Correctly rounded floating-point division for DSP-enabled FPGAs

Bogdan Pasca
FPL 29-31 August 2012
Outline

1 Background
   - Floating-point division
   - Division algorithms

2 Faithfully rounded dividers
   - Error analysis walk-through
   - Implementation discussion

3 Correctly rounded dividers
   - Obtaining correctly rounded dividers
   - The cost of correct rounding

4 Results

5 Conclusion
Outline

1 Background
   • Floating-point division
   • Division algorithms

2 Faithfully rounded dividers
   • Error analysis walk-through
   • Implementation discussion

3 Correctly rounded dividers
   • Obtaining correctly rounded dividers
   • The cost of correct rounding

4 Results

5 Conclusion
Let $x, y$ floating-point numbers in format $\mathbb{F}_{WE,WF}$ with:

- $x = (-1)^{s_x} 2^{e_x} 1.f_x$
- $y = (-1)^{s_y} 2^{e_y} 1.f_y$

Let:

$$q = \frac{x}{y} = (-1)^{s_x \oplus s_y} 2^{e_x - e_y} \frac{1.f_x}{1.f_y}$$

where $Q = \frac{X}{Y} = \frac{1.f_x}{1.f_y} \in (1/2, 2)$
Let $x, y$ floating-point numbers in format $\mathbb{F}_{w_E,w_F}$ with:

- $x = (-1)^{s_x}2^{e_x}1.f_x$
- $y = (-1)^{s_y}2^{e_y}1.f_y$

Let:

$$q = \frac{x}{y} = (-1)^{s_x \oplus s_y}2^{e_x-e_y} \frac{1.f_x}{1.f_y}$$

where $Q = \frac{X}{Y} = \frac{1.f_x}{1.f_y} \in (1/2, 2)$

**Fixed-point division is the core of floating-point implementation**
Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long latency $O(wF)$
  - negative routing impact

Functional iterations for approximating $1/Y$ Newton-Raphson or Goldschmidt start with an initial low accuracy approximation. Quadratic convergence requires multipliers and memories. Polynomial approximation for $1/Y$ is a general technique, piecewise-polynomial approximation for range-reduction. Combining techniques: use polynomial approx. for initial approximation, then functional iterations.
Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long latency $\mathcal{O}(wF)$
  - negative routing impact

- **Functional iterations** for approximating $1/Y$
  - Newton-Raphson or Goldschmidt
  - start with an initial low accuracy approximation
  - quadratic convergence
  - require multipliers and memories
Digit-recurrence algorithms
- correctly rounded
- long latency $O(wF)$
- negative routing impact

Functional iterations for approximating $1/Y$
- Newton-Raphson or Goldschmidt
- start with an initial low accuracy approximation
- quadratic convergence
- require multipliers and memories

Polynomial approximation for $1/Y$
- general technique
- piecewise-polynomial approximation for range-reduction
Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long latency $O(wF)$
  - negative routing impact

- **Functional iterations** for approximating $1/Y$
  - Newton-Raphson or Goldschmidt
  - start with an initial low accuracy approximation
  - quadratic convergence
  - require multipliers and memories

- **Polynomial approximation** for $1/Y$
  - general technique
  - piecewise-polynomial approximation for range-reduction

Combining techniques
Use polynomial approx. for initial approximation, then functional iterations
Outline

1 Background
   • Floating-point division
   • Division algorithms

2 Faithfully rounded dividers
   • Error analysis walk-through
   • Implementation discussion

3 Correctly rounded dividers
   • Obtaining correctly rounded dividers
   • The cost of correct rounding

4 Results

5 Conclusion
Let:

- $x, y$ be two floating-point numbers and
- $q^* = \frac{x}{y}$ be the infinitely accurate quotient.
Faithfully rounded dividers
Faithful and correct rounding

Let:

- $x, y$ be two floating-point numbers and
- $q^* = \frac{x}{y}$ be the infinitely accurate quotient.

The IEEE-754 Standard for Floating-Point Arithmetic: $q = \circ(q^*)$

- directed rounding modes + round to nearest mode (2)
- round to nearest, tie breaks to even - most used
Let:

- $x, y$ be two floating-point numbers and 
- $q^* = \frac{x}{y}$ be the infinitely accurate quotient.

The IEEE-754 Standard for Floating-Point Arithmetic: $q = \circ(q^*)$

- directed rounding modes + round to nearest mode (2)
- round to nearest, tie breaks to even - most used

---

**Faithfully rounded dividers**

Faithful and correct rounding

Let:

- $x, y$ be two floating-point numbers and 
- $q^* = \frac{x}{y}$ be the infinitely accurate quotient.

The IEEE-754 Standard for Floating-Point Arithmetic: $q = \circ(q^*)$

- directed rounding modes + round to nearest mode (2)
- round to nearest, tie breaks to even - most used

---

**FR** – faithful rounding

**CR** – correct rounding

floating-point numbers

---

**FR** – faithful rounding

**CR** – correct rounding

floating-point numbers
Faithfully rounded result requires:

\[ E_{\text{total}} = E_{\text{round}} + E_{\text{approx}} \leq 1ulp \]
Faithfully rounded result requires:

\[ E_{\text{total}} = E_{\text{round}} + E_{\text{approx}} \leq 1\,\text{ulp} \]

- \( E_{\text{round}} \) - packing the result to the output format. (1/2ulp for RN)
- \( E_{\text{approx}} \) - sums the method and computational errors. (must be bounded by 1/2ulp)
Sequence of operations for fixed-point division:

- **Infinitely accurate**
  \[ Z = \frac{1}{Y} \]
  \[ Q = Z \times X \]

- **Implemented operations**
  \[ Z' = o\left(\frac{1}{Y}\right) \]
  \[ Q' = Z' \times X \]

The approximation error \( E_{\text{approx}} \):

\[
|Q - Q'| = |ZX - Z'X| \\
\leq |Z - Z'| |X|
\]

as \( X \in [1, 2) \) → 

\[
|Z - Z'| \leq \frac{1}{4} \text{ulp}
\]

but \( Z \in (\frac{1}{2}, 1] \) so a faithful approximation on \( wF + 3 \) bits is required.
Faithfully rounded dividers
Error analysis walk-through

Sequence of operations for fixed-point division:

Infinitely accurate | Implemented operations
\[ Z = 1/Y \] | \[ Z' = \circ(1/Y) \]
\[ Q = Z \times X \] | \[ Q' = Z' \times X \]

The approximation error \( E_{\text{approx}} \):
\[
|Q - Q'| = |ZX - Z'X| \\
= |(Z - Z')X| \\
\leq |Z - Z'| |X|
\]
Faithfully rounded dividers
Error analysis walk-through

Sequence of operations for fixed-point division:

- Infinitely accurate
  \[ Z = 1/Y \]
  \[ Q = Z \times X \]

- Implemented operations
  \[ Z' = o(1/Y) \]
  \[ Q' = Z' \times X \]

The approximation error \( E_{\text{approx}} \):

\[
|Q - Q'| = |ZX - Z'X| \\
= |(Z - Z')X| \\
\leq |Z - Z'| |X|
\]

- as \( X \in [1, 2) \rightarrow |Z - Z'| \leq 1/4 \text{ulp} \) but \( Z \in (1/2, 1] \) so a faithful approximation on wF+3 bits is required.
Higher precisions allow saving DSPs using a truncated multiplier:

\[ Z' = \circ(1/Y) \]
\[ P' = Z' \times X \]
\[ Q' = \text{trunc}(P') \]
Higher precisions allow saving DSPs using a truncated multiplier:

\[ Z' = o(1/Y) \]
\[ P' = Z' \times X \]
\[ Q' = \text{trunc}(P') \]

The approximation error \( E_{\text{approx}} \):

\[
|Q - Q'| = |ZX - \text{trunc}(P')| \\
= |ZX - Z'X + Z'X - \text{trunc}(Z'X)| \\
= |(Z - Z')X + \text{truncerror}\text{for}Z'X| \\
\leq |Z - Z'||X| + |\text{truncerror}\text{for}Z'X|
\]
Faithfully rounded dividers
Error analysis walk-through

Higher precisions allow saving DSPs using a truncated multiplier:

\[
Z' = o(1/Y) \\
P' = Z' \times X \\
Q' = \text{trunc}(P')
\]

The approximation error \( E_{\text{approx}} \):

\[
|Q - Q'| = |ZX - \text{trunc}(P')| \\
= |ZX - Z'X + Z'X - \text{trunc}(Z'X)| \\
= |(Z - Z')X + \text{truncerrorforZ'}X| \\
\leq |Z - Z'| |X| + |\text{truncerrorforZ'}X|
\]

- faithful approx. on \( wF + 4 \) for \( Z' \)
- faithful multiplier on \( wF + 3 \) for \( Z'X \).
The **Newton-Raphson iteration**:

\[ Z_{n+1} = 2Z_n - Z_n^2 Y \]

Two solutions:

1. bootstrap with \( Z_0 \) accurate to \( 2^{-14} \)
2. preform one iteration: \( Z_1 = 2Z_0 - Z_0^2 Y \)
3. 6M20K (StratixV)/ 13M10K (AriaV/CycloneV) + 2DSPs + logic
Faithfully rounded dividers

Single-precision implementation using Newton-Raphson for $Z$

The **Newton-Raphson iteration**:

$$Z_{n+1} = 2Z_n - Z_n^2 Y$$

Two solutions:

1. bootstrap with $Z_0$ accurate to $2^{-14}$
   - preform one iteration: $Z_1 = 2Z_0 - Z_0^2 Y$
   - 6M20K (StratixV)/ 13M10K (AriaV/CycloneV) + 2DSPs + logic

2. bootstrap with $Z_0$ accurate to $2^{-10}$
   - perform two iterations:
     $$Z_1 = 2Z_0 - Z_0^2 Y$$
     $$Z_2 = 2Z_1 - Z_1^2 Y$$
     - 1M20K/1M10K + 4DSPs + logic
     - has a longer latency
Two solutions:

1. bootstrap \( Z_0 \) accurate to \( 2^{-15} \)
2. perform two iterations, \( Z_1, Z_2 \)
3. large memory requirement
Faithfully rounded dividers

**Double-precision implementation using Newton-Raphson for Z**

Two solutions:

1. bootstrap $Z_0$ accurate to $2^{-15}$
2. perform two iterations, $Z_1$, $Z_2$
3. large memory requirement

2. bootstrap with $Z_0$ accurate to $2^{-10}$
2. perform 3 iterations (several optimizations possible)
   - $Z_1 = 2Z_0 - Z_0^2 Y$ 10-bit squarer + 20x53 mult. $\rightarrow 20 \times 27$ (1 DSP)
   - $Z_2 = 2Z_1 - Z_1^2 Y$ 20-bit squarer + 40x53 mult. (4 DSPs but can be red. to 3)
   - $Z_3 = 2Z_2 - Z_2^2 Y$ 40-bit squarer (3DSPs) + 54x53 mult (4DSPs)

14 DSPs + 1 memory block for the inverse
Faithfully rounded dividers
Implementation using Polynomial Approximation

Single-precision

- degree 2 polynomial on 256 subintervals: 1 M20K, 2M10K
- 2 DSPs
Faithfully rounded dividers
Implementation using Polynomial Approximation

**Single-precision**
- degree 2 polynomial on 256 subintervals: 1 M20K, 2M10K
- 2 DSPs

**Double-precision:**
- degree 5 poly. on 256 subintervals: 6M10K on AriaV/CycloneV
- degree 4 poly. on 1K subintervals: 19M20K on StratixV
- truncated datapath (6DSPs + logic)

*Horner Evaluation*

\[
\begin{align*}
18x18 &\rightarrow 20 \\
29x31 &\rightarrow 30 \\
38x40 &\rightarrow 40 \\
42x50 &\rightarrow 51
\end{align*}
\]
Faithfully rounded dividers
Implementation using combined techniques

For double precision:

- Start with initial polynomial approx. $2^{-28}$
  - Degree 2 poly. $2M20K/4M10K$.
  - 2 DSPs
- Perform one Newton-Raphson iteration
  - 28-bit squarer (1 DSP + logic)
  - $56 \times 53 \rightarrow 56$ (3 DSPs + logic)

Same number of DSPs as polynomial approximation, less memory blocks.
Outline

1 Background
   - Floating-point division
   - Division algorithms

2 Faithfully rounded dividers
   - Error analysis walk-through
   - Implementation discussion

3 Correctly rounded dividers
   - Obtaining correctly rounded dividers
   - The cost of correct rounding

4 Results

5 Conclusion
Correctly rounded dividers
Obtaining correctly rounded dividers

Technique:

- compute $\tilde{d} = \frac{x}{y}$ faithfully rounded on $w_F + 1$ fraction bits
- with respect to the target $w_F$ format, $\tilde{d}$ is either a FP number, or a midpoint.
Correctly rounded dividers

Obtaining correctly rounded dividers

Technique:
- Compute $\tilde{d} = x/y$ faithfully rounded on $w_F + 1$ fraction bits.
- With respect to the target $w_F$ format, $\tilde{d}$ is either a FP number, or a midpoint.

- If $\tilde{d}$ is a FP in $w_F$ format, then is the correct result.
- If $\tilde{d}$ is a midpoint, then compute $\tilde{d} \cdot y$ and compare with $x$.
  - If $\tilde{d} \cdot y > x$ then return $\text{trunc}(\tilde{d})$.
  - Else return $\tilde{d} + \text{ulp}$.

Technical detail for optimizing the computation is in the paper by Bogdan Pasca (ALTERA)
Correctly rounded dividers

Obtaining correctly rounded dividers

Technique:

- Compute $\tilde{d} = x/y$ faithfully rounded on $w_F + 1$ fraction bits.
- With respect to the target $w_F$ format, $\tilde{d}$ is either a FP number, or a midpoint.

If $\tilde{d}$ is a FP in $w_F$ format, then is the correct result.
If $\tilde{d}$ is a midpoint, then compute $\tilde{d} \times y$ and compare with $x$.
  - If $\tilde{d} \times y > x$ then return $\text{trunc}(\tilde{d})$.
  - Else return $\tilde{d} + \text{ulp}$.
Correctly rounded dividers

Obtaining correctly rounded dividers

**Technique:**
- Compute $\tilde{d} = x/y$ *faithfully rounded on* $w_F + 1$ fraction bits
- With respect to the target $w_F$ format, $\tilde{d}$ is either a FP number, or a midpoint.

- If $\tilde{d}$ is a FP in $w_F$ format, then it is the correct result.
- If $\tilde{d}$ is a midpoint, then compute $\tilde{d} \times y$ and compare with $x$.
  - If $\tilde{d} \times y > x$ then return $\text{trunc}(\tilde{d})$
  - Else return $\tilde{d} + \text{ulp}$.

**Technical detail for optimizing the computation** is in the paper.
Correctly rounded dividers
The cost of correct rounding

Requires:

1. a faithful division on wF+1 bits and
   - the combined techniques: 29-bit initial approx.
2. product $\tilde{d}_{wF+1} \times y$ with only the LSB wF+3 bits.
3. integer subtraction (making good use of internal adders)
Outline

1. Background
   - Floating-point division
   - Division algorithms

2. Faithfully rounded dividers
   - Error analysis walk-through
   - Implementation discussion

3. Correctly rounded dividers
   - Obtaining correctly rounded dividers
   - The cost of correct rounding

4. Results

5. Conclusion
Results

\[
\text{PA}(d) = \text{Polynomial Approximation of degree } d \\
\text{NR} = \text{Newton-Raphson iteration} \\
\text{R4 DR} = \text{Radix-4 Digit Recurrence}
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Acc.</th>
<th>Published</th>
<th>FPGA</th>
<th>Freq., Lat., Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4 DR</td>
<td>CR</td>
<td>FloPoCo</td>
<td>StratixV</td>
<td>233MHz, 16, 1210ALUT, 1308REG</td>
</tr>
<tr>
<td><strong>PA(2)</strong></td>
<td>FR</td>
<td>ours</td>
<td>StratixV</td>
<td>400MHz, 11, 274ALUT, 291REG, 2M20K, 3DSP</td>
</tr>
<tr>
<td></td>
<td>CR</td>
<td></td>
<td></td>
<td>400MHz, 15, 426ALUT, 408REG, 2M20K, 4DSP</td>
</tr>
<tr>
<td>R4 DR</td>
<td>CR</td>
<td>FloPoCo</td>
<td>StratixV</td>
<td>219MHz, 36, 5209ALUT, 5473REG</td>
</tr>
<tr>
<td>-</td>
<td>FR</td>
<td>FP_DIV</td>
<td>StratixV</td>
<td>196MHz, 24, 810ALUT, 1629REG, 9M20K, 14DSP</td>
</tr>
<tr>
<td><strong>PA(4)</strong></td>
<td>FR</td>
<td></td>
<td>StratixV</td>
<td>380MHz, 33, 1113ALUT, 1825REG, 10M20K, 9DSP</td>
</tr>
<tr>
<td><strong>PA(2) + NR</strong></td>
<td></td>
<td>ours</td>
<td></td>
<td>268MHz, 18, 887ALUT, 823REG, 2M20K, 9DSP</td>
</tr>
<tr>
<td><strong>PA(2) + NR</strong></td>
<td></td>
<td></td>
<td></td>
<td>400MHz, 25, 947ALUT, 1296REG, 2M20K, 9DSP</td>
</tr>
<tr>
<td><strong>Multiplicative 2ulp</strong></td>
<td></td>
<td>VII-Pro</td>
<td></td>
<td>275MHz, 36, 2097SLICE, 1 18KBRAM, 28DSP</td>
</tr>
</tbody>
</table>

Outline

1 Background
   • Floating-point division
   • Division algorithms

2 Faithfully rounded dividers
   • Error analysis walk-through
   • Implementation discussion

3 Correctly rounded dividers
   • Obtaining correctly rounded dividers
   • The cost of correct rounding

4 Results

5 Conclusion
approximation-based techniques provide fast and resource-balanced implementations

correctly-rounded dividers obtained at reduced cost

faithfully accurate divider \((wF + 1\) bits) \(\equiv\) correctly rounded on \(wF\) bits

IEEE-754 compliance is impossible if elementary functions are used: faithful dividers allow reducing implementation cost.

divider architectures: available via *Altera DSP Builder Advanced blockset* but also used by the Altera OpenCL initiative.