Research Paper

FPGA IMPLEMENTATION OF FLOATING POINT RECIPROCATOR

B. Sujatha[1], D. Raghunatha Rao[2]

Address for Correspondence

[1]P.G.Student, [2]Associate Professor, Dept. of ECE, RGMCET, Nandyal, A.P, India

ABSTRACT

In this paper an efficient FPGA implementation of a reciprocator for both IEEE single-precision and double-precision floating numbers is presented. This method is based on the use of look-up-tables (LUTs) and partial block multipliers. Previously the LUTs and Multipliers are mostly used, in these methods accuracy is not achieved and area occupied by them is also more. In the proposed method, number of LUTs and multipliers are reduced such that performance in terms of frequency and accuracy is improved and also latency is reduced. The designs trade off either 1 unit in last –place (ulp) or 2 ulp of accuracy (for double or single precision respectively), without rounding, to obtain a better implementation. Rounding can also be added to the design to restore some accuracy at a slight sacrificing in area.

KEY WORDS: Floating-point arithmetic, reciprocator, FPGA, double- precision, partial block multipliers, binomial expansion.

I. INTRODUCTION

Floating point arithmetic is widely used in many scientific and signal processing applications. The greater range and need to lack of scale the numbers make development of algorithms much easier. However, implementing arithmetic operations for floating point numbers in hardware is very challenging. Among the operations (add, subtract, multiply, divide), division is generally the most difficult to implement in hardware. Division is a fairly common operation in many scientific and signal processing applications, so there is a need of hardware implementation for division.

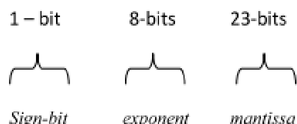
The IEEE standard for floating point (IEEE-754) defines the format of the numbers, and also specifies various rounding modes that determine the accuracy of the result. For many signal processing, and graphics applications, it is acceptable to trade off some accuracy [1] (in the least significant bit positions) for faster and better implementations.

A lot of work has been done on obtaining efficient implementations for this operation. Generally, this operation is done in two parts, first consider the inverse of divisor and then multiply with dividend. Because of this, many hardware dividers focus on efficiently obtaining the reciprocal of floating-point number. Most of the existed architectures in the literature are based on Newton-Raphson method [2],[3],[4],[6]&[7] digit recurrence method [2],[5],[7]&[8],seed-architecture [9]etc. Previous works had used huge look-up-tables, along with wider multipliers, which affects the area and performance. Our approach discussed in the paper also focused on finding the reciprocal. It is based on the well known binomial expansion, contains small look-up table and uses partial block multipliers, resulting less area, less delay, and correct up to required level (accuracy trade off). Here it is restricted to only normalized numbers. To implement this Xilinx ISE-8.2 [10] synthesis tool. Modelsim SE 6.1b simulation tool and X2VP30-7ff896 are used.

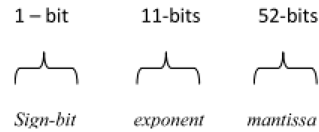
II. APPROACH

The format of a floating-point number is as follows

For Single-Precision



For Double –Precision



The benefits of proposed implementation are in the computation of the inverse of the mantissa.

Let y be the inverse of the mantissa in *a*. Then, $y = \frac{1}{1.a}$, *where in 1.a, 1 is hidden bit of mantissa.*

We have divided the mantissa in two parts,a_1 and a_2.a_1 is used to fetch some pre-calculated data from a look-up-table.

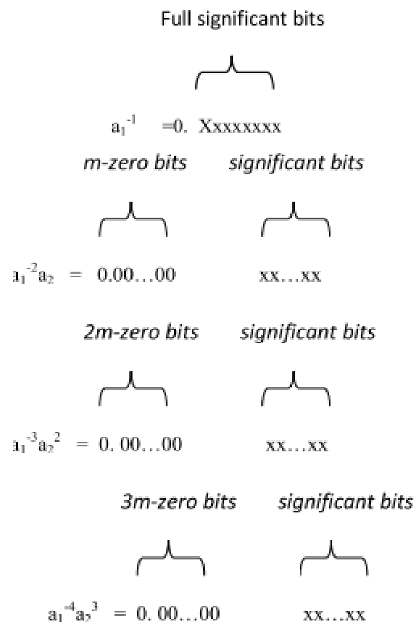
Now, since

$$y = \frac{1}{(a_1 + a_2)}$$

$= (a_1+a_2)^{-1}$

$=a_1^{-1}-a_1^{-2}.a_2+a_1^{-3}.a_2^{2}-a_1^{-4}.a_2^{3}+\dots$ (1)

The content of each term of equation (1) will be as follows



….and so on

Where m is the number of bits of a_1.

Moving towards higher terms their contribution to main result is decreasing. Thus, depending upon precision we can choose suitable number of terms from equation(1) for calculating inverse term, based on value of *m*.

In this implementation, based on experiments over a large number of random test cases, we have chosen the number of terms as described below. In case of single-precision first three terms are considered, while for the case of double precision 7 terms have been taken. The value of m is chosen as 8 for both cases. These values are selected based on available FPGA resources of simplified the desired terms in such a way that the use less hardware with low latency and good accuracy.

For single precision we have taken all the three terms as,

$$y = a_1^{-1} - a_1^{-2}.a_2 + a_1^{-3}.a_2^2 \quad (2)$$

For double precision, simplified form will be as,

$$y = a_1^{-1} - a_1^{-1}[(a_1^{-1}.a_2 - a_1^{-2}.a_2^2)\ (1 + a_1^{-2}.a_2^2 + a_1^{-4}.a_2^4)] \quad (3)$$

The simplification of above equations will affect the area, latency and accuracy. The accuracy is affected due to the fact that floating-point operations are not completely associative.i.e. u (v+w) may not be exactly equal to (uv+uw).This is due to the finite number of bits used to represent the numbers.

III. IMPLEMENTATION

We have shown the implementations for single precision and double precision separately as different issues arise in each case.

Some of the designed decisions are based on the fact that multipliers of size 18x18 are readily available as hard IP cores on many common FPGA families. We have based our computations on the Xilinx Virtex II platform. However, the basic ideas of saving some of the block multiplications hold even if a different sized multiplier core is used, although the exact numbers would change.

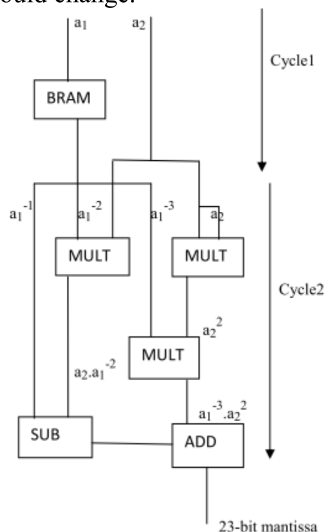


Fig.1. Architecture for single-precision floating-point reciprocator

A. Single-precision Floating-point

The architecture of single precision floating point reciprocator is shown in Fig.1.It includes a Block-Memory(BRAM) which contains pre-calculated values of a_1^{-1}(24-bits),a_1^{-2}(17-bits),and a_1^{-3}(17-bits) in a single data word(58 bits),with 8-bit (content of a_1) as address bits. The contents of the BRAM have been calculated using a separate program written in C, with floating data type for the numbers. The content of a_1^{-1} has been extended to 30-bits (by appending 6-bits "111111" at least significant bit (LSBs)) for addition/subtraction purpose. Here we can also do above operation with only value of a_1^{-1}, but it will increase the total operation latency and size of multipliers.

The architecture has latency of two though we can include the BRAM access in the first stage with a slight loss in maximum operating frequency. By using pipelined multiplier the overall frequency approximately, doubles the result with the latency two is shown.

B.Double-precision Floating-point

The architecture of double-precision floating point reciprocator is shown in Fig.2..It also includes BRAM which contains pre-calculated values of only a_1^{-1}(54-bits) with 8-bit (content of a_1) as address bits. The content of BRAM has been calculated using a C-program, with double as data-type of floating-point numbers. The content of a_1^{-1} has been extended to 60-bits (by appending 6-bits "111111" at LSB's) for addition/subtraction purpose. By this there is saving in Block memory.

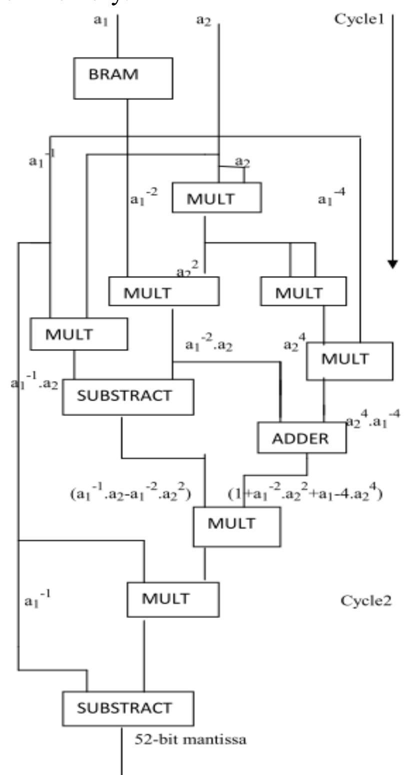


Fig.2. Architecture for Double-precision floating-point reciprocator

Overall latency of module is two with a slight loss in maximum operating frequency. By using pipelined multiplier we can approximately double the overall frequency. The result with latency TWO is shown.

IV. RESULTS

Hardware utilization and performance of both the single-precision and double-precision is shown in Table-I.

TABLE I: HARDWARE UTILISATION AND PERFORMANCE TABLE

Parameters	Single-precision	Double-precision
MULT18X18	1	16
BRAM	1	1
Slices	87	120
Freq[MHz]	217	257
Latency	2	2

Since our implementation neglects some of the lower order bits in the computation, it is important to estimate the impact of this on overall accuracy of results. For the error performance 5 millions

randomly generated test cases were used to check the errors. The error performance is shown in Table-II for both versions of floating-point numbers. The error was obtained by comparing results from the proposed module with the results produced by a C compiler on a workstation. In all cases, it was found that the maximum error in the case of single precision was 2 ulp (unit in last place), while in the case of double precision numbers, it was 1 ulp. The error obtained is without rounding.

TABLE II ERROR PERFORMANCE

Error	Singleprecision	Doubleprecision
Max ULP	2	1
Mean	4.7867e-0.8	7.7636e-1.7
Mean(absolute)	5.0060e-0.8	7.8125e-1.7
Variance	2.5959e-1.5	7.5177e-3.3
variance(absolute)	2.3812e-1.5	7.4415e-3.3

V. CONCLUSION

Implementation of reciprocal unit on FPGA for both single-precision and double-precision floating point numbers is done. The hardware was realized on XILINX FPGA kit. The processor was coded with Verilog and simulated using Modelsim 6.5e.[9].

The implementation can thus form a useful core for use in hardware dividers, especially for applications like signal processing that could be more tolerant of inaccuracies in the least significant bits.

REFERENCES

1. J. Hop, “A parameterizable HandleIC divider generator for FPGAs with embedded hardware multipliers”, *IEEE International Conference on Field-Programmable Technology*, Pages 355-358, Dec-2004.
2. P. Montuschi, L.Ciminiera, A. Giustina, “Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient”, *IEE Proceedings - Computers and Digital Techniques*, Issue 6, Vol. 141, Pages 317 - 324, Nov-1994.
3. M. Ito, N.Takagi, and S.Yajima, “Efficient Initial Approximation and Fast Converging Methods for Division and Square Root”, *Proceedings of the 12th Symposium on Computer Arithmetic*, Pages 2-9, July-1995.
4. Milos D. Ercegovac, Tomas Lang, Jean-Michel Muller, Arnaud Tisserand, “Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers”, *IEEE Transactions on Computers*, Issue 7, VOL. 49, July-2000.
5. Xiaojun Wang, B. E. Nelson, “Tradeoffs of designing floating-point division and square root on Virtex FPGAs”, *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM2003)*, Pages 195- 203, Apr-2003.
6. U. Kucukkabak, A. Akkas, “A Combined Interval and Floating-Point Reciprocal Unit”, *Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, pages 1366- 1371, Nov-2005.
7. E. Antelo, T. Lang, P. Montuschi, A. Nannarelli, “Low latency digit recurrence reciprocal and square-root reciprocal algorithm and architecture”,*17th IEEE Symposium on Computer Arithmetic*, Pages 147-154,June-2005.
8. H. Bessalah, M. Anane, M. Issad, N. Anane, K. Messaoudi, “Digit recurrence divider: Optimization and verification”, *International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, Pages 70-75, Sept-2007.
9. Stratix II vs. Virtex-4 Density Comparison. [Online]. Available: http://www.altera.com/literature/wp/wpstxiixlnx.pdf
10. Xilinx Floating-point unit v2.0. [Online]. Available: www.xilinx.com