Correctly rounded floating-point division for DSP-enabled FPGAs

Bogdan Pasca

FPL 29-31 August 2012

Outline

Outline

Background
Floating-point division

Let x, y floating-point numbers in format $\mathbb{F}_{\mathsf{w_E},\mathsf{w_F}}$ with :

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

Let :

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

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

Background

Floating-point division

Let x, y floating-point numbers in format $\mathbb{F}_{\mathsf{w_E},\mathsf{w_F}}$ with :

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

Let :

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

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

Fixed-point division is the core of floating-point implementation

Background
Division algorithms

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

Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long lantency $\mathcal{O}(wF)$
  - 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

Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long lantency $\mathcal{O}(wF)$
  - 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
- **Polynomial approximation** for $1/Y$
  - general technique
  - piecewise-polynomial approximation for range-reduction

Background
Division algorithms

- **Digit-recurrence algorithms**
  - correctly rounded
  - long lantency $\mathcal{O}(wF)$
  - 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
- **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

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.

Faithfully rounded dividers

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

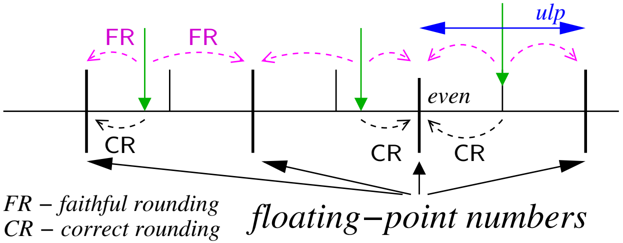
Faithful and correct rounding

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



Faithfully rounded dividers

Error analysis walk-through

Faithfully rounded result requires :

$$E_{\text{total}} = E_{\text{round}} + E_{\text{approx}} \leq 1ulp$$

Faithfully rounded dividers

Error analysis walk-through

Faithfully rounded result requires :

$$E_{\text{total}} = E_{\text{round}} + E_{\text{approx}} \leq 1ulp$$

- E_{round}- packing the result to the output format. (1/2ulp for RN)
- E_{approx}- sums the method and computational errors. (must be bounded by 1/2ulp)

Faithfully rounded dividers

Error analysis walk-through

Sequence of operations for fixed-point division :

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

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_{approx} :

$$\begin{aligned} |Q - Q'| &= |ZX - Z'X| \\ &= |(Z - Z')X| \\ &\leq |Z - Z'||X| \end{aligned}$$

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_{approx} :

$$\begin{aligned} |Q - Q'| &= |ZX - Z'X| \\ &= |(Z - Z')X| \\ &\le |Z - Z'||X| \end{aligned}$$

- as $X \in [1, 2) \rightarrow |Z - Z'| \le 1/4ulp$ but $Z \in (1/2, 1]$ so a faithful approximation on wF+3 bits is required.

Faithfully rounded dividers
Error analysis walk-through

Higher precisions allow saving DSPs using a truncated multiplier :

$$\begin{aligned} Z' &= \circ(1/Y) \\ P' &= Z' \times X \\ Q' &= trunc(P') \end{aligned}$$

Faithfully rounded dividers
Error analysis walk-through

Higher precisions allow saving DSPs using a truncated multiplier :

$$\begin{aligned} Z' &= \circ(1/Y) \\ P' &= Z' \times X \\ Q' &= \mathit{trunc}(P') \end{aligned}$$

The approximation error E_{approx} :

$$\begin{aligned} |Q - Q'| &= |ZX - \mathit{trunc}(P')| \\ &= |ZX - Z'X + Z'X - \mathit{trunc}(Z'X)| \\ &= |(Z - Z')X + \mathit{truncerrorforZ'X}| \\ &\leq |Z - Z'||X| + |\mathit{truncerrorforZ'X}| \end{aligned}$$

Faithfully rounded dividers

Error analysis walk-through

Higher precisions allow saving DSPs using a truncated multiplier :

$$\begin{aligned} Z' &= \circ(1/Y) \\ P' &= Z' \times X \\ Q' &= trunc(P') \end{aligned}$$

The approximation error E_{approx} :

$$\begin{aligned} |Q - Q'| &= |ZX - trunc(P')| \\ &= |ZX - Z'X + Z'X - trunc(Z'X)| \\ &= |(Z - Z')X + truncerrorforZ'X| \\ &\leq |Z - Z'||X| + |truncerrorforZ'X| \end{aligned}$$

- faithful approx. on $wF + 4$ for Z'
- faithful multiplier on $wF + 3$ for Z'X.

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

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

Faithfully rounded dividers

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

Two solutions :

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

Faithfully rounded dividers

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

Two solutions :

1. - bootstrap Z_0 accurate to 2^{-15}
   - perform two iterations, Z_1, Z_2
   - large memory requirement
2. - bootstrap with Z_0 accurate to 2^{-10}
   - perform 3 iterations (several optimizations possible)
     - $Z_1 = 2Z_0 - Z_0^2 Y$ 10-bit squarer + 20x53 mult. $\rightarrow$ 20 x 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

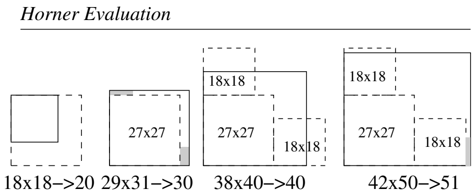
Implementation using Polynomial Approximation

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)



Faithfully rounded dividers
Implementation using combined techniques

For double precision :

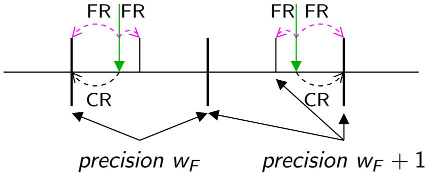
- start with initial polynomial approx. 2^{-28}
  - degree 2 poly. 2M20K/4M10K.
  - 2 DSPs
- perform one Newton-Raphson iteration
  - 28-bit squarer (1 DSP + logic)
  - $56 \times 53 \rightarrow 56$ (3 DSPs + logic)

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

Outline

Technique :

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

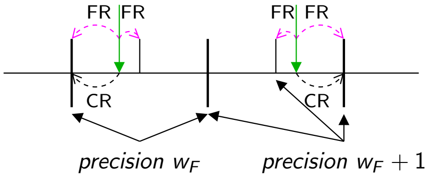


Correctly rounded dividers

Obtaining correctly rounded dividers

Technique :

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



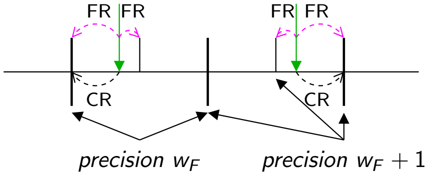
- if $\tilde{d}$ is a FP in wF format, then is the correct result

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 *wF* format, $\tilde{d}$ is either a FP number, or a midpoint.

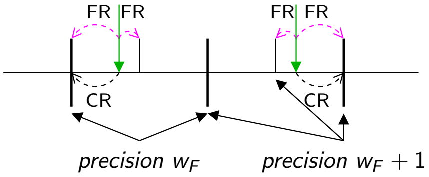


- if $\tilde{d}$ is a FP in *wF* format, then is the correct result
- if $\tilde{d}$ is a midpoint, then compute $\tilde{d} * y$ and compare with x.
  - if $\tilde{d} * y > x$ then return $trunc(\tilde{d})$
  - else return $\tilde{d}$ + ulp.

Correctly rounded dividers

Technique :

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



- if $\tilde{d}$ is a FP in *wF* format, then is the correct result
- if $\tilde{d}$ is a midpoint, then compute $\tilde{d} * y$ and compare with x.
  - if $\tilde{d} * y > x$ then return $trunc(\tilde{d})$
  - else return $\tilde{d}$ + 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

Results

PA(d) = Polynomial Approximation of degree d
NR = Newton-Raphson iteration
R4 DR = Radix-4 Digit Recurrence

Algorithm	**Acc.**	**Published**	**FPGA**	**Freq., Lat., Resources**
R4 DR	CR	FloPoCo	StratixV	233MHz, 16, 1210ALUT, 1308REG
PA(2)	FR	ours	StratixV	400MHz, 11, 274ALUT, 291REG, 2M20K, 3DSP
PA(2)	CR	ours	StratixV	400MHz, 15, 426ALUT, 408REG, 2M20K, 4DSP
Goldberg iterations + Booth Radix-8 Mult.	CR	[1]	StratixII	131MHz, 11/8, 5800ALUT, 3592ALM+12 M20K
R4 DR	CR	FloPoCo	StratixV	219MHz, 36, 5209ALUT, 5473REG
-	FR	FP_DIV	StratixV	196MHz, 24, 810ALUT, 1629REG, 9M20K, 14DSP
PA(4)	FR	ours	StratixV	380MHz, 33, 1113ALUT, 1825REG, 10M20K, 9DSP
PA(2) + NR	FR	ours	StratixV	268MHz, 18, 887ALUT, 823REG, 2M20K, 9DSP
PA(2) + NR	FR	ours	StratixV	400MHz, 25, 947ALUT, 1296REG, 2M20K, 9DSP
Multiplicative	2ulp	[2]	VII-Pro	275MHz, 36, 2097SLICE, 1 18KBRAM, 28DSP

[1] R. Goldberg, G. Even, and P.-M. Seidel, "An FPGA implementation of pipelined multiplicative division with IEEE rounding", FCCM 2007
[2] M. K. Jaiswal and R. C. C. Cheung, "High performance reconfigurable architecture for double precision floating point division", ARC 2012

Outline

Conclusion

- *approximation-based techniques* provide *fast and resource-balanced* implementations
- *correctly-rounded dividers* obtained at *reduced cost*
- faithfully accurate divider ($wF + 1$ bits) $==$ 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.