Multiplier-Based Double Precision Floating Point Divider According to the IEEE-754 Standard

Vítor Silva[1], Rui Duarte[1], Mário Véstias[2], and Horácio Neto[1]

[1] INESC-ID/IST/UTL, Technical University of Lisbon, Portugal
[2] INESC-ID/ISEL/IPL, Polytechnic Institute of Lisbon, Portugal

Abstract. This paper describes the design and implementation of a unit to calculate the significand of a double precision floating point divider according to the IEEE-754 standard. Instead of the usual digit recurrence techniques, such as SRT-2 and SRT-4, it uses an iterative technique based on the Goldsmith algorithm. As multiplication is the main operation of this algorithm, its implementation is able to take advantage of the efficiency of the embedded multipliers available in the FPGAs. The results obtained indicate that the multiplier-based iterative algorithms can achieve better performance than the alternative digit recurrence algorithms, at the cost of some area overhead.

1 Introduction

FPGAs are becoming a commonplace in the design of computational units to accelerate many applications. Some of these applications require the use of single or double floating point arithmetic. Therefore, hardware designers are looking for efficient solutions to implement floating point arithmetic in FPGAs using the available resources of the programmable hardware units. The works from [1], [2] and [3] are some of the first approaches implementing single precision (or lower) floating-point division in FPGA. [4] extended the previous work with a floating-point divider. [9] presents a divider core for both single and double precision. [10] describes a double precision floating-point core for division and other operations, that supports the complete IEEE-754 standard.

In the case of the floating point division operation, most implementations are based on the well-known digit recurrence algorithms. However, today's FPGAs include a significant number of built-in multipliers, which makes multiplication-based division algorithms increasingly interesting.

In this paper, a new architecture to calculate the division of the significands of a double precision floating point divider according to standard IEEE-754 is presented. The architecture is completely pipelined and is based on the Goldsmith method for the calculation of division, instead of the more usual NRD or SRT4 algorithms. The method is based on successive iterations and calculates the division with better performance than previous approaches.

Section 2 describes the Goldsmith iterative division algorithm. Section 3 is concerned with the generation of the seed. Section 4 presents the architecture

R. Woods et al. (Eds.): ARC 2008, LNCS 4943, pp. 262–267, 2008.

of the divisor. Section 5 describes the implementation results. Finally, section 6 ends the document with the conclusions.

2 Iterative Algorithm for Division

The Goldsmith method for division uses the Taylor series expansion to calculate the reciprocal of the denominator (divisor), that is, $q = \frac{N}{D} = N \times g(y)$, such that $g(y)$ may be efficiently calculated using an iterative method.

A straightforward approach is to consider g(y) = $1/y$ with p = 1 and then calculate the series. However, it is computationally more efficient to consider the Maclaurin series g(y) = $1/(y + 1)$ (with p = 0). In this case, g(y) is given by

$$g(y) \approx \frac{1}{(1+y)} = 1 - y + y^2 - y^3 + y^4 + \cdots \tag{1}$$

To obtain $\frac{1}{D}$ from g(y), y must be replaced by D - 1, with D normalized such that $0,5 \leq D$ and $|y| \leq 0,5$. In this case, the quotient is given by:

$$q = N \times [(1-y)(1+y)^2(1+y)^4(1+y)^8...] \tag{2}$$

This equation can be calculated iteratively, considering that an approximation to the quotient can be calculated from $q_i = \frac{N_i}{D_i}$, where N_i and D_i are the values of the numerator and the denominator after iteration i of the algorithm.

The Goldsmith algorithm starts with $N_0 = N$ and $D_0 = D$. Then, in the first iteration both N_0 and D_0 are multiplied by $R_0 = 1 - y = 2 - D$ to generate a new approach for the numerator, N_1, and for the denominator, D_1.

Generically, the iterative process calculates the following recurrent equations:

$$N_{i+1} = N_i \times R_i, D_{i+1} = D_i \times R_i, R_{i+1} = 2 - D_{i+1} \tag{3}$$

where N and D have quadratic convergence to q and 1, respectively.

To reduce the number of iterations needed to achieve a certain precision, a better precision for R_0 may be used, which is usually designated seed. Based on this idea, [6] proposed a modified Goldsmith algorithm that uses an initial seed with enough precision to guarantee that a single iteration is sufficient to obtain double precision. In this case, equations (3) are replaced by the equations,

$$\mathbf{G} = R_0 \times N_0, \mathbf{V} = 1 - R_0 \times D_0, \mathbf{Q} = G + G \times V \tag{4}$$

which are mathematically equivalent but computationally more efficient.

3 Seed Generation

The seed is computed using an efficient algorithm based on a second degree polynomial approximation, similar to the one proposed by Piñeiro [7] with the modifications proposed by Muller [5]. The seed is calculated using second order polynomial approximations calculated with the *Minimax* technique. The domain

of the reciprocal is first divided into a power of two number of intervals and then the two order polynomial approximation is found for each interval.

Assuming all operands are in IEEE-754 normalized form, in order to implement the 2^{nd} degree minimax polynomial approximation the significand of the input of the seed generator is split into an upper part $X_1 = [1.x_1x_2..x_{m1}]$, a middle part $X_2 = [x_{m1+1}..x_{m2}] \times 2^{-m_1}$ and a lower part $X_3 = [x_{m2+1}..x_n] \times 2^{-m_2}$.

An approximation to the reciprocal in the range $X_1 \leq X < X_1 + 2^{-m_1}$ can be found by evaluating the polynomial:

$$X^p \approx C_0 + C_1 \times X_2 + C_2 \times X_2^2 \tag{5}$$

The coefficients C_0, C_1 and C_2 are computed for each of the 2^{m_1} intervals. So, a different polynomial is used for each interval. X_3 does not play any part in the computation of the approximation.

The coefficients are then are stored in three lookup tables, each with 2^{m_1} entries, one for each of the above mentioned segments. Each table is addressed by the m_1 most significant bits of the fractional part of the significand.

To determine the set of coefficients, the following steps are considered:

1. Step 1 - Given the desired precision (ϵ_d) the space of the 2^{nd} degree minimax polynomials approximation must be swept to find a good candidate for the number of segments that the approximation function must be split into.
2. Step 2 - Find the appropriate word length for C_0, C_2, X_2, and the number of bits of X_2 used as inputs to the squarer unit.
3. Step 3 - Fill in the tables of coefficients.

Step one. uses a semi-automatic iterative procedure in which a script is executed with the following inputs:

1. Word length of C_1, denoted k from now on.
2. Number of bits used to address the tables, m_1.

For each pair (k,m_1), the algorithm determines the coefficients C_0, C_1, and C_2 with the computer algebra system Maple, which performs the minimax approximations using the Remez algorithm. Then the Partially Rounded (PR) approach from Muller is applied. According to PR, C_1 is rounded to k bits and, from $C_{1rounded}$, the new values of C_0, and C_2 are found as follows [5]:

$$C_{0new} = C_0 + (C_1 - C_{1rounded}) \times 2^{-m_1-3} \tag{6}$$

$$C_{2new} = C_2 + (C_1 - C_{1rounded}) \times 2^{m_1} \tag{7}$$

The difference between the polynomial approximation with both sets of values is the error of the approximation with the pair of values (k,m_1).

The script is run as many times as needed to find a good candidate, that is, a tuple (k,m_1) for which the accuracy found with the PR approach is greater than the desired accuracy, ϵ_d.

Step two. consists in finding the appropriate word length for the parameters of the polynomial. Once the value for the tuple (k,m_1) and the accuracy (ϵ_{pr})

for the PR approximation is computed, the missing parameters may be easily figured out. Due to the rounding of C_0, C_2 and X_2 in the PR approximation, the final accuracy is lower than ϵ_{pr}. Therefore, we try to achieve an accuracy $(\epsilon'_{pr}) = (\epsilon_{pr}) - 1$, that is, an error less than $2^{-\epsilon'_{pr}}$. Formally, we want an error on the computation of $C_{0new} + C_{2new} \times (X - X_1)^2 \leq 2^{-\epsilon'_{pr}}$ (from now on we will refer to $(X - X_1)$ as h). Algebraically, we want to guarantee that

$$|C_{0new} - C_{0newRounded}| + \left|C_{2new} \times h^2 - C_{2newRounded} \times (h + h_{error})^2\right| \leq 2^{-\epsilon'_{pr}} \tag{8}$$

Given that $h \leq 2^{-m_1}$ then the word lengths of C_0 ($C_{0_{WL}}$), C_2 ($C_{2_{WL}}$), h (h_{WL}), and X_2 ($X_{2_{WL}}$) are as follows:

$$C_{0_{WL}} = \epsilon'_{pr} + 1 \tag{9}$$
$$C_{2_{WL}} = \epsilon'_{pr} + 2 - 2 \times m_1 \tag{10}$$
$$C_{h_{WL}} = \epsilon'_{pr} + 4 - 2 \times m_1 \tag{11}$$
$$X_{2_{WL}} = \epsilon'_{pr} + 2 - m_1 \tag{12}$$

C_0 will have 1 bit for the integer part and $C_{0_{WL}}$ for the fractional part.

Step three. A script in Maple receives the size of the tables (m_1), and the word length of C_0, C_1 and C_2 and generates the tables.

4 Architecture to Calculate the Significand

The architecture to calculate the significand consists in the unit to determine the seed and the unit to calculate the significand with double precision.

Using the rounding criteria defined in [8] and knowing that the Goldsmith algorithm has quadratic convergence, we conclude the seed must have at least 29 bits of precision so that a single iteration is enough to complete the division.

Figure (1) shows the hardware structure of the circuit used to calculate X^{-1}. The circuit has three coefficient tables, two multipliers, one squarer, one adder and one subtractor. The size of each coefficient and of the arithmetic units were obtained with the process described previously with $\epsilon'_{pr} = 29$.

From the seed, equations (4) are used to calculate the significand. Figure (2) shows the structure of the unit to calculate the significand. As shown, the Add/Sub operator implements an adder/subtractor controlled by bit 28 of V. If V$<28>$ is '0' then the adder unit is used. Otherwise, the operation is a subtraction. Thus, it was possible to reduce the size of the operands of (4) and the product $G \times V$, that appears at the second term of equation (4). This is due to the fact that the seed has 29 bits of precision, which guarantees that $|V| \leq 2^{-29}$ and,therefore, at least the most significant 29 bits from the equation (4) are fixed. Hence, it is possible to reduce 28 bits in the two's complement operator and, in the case of the multiplication G $\times$ V it allows the replacement of a 57 $\times$ 57 multiplication for a 57 $\times$ 28 multiplication.

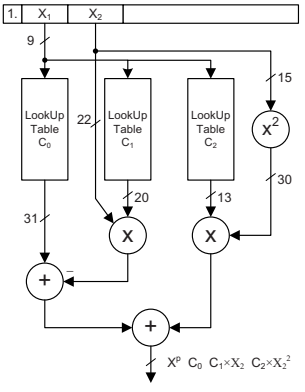

Fig. 1. Hardware to compute the seed

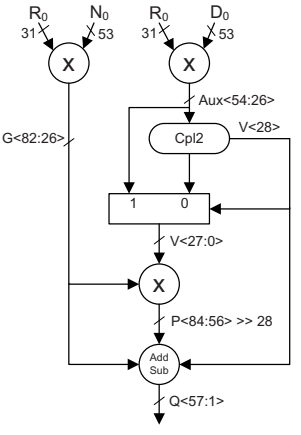

Fig. 2. Hardware to compute Q

5 Implementation and Results

The architecture was specified in VHDL, synthesized with Xilinx ISE8.2 and then implemented in a Xilinx Virtex-2 6000 FPGA [13] with 144 multipliers included in a Celoxica ADMXRC2 board.

From Table 1 we observe that our circuit calculates the significand in 33 cycles at more than 350 MHz. Also observing the results of a straightforward approach for the pre and postprocessing circuits we conclude the complete circuit can achieve a frequency of 325 MHz in about 70 cycles. The performance of the circuit is very good when compared to other recent approaches (see table 2).

Table (2) includes the works from [9], [10], [11] and [12]. The first two were obtained with a Virtex 2/2P, while the others were obtained with a Virtex-4. All algorithms use the digit-recurrence technique to calculate the significand, except ours and the Xilinx IP core 1.0, which are iterative.

When compared to the works of [9] and [10], ours is much faster with about the same number of cycles. While the former approach does not support exceptions and only supports rounding to nearest, the second one supports the complete standard. The main difference to our approach has to do with the algorithm used to calculate the significand.

Table 1. Implementation of our circuit in a Virtex2$^{\mathrm{TM}}$post-$P\&R$

Stages	Slices	FFs	MULT18×18s	Mhz	Latency
Preprocessing	1922	3220	0	325 MHz	12
Seed	2170	3802	3	372 MHz	11
Significand	6900	12331	28	361 MHz	22
Post processing	1549	2635	0	372 MHz	18

Table 2. Comparison with our approach

Author	Algorithm	Mhz	Latency
Underwood	RD	83	67
Prasanna	RD	140	68
Xilinx Div 1.0	Iter	284	99
Xilinx FP 3.0	RD	266	57
Ours	Iter	325	70

6 Conclusions

The significand divider implemented in this work based on the Goldsmith algorithm is very competitive when compared to algorithms based on digit recurrence. The proposed circuit achieves higher frequencies than previous works with a similar number of cycles.

The next step of the project is to optimize the architecture to reduce the latency of the circuit. Also, the complete double precision floating point divider according to standard IEEE-754 is already being tested in a Virtex-II platform.

References

1. Wang, X., Leeser, M.: Variable Precision Floating Point Division and Square Root. In: 8th Annual High Performance Embedded Computing Workshop (October 2004)
2. Lienhart, G., Kugel, A., Manner, R.: Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations. In: FCCM 2002 (April 2002), pp. 182–191
3. Liang, J., Tessier, R., Mencer, O.: Floating point unit generation and evaluation for FPGAs. In: FCCM 2003 (April 2003)
4. Daga, V., Govindu, G., Prasanna, V.K., Gangadharpalli, S., Sridhar, V.: Floating-point based block LU decomposition on FPGAs. In: ERSA 2004(June 2004)
5. Muller, J.-M.: Partially rounded. Small-Order Approximations for Accurate Hardware-Oriented, Table-Based Methods, Reserach Report 4593, INRIA (2002)
6. Piñeiro, J.A., Bruguera, J.D.: High-Speed Double-Precision Computation of Reciprocation and Division. In: DCIS 2001 (November 2001)
7. Piñeiro, J.A., Bruguera, J.D., Muller, J.M.: Faithful Powering Computation Using Table Look-Up and a Fused Accumulation Tree. In: ARITH 2001 (June 2001)
8. Stuart, F.: Oberman and Michael J. Flyn, Fast IEEE rounding for division by functional iteration, TR CSL-TR-96-700, DEECS, Stanford University (1996)
9. Underwood, K.: FPGA vs. CPUs: Trends in Peak Floating Point Performance. In: FPGA 2004 (February 2004)
10. Govindu, G., Scrofano, R., Prasanna, V.: A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing. In: ERSA 2005 (June 2005)
11. `www.xilinx.com/bvdocs/ipcenter/data/_sheet/div/_gen/_ds530.pdf`
12. `http://www.xilinx.com/bvdocs/ipcenter/data_sheet/floating_point_ds335.pdf`
13. ADM-XRC-II Reconfigurable Computer Documentation, Alpha-Data (2002), `http://www.alphadata.com/adm-xrc-ii.html`