Lecture 8

FPGA Multipliers

Radix 2 Sequential Multipliers
Required Reading

Behrooz Parhami,
Computer Arithmetic: Algorithms and Hardware Design

Chapter 9, Basic Multiplication Scheme
Chapter 10, High-Radix Multipliers
Chapter 12.3, Bit-Serial Multipliers
Chapter 12.4, Modular Multipliers
FPGA Multipliers
Notation

Y  Multiplicand  \( Y_{k-1} Y_{k-2} \cdots Y_1 Y_0 \)

X  Multiplier  \( x_{m-1} x_{m-2} \cdots x_1 x_0 \)

P  Product (Y \cdot X)  \( p_{m+k-1} p_{m+k-2} \cdots p_2 p_1 p_0 \)

If multiplicand and multiplier are of different sizes, usually multiplier has the smaller size.
Xilinx FPGA Implementation
Equations

\[ Z = (2x_{m-1}+x_{m-2}) \cdot Y \cdot 2^{m-2} + \ldots + (2x_{i+1}+x_i) \cdot Y \cdot 2^i + \ldots + (2x_3+x_2) \cdot Y \cdot 2^2 + (2x_1+x_0) \cdot Y \cdot 2^0 \]

\[ (2x_{i+1}+x_i) \cdot Y = p_{i(k+1)}p_{i(k)}p_{i(k-1)}\ldots p_{i2}p_{i1}p_{i0} \]

\[ p_{ij} = x_i \cdot y_j \ \text{xor} \ x_{i+1} \cdot y_{j-1} \ \text{xor} \ c_j \]

\[ c_{j+1} = (x_i \cdot y_j)(x_{i+1} \cdot y_{j-1}) + (x_i \cdot y_j) \cdot c_j + (x_{i+1} \cdot y_{j-1}) \cdot c_j \]

\[ c_0 = c_1 = 0 \]
Modified Basic Cell

Xilinx FPGA Implementation

\[ x_{i+1}, x_i, c_j+1, y_j, y_{j-1}, p_{ij}, c_j \]
Modified Basic Cell
Xilinx FPGA Implementation

LUT: \( x_i \cdot y_j \) \( \text{xor} \) \( x_{i+1} \cdot y_{j-1} \)

\[
p_{ij} = x_i \cdot y_j \text{xor} x_{i+1} \cdot y_{j-1} \text{xor} c_j
\]

\[
c_{j+1} = (x_i \cdot y_j)(x_{i+1} \cdot y_{j-1}) + (x_i \cdot y_j) \cdot c_j + (x_{i+1} \cdot y_{j-1}) \cdot c_j
\]
Xilinx FPGA Multiplier
Radix 2
Sequential Multipliers
Notation

a    Multiplicand       $a_{k-1}a_{k-2} \ldots a_1a_0$

x    Multiplier         $x_{k-1}x_{k-2} \ldots x_1x_0$

p    Product (a · x)     $p_{2k-1}p_{2k-2} \ldots p_2p_1p_0$

If multiplicand and multiplier are of different sizes, usually multiplier has the smaller size.
Multiplication of two 4-bit unsigned binary numbers in dot notation

Number of partial products = number of bits in multiplier $x$
Bit-width of each partial product = bit-width of multiplicand $a$
Basic Multiplication Equations

\[ p = a \cdot x \]

\[ x = \sum_{i=0}^{k-1} x_i \cdot 2^i \]

\[ p = a \cdot x = \sum_{i=0}^{k-1} a \cdot x_i \cdot 2^i = x_0 a 2^0 + x_1 a 2^1 + x_2 a 2^2 + \ldots + x_{k-1} a 2^{k-1} \]
Shift/Add Algorithm
Right-shift version
Shift/Add Algorithms
Right-shift algorithm

\[ p = a \cdot x = x_0a2^0 + x_1a2^1 + x_2a2^2 + \ldots + x_{k-1}a2^{k-1} = \]

\[ = \underbrace{((0 + x_0a2^k)/2 + x_1a2^k)/2 + \ldots + x_{k-1}a2^k)/2}_{k \text{ times}} = \]

\[ p^{(0)} = 0 \]

\[ p^{(j+1)} = (p^{(j)} + x_ja2^k) / 2 \quad j=0..k-1 \]

\[ p = p^{(k)} \]
Sequential shift-and-add multiplier for right-shift algorithm
Right-shift multiplication algorithm:

Example
Area optimization for the sequential shift-and-add multiplier with the right-shift algorithm

Adder's carry-out

Adder's sum

Partial Product

Unused part of the multiplier

To adder

To mux control
Shift/Add Algorithms
Right-shift algorithm: multiply-add

\[ p^{(0)} = y2^k \]

\[ p^{(j+1)} = \frac{(p^{(j)} + x_j a 2^k)}{2} \quad j=0..k-1 \]

\[ p = p^{(k)} \]

\[ = \underbrace{(\ldots((y2^k + x_0a2^k)/2 + x_1a2^k)/2 + \ldots + x_{k-1}a2^k)/2 =}_{k \text{ times}} \]

\[ = y + x_0a2^0 + x_1a2^1 + x_2a2^2 + \ldots + x_{k-1}a2^{k-1} = y + a \cdot x \]
Signed Multiplication

- Previous sequential multipliers are for unsigned multiplication
- For signed multiplication:
  - assume sign-extended operation for \( p(j) + x_ja \)
  - if 2's complement multiplier is POSITIVE
    right-shift sequential algorithms (shift-add) will work directly
  - if 2's complement multiplier is NEGATIVE than we must use
    "negative weight" for \( x_{k-1} \) and subtract \( x_{k-1}a \) in the last cycle
- Slight increase in area due to control and one-bit sign extension on
  inputs of adder
  - Unsigned: \( k \) bit number + \( k \) bit number \( \Rightarrow \) \( k+1 \) bit number
  - Signed: \( k+1 \) bit sign extended number + \( k+1 \) bit sign extended
    number \( \Rightarrow \) \( k+1 \) bit number
Sequential multiplication of 2’ s-complement numbers with right shifts (positive multiplier)

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>p⁰</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>+xa</td>
<td>10110</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p⁽¹⁾</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>110110</td>
<td></td>
</tr>
<tr>
<td>+xa</td>
<td>10110</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p⁽²⁾</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1100010</td>
<td></td>
</tr>
<tr>
<td>+xa</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p⁽³⁾</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11100010</td>
<td></td>
</tr>
<tr>
<td>+xa</td>
<td>10110</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p⁽⁴⁾</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>110010010</td>
<td></td>
</tr>
<tr>
<td>+xa</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p⁽⁵⁾</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1110010010</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>1110010010</td>
<td></td>
</tr>
</tbody>
</table>
```
Sequential multiplication of 2’s-complement numbers with right shifts (negative multiplier)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|}
\hline
p^{(0)} & 0 0 0 0 0 \\
+x_0a & 1 0 1 1 0 \\
\hline
2p^{(1)} & 1 1 0 1 1 0 \\
p^{(1)} & 1 1 0 1 1 0 \\
+x_1a & 0 0 0 0 0 \\
\hline
2p^{(2)} & 1 1 1 0 1 1 0 \\
p^{(2)} & 1 1 1 0 1 1 0 \\
+x_2a & 1 0 1 1 0 \\
\hline
2p^{(3)} & 1 1 0 0 1 1 1 0 \\
p^{(3)} & 1 1 0 0 1 1 1 0 \\
+x_3a & 0 0 0 0 0 \\
\hline
2p^{(4)} & 1 1 1 0 0 1 1 1 0 \\
p^{(4)} & 1 1 1 0 0 1 1 1 0 \\
+(−x_4a) & 0 1 0 1 0 \\
\hline
2p^{(5)} & 0 0 0 1 1 0 1 1 1 0 \\
p^{(5)} & 0 0 0 1 1 0 1 1 1 0 \\
\hline
\end{array}
\]
Shift/Add Algorithm
Left-shift version
Shift/Add Algorithms

Left-shift algorithm

\[ p = a \cdot x = x_0a2^0 + x_1a2^1 + x_2a2^2 + \ldots + x_{k-1}a2^{k-1} = \]

\[ = (\ldots((0 \cdot 2 + x_{k-1}a) \cdot 2 + x_{k-2}a) \cdot 2 + \ldots + x_1a) \cdot 2 + x_0a = \]

k times

\[ p^{(0)} = 0 \]

\[ p^{(j+1)} = (p^{(j)} \cdot 2 + x_{k-1-j}a) \quad j=0..k-1 \]

\[ p = p^{(k)} \]
Sequential shift-and-add multiplier for left-shift algorithm

Left shifts are not as efficient for two's complement because must sign extend multiplicand by $k$ bits.
### Left-shift multiplication algorithm: Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-shift algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>p(^{(0)})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2p(^{(0)})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>+x(_3)a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>p(^{(1)})</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2p(^{(1)})</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+x(_2)a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>p(^{(2)})</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2p(^{(2)})</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+x(_1)a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>p(^{(3)})</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2p(^{(3)})</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>+x(_0)a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>p(^{(4)})</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Shift/Add Algorithms
Left-shift algorithm: multiply-add

\[ p^{(0)} = y2^{-k} \]

\[ p^{(j+1)} = (p^{(j)} \cdot 2 + x_{k-(j+1)}a) \quad j=0..k-1 \]

\[ p = p^{(k)} \]

\[ = \left( \left( (y2^{-k} \cdot 2 + x_{k-1}a) \cdot 2 + x_{k-2}a \right) \cdot 2 + ... + x_{1}a \right) \cdot 2 + x_{0}a = \]

\[ = y + x_{k-1}a2^{k-1} + x_{k-2}a2^{k-2} + ... + x_{1}a2^{1} + x_{0}a = y + a \cdot x \]
Shift/Add Algorithm
Right-shift version
with Carry-Save Adder
Sequential shift-and-add multiplier with a carry save adder