + \]

\[ \sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_iy_j2^{i+j} - 2^{n+m-1} + 2^{n-2} + 2^{m-2} \]

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

---

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
BAUGH-WOOLEY SIGNED 4X4 ARRAY MULTIPLIER

BOOTH ENCODING

- The wanted product: \( x \cdot y \).
- 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 \( 2^a = 2^{a+1} - 2^a \):
  \[
  y = 2^n (y_{n+1} y_n) + 2^{n-2} (y_{n+2} - y_{n+1}) + 2^{n-3} (y_{n+3} - y_{n+2}) + \ldots
  \]
  \[
  y = 2^{n-1} (2(y_{n+1} - y_n) + (y_{n+2} - y_{n+1})) + 2^{n-3} (2(y_{n+3} - y_{n+2}) + (y_{n+4} - y_{n+3})) + \ldots
  \]
  Taking steps of 2
- 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 )</th>
<th>( y_{i-1} )</th>
<th>( y_{i-2} )</th>
<th>\text{increment} (2(y_{i-1} - y_i) + y_{i-2} - y_{i-1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**BOOTH EXAMPLE**

- $x = \text{011001} (25_{10})$, $y = \text{101110} (-18_{10})$.
- $y_3y_2y_1 = 100$, $P_1 = P_0 - (10 \cdot \text{011001}) = 11111001110$. $-2 \cdot 1 \cdot x -50_{10}$
- $y_3y_2y_1 = 111$, $P_2 = P_1 + 0 = 11111001110$. $0 \cdot 4 \cdot x$ $-50_{10}$
- $y_3y_4y_3 = 101$, $P_3 = P_2 \cdot \text{0110010000} = 11000111110 (-450_{10})$. $-50_{10}$ $-400_{10}$ $-1 \cdot 16 \cdot x$

**BOOTH STRUCTURE**

Comparison with array multiplier:
- Depth for partial product generation is half, which should result in faster and smaller solution [not always].
- Some extra overhead for Booth encoding, etc.

**FIR-FILTER DIRECT FORM (1)**

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

**FIR-FILTER DIRECT FORM (2)**

- Use a binary tree structure for the additions:
  $$x[n]$$
  $$\rightarrow T_0 \rightarrow T_0 \rightarrow T_0 \rightarrow T_0 \rightarrow T_0$$
  $$b_0 \times b_1 \times b_2 \times b_{N-1} \times b_N$$
  $$+ + + +$$
  $$y[n]$$

- Where is the critical path?
- How long is it as function of $N$?
**CLASSICAL RETIMING**

- It is allowed to “push delay elements” through a computation:
  - From inputs to outputs or
  - From outputs to inputs
- Compute-and-then-delay is the same as delay-and-then-compute.
- Allowed in cyclic DFGs.

**CUT-SET RETIMING**

- Generalization of classical retiming.
- Cut-set = set of edges that cuts a graph in two when removed.
- Given a cut-set of any DFG, the DFG’s behavior remains unchanged if the same number of delays are added (removed) on incoming edges as are removed (added) on outgoing edges.

**FIR-FILTER DIRECT FORM (3)**

- Reverse order of additions:

**CUT-SET RETIMED FIR-FILTER**

- Reverse order of additions:
**FIR-FILTER TRANSPOSED FORM**

- Computationally equivalent to direct form
- Can be obtained by systematically applying cut-set retiming.
- Now, all multiplications share one input

\[ x[n] \rightarrow b_0 \times b_1 \times b_2 \times b_{N-1} \times b_N \rightarrow y[n] \]

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

---

**IIR FILTER**

- IIR = infinite impulse response
- Difference equation:

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

---

**FIR FILTER HYBRID FORM**

- The direct-form-implementation has all its delays in the input line.
- The transposed-form implementation has all delays on the output line.
- Hybrid-form implementation has part of the delays in the input line and part on the output line. See paper by Aksoy et al. for more details.
### IIR-FILTER TRANSPOSED FORM

![Diagram of IIR filter transposed form](image)

### 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 $\overline{1}$ 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: $X = x_n-1x_{n-2} \ldots x_1x_0$
- 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 (sign extension): $x_n = x_{n-1}$
- 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 & Swartzlander, 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 $45 = 101101_2 = 1010101$

![Diagram](image)

$65x = x + 64x$
$49x = 65x - 16x$
$45x = 49x - 4x$

$9x = 8x + x$
$45x = 5(9x) = 9x + 4(9x)$

3x add/sub

**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:

![Diagram](image)

$9x = 8x + x$
$23x = 32x - 9x$
$81x = 8(9x) + 9x$

2x add/sub

**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)

**CONSTANT MATRIX-VECTOR MULT. (1)**

Applications in hybrid implementations of FIR filters

$y_1 = 11x_1 + 17x_2$
$y_2 = 19x_1 + 33x_2$

Unoptimized: 8 add/sub

Aksoy et al., Figure 3
CHOOSING THE COEFFICIENTS

- Until now, the discussion was about implementing filters with given constant coefficients as efficiently as possible.
- Classical approach starts from floating-point coefficients as e.g. computed in Matlab and a “blind” fixed-point conversion.
- 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.
- See e.g. tools at: https://at1x.nl/asic-tools/