Vendor Agnostic, High Performance, Double Precision Floating Point Division for FPGAs

Presented by Xin Fang

Advisor: Professor Miriam Leeser

ECE Department Northeastern University
Outline

- Background
- Algorithm Description
- Components of Divider
- Results & Performance
- Conclusions and Future work
Background

- Floating Point Format
- VFloat Library
- Three Main Algorithms
Floating Point Representation


<table>
<thead>
<tr>
<th>Format</th>
<th>Sign Bit (s)</th>
<th>Exponent Bits (e)</th>
<th>Mantissa Bits (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Precision</td>
<td>1</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>Double Precision</td>
<td>1</td>
<td>11</td>
<td>52</td>
</tr>
</tbody>
</table>

b is the base, s is the sign bit, e is the exponent, c is the mantissa. The value of floating point number is

\[
(-1)^s \times 1.c \times b^{e-\text{bias}}
\]
Floating Point Format (Cont)

\[ (-1)^s \times 1.c \times b^{e-bias} \]

- Mantissa Representation
- Exponent Bias

<table>
<thead>
<tr>
<th>Format</th>
<th>Exponent in IEEE754</th>
<th>Exponent Bias</th>
<th>Real Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Precision</td>
<td>1 to 254</td>
<td>127</td>
<td>-126 to 127</td>
</tr>
<tr>
<td>Double Precision</td>
<td>1 to 2046</td>
<td>1023</td>
<td>-1022 to 1023</td>
</tr>
</tbody>
</table>
VFloat Library

- Vfloat*: library of variable precision floating point units written in VHDL, targeting Altera and Xilinx FPGAs
- Components include floating point arithmetic (addition, subtraction, multiplication, reciprocal, division, square root, accumulation) and format conversion (fix2float).
- Operand can be variable precision floating point number
  Any bitwidth exponent or mantissa is supported.

Reconfigurable Computing Laboratory of Northeastern University
VFloat Library (Cont)

Blackbox for Variable Precision Floating Point Operation:

Components of VFloat can be easily fit into a large project because of the ports Reset, Stall, Ready, Exception_in, Exception_out and Done.
Three Main Approaches

For Division:

1. Digit Recurrence Method
2. Iterative Method
3. Table-based Method
Digit Recurrence Division

Example: SRT Algorithm

- Step 1: Quotient Digit Selection

\[
q_{j+1} = \begin{cases} 
1 & 2 \times W_j \geq 1/2; \\
0 & -1/2 \leq 2 \times W_j < 1/2; \\
-1 & 2 \times W_j < -1/2;
\end{cases}
\]

- Step 2: Residual Recurrence

\[
w[j + 1] = r \times w[j] - d \times q[j + 1]
\]
Iterative Division

Example: Newton Raphson Algorithm

- Step 1: find an approximation of $1/d$ from LUT
- Step 2: Iteration: $X_{(i+1)} = X_i \times (2 - d \times X_i)$

Newton Raphson Algorithm Diagram
**Table-based Division**

- Table-based algorithm* that uses small look-up tables and multipliers
  - Based on Taylor series, but more sophisticated

- Gets good tradeoff between clock cycle latency, maximum frequency and resource utilization by implementing this table-based algorithm

- Double Precision Division in this presentation is based on this algorithm

Outline

- Background
- Algorithm Description
- Components of Divider
- Results & Performance
- Conclusions and Future work
Algorithm Description

- Algorithm Overview
- Mantissa Computation:
  1. Reduction Step
  2. Evaluation Step
  3. Post-processing Step
Algorithm Overview

Top Components:

1. Denormalizer
2. Divider
3. Round to normal
Algorithm Overview (Cont)

- Three parts: sign bit, exponent and mantissa

- **Sign bit**: XOR Logic

- **Exponent**: Subtract the exponent of divisor from that of dividend

- **Mantissa**: where Table-based algorithm applies
Mantissa Computation

- Three steps to compute Mantissa:
  - 1. Reduction Step (To compute M)
  - 2. Evaluation Step (To compute B)
  - 3. Post-processing Step (To compute Result)
1. Reduction Step

After the denormalizer, inputs \((X, Y)\) are the significand of the floating point number with format 1.c each of them has \(m\) bits.
For double precision format, \(m = 53\).

Let \(K = \text{floor}((m+2)/4)+1\)

\(Y(K)\) represents the truncation of \(Y\) to the first \(K\) bits

Let \(R = 1/ Y(K)\);

\(M = R\)
2. Evaluation Step

- Let $B = f(A) = 1/(1+A)$
- Where $A = (Y*R)^{-1}, \quad (-2^{-k} < A < 2^{-k})$
- Let $z = 2^{-k}$, then $A = A_2z^2 + A_3z^3 + A_4z^4 + ...$

<table>
<thead>
<tr>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
</table>

Using Taylor series expansion,

$$B = f(A) = C_0 + C_1 A + C_2 A^2 + C_3 A^3 + C_4 A^4 + ...$$

$$\approx C_0 + C_1 (A_2 z^2 + A_3 z^3 + A_4 z^4)$$

$$+ C_2 (A_2 z^2 + A_3 z^3 + A_4 z^4)$$

$$+ C_3 (A_2 z^2 + A_3 z^3 + A_4 z^4)^3$$

$$+ C_4 (A_2 z^2 + A_3 z^3 + A_4 z^4)^4$$

$$\approx C_0 + C_1 A + C_2 A_2 z^4 + 2C_2 A_2 A_3 z^5 + C_3 A_2^3 z^6$$

$C_i = 1$ when $i$ is even, $C_i = 0$ when $i$ is odd.
Evaluation Step (Cont)

- Simplifying the equation:

\[ B = f(A) \approx 1 - A_2z^2 - A_3z^3 + (-A_4 + A_2^2)z^4 + 2A_2S_3z^5 - A_2^3z^6 \]
\[ \approx (1 - A) + A_2^2z^4 + 2A_2A_3z^5 - A_2^3z^6 \]
3. Post-processing Step

- The result of reciprocal, divider or square root is given by $M \times B$

  M is from the Reduction step
  B is from the Evaluation step
Outline

- Background
- Algorithm Description
- Components of Divider
- Results & Performance
- Conclusions and Future work
Components of Divider
Outline

- Background
- Algorithm Description
- Components of Divider
- Results & Performance
- Conclusions and Future work
Results and Performance

- FPGA Device:
  1. Stratix V device
  2. Virtex 6 device

- Software IDE:
  1. Altera Quartus II 13.0
  2. Xilinx ISE Design Suite 13.4
Altera Stratix V

- Optimized for high performance, high bandwidth applications. (28-ns)
- The Stratix V device family contains GT, GX, GS and E sub-families.
- 5SGXB6 for synthesis.
Altera Stratix V (5SGXB6)

- 597K Logic Elements
- 902K Register
- 2660 M20K (52 Mb)

- 798 18*18 Mul
- 399 27*27 Mul
Xilinx Virtex 6

- High performance programmable silicon
- Three sub-family: LXT, SXT and HXT.
- XC6VLX75T for synthesis.
  - 74,496 Logic Cells
  - 11,640 Slices
  - 1,045 Kb Distributed RAM
  - 288 DSP48E1 Slices
  - 156 36Kb Block RAM
Results & Performance (Double Precision Division)

- For Xilinx:

<table>
<thead>
<tr>
<th>Method</th>
<th>Device</th>
<th>Latency</th>
<th>Max Freq.</th>
<th>Total Latency</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>Virtex 6</td>
<td>14 CC</td>
<td>148 MHz</td>
<td>95 ns</td>
<td>934 Reg, 6957 Slice LUTs</td>
</tr>
<tr>
<td>IP Core</td>
<td>Virtex 6</td>
<td>8 CC</td>
<td>74 MHz</td>
<td>108 ns</td>
<td>722 Reg, 3210 Slice LUTs</td>
</tr>
<tr>
<td>IP Core</td>
<td>Virtex 6</td>
<td>14 CC</td>
<td>117 MHz</td>
<td>120 ns</td>
<td>1188 Reg, 3234 Slice LUTs</td>
</tr>
<tr>
<td>IP Core</td>
<td>Virtex 6</td>
<td>20 CC</td>
<td>192 MHz</td>
<td>104 ns</td>
<td>2035 Reg, 3216 Slice LUTs</td>
</tr>
<tr>
<td>Iterative</td>
<td>Virtex 6</td>
<td>36 CC</td>
<td>275 MHz</td>
<td>131 ns</td>
<td>2097 Slices, 118K BRAM, 28 18*18</td>
</tr>
</tbody>
</table>

Results & Performance (Double Precision Division)

For Altera:

<table>
<thead>
<tr>
<th>Method</th>
<th>Device</th>
<th>Latency</th>
<th>Max Freq.</th>
<th>Total Latency</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our 1st</td>
<td>Stratix V</td>
<td>14 CC</td>
<td>121 MHz</td>
<td>116 ns</td>
<td>818 ALMs, 931 Logic Reg, 11 DSP block</td>
</tr>
<tr>
<td>Our 2nd</td>
<td>Stratix V</td>
<td>16 CC</td>
<td>145 MHz</td>
<td>110 ns</td>
<td>1004 ALMs, 1105 Logic Reg, 13 DSP block</td>
</tr>
<tr>
<td>MegaCore</td>
<td>Stratix V</td>
<td>10 CC</td>
<td>176 MHz</td>
<td>57 ns</td>
<td>525 ALMs, 1247 Logic Reg, 14 DSP block</td>
</tr>
<tr>
<td>MegaCore</td>
<td>Stratix V</td>
<td>24 CC</td>
<td>237 MHz</td>
<td>101 ns</td>
<td>849 ALMs, 1809 Logic Reg, 14 DSP block</td>
</tr>
<tr>
<td>MegaCore</td>
<td>Stratix V</td>
<td>61 CC</td>
<td>332 MHz</td>
<td>184 ns</td>
<td>9379 ALMs, 13493 Logic Reg</td>
</tr>
<tr>
<td>Digital Recur*</td>
<td>Stratix V</td>
<td>36 CC</td>
<td>219 MHz</td>
<td>164 ns</td>
<td>2605 ALMs, 5473 Logic Reg</td>
</tr>
<tr>
<td>Newton Raphson*</td>
<td>Stratix V</td>
<td>18 CC</td>
<td>268 MHz</td>
<td>67 ns</td>
<td>444 ALMs, 823 Logic Reg, 2 M20K, 9 DSP</td>
</tr>
</tbody>
</table>

Outline

- Background
- Algorithm Description
- Components of Divider
- Results & Performance
- Conclusions and Future work
Conclusions

- Double Precision Floating Point Division component using table-based algorithm
- Flexible: Easy to adjust pipeline parameters
- Adaptable: Cross-platform portability
- Good tradeoff between clock cycle latency, maximum frequency and resource utilization
Future Work

- Try different pipelines to find better tradeoffs
- Add more components to the library
- Apply to applications
Thank you!

- Xin Fang  fang.xi@husky.neu.edu
  ECE Department, Northeastern University
- Miriam Leeser  mel@coe.neu.edu
  ECE Department, Northeastern University

- VFloat website: